

تم تحميل الملف من موقع
البوصلة التقنية
www.boosla.com

المملكة العربية السعودية
المؤسسة العامة للتدريب التقني والمهني
الكلية التقنية بالرياض

بكالوريوس هندسة تقنية

قسم تقنية الحاسب

ترجمة كتاب:

تركيب البيانات باستخدام C++

دي اس ماليك

برنامج تدريبي في التكنولوجيا
طومسون

طومسون
برنامج تدريبي في التكنولوجيا

تركيبات البيانات باستخدام C++

دي اس ماليك

طومسون
برنامج تدريبي في التكنولوجيا
تركيبات البيانات باستخدام C++
دي اس ماليك

| | | |
|---------------------|--------------------|------------------|
| كبير المحررين: | مساعد التحرير: | مصمم الغلاف: |
| جنيفر موروف | كريستي أوربان | ستيف ديشيني |
| مدير التحرير: | محرر الانتاج: | المؤلف: |
| جنيفر لوك | ايمي بورير | خدمات جيكس للنشر |
| محرر تطويري: | مدير تسويق المنتج: | المنسق التصنيعي: |
| سوزان جلبرت، ايديكس | انجي لاغلين | لورا برنز |
| مدير المنتج: | | |
| جانيت أراس | | |

حقوق الطبع © 2003 برنامج التكنولوجيا التدريبي، قسم من شركة طومسون التعليمية. شركة طومسون التعليمية علامة تجارية يتم استخدامها هنا تحت الترخيص.

© builder C++ علامة تجارية مسجلة لشركة بورلاند الدولية.

© codeWarrior علامة تجارية مسجلة لشركة موتورولا © Metrowerks. العلامات 6.0 C++ © visual

و visual Studio و © NET و PowerPoint علامات تجارية مسجلة لشركة مايكروسوفت.

© pentium علامة تجارية مسجلة لشركة انتل و © IBM علامة تجارية مسجلة لماكينات الأعمال الصناعية.

جميع الشخصيات الكرتونية والأفلام المتحركة المشار إليها في هذا الكتاب علامات تجارية مسجلة لشركة والت ديزني.

تم الطبع في كندا

WC 07 06 3 4 5 6 7 8 9

لمزيد من المعلومات اتصل ببرنامج التكنولوجيا التدريبي في 25 طومسون في بوسطون بولاية ماساشوسيتس 02210.

أو جدنا على شبكة الانترنت على: www.course.com

جميع الحقوق محفوظة. لن يتم إعادة انتاج أو استخدام أي جزء من هذا العمل المغطى بحق الطبع بأي صيغة أو أي وسيلة – تصويرية، أو الكترونية، أو آلية بما فيها التصوير والتسجيل والتوزيع الشبكي أو تخزين المعلومات وأنظمة الاستعادة – دون اذن كتابي من الناشر.

من أجل الحصول على اذن باستخدام هذا الكتاب أو المنتج اتصل بنا على:

هاتف: (800) 730-2214

فاكس: (800) 730-2215

www.thomsonrights.com

تنازل:

يحتفظ برنامج التكنولوجيا التدريبي بحق مراجعة هذا المنشور وعمل تغييرات من حين لآخر في محتواه دون اخطار. البرامج في هذا الكتاب موجودة لأهداف تعليمية فقط. لقد تم اختبارها بحرص ولا يتم ضمانها ضد أي قصد محدد بخلاف أهداف تعليمية. لا يقوم الكاتب والناشر بتقديم أية ضمانات أو تمثيلات ولا يقبلوا أي مسؤوليات بشأن البرامج.

ISBN 0-619-15907-3

محتويات

تمهيد:

| | |
|-----|--|
| 25 | 1. مبادئ هندسة البرمجيات وفئات C++ |
| 27 | 2. التصميم والقائم على الهدف وC++ |
| 161 | 3. المؤشرات والقوائم المبنية على المصفوفات |
| 234 | 4. مكتبة القالب المعياري |
| 303 | 5. القوائم المتصلة |
| 394 | 6. التكرار |
| 433 | 7. المرصوصات |
| 490 | 8. الطوابير |
| 548 | 9. خوارزميات البحث |
| 584 | 10. خوارزميات الترتيب |
| 644 | 11. الأشجار الثنائية |
| 727 | 12. الأشكال البيانية |
| 777 | 13. مكتبة القالب المعياري 2 |
| 859 | الملحق أ - الكلمات المحفوظة |
| 860 | الملحق ب - أسقية المعامل |
| 861 | الملحق ج - مجموعات الرموز |
| 863 | الملحق د - ائقال المعامل |
| 864 | الملحق هـ - الملفات الرئيسية |
| 870 | الملحق و - موضوعات C++ اضافية |
| 880 | الملحق ز - C++ لمبرمجي الجافا |
| 903 | الملحق ح - المرجعيات |
| 905 | الملحق ط - حلول تمارين مختارة |
| 919 | الفهرست |

جدول المحتويات

تمهيد

| | |
|----|--|
| 25 | 1. مبادئ هندسة البرمجيات وفئات C++ |
| 26 | دورة حياة البرنامج |
| 27 | مرحلة تطوير البرنامج |
| 27 | التحليل |
| 28 | التصميم |
| 29 | التنفيذ |
| 30 | الاختبار والتصحيح |
| 32 | تحليل الخوارزمية: ترميز O الكبيرة |
| 37 | الفئات |
| 39 | رسومات لغة التشكيل الموحدة |
| 40 | اعلان (هدف) المتغير |
| 41 | تناول عناصر الفئة |
| 45 | عمليات مدمجة على الفئات |
| 45 | معامل الاسناد والفئات |
| 47 | مجال الفئة |
| 48 | الدالات والفئات |
| 49 | عوامل الاشارة وأهداف (متغيرات) الفئة |
| 50 | تنفيذ دالات العنصر |
| 52 | المقومات |
| 57 | الدممرات |
| 57 | البنيات |
| 59 | ازالة البيانات والفئات وأنواع البيانات المجردة |
| 62 | اخفاء المعلومات |
| 68 | مثال برمجة: ماكينة الحلوى |
| 69 | تعريف الفئات والأهداف والعمليات |
| 72 | مراجعة سريعة |
| 75 | تمارين |
| 83 | تمارين برمجة |

| | |
|-----|--|
| 92 | التصميم القائم على الهدف وC++ |
| 92 | التوريث |
| 95 | اعادة تعريف دالت عنصر الفئة الأساسية |
| 98 | مقومات الفئات المستمدة والأساسية |
| 104 | الملف الرئيسي للفئة المستمدة |
| 103 | ادراجات متعددة لملف رئيسي |
| 106 | عناصر الفئة المحمية |
| 109 | التوريث العام والمحمي والخاص |
| 119 | التكوين |
| 110 | تعدد الأشكال: ائقال المعامل والدالة |
| 111 | ائقال المعامل |
| 112 | سبب الحاجة الى ائقال المعامل |
| 113 | ائقال المعامل |
| 114 | تركيب دالات المعامل |
| 115 | ائقال معامل: بعض القيود |
| 116 | المؤشر this |
| 120 | الدالات الصديقة للفئات |
| 122 | دالات المعامل كدالات عنصر وكدالات غير عنصر |
| 123 | ائقال عوامل ثنائية |
| 125 | ائقال عوامل ثنائية (رياضية أو مترابطة) كدالات عنصر |
| 126 | ائقال عوامل ثنائية (رياضية أو مترابطة) كدالات غير عنصر |
| 127 | ائقال عوامل ادخال التدفق (<<) واستخلاص التدفق (>>) |
| 127 | ائقال عامل ادخال التدفق (<<) |
| 128 | ائقال عامل استخلاص التدفق (>>) |
| 130 | ائقال عوامل أحادية |
| 132 | ائقال المعامل: العنصر مقابل اللا عنصر |
| 135 | تمرين برمجة: الأعداد المركبة |
| 136 | ائقال الدالة |
| 136 | القوالب |
| 142 | قوالب الدالة |
| 146 | قوالب الفئة |

| | |
|-----|--|
| 148 | الملف الرئيسي وملف تنفيذ قالب الفئة |
| 148 | مراجعة سريعة |
| 152 | تمارين |
| 158 | تمارين برمجة |
| 161 | 3. المؤشرات والقوائم المبنية على المصفوفات |
| 162 | أنواع بيانات المؤشر ومتغيرات المؤشر |
| 165 | اعلان متغيرات المؤشر |
| 166 | عنوان العامل (&) |
| 168 | عنوان ازالة الاشارة (*) |
| 170 | الفئات والبنيات ومتغيرات المؤشر |
| 173 | تهيئة متغيرات المؤشر |
| 175 | متغيرات حركية |
| 176 | عمليات على متغيرات المؤشر |
| 179 | مصفوفات حركية |
| 180 | دالات ومؤشرات |
| 181 | النسخ السطحي والعميق والمؤشرات |
| 183 | الفئات والمؤشرات: بعض الخصائص |
| 185 | المدمر |
| 187 | معامل الاسناد |
| 190 | اثقال معامل الاسناد |
| 193 | مقوم النسخ |
| 195 | اثقال معامل مؤشر المصفوفة ([]) |
| 196 | تمرين البرمجة: newString |
| 180 | قوائم مبنية على المصفوفات |
| 195 | التركيب الزمني لعمليات القائمة |
| 198 | مثال البرمجة: عمليات متعددة الحدود |
| 201 | مراجعة سريعة |
| 228 | تمارين |
| 230 | تمارين برمجة |
| 243 | 4. مكتبة القالب المعياري |
| 243 | مكونات مكتبة القالب المعياري |

| | |
|-----|--|
| 243 | أنواع الحاوية |
| 244 | الحاويات المتتابة |
| 245 | الحاوية المتتابة: vector |
| 246 | اعلان عناصر الحاوية vector |
| 247 | اعلان مكرر داخل حاوية vector |
| 255 | دالات عنصر مشتركة بين جميع الحاويات |
| 228 | دالات عنصر مشتركية بين الحاويات المتتابة |
| 229 | خوارزمية النسخ |
| 231 | مكرر ostream والدالة copy |
| 233 | الحاوية المتتابة: deque |
| 260 | المكررات |
| 274 | أنواع المكررات |
| 281 | مثال البرمجة: تقرير الدرجات |
| 286 | مراجعة سريعة |
| 290 | تمارين |
| 297 | تمارين البرمجة |
| 303 | 5. القوائم المتصلة |
| 304 | القوائم المتصلة |
| 305 | القوائم المتصلة: بعض الخصائص |
| 307 | اجتياز قائمة متصلة |
| 311 | ادخال وحذف العنصر |
| 313 | الادخال |
| 315 | الحذف |
| 317 | بناء قائمة متصلة |
| 319 | بناء قائمة متصلة من الأمام |
| 320 | بناء قائمة متصلة من الخلف |
| 320 | القائمة المتصلة كنوع بيانات مجرد |
| 322 | المقوم الافتراضي |
| 322 | تدمير القائمة |
| 323 | تهيئة القائمة |
| 324 | اثقال عامل ادخال التدفق |

| | |
|-----|--|
| 325 | طول القائمة |
| 326 | استعادة بيانات العقدة الأولى |
| 328 | استعادة بيانات العقدة الأخيرة |
| 330 | بحث القائمة |
| 331 | ادخال أول عقدة |
| 331 | ادخال آخر عقدة |
| 331 | حذف عقدة |
| 332 | نسخ القائمة |
| 333 | المدمر |
| 333 | مقوم النسخ |
| 334 | اثقال معامل الاسناد |
| 337 | قوائم متصلة مرتبة |
| 337 | بحث القائمة |
| 341 | ادخال عقدة |
| 342 | حذف عقدة |
| 343 | الملف الرئيسي للقائمة المتصلة المرتبة |
| 346 | القوائم المتصلة من الطرفين |
| 247 | المقوم الافتراضي |
| 348 | isEmptyList |
| 348 | تدمير القائمة |
| 348 | تهيئة القائمة |
| 348 | طول القائمة |
| 349 | اثقال معامل ادخال التدفق |
| 350 | عكس طباعة القائمة |
| 353 | بحث القائمة |
| 354 | العنصر الأول والأخير |
| 354 | ادخال عقدة |
| 367 | حذف عقدة |
| 350 | الحاوية المتتابعة لمكتبة القالب المعياري: list |
| 355 | القوائم المتصلة ذات العقد الأساسية والعقد header |
| 356 | قوائم متصلة دائرية |

| | |
|-----|---------------------------------------|
| 367 | مثال برمجة: متجر الفيديو |
| 367 | مراجعة سريعة |
| 378 | تمارين |
| 383 | تمارين برمجة |
| 394 | 6. التكرار |
| 368 | تعريفات تكرارية |
| 371 | تكرار مباشر وغير مباشر |
| 371 | تكرار مطلق |
| 372 | حل المسألة باستخدام التكرار |
| 390 | التكرار أم المكررات؟ |
| 391 | التكرار والتتبع الخلفي للغز الملكات n |
| 391 | التتبع الخلفي |
| 394 | التتبع الخلفي ولغز الأربع ملكات |
| 395 | لغز الثماني ملكات |
| 398 | مراجعة سريعة |
| 399 | تمارين |
| 402 | تمارين البرمجة |
| 433 | 7. المرصوصات |
| 433 | المرصوصات |
| 434 | عمليات المرصوصة |
| 436 | تطبيق المرصوصات كمصفوفات |
| 339 | تهيئة المرصوصة |
| 340 | تدمير المرصوصة |
| 341 | مرصوصة فارغة |
| 441 | مرصوصة ممثلة |
| 441 | الدفع |
| 442 | إعادة العنصر العلوي |
| 442 | الطرح |
| 443 | نسخ المرصوصة |
| 445 | المقوم والمدمر |
| 446 | مقوم النسخ |

| | |
|-----|---|
| 446 | انقال معامل الاسناد (=) |
| 447 | الملف الرئيسي للمرصوفة |
| 447 | مثال البرمجة: أعلى متوسط درجات |
| 448 | التطبيق المتصل للمرصوفات |
| 452 | المقوم الافتراضي |
| 452 | تدمير المرصوفة |
| 457 | تهيئة المرصوفة |
| 457 | الدفع |
| 457 | اعادة العنصر العلوي |
| 458 | الطرح |
| 459 | المرصوفة كمستمددة من الفئة linkedListType |
| 461 | استخدام المرصوفات: حاسب التعبير postfix |
| 467 | الخوارزمية الرئيسية |
| 469 | تحديد البرنامج الكامل |
| 472 | ازالة التكرار: خوارزمية غير تكرارية لطباعة قائمة متصلة من الخلف |
| 480 | فئة مكتبة القالب المعياري stack (مكيف حاوية المرصوفة) |
| 482 | مراجعة سريعة |
| 482 | تمارين |
| 485 | تمارين برمجة |
| 490 | 8. الطوابير |
| 470 | الطوابير |
| 470 | عمليات الطابور |
| 471 | تطبيق الطوابير كمرصوفات |
| 484 | التطبيق المتصل للطوابير |
| 487 | العمليات addQueue، front، وback، وdeleteQueue |
| 488 | طابور مستمد من الفئة linkedListType |
| 492 | فئة مكتبة القالب المعياري queue (مكيف حاوية الطابور) |
| 493 | طوابير الأسبقية |
| 494 | فئة مكتبة القالب المعياري priority_queue |
| 494 | استخدام الطوابير: المحاكاة |
| 495 | تصميم نظام للطابور |

| | |
|-----|--|
| 496 | العميل |
| 499 | مقدم الخدمة |
| 502 | قائمة مقدم الخدمة |
| 507 | طابور العملاء المنتظرين |
| 517 | مراجعة سريعة |
| 517 | تمارين |
| 521 | تمارين برمجة |
| 523 | 9. خوارزميات البحث |
| 524 | خوارزميات البحث |
| 526 | البحث المتتابع |
| 529 | القوائم المرتبة |
| 529 | البحث الثنائي |
| 533 | أداء البحث الثنائي |
| 535 | الادخال في قائمة مرتبة |
| 537 | الحد الأدنى على خوارزميات البحث القائمة على المقارنة |
| 538 | البعثة |
| 539 | دالات البعثة: بعض الأمثلة |
| 540 | حل التعارض |
| 540 | حل التعارض: مواجهة مفتوحة |
| 545 | الحذف: مواجهة مفتوحة |
| 547 | البعثة: التطبيق باستخدام التحقيق الرباعي |
| 549 | حل التعارض: التسلسل (البعثة المفتوحة) |
| 551 | تحليل البعثة |
| 552 | مراجعة سريعة |
| 555 | تمارين |
| 557 | تمارين برمجة |
| 559 | 10. خوارزميات الترتيب |
| 560 | خوارزميات الترتيب |
| 560 | الترتيب بالاختيار: قوائم مبنية على المصفوفات |
| 566 | التحليل: الترتيب بالاختيار |
| 566 | الترتيب بالادخال: قوائم مبنية على المصفوفات |

| | |
|-----|--|
| 573 | الترتيب بالادخال: قوائم مبنية على القوائم المتصلة |
| 578 | التحليل: الترتيب بالادخال |
| 578 | الحد الأدنى على خوارزميات الترتيب المبنية على المقارنة |
| 579 | الترتيب السريع: القوائم المبنية على المصفوفة |
| 586 | التحليل: الترتيب السريع |
| 587 | الترتيب بالدمج: القوائم المبنية على القوائم المتصلة |
| 589 | القسمة |
| 591 | الدمج |
| 595 | تحليل: الترتيب بالدمج |
| 595 | الترتيب بالتكدس: القوائم المبنية على المصفوفة |
| 596 | بناء التكدس |
| 604 | تحليل: الترتيب بالتكدس |
| 605 | طوابير الأسبقية (مرة أخرى) |
| 606 | مثال البرمجة: نتائج الانتخابات |
| 626 | مراجعة سريعة |
| 627 | تمارين |
| 628 | تمارين برمجة |
| 631 | 11. أشجار ثنائية |
| 632 | أشجار ثنائية |
| 638 | نسخ الشجرة |
| 639 | اجتياز شجرة ثنائية |
| 639 | اجتياز في الترتيب |
| 640 | الاجتياز قبل الترتيب |
| 640 | الاجتياز بعد الترتيب |
| 643 | تطبيق الأشجار الثنائية |
| 651 | أشجار البحث الثنائية |
| 655 | البحث |
| 656 | الادخال |
| 658 | الحذف |
| 666 | شجرة البحث الثنائية: تحليل |
| 668 | خوارزميات اجتياز الشجرة الثنائية الغير تكرارية |

| | |
|-----|--|
| 668 | الاجتياز الغير تكراري في الترتيب |
| 670 | الاجتياز الغير تكراري قبل الترتيب |
| 671 | الاجتياز الغير تكراري بعد الترتيب |
| 672 | اجتياز الشجرة الثنائية والدالات كمعاملات |
| 675 | أشجار AVL (متوازنة الارتفاع) |
| 679 | الادخال في أشجار AVL |
| 685 | دورانات الشجرة AVL |
| 699 | الحذف من شجرة AVL |
| 700 | تحليل: أشجار AVL |
| 701 | مثال البرمجة: متجر الفيديو (مرة أخرى) |
| 710 | مراجعة سريعة |
| 712 | تمارين |
| 716 | تمارين برمجة |
| 719 | 12. الرسوم البيانية |
| 720 | مقدمة |
| 721 | تعريفات ورموز الرسم البياني |
| 723 | تمثيل الرسم البياني |
| 723 | مصفوفة التجاور |
| 724 | قوائم التجاور |
| 726 | العمليات على الرسوم البيانية |
| 727 | القوالب (مرة أخرى) |
| 728 | الرسوم البيانية كنوع بيانات مجرد |
| 731 | اجتيازات الرسم البياني |
| 732 | الاجتياز الأول للعمق |
| 734 | الاجتياز الأول للعرض |
| 737 | خوارزمية المسار الأقصر |
| 738 | المسار الأقصر |
| 744 | شجرة الامتداد الأدنى |
| 754 | الترتيب الطوبوغرافي |
| 755 | الترتيب الطوبوغرافي الأول للعرض |
| 762 | مراجعة سريعة |

| | |
|-----|---|
| 763 | تمارين |
| 766 | تمارين برمجة |
| 769 | 13. مكتبة القالب المعياري 2 |
| 770 | الفئة pair |
| 771 | مقارنة أهداف من النوع pair |
| 772 | النوع pair والدالة make_pair |
| 774 | الحاويات المترابطة |
| 775 | الحاويات المترابطة: set و multiset |
| 780 | الحاويات المترابطة: map و multimap |
| 785 | الحاويات والملفات الرئيسية المترابطة ودعم المكررات |
| 786 | الخوارزميات |
| 786 | تصنيف خوارزميات مكتبة القالب المعياري |
| 786 | خوارزميات غير تعديلية |
| 787 | خوارزميات تعديلية |
| 788 | خوارزميات رقمية |
| 788 | خوارزميات التكدس |
| 789 | أهداف الدالة |
| 794 | الأصول |
| 794 | ادخال مكرر |
| 796 | خوارزميات مكتبة القالب المعياري |
| 796 | الدالات fill و fill_n |
| 798 | الدالات generate و generate_n |
| 800 | الدالات find، find_if، find_end، و find_first_of |
| 802 | الدالات remove، remove_if، و remove_copy و remove_copy_if |
| 806 | الدالات replace، replace_if، و replace_copy و replace_copy_if |
| 808 | الدالات swap، iter_swap، و swap_ranges |
| 811 | الدالات search، search_n، sort، و binary_search |
| 815 | الدالات adjacent_find، merge، و inplace_merge |
| 818 | الدالات reverse، reverse_copy، rotate، و rotate_copy |
| | الدالات count، count_if، max، و max_element، min، |
| 821 | و min_element، و random_shuffle |

| | |
|-----|---|
| 825 | الدالات for_each و transform |
| | الدالات includes و set_intersection و set_union |
| 827 | و set_difference و set_symmetric_difference |
| | الدالات accumulate و adjacent_difference |
| 835 | و inner_product و partial_sum |
| 840 | مراجعة سريعة |
| 844 | تمارين |
| 846 | تمارين برمجة |
| 847 | الملحق أ كلمات محفوظة |
| 849 | الملحق ب أسبقية المعامل |
| 851 | الملحق ج مجموعات الرموز |
| 855 | الملحق د ائصال المعامل |
| 857 | الملحق هـ الملفات الرئيسية |
| 857 | الملف الرئيسي cassert |
| 857 | الملف الرئيسي cctype |
| 859 | الملف الرئيسي cmatch |
| 860 | الملف الرئيسي cstddef |
| 860 | الملف الرئيسي cstring |
| 861 | الملف الرئيسي string |
| 865 | الملحق (و) موضوعات C++ اضافية |
| 865 | التوريث والمؤشرات والدالات الافتراضية |
| 871 | الفئات والمدمرات الافتراضية |
| 872 | عنوان المعامل والفئات |
| 875 | الملحق ز C++ لمبرمجي الجافا |
| 875 | أنواع البيانات |
| 876 | العوامل والتعبيرات الرياضية |
| 876 | الثوابت والمتغيرات المسماة وبيانات الاسناد |
| 877 | مكتبة C++ : موجهات قبل المعالجة |
| 878 | برنامج C++ |
| 879 | المدخلات والمخرجات |
| 879 | المدخلات |

| | |
|-----|---------------------------------------|
| 882 | المخرجات |
| 887 | ادخال / اخراج ملف |
| 889 | بنيات التحكم |
| 890 | الأسماء |
| 895 | الدالات والعوامل |
| 895 | دالات منتجة للقيمة |
| 896 | دالات void |
| 898 | عوامل الاشارة والدالات المنتجة للقيمة |
| 898 | دالات ذات عوامل افتراضية |
| 900 | المصفوفات |
| 900 | تناول مكونات المصفوفة |
| 901 | مؤشر المصفوفة خارج الحدود |
| 901 | المصفوفات كمعاملات للدالات |
| 903 | الملحق ح المرجعيات |
| 905 | الملحق ط حلول تمارين مختارة |
| 905 | الفصل 1 |
| 906 | الفصل 2 |
| 907 | الفصل 3 |
| 908 | الفصل 4 |
| 909 | الفصل 5 |
| 909 | الفصل 6 |
| 910 | الفصل 7 |
| 911 | الفصل 8 |
| 912 | الفصل 9 |
| 913 | الفصل 10 |
| 914 | الفصل 11 |
| 916 | الفصل 12 |
| 918 | الفصل 13 |
| 919 | الفهرس |

تمهيد

أهلاً بكم في بنيات البيانات باستخدام C++. هذا الكتاب المصمم من أجل برنامج تدريبي ثاني في علوم الحاسب على C++ سوف يقدم لك ولطلابك نسمة من الهواء المنعش. برنامج علوم الحاسب الثاني يقوم باكمال متطلبات برمجة منهج علوم الحاسب الآلي وهذا الكتاب عبارة عن تنويع وتطوير لملاحظات فصلي الدراسي على مدار ما يزيد عن خمسين فصل دراسي من تدريس البرمجة وبنيات البيانات بشكل ناجح لطلاب علوم الحاسب الآلي.

هذا الكتاب استمرار للعمل الذي بدأ في كتاب البرنامج الأول لعلوم الحاسب وهو برمجة C++: من تحليل المشكلة الى تصميم البرنامج الذي تم نشره عن طريق كورس تكنولوجياي. المنهج المأخوذ في هذا الكتاب مماثل للمنهج المستخدم في كتاب برنامج علوم الحاسب الأول وهو تقديم مادة تم استمدادها عن طريق طلب الطلاب على الوضوح والقراءة. لقد تمت كتابة المادة واعادة كتابتها حتى شعر الطلاب بالارتياح لها. غالبية الأمثلة المذكورة في هذا الكتاب نتجت من تفاعل الطالب في الحجرة الدراسية.

هذا الكتاب يفترض أن القارئ على علم بالعناصر الأساسية من C++ مثل أنواع البيانات وبنيات التحكم والدالات والعوامل والمصفوفات وبالرغم من هذا اذا احتجت الى مراجعة هذه المفاهيم أو أخذت الجافا كلغة برمجة أولى فسوف تجد أن المادة الموجودة في الملحق (ز) مفيدة. واذا احتجت الى مراجعة موضوعات برنامج علوم الحاسب الأول بمزيد من التفصيل أكثر مما هو معطى في الملحق (ز) ارجع الى كتاب برمجة C++ بواسطة الكاتب المذكور في الفقرة السابقة وكذلك الموارد المذكورة في الملحق (ط). بالإضافة الى هذا هناك حاجة الى بعض من الخلفية الرياضية المناسبة مثل جبر الجامعة،

المنهج:

هذا الكتاب كبرنامج تدريبي ثاني في برمجة الحاسب يقوم بالتركيز على بنيات البيانات وكذلك على التصميم القائم على الهدف. أمثلة البرمجة المعطاة في هذا الكتاب تقوم باستخدام أساليب التصميم القائم على العنصر بشكل فعال من أجل حل وبرمجة مشكلة محددة. الفصل الأولي يقدم مبادئ هندسة البرمجيات وبعد وصف دورة حياة البرنامج يقوم هذا الفصل بمناقشة سبب أهمية تحليل الخوارزمية ويقوم بتقديم رمز O الكبير المستخدم في تحليل الخوارزمية. التصميم القائم على الهدف له ثلاثة مبادئ رئيسية – التغليف، والتوريث، وتعدد الأشكال. يتم تحقيق التغليف في C++ عبر استخدام الفئات. يقوم النصف الثاني من الفصل الأول بمناقشة الفئات المعرفة بواسطة المستخدم. اذا كنت على علم بكيفية عمل واستخدام فئات خاصة بك يمكنك تجاوز هذا الجزء. كما يقوم هذا الفصل بمناقشة أسلوب أساسي من التصميم القائم على الهدف لحل مشكلة معينة.

يستمر الفصل 2 في مبادي التصميم المبني على الهدف ويناقش التوريث وكذلك نوعين من تعدد الأشكال. (تتم مناقشة النوع الثالث من تعدد الأشكال في الملحق و). اذا كان القارئ على علم بكيفية عمل التوريث واثقال المعامل والقوالب في C++ يمكن للمستخدم أن يتجاوز هذا الفصل.

C++ لها ثلاثة أنواع رئيسية من البيانات: بسيطة، ومركبة، ومؤشرات. يفترض هذا الكتاب أن القارئ على علم بأنواع البيانات البسيطة وكذلك المصفوفات (نوع بيانات مركبة). يتم تقديم فئات نوع البيانات المركبة في الفصل 1 ويقوم الفصل 3 بمناقشة كيفية عمل نوع بيانات المؤشر في C++. يقوم الفصل 3 بوصف العلاقة بين المؤشرات والفئات. باستخدام المؤشرات والقوالب يقوم هذا الفصل بتوضيح وتطوير كود عام لتطبيق القوائم باستخدام مصفوفات حركية.

C++ مزودة بمكتبة القالب المعياري. تقوم مكتبة القالب المعياري من ضمن أشياء أخرى بتقديم كود لمعالجة البيانات (متجاورة أم متصلة) والمرصوصات والطوابير. يقوم الفصل 4 بمناقشة بعض من سمات مكتبة القالب المعياري الهامة ويوضح كيفية استخدام أدوات محددة من مكتبة القالب المعياري في برنامج ما. يقوم هذا الفصل بوجه خاص بمناقشة الحاويات المتتابعة vector و deque. الفصول التالية توضح كيفية عمل كود خاص بك لتطبيق البيانات والسيطرة عليها وكذلك كيفية استخدام الكود المكتوب بشكل جيد.

الفصل 5 يناقش القوائم المتصلة ويقوم أولاً بتوضيح الخصائص الأساسية للقوائم المتصلة مثل ادخال وحذف عنصر وكيفية بناء قائمة متصلة ثم يقوم بعمل كود عام لمعالجة البيانات في قائمة متصلة واحدة. يقوم الفصل 5 كذلك بمناقشة القوائم المتصلة من الطرفين والقوائم المتصلة ذات العقد header و trailer والقوائم المتصلة الدائرية. فضلاً عن هذا يناقش الفصل فئة مكتبة القالب المعياري list. يقوم الفصل 6 بتقديم التكرار ويعطي أمثلة متنوعة لتوضيح كيفية استخدام التكرار لحل مسألة وكذلك التفكير على أساس التكرار.

يقوم كل من الفصل 7 و 8 بمناقشة المرصوصات والطوابير وبالإضافة الى توضيح كيفية عمل كود عام خاص بك لتطبيق المرصوصات والطوابير يقوم هذان الفصلان بتوضيح كيفية عمل فئات مكتبة القالب المعياري stack و queue (المرصوصة والطابور). كود البرمجة الذي يتم وضعه في هذين الفصلين كود عام.

يقوم الفصل 9 بوصف خوارزميات البحث وبعد تحليل خوارزمية البحث التتابعي يناقش خوارزمية البحث الثنائي ويقدم تحليل موجز لهذه الخوارزمية. يقوم هذا الفصل بمناقشة البعثة بعد اعطاء الحد الأدنى على خوارزميات البحث القائمة على المقارنة.

يتم تقديم خوارزميات الترتيب مثل الترتيب بالاختيار، والترتيب بالادخال، والترتيب السريع، والترتيب بالدمج، والترتيب بالتكدس في تافصل رقم 10. يقوم الفصل 11 بمناقشة الأشجار الثنائية ويقوم الفصل 12 بتقديم الرسوم البيانية ويناقش خوارزميات الرسم البياني مثل أقصر مسار، وشجرة الامتداد الأدنى، والترتيب الطوبوغرافي.

يستمر الفصل 13 في مناقشة مكتبة القالب المعياري التي بدأت في الفصل 4 ويقوم بوجه خاص بتقديم الحاويات المتتابعة وخوارزميات مكتبة القالب المعياري.

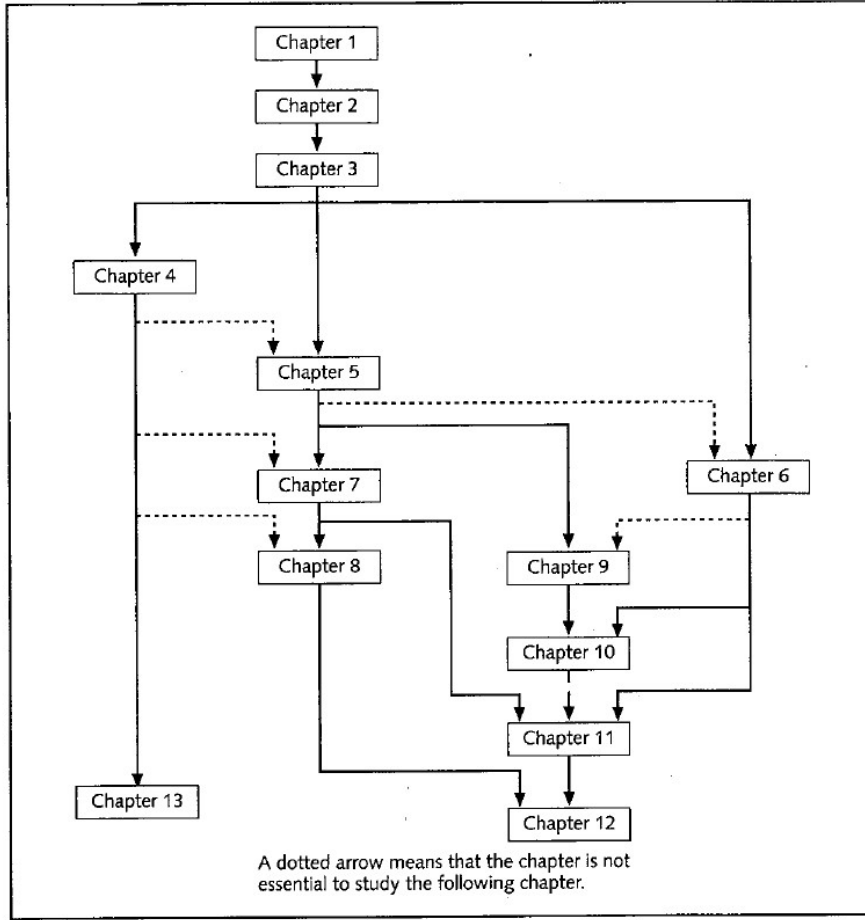
الملحق (أ) يحدد الكلمات المحفوظة في C++ ويقوم الملحق (ب) بتوضيح أسبقية وترابط عوامل C++. الملحق (ج) يحدد الكود المعياري الأمريكي لتبادل المعلومات ومجموعات رموز التبادل العشري للكود الثنائي الممتد. يقوم الملحق (د) بتحديد عوامل C++ التي يمكن ائصالها. يقوم الملحق (هـ) بوصف بعض من تقاليد المكتبة التي يتم استخدامها على نطاق واسع. يقوم الملحق (و) بمناقشة موضوعات C++ اضافية – دالات افتراضية، وعنوان المعامل والفئات. يقوم الملحق (ز) باستعراض العناصر الأساسية من C++ ويقارن المفاهيم الأساسية للغة C++ ولغة الجافا مثل أنواع البيانات وبنيات التحكم والدالات والعوامل والمصفوفات. لهذا اذا كنت قد أخذت الجافا كلغة برمجة أولى يساعد الملحق (ز) على تعريفك بهذه العناصر الأساسية من C++. الملحق (ح) يقدم قائمة بالمراجع من أجل المزيد من الدراسة وكيفية ايجاد موضوعات C++ اضافية لم يتم استعراضها في الملحق (ز). يقوم الملحق (ط) باعطاء الاجابات لتمرين مختارة في الكتب.

كيفية استخدام هذا الكتاب:

الهدف الأساسي من هذا الكتاب هو تدريس موضوعات بنية البيانات باستخدام C++ والتصميم المبني على الهدف من أجل حل مسألة معينة. للقيام بهذا يقوم الكتاب بمناقشة بنيات بيانات مثل القوائم المتصلة والمرصوصات والطوابير والأشجار الثنائية. كما أن مكتبة القالب المعياري C++ تقدم الكود اللازم لتطبيق بنيات البيانات هذه. ومع هذا تأكيدنا يكون على تعليمك كيفية عمل كود خاص بك وفي نفس الوقت نريدك أن تتعلم كيفية استخدام الكود المكتوب باحتراف.

يقدم الفصل 4 مكتبة القالب المعياري وفي الفصول التالية نقوم بعد توضيح كيفية عمل الكود الخاص بك بتوضيح كيفية استخدام الكود الحالي من مكتبة القالب المعياري. لهذا يمكن استخدام هذا الكتاب بطرق متعددة. الا لم تكن مهتماً بمكتبة القالب المعياري لنقل في القراءة الأولى اذن يمكنك تجاوز الفصل 4 وفي الاقسام التالية الأقسام التي نناقش فيها مكون محدد من مكتبة القالب المعياري.

الفصل 6 يناقش التكرار وبالرغم من هذا فان الفصل رقم 6 ليس ضروري من أجل الفصلين 7 و8. اذا قرأت الفصل 6 بعد هذه الفصول اذن يمكنك تجاوز الجزء "ازالة التكرار" في الفصل 7 وقرأ هذا الجزء بعد قراءة الفصل 6. بالرغم من أن الفصل رقم 6 غير لازم لاستيعاب الفصل 9 الا أن الفصلين 9 و10 يجب قراءتهما بالتتابع. لهذا نوصيك بدراسة الفصل 6 قبل الفصل 9. الشكل التالي يوضح اعتماد الفصول على بعضها البعض.



شكل 1: رسم اعتماد الفصول على بعضها البعض

خصائص:

يشتمل هذا الكتاب على الخصائص التالية في كل فصل وهذه الخصائص تكون مساعدة على التعلم وتجعل من الممكن للقراء أن يتعلموا المادة بالخطى الخاصة بهم.

- الأهداف تقدم مخطط لمفاهيم برمجة C++ التي تتم مناقشتها في الفصل.
- الملاحظات تبرز حقائق هامة بشأن المفاهيم التي يتم تقديمها في الفصل.
- الرسوم المرئية شاملة ومفصلة توضح مفاهيم صعبة والكتاب يحتوي على أكثر من 350 شكل.

- الأمثلة المرقمة داخل كل فصل عبارة عن برامج صغيرة توضح المفاهيم الرئيسية ذات الكود المرتبط بها ويكون كل سطر في كود البرمجة في هذه الأمثلة مرقم. بعد هذا يتم شرح كل برنامج موضح من خلال تنفيذ العينة سطرا تلو الآخر وتتم مناقشة السبب وراء كل سطر بالتفصيل.

- أمثلة البرمجة عبارة عن برامج موجودة في نهاية كل فصل وهذه الأمثلة تتضمن المراحل الدقيقة والمحددة للدخال، والاخراج، وتحليل المشكلة، وتصميم الخوارزمية، وتحديد

البرنامج. فضلاً عن هذا يتم حل وبرمجة المسائل الموجودة في أمثلة البرمجة هذه باستخدام التصميم القائم على الهدف. ان أمثلة البرمجة تشكل العمود الفقري للكتاب ويتم ابرازها بأيقونة في الهامش. البرامج يتم تصميمها حتى تكون منهجية وسهلة على المستخدم. بدءاً من تحليل المشكلة يتبع مثال البرمجة تصميم الخوارزمية. بعد هذا يتم ترميز كل خطوة من الخوارزمية في C++. بالإضافة الى تعليم أساليب حل المسائل توضح هذه البرامج التفصيلية للمستخدم كيفية تطبيق المفاهيم في برنامج C++ معين. أوصي بشدة أن يكون الطلاب الدارسين لأمثلة البرمجة حريصين للغاية من أجل تعلم C++ بشكل فعال.

- أقسام المراجعة السريعة في نهاية كل فصل تدعم التعلم عن طريق تلخيص المفاهيم التي يتم تغطيتها داخل الفصل. بعد قراءة الفصل يمكن للقراء أن يمرون خلال أساسيات الفصل بشكل سريع ثم يقومون باختبار أنفسهم عن طريق استخدام التمارين التالية. الكثير من القراء يعودون الى المراجعة السريعة كطريقة لمراجعة الفصل بسرعة قبل الاختبار.
 - تقوم التمارين بتعزيز التعليم وتضمن أن الطلاب قد تعلموا المادة بالفعل.
 - تمارين البرمجة تتحدى الطلاب في أن يكتبوا برامج C++ ذات نتيجة معينة.
- من البداية الى النهاية يتم تقديم المفاهيم بشكل مساعد على التعلم. أسلوب كتابة هذا الكتاب بسيط وواضح. قبل تقديم مفهوم رئيسي نقوم بتوضيح سبب أهمية عناصر محددة وبعد هذا يتم تفسير المفاهيم المقدمة باستخدام أمثلة وبرامج صغيرة.
- لقد تمت كتابة وتجميع جميع الحلول وكود المصدر وتم اختبار ضمان الجودة. يمكن تجميع البرامج بالمايكروسوفت فيجوال C++ نت أو مترووركس كودواريور.

الأدوات التعليمية:

المواد التكميلية التالية تكون متوفرة عندما يتم استخدام هذا الكتاب في وضع الفصل الدراسي. جميع الأدوات التعليمية المتوفرة في هذا الكتاب يتم تقديمها الى الموجه على أسطوانة واحدة.

كتيب الموجه الالكتروني: كتيب الموجه الذي يصاحب هذا الكتاب يشمل:

- مادة تعليمية اضافية تقوم بالمساعدة في اعداد الفصل وتتضمن الاقتراحات لموضوعات المحاضرة.

- حلول لجميع المواد الموجودة في نهاية الفصل بما فيها تمارين البرمجة.

رؤية اختبارية: هذا الكتاب مصحوب ببرنامج ايجزامفيو وهو مجموعة برامج اختبار قوية تسمح للمعلمين بعمل وإدارة اختبارات مطبوعة وحاسوبية (مبنية على LAN) وانترنت. هذا البرنامج يحتوي على مئات من الأسئلة التي تتوافق مع الموضوعات المغطاة في هذا الكتاب وهو بهذا يمكن الطلاب من عمل ارشادات دراسية تفصيلية تشمل اشارات الصفحات للمزيد من المراجعة. مكونات اختبارات الانترنت والاختبارات القائمة على الحاسوب تسمح للطلاب بأخذ اختبارات على حاسوباتهم الخاصة كما تقوم بتوفير الوقت للمعلم عن طريق تحديد درجات كل اختبار بشكل آلي.

تمثيلات الباوربوينت: يأتي هذا الكتاب بشرائح من مايكروسوفت باوربوينت لكل فصل وهي متضمنة كمساعدة تعليمية لتمثيلات الحجرة الدراسية اما لكل تكون متاحة للطلاب على الشبكة من أجل مراجعة الفصل أو حتى يتم طبعها من أجل التوزيع في الحجرة الدراسية. يمكن للمعلمين أن يضيفوا شرائحهم الخاصة من أجل الموضوعات الاضافية التي يقدمونها الى الفصل.

التعليم عن بعد: كورس تكنولوجيا تفخر بأنها تقدم برامج تدريبية على الانترنت في WebCT و Blackboard وكذلك عند MyCourse.com وهو أداة دعم البرنامج التدريبي الخاصة بكورس تكنولوجيا لتوفير أكثر خبرة تعليمية حركية ممكنة. عندما تضيف محتوى على الانترنت لواحد من برامجك التدريبية فانك بهذا تضيف الكثير من الاختبارات الذاتية والوصلات والملخصات و بوابة مرور الى أهم موارد معلومات القرن 21. اننا نتمنى أن تجتاز غالبية برنامجك التدريبي سواء الموجودة على الانترنت أو غيرها. لمزيد من المعلومات عن كيفية جلب التعليم عن بعد لبرنامجك التدريبي اتصل بمندوب المبيعات المحلي لكورس تكنولوجيا.

كود المصدر: كود المصدر متوفر على www.course.com وكذلك على أسطوانة الأدوات التعليمية. ملفات الادخال اللازمة لتشغيل بعض من البرامج موجودة كذلك مع كود المصدر. بالرغم من هذا يجب أن يتم حفظ ملفات الادخال أولاً على قرص مرن في الجزء A:

ملفات الحل: كود المصدر لجميع تمارين البرمجة متوفر في www.course.com وعلى أسطوانة الأدوات التعليمية. ملفات الادخال اللازمة لتشغيل بعض من تمارين البرمجة موجودة كذلك مع كود المصدر. بالرغم من هذا يجب أن يتم حفظ ملفات الادخال أولاً على قرص مرن في الجزء A:

شكر:

انني أدين بالكثير للمراجعين التاليين الذين قاموا بصبر بقراءة كل صفحة من كل فصل من النسخة الحالية وعمل تعليقات هامة لتحسين الكتاب: ريتشارد البرايت بجامعة ديلاوير، وستيف أرمسترونج بجامعة ليتورنيه، وجون دابونتيه بجامعة ولاية كونيتيكت الجنوبية، وتشارلز داولنج بجامعة بالتيموركاونتي في كاتونسفيل، ومارجريت ندربرج بجامعة ولاية ينجستاون. سوف يدرك المراجعون أن اقتراحاتهم لم يتم اغفالها وفي الحقيقة فقد جعلت هذا الكتاب أفضل. الشكر لمحرة التطوير سوزان جلبرت من أجل تحريرها الحريص لكل فصل وكل هذا لم يكن ممكناً بدون تخطيط كبيرة المحررين جنيفر موروف. شكري الخاص لجنيفر وكذلك لمحرر الانتاج ايمي بوريير وكذلك لقسم ضمان الجودة في كورس تكنولوجيا من أجل اختبارهم الحريص للكود.

أنني شاكر لوالدي ووالدتي الى من أهديهم هذا الكتاب من أجل مباركاتهم كما أريد أن أشكر أخوتي على تشجيعهم ومساندتهم.

أخيراً انني شاكر لمساندة زوجتي سادهانا وخاصة ابنتي شيلي ماليك فقد كانت تبهجني عندما أكون مغموراً أثناء كتابة هذا الكتاب. ان شيلي تقوم دائماً برسم بهجة خاصة عندما أقوم بتولي مثل هذه المشروعات.

انني أرحب بأية تعليقات بشأن الكتاب ويمكن ارسال التعليقات الى عنوان البريد الالكتروني التالي:

malik@creighton.edu

دي اس ماليك

الفصل

1

هندسة البرمجيات

المبادئ والفئات C++

في هذا الفصل سوف:

- تتعلم مبادئ هندسة البرمجيات.
- تكتشف نظام الحلول الحسابية وتكتشف أساليب حل المشكلات.
- تصبح على علم بالتصميم المنظم والتصميم القائم على الموضوع ومنهجيات البرمجة.
- تتعلم الفئات.
- تتعلم أعضاء الفئة الخاصة والمحمية والعامة.
- تكتشف كيفية تطبيق الفئات.
- تصبح على علم بلغة التكوين الموحدة.
- تدرس constructors & destructors
- تتعلم نوع البيانات المجردة.
- تكتشف كيفية استخدام الفئات لتطبيق نوع البيانات المجردة.
- تتعلم إخفاء المعلومات.
- تكتشف كيفية تطبيق إخفاء المعلومات في C++.

ان كل فرد تقريباً يعمل باستخدام الحاسبات الآلية يكون على علم بالمصطلح "برمجيات". البرمجيات عبارة عن برامج حاسب آلي يتم تصميمها لانجاز مهمة محددة. على سبيل المثال برنامج معالجة الكلمات عبارة عن برنامج يمكنك من كتابة أوراق للمصطلحات، وابتكار سيرة ذاتية مثيرة للاعجاب، وحتى من كتابة كتاب كامل. هذا الكتاب على سبيل المثال تم ابتكاره بمساعدة معالج الكلمات. لم يعد للطلاب يقومون بكتابة أوراقهم على الآلات الكاتبة أو كتابتها يدوياً ولكنهم بدلاً من ذلك يستخدمون برنامج معالج الكلمات لكي يقوموا باكمال أوراق فصلهم الدراسي. العديد من الأفراد يقومون بالمحافظة على وموازنة دفاتر شيكاتهم على الحاسبات الآلية.

إن البرامج القوية السهلة الاستخدام غيرت الطريقة التي نعيش ونتواصل بها بشكل كبير. إن مصطلحات مثل "الانترنت" التي كانت غير مألوفة منذ سنوات قليلة فقط أصبحت شائعة للغاية في الوقت الحالي. بمساعدة الحاسبات الآلية والبرامج التي يتم تشغيلها عليها يمكنك أن ترسل خطابات الى وتستقبل خطابات ممن تحبهم خلال ثوان. كما أنك لم تعد في حاجة الى ارسال السيرة الذاتية بالبريد من أجل التقدم الى وظيفة ما ففي العديد من الحالات يمكنك ببساطة تقديم طلب التقدم الى الوظيفة عبر الانترنت، كما يمكنك كيفية أداء الأسهم في الوقت الحالي وشرائها وبيعها على الفور. يقوم الأطفال في المدارس الابتدائية "بتصفح" الانترنت بانتظام ويستخدمون الحاسب الآلي لتصميم مشروعات فصولهم الدراسية.

بدون البرمجيات تكون الحاسبات الآلية عديمة الفائدة فالبرمجيات هي التي تمكنك من القيام بالأشياء التي ربما كانت خيالية منذ القليل من السنوات. بالرغم من هذا فانه لم يتم اختراع البرمجيات بين عشية وضحاها فالبرنامج منذ الوقت الذي يتم تصويره فيه زحى يتم توصيله يمر خلال عدة مراحل. هناك فرع من علوم الحاسب الآلي يسمى هندسة البرمجيات وهو متخصص في هذا المجال والعديد من الكليات والجامعات تقدم برنامج تدريبي في هندسة البرمجيات. هذا الكتاب لا يهتم بتدريس مبادئ هندسة البرمجيات ولكنه بالرغم من هذا يوضح بايجاز البعض من المبادئ الأساسية لهندسة البرمجيات التي يمكنها تبسيط تصميم برنامج ما.

دورة حياة البرمجيات:

يمر البرنامج خلال العديد من المراحل من وقت تصويره في البداية وحتى انتهاء خدمته وهذه الفترة يطلق عليها **دورة حياة البرنامج**. هناك ثلاثة مراحل أساسية يمر بها أي برنامج وهي: **التطوير والاستخدام والصيانة**. عادةً يتم تصور البرنامج في البداية بواسطة مطور البرمجيات لأن هناك عميل ما لديه مشكلة تحتاج الى حلها والعمل يرغب في دفع المال من أجل حلها. يتم ابتكار البرنامج الجديد في مرحلة **تطوير البرمجيات** والقسم التالي يصف هذه المرحلة ببعض من التفصيل.

بمجرد اعتبار البرنامج مكتمل يتم إصداره للمستخدم لكي يقوم باستخدامه وبمجرد بدء المستخدمين في استخدام البرنامج فانهم بالتأكيد يكتشفون المشكلات أو يكون لديهم اقتراحات لتحسينها. يتم نقل المشكلة و/أو أفكار التحسين الى مطور البرمجيات ويمر البرنامج خلال مرحلة الصيانة.

في عملية **صيانة البرمجيات** يتم تعديل البرنامج من أجل اصلاح المشكلات (المحددة) أو لتحسينها. اذا كان هناك تغيرات خطيرة أو متعددة يتم ابتكار نسخة جديدة من البرنامج ويتم إصداره للاستخدام. عندما يتم اعتبار برنامج ما مكلف للغاية في الصيانة فقد يقرر المطور أن يحيل البرنامج الى التقاعد ولا يتم إصدار نسخة جديدة من البرنامج.

ان مرحلة تطوير البرمجيات هي المرحلة الأولى وربما الأكثر أهمية من دورة حياة البرمجيات حيث أن البرنامج الذي من دورة حياة البرمجيات حيث أن البرنامج الذي يتم تطويره جيداً تكون صيانتة أسهل وأقل تكلفة. القسم التالي يصف هذه المرحلة.

مرحلة تطوير البرمجيات:

يقوم مهندسيون البرمجيات عادةً بتقسيم عملية تطوير البرمجيات الى المراحل الأربعة التالية:

- التحليل.
- التصميم.
- التطبيق.
- الاختبار والمعالجة.

الأقسام القليلة التالية تصف تلك المراحل الأربعة ببعض من التفاصيل.

التحليل:

تحليل المشكلة هو الخطوة الأولى والضرورية وتتطلب منك هذه الخطوة أن تقوم بما يلي:

- فهم المشكلة تماماً .
- **متطلبات التحليل:** استيعاب متطلبات المشكلة وقد تتضمن المتطلبات ما اذا كان البرنامج يحتاج تفاعل مع المستخدم، وما اذا كان يضارب في البيانات، وما اذا كان ينتج مخرج وما يبدو عليه هذا المخرج.
- على سبيل المثال افترض أنك تحتاج الى تطوير برنامج ما لجعل ماكينة الصراف الآلي جاهزة للعمل. في مرحلة التحليل تقوم بتحديد وظيفة الماكينة وهنا سوف تحدد العمليات الضرورية التي تؤديها الماكينة مثل السماح بعمليات السحب والايداع والتحويل، وتوفير أرصدة الحساب، وهكذا. خلال هذه المرحلة تتحدث كذلك الى العملاء المحتملين الذين سوف يستخدمون الماكينة. لجعل الماكينة سهلة الاستعمال بالنسبة للمستخدمين يجب أن تفهم متطلباتهم وتضيف أي عمليات لازمة.
- اذا كان البرنامج يتلاعب في البيانات يجب أن يعلم المبرمج ما هي البيانات وكيف يتم تمثيلها وهذا يعني أنك في حاجة الى النظر الى بيانات العينة. اذا كان البرنامج يقدم مخرجات يجب أن تعلم كيفية توليد وتشكيل النتائج.
- اذا كانت المشكلة معقدة قم بتقسيمها الى مشكلات فرعية وقم بتحليل كل مشكلة فرعية منها واستوعب متطلبات كل مشكلة فرعية.

التصميم:

بعد قيامك بتحليل المشكلة جيداً الخطوة التالية تكون تصميم حلول حسابية من أجل حل المشكلة وإذا قمت بتقسيم المشكلة الى مشكلات فرعية سوف تحتاج الى تصميم حل حسابي لكل مشكلة فرعية.

الحلول الحسابية: عملية حل المشكلة خطوة خطوة حيث يتم الوصول الى الحل في قدر محدود من الزمن.

التصميم المنظم:

تقسيم المشكلة الى مشكلات فرعية أصغر يسمى **تصميم منظم** وأيضاً يعرف منهج التصميم المنظم **بالتصميم من الأعلى الى الأسفل**، و **التحسن التدريجي**، و**برمجة الوحدات**. في التصميم المنظم يتم تقسيم المشكلة الى مشكلات فرعية أصغر ثم يتم تحليل كل مشكلة فرعية ويتم الوصول الى حل من أجل حل المشكلة. ثم يتم دمج حل جميع المشكلات الفرعية لحل المشكلة الكلية. هذه العملية المرتبطة بتنفيذ التصميم المنظم يطلق عليها **البرمجة المنظمة**.

التصميم القائم على الموضوع:

في التصميم القائم على الموضوع تكون الخطوة الأولى في عملية حل المشكلة هي تحديد المكونات التي تسمى "موضوعات" والتي تشكل أساس الحل وتحديد كيفية تفاعل تلك الموضوعات مع بعضها البعض. على سبيل المثال افترض أنك تريد كتابة برنامج يجعل عملية تأجير الفيديو أوتوماتيكية من أجل متجر فيديو محلي. الموضوعان الرئيسيان في هذه المشكلة هما الفيديو والعمل.

بعد تحديد الموضوعات تكون الخطوة التالية تحديد البيانات المرتبطة بكل موضوع والعمليات الممكنة التي يمكن اجرائها على تلك البيانات. على سبيل المثال وبالنسبة لموضوع الفيديو قد تتضمن البيانات اسم الفيلم، والممثلين الأبطال، والمنتج، وشركة الانتاج، وعدد النسخ الموجودة في المخزن، وهكذا. البعض من عمليات موضوع الفيديو قد تتضمن التأكد من اسم الفيلم، وتقليل عدد النسخ في المخزن نسخة بعد تأجير النسخة، وزيادة عدد النسخات بعدد نسخة واحدة عندما يعيد العميل شريط فيديو معين. هذا يوضح أن كل موضوع يتكون من بيانات وعمليات خاصة بتلك البيانات. يقوم الموضوع بدمج البيانات والعمليات الخاصة بالبيانات في وحدة واحدة. في التصميم القائم على الموضوع يكون البرنامج النهائي هو مجموعة من الموضوعات المتفاعلة. لغة البرمجة التي تقوم بتطبيق التصميم القائم على الموضوع تسمى **لغة برمجة قائمة على الموضوع**. سوف تعرف الكثير عن مميزات التصميم القائم على الموضوع في الفصول اللاحقة.

التصميم القائم على الموضوع له ثلاثة مبادئ أساسية:

- التغليف – القدرة على دمج البيانات والعمليات في وحدة واحدة.
- التوريث – القدرة على خلق أنواع جديدة من البيانات من أنواع البيانات المتوفرة.
- تعدد الأشكال – القدرة على استخدام التعبير نفسه للإشارة الى مختلف العمليات.

في C++، يتم تحقيق التغليف من خلال استخدام أنواع البيانات المسماة "فئات". في جزء لاحق من هذا الفصل يتم توضيح كيفية تطبيق الفئات في C++. كما يناقش الفصل رقم 2 التوريث وتعدد الأشكال.

في التصميم القائم على الموضوع تقرر نوع الفئات التي تحتاجها وأعضاء البيانات المرتبطة بها ووظائف الأعضاء ثم تقوم بوصف الطريقة التي تتفاعل بها الفئات مع بعضها البعض.

التطبيق:

في مرحلة التطبيق تقوم بكتابة تأليف نظام برمجة لتطبيق الفئات والوظائف التي تم اكتشافها في مرحلة التصميم.

يقوم هذا الكتاب باستخدام أسلوب التصميم القائم على الموضوع (بالارتباط مع البرمجة المنظمة) لحل مشكلة محددة كما يحتوي على العديد من دراسات الحالة – التي تعرف بأمثلة البرمجة – لحل مشكلة واقعية.

في الصيغة النهائية من البرنامج يتكون البرنامج من العديد من الدوال التي يحقق كل منها هدف محدد. بعض الدوال تعتبر جزء من البرنامج الرئيسي والبعض الآخر يتم استخدامه لتطبيق عمليات متنوعة على الموضوعات. تتفاعل الدوال بشكل واضح مع بعضها البعض وتستفيد من إمكانيات بعضها البعض. من أجل استخدام دالة ما يحتاج المستخدم إلى معرفة فقط كيفية استخدام الدالة وما تقوم به الدالة. يجب ألا يهتم المستخدم بتفاصيل الدالة أي كيفية كتابتها. المثال التالي يوضح هذا المفهوم. افترض أنك تريد كتابة دالة تقوم بتحويل قياس معطى بالبوصة إلى سنتيمترات. تكون صيغة التحويل $1 \text{ بوصة} = 2.54 \text{ سنتيمتر}$. الدالة التالية تؤدي المهمة.

```
double inchesToCentimeters(double inches)
{
    if(inches < 0)
    {
        cerr<<"The given measurement must be nonnegative"<<endl;
        return -1.0;
    }
    else
        return 2.54 * inches;
}
```

الموضوع cerr يتوافق مع تدفق الخطأ المعياري الغير مصفون وبلا حنرف عن الموصوح cout (المخرج الذي يذهب أولاً إلى الصقل) يتم إرسال مخرج cerr إلى سيل الخطأ المعياري الذي يكون عادةً الشاشة.

إذا نظرت إلى هيكل الدالة يمكنك إدراك أنه إذا كانت قيمة البوصات أقل من صفر أي سالبة فإن الدالة تعيد -1.0 وبخلاف ذلك تعيد الدالة الطول المكافئ بالسنتيمترات. لا يحتاج مستخدم هذه الدالة إلى معرفة التفاصيل الخاصة بكيفية تطبيق الحلول الحسابية التي تجد الطول المكافئ بالسنتيمترات. بالرغم

من هذا يجب أن يعرف المستخدم أنه من أجل الحصول على اجابة صحيحة يجب أن تكون المدخلات رقد غير سالب. اذا كانت مدخلات هذه الدالة رقم سالب فان البرنامج ينتج -1.0. يمكن تقديم هذه المعلومات كجزء من توثيق هذه الدالة باستخدام بيانات معينة تسمى شروط مسبقة وشروط تالية.

الشرط المسبق: بيان يحدد الشروط التي يجب أن تكون صحيحة قبل استدعاء الدالة.

الشرط التالي: بيان يحدد الشروط التي يجب أن تكون صحيحة بعد اكمال استدعاء الدالة.

الشرط المسبق والشرط التالي لدالة التحويل من بوصة الى سنتيمتر يمكن تحديدهما كما يلي:

الشرط المسبق: يجب أن تكون قيمة البوصات غير سالبة.

الشرط التالي: اذا كانت قيمة البوصات $0 >$ فان الدالة تنتج -1.0 وبخلاف هذا تنتج الدالة الطول المكافئ بالسنتيمترات.

```
double inchesToCentimeters(double inches)
{
    if(inches < 0)
    {
        cerr<<"The given measurement must be nonnegative"<<endl;
        return -1.0;
    }
    else
        return 2.54 * inches;
}
```

في حالات معينة يمكنك استخدام بيان موافقة C++ لاثبات المدخلات. على سبيل المثال يمكن كتابة الدالة السابقة كما يلي:

الشرط المسبق: يجب أن تكون قيمة البوصات غير سالبة.

الشرط التالي: اذا كانت قيمة البوصات $0 >$ فان الدالة تتوقف وبخلاف هذا تنتج الدالة الطول المكافئ بالسنتيمترات.

```
double inchesToCentimeters(double inches)
{
    assert(inches >= 0);
    return 2.54 * inches;
}
```

اذا فشلت بيانات الاثبات يتوقف كامل البرنامج وهذا قد يكون مناسباً اذا اعتمد باقي البرنامج على تنفيذ الدالة. على الجانب الآخر يمكن للمستخدم التأكد من القيمة التي تنتجها الدالة وتحديد ما اذا كانت القيمة المنتجة صحيحة أم لا ويتقدم الى الأمام بناءاً على ذلك. لاستخدام دالة الاثبات تحتاج الى ادخال الملف cassert في برنامجك.

لاطفاء بيانات assert في برنامج ما قم باستخدام المعالج المسبق التوجيهي

```
# define NDEBUG
```

هذا الاتجاه يجب وضعه أمام البيان

```
# include <cassert>
```

كما ترى يمكن تطبيق نفس الدالة بطريقة مختلفة وبرامج مختلفة. لأن مستخدم الدالة لا يحتاج الى الاهتمام بتفاصيل الدالة فان الشروط المسبقة والشروط التالية تكون محددة بنموذج الدالة. هذا يعني أن يتم اعطاء المعلومات التالية للمستخدم:

Double inchesToCentimeters (double inches);

الشرط المسبق: يجب أن تكون قيمة البوصات غير سالبة.

الشرط التالي: اذا كانت قيمة البوصات $0 >$ فان الدالة تنتج -1.0 وبخلاف هذا تنتج الدالة الطول المكافئ بالسنتيمترات.

كمثال آخر من أجل استخدام الدالة التي تبحث عن عنصر معين في قائمة يجب أن تكون القائمة موجودة قبل استدعاء الدالة. وبعد انتهاء البحث تنتج الدالة صحيح أو خطأ بناءً على ما اذا كان البحث ناجح أم لا.

Bool search (int list [], int listLength, int searchItem);

الشرط المسبق: يجب أن تكون القائمة متوفرة.

الشرط التالي: الدالة تنتج صحيح اذا كان عنصر البحث موجود في القائمة وبخلاف هذا تنتج الدالة خطأ.

الاختبار والتصحيح:

مصطلح "اختبار" يشير الى اختبار صحة البرنامج أي التأكد من أن البرنامج ما يؤدي ما هو مفترض منه كما يشير مصطلح "تصحيح" الى العثور على الأخطاء وتصحيحها اذا وجدت.

بمجرد كتابة دالة ما أو حلول حسابية ما تكون الخطوة التالية هي التأكد من أنها تعمل بشكل صحيح. بالرغم من هذا وفي برنامج طويل ومعقد تتواجد الأخطاء بالتأكيد. لهذا ومن أجل زيادة دقة البرنامج يجب اكتشاف الأخطاء واصلاحها قبل أن يتم اصدار البرنامج الى المستخدم.

بالتأكيد يمكنك اثبات صحة البرنامج باستخدام البعض من تحليل (ربما رياضي) للبرنامج. ومع هذا وبالنسبة الى البرامج الكبيرة والمعقدة قد لا يكون هذا الأسلوب وحده كافياً لأنه يمكن ارتكاب أخطاء في الاثبات. لهذا فاننا أيضاً نعتمد على الاختبار من أجل تحديد جودة البرنامج حيث يمر البرنامج خلال سلسلة من الاختبارات المحددة التي تسمى **حالات اختبار** في محاولة للعثور على أية مشكلات.

تتكون حالة الاختبار من مجموعة من المدخلات، أو أفعال المستخدم، أو ظروف مبدئية أخرى، والمخرجات المتوقعة. لأن حالة الاختبار يمكن اعادةها عدة مرات يجب أن يتم توثيقها بشكل صحيح. من المعتاد أن يقوم أي برنامج بالتلاعب في مجموعة كبيرة من البيانات ولهذا فانه من غير العملي (بالرغم من أنه ممكن) عمل حالات اختبار لجميع المدخلات الممكنة. افترض على سبيل المثال أن البرنامج يتلاعب بالأعداد الصحيحة. من الواضح أنه من غير الممكن عمل حالة اختبار لكل عدد صحيح. يمكنك تصنيف حالات الاختبار الى فئات منفصلة تسمى فئات التعادل. فئة التعادل عبارة عن مجموعة من القيم المدخلة التي من المحتمل أن تنتج نفس المخرجات. افترض على سبيل المثال أن

لديك دالة تأخذ العدد الصحيح كمدخل وتنتج كلمة صحيح اذا كان العدد الصحيح غير سالب وكلمة خطأ بخلاف هذا. في هذه الحالة يمكنك تكوين فئتين تعادل واحدة مكونة من أرقام سالبة والأخرى مكونة من أرقام غير سالبة.

هناك نوعان من الاختبار – اختبار الصندوق الأسود واختبار الصندوق الأبيض. في اختبار الصندوق الأسود، لا تعرف العمل الداخلي للحلول الحسابية أو الدالة وتعرف فقط ما تقوم به الدالة. اختبار الصندوق الأسود مبني على مدخلات ومخرجات. حالات الاختبار الخاصة باختبار الصندوق الأسود يتم اختيارها عادةً عن طريق عمل فئات تعادل. اذا كانت الدالة تعمل لمدخل واحد في فئة التعادل فمن المتوقع أن تعمل للمدخلات الأخرى في نفس الفئة.

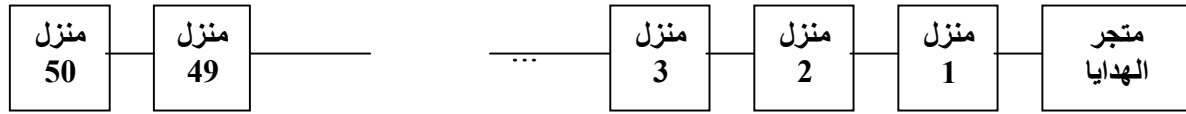
افترض أن الدالة isWithinRange تنتج القيمة (صحيح) اذا كان العدد الصحيح أكبر من أو يساوي صفر وأقل من أو يساوي 100. في اختبار الصندوق الأسود يتم اختبار الدالة على القيم المحيطة والواقعة على الحدود المسماة بالقيم الحدية وكذلك قيم عامة من فئات التعادل. بالنسبة الى دالة isWithinRange قد تكون القيم الحدية في اختبار الصندوق الأسود: -1.0، و1، و99، و100 ولهذا قد تكون قيم الاختبار -500، و-1.0، و1، و50، و99، و100، و101، و500.

يعتمد اختبار الصندوق الأبيض على البنية الداخلية وعلى تطبيق الدالة أو الحلول الحسابية. الهدف هو التأكد من أن كل جزء من الدالة أو الحل الحسابي تم تنفيذه مرة واحدة على الأقل. افترض أنك تريد التأكد مما اذا كانت الدالة تعمل بشكل صحيح. يجب اذن أن تتكون حالات الاختبار من مدخل واحد على الأقل تقوم ببيان اذا بتقييم أنه صحيح وحالة واحدة على الأقل تقيمها بأنها خطأ. يمكن بالمثل اختبار الحلقات والبنىات الأخرى.

تحليل الحلول الحسابية: رمز O الكبيرة:

كما يتم تحليل المشكلة قبل كتابة الحلول الحسابية وبرنامج الحاسب الآلي يجب تحليله مرة أخرى بعد تصميم الحلول الحسابية. عادةً توجد طرق متنوعة لتصميم حل حسابي معين وهناك حلول حسابية محددة تأخذ في التنفيذ القليل للغاية من وقت الحاسب الآلي بينما توجد حلول أخرى تأخذ قدر كبير من الوقت.

قم بالتفكير في المشكلة التالية. موسم الاجازات يقترب ويتوقع متجر للهدايا أن تتضاعف مبيعاته مرتين أو ثلاث مرات عن المبيعات العادية. قام المتجر بتعيين أفراد توصيل اضافيين لتوصيل الهدايا في الوقت المناسب. تقوم الشركة بحساب المسافة الأقرب من المتجر الى مكان معين وتقوم بتسليم المسار الى السائق. افترض أنه يجب ايصال 50 هدية الى 50 منزل مختلف وأن المتجر أثناء تحديده للمسار سوف يجد أن الخمسين منزل يقع على بعد ميل واحد وفي منطقة واحدة وأن المنزل الأول يقع كذلك على بعد ميل واحد من المتجر (انظر شكل 1-1).



شكل 1-1: متجر الهدايا والخمسين منزل.

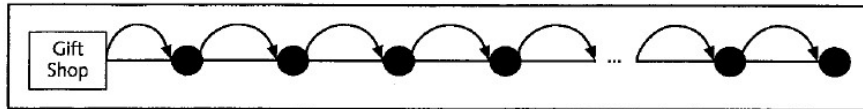
لتبسيط هذا الشكل نستخدم شكل 2-1.

كل نقطة تمثل منزل ز المسافة بين المنازل ميل واحد كما هو موضح في شكل 2-1.



شكل 2-1: متجر الهدايا وكل نقطة تمثل منزل.

لتوصيل 50 هدية الى الأماكن المرسله اليها يقوم أحد السائقين بالتقاط الخمسين هدية ويقود مسافة ميل واحد الى المنزل الأول ويقوم بتسليم الهدية الأولى ثم يقود مسافة ميل آخر ويقوم بتسليم الهدية الثانية وميل آخر لتسليم الهدية الثالثة وهكذا. شكل 3-1 يوضح خطة التسليم هذه.



شكل 3-1: خطة توصيل الهدايا.

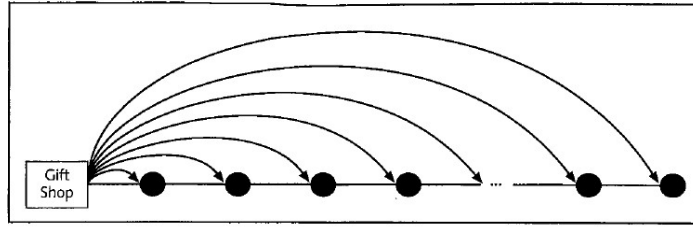
باستخدام هذا المخطط تكون المسافة التي يقودها السائق لتوصيل الهدايا هي:

$$50 = 1 + \dots + 1 + 1 + 1$$

لهذا تكون المسافة الكلية التي يسافرها السائق لتوصيل الهدايا ثم الرجوع مرة أخرى الى المتجر هي:

$$100 = 50 + 50$$

هناك سائق آخر له مسار مماثل لتوصيل مجموعة أخرى مكونة من 50 هدية. ينظر السائق الى المسار ويقوم بتوصيل الهدايا كما يلي: يجمع السائق الهدية الأولى ويقود ميل واحد الى المنزل الأول ثم يعود مرة أخرى الى المتجر. بعد ذلك يلتقط السائق الهدية الثانية ويقود 2 ميل ويقوم بتوصيل الهدية الثانية ثم يعود الى المتجر. بعد هذا يلتقط السائق الهدية الثالثة ويقود 3 أميال ويقوم بتوصيل الهدية ويعود الى المتجر. شكل 4-1 يوضح خطة التوصيل هذه.



شكل 4-1: خطة أخرى لتوصيل الهدايا.

هذا القائد يقوم بتوصيل هدية واحدة فقط كل مرة وبعد توصيل الهدية يعود مرة أخرى الى المتجر لكي يأخذ الهدية التالية ويقوم بتوصيلها. باستخدام هذه الخطة تكون المسافة الكلية التي يسافرها السائق لتوصيل الهدايا ثم العودة الى المتجر:

$$2 * (1 + 2 + 3 + \dots + 50) = 2550 \text{ ميل.}$$

افترض الآن أن هناك عدد n من الهدايا يجب توصيلها الى عدد n من المنازل وكل منزل يقع على بعد ميل من كل منزل كما هو موضح في شكل 4-1. اذا تم توصيل الهدايا باستخدام الخطة الأولى فان المعادلة التالية تعطي المسافة الكلية التي تم قطعها:

$$\underbrace{1+1+1+\dots+1}_{n \text{ times}} + n = 2n \quad (1-1)$$

اذا تم توصيل الهدايا باستخدام الطريقة الثانية فان المسافة المقطوعة تكون:

$$2 * (1 + 2 + 3 + \dots + n) = 2 * \left(\frac{n(n+1)}{2}\right) = n^2 + n \quad (1-2)$$

في المعادلة (1-1) نقول أن المسافة المقطوعة هي دالة n . لنقم بالتفكير في المعادلة (2-1) ففي هذه المعادلة وبالنسبة الى القيم الكبيرة من n سوف نجد أن الطرف المكون من n^2 يصبح الطرف السائد والطرف المكون من n يصبح مهملاً. في هذه الحالة نقول أن المسافة المقطوعة دالة من n^2 . يقوم جدول 1-1 بتقييم المعادلات (1-1) و (2-1) لنسب معينة من n . (الجدول يوضح أيضاً قيمة n^2).

جدول 1-1: قيم n ، و $2n$ ، و n^2 ، و $n^2 + n$.

| $n^2 + n$ | n^2 | $2n$ | n |
|-----------|-------------|--------|--------|
| | 10 | 2 | 1 |
| | 100 | 20 | 10 |
| | 10000 | 200 | 100 |
| | 1000000 | 2000 | 1000 |
| | 100.000.000 | 20.000 | 10.000 |

أثناء تحليل حل حسابي معين نقوم عادةً باحتساب عدد العمليات التي ينفذها الحل الحسابي. نقوم بالتركيز على عدد العمليات وليس على الزمن الفعلي الذي يأخذه الحاسب الآلي في تنفيذ الحل الحسابي وهذا يرجع الى حقيقة أنه يمكن تطبيق حل حسابي معين على مجموعة متنوعة من الحاسبات الآلية ويمكن لسرعة الحاسب أن تؤثر على زمن التنفيذ. بالرغم من هذا فسوف يكون عدد العمليات التي يؤديها الحل الحسابي هو نفس العدد على كل حاسب. لنقم بالتفكير في الأمثلة التالية:

مثال 1-1:

انظر في الحل الحسابي التالي (افترض أن جميع المتغيرات يتم اعلانها بشكل صحيح).

```
cout<<"Enter two numbers";           //Line 1
cout<<flush;                          //Line 2

cin>>num1>>num2;                     //Line 3

if(num1 >= num2)                      //Line 4
    max = num1;                       //Line 5
else                                  //Line 6
    max = num2;                       //Line 7

cout<<"The maximum number is: "<<max<<endl; //Line 8
```

الصف 1 به عملية واحدة، >> الصف 2 به عملية واحدة، >> الصف 3 به اثنان، << عمليات،
الصف 4 به عملية واحدة، <= الصف 5 به عملية واحدة، = الخط 7 به عملية واحدة، =، و
الصف 8 به ثلاثة، >>، عمليات. كلا من الصف 5 والصف 7 لا يطبقون. لهذا العدد الكلي للعمليات
المنفذة في الكود السابق هو $9 = 3 + 1 + 1 + 2 + 1 + 1$. في هذا الحل الحسابي يكون عدد
العمليات ثابت.

مثال 2-1:

انظر في الحل الحسابي التالي:

```
cout<<"Enter positive integers ending with -1"<<endl; //Line 1

count = 0                                           //Line 2
sum = 0;                                           //Line 3

cin>>num;                                          //Line 4

while(num != -1)                                   //Line 5
{
    sum = sum + num;                               //Line 6
    count++;                                       //Line 7
    cin>>num;                                       //Line 8
}
```

```

cout<<"The sum of the numbers is: "<<sum<<endl;           //Line 9

if(count != 0)                                             //Line 10
    average = sum / count;                                //Line 11
else                                                       //Line 12
    average = 0;                                           //Line 13

cout<<"The average is: "<<average<<endl;                 //Line 14

```

هذا الحل الحسابي به 5 عمليات (الصف 1 وحتى 4) قبل الحلقة وبالمثل هناك 9 أو 8 عمليات بعد الحلقة بناءً على ما اذا كان الصف 11 أو الصف 13 يطبق.

الصف 5 به عملية واحدة و 4 عمليات ضمن الحلقة (الصف 6 وحتى 8). لهذا من الصف 5 والى الصف 8 بهم 5 عمليات. اذا طبقت الحلقة 10 مرات يتم تطبيق تلك العمليات الخمس 10 مرات. كما يتم تنفيذ عملية واحدة اضافية عند الصف 5 لايقاف الحلقة. بهذا يكون عدد العمليات المنفذة 51 من الصف 5 والى 8.

اذا تم تطبيق الحلقة 10 مرات فان العدد الكلي للعمليات المنفذة يكون:

$$10 * 5 + 1 + 5 + 9 \text{ أو } 10 * 5 + 1 + 5 + 8$$

وهذا يعني:

$$10 * 5 + 15 \text{ أو } 10 * 5 + 14$$

يمكننا تعميمها في حالة تطبيق الحلقة n مرات واذا تم تنفيذ الحلقة n مرات يكون عدد العمليات المنفذة هو:

$$15 + 5n \text{ أو } 14 + 5n$$

في هذه التعبيرات وبالنسبة الى القيم الكبيرة من n يصبح الطرف 5n الطرف المسيطر والطرف 15 و14 مهمل.

في الحلول الحسابية هناك عمليات محددة تكون عادةً مسيطرة. على سبيل المثال في الحل الحسابي السابق لاضافة ارقام تكون العملية السائدة في الصف 6. وبالمثل في البحث عن حل حسابي ولأنه تتم مقارنة عنصر البحث مع العناصر في القائمة فسوف تكون العملية السائدة هي المقارنة وهذا يعني العملية ذات الصلة. لهذا وفي حالة بحث الحل الحسابي نحصي عدد المقارنات. افترض كمثال آخر أننا نكتب برنامج لضرب المصفوفات وضرب المصفوفات يتضمن الجمع والضرب. اننا نقوم باحصاء عدد عمليات الضرب لأن الضرب يأخذ زمن أطول من الحاسب الآلي لكي يقوم به ولكي يحلل الحل الحسابي لضرب المصفوفة.

ان هذا الكتاب لا يضع حلول حسابية فقط ولكنه يوفر أيضاً تحليل منطقي لكل حل حسابي. في الحقيقة اذا كان هناك عدة حلول حسابية تحقق مهمة معينة فان تحليل الحل الحسابي يسمح للمبرمج بالاختيار من بين الخيارات المتنوعة.

افترض أن هناك حل حسابي يؤدي $f(n)$ عمليات أساسية لانجاز مهمة ما حيث n هي حجم المسألة. افترض انك تريد تحديد ما اذا كان عنصر ما بالقائمة أم لا. فضلاً عن هذا افترض أن حجم القائمة هو n . من أجل تحديد ما اذا العنصر في القائمة أم لا هناك العديد من الحلول الحسابية كما سوف ترى في الفصل 9. بالرغم من هذا فان الطريقة الأساسية هي مقارنة العنصر بالعناصر الموجودة في القائمة. بهذا يعتمد أداء الحل الحسابي على عدد المقارنات.

في حالة البحث يكون n هو حجم القائمة ويصبح $f(n)$ دالة العد وهذا يعني أن $f(n)$ تعطي عدد المقارنات التي يتم عملها من جانب حل البحث الحسابي. افترض أنه بحاسب آلي معين يأخذ تنفيذ عملية واحدة c وحدات من زمن الحاسب الآلي. بهذا يكون الزمن الذي يستهلكه الحاسب الآلي في تنفيذ $f(n)$ عملية هو $cf(n)$. يعتمد الثابت c بوضوح على سرعة الحاسب الآلي ولهذا يتنوع من حاسب الى آخر. بالرغم من هذا فان عدد العمليات الأساسية $f(n)$ يكون نفسه في كل حاسب. اذا كنا نعلم كيف تزداد الدالة $f(n)$ مع زيادة حجم المسألة يمكننا تحديد كفاءة الحل الحسابي. انظر الى جدول 2-1.

جدول 2-1 معدل نمو دوال متنوعة:

| n | $\log_2 n$ | $N \log_2 n$ | n^2 | $2n$ |
|-----|------------|--------------|-------|---------------|
| 1 | 0 | 0 | 1 | 2 |
| 2 | 1 | 2 | 2 | 4 |
| 4 | 2 | 8 | 16 | 15 |
| 8 | 3 | 24 | 64 | 256 |
| 16 | 4 | 64 | 256 | 65.536 |
| 32 | 5 | 160 | 1.024 | 4.298.967.296 |

يوضح جدول 2-1 كيف تنمو دوال معينة مع نمو العامل n وهو حجم المشكلة. افترض أن حجم المسألة تضاعف. من جول 2-1 ينتج أنه اذا كان عدد العمليات الأساسية هو دالة من $f(n) = n^2$ فان عدد العمليات الأساسية يصبح أربعة أضعاف. اذا كان عدد العمليات الأساسية دالة من $f(n) = 2^n$ فان عدد العمليات الأساسية يكون مربع. بالرغم من هذا اذا كان عدد العمليات الأساسية دالة من $f(n) = \log_2 n$ فان التغير في عدد العمليات الرئيسية يكون غير مؤثر.

افترض أن الحاسب يمكنه تنفيذ 1 بليون خطوة في الثانية فان جدول 3-1 يوضح الوقت الذي يستهلكه الحاسب الآلي في تنفيذ $f(n)$ خطوات.

| $f(n)=2^n$ | $f(n)=n^2$ | $f(n)=n\log_2 n$ | $f(n)=\log_2 n$ | $f(n)=n$ | n |
|------------|-------------|------------------|-----------------|--------------|----|
| 1 | μs 0.1 | μs 0.033 | μs 0.003 | μs 0.01 | 10 |
| ms1 | μs 0.4 | μs 0.086 | μs 0.004 | μs 0.02 | 20 |
| s1 | μs 0.9 | μs 0.147 | μs 0.005 | μs 0.03 | 30 |

في هذا الجدول $10^{-6} \mu s = 1 \text{ ثانية}$ و $10^{-3} \text{ ثانية} = 1 \text{ ms}$.

جدول 1-3 زمن $f(n)$ أمر على حاسب آلي يقوم بتنفيذ 1 بليون أمر كل ثانية (يتبع)

| $f(n)=2^n$ | $f(n)=n^2$ | $f(n)=n\log_2 n$ | $f(n)=\log_2 n$ | $f(n)=n$ | n |
|------------|------------|------------------|-----------------|----------|-------------|
| | | 0.218 | 0.005 | 0.04 | 4 |
| | | 0.282 | 0.006 | 0.05 | 50 |
| | | 0.664 | 0.007 | 0.10 | 100 |
| | | 9.966 | 0.010 | 1.00 | 1000 |
| | | 130 | 0.013 | 10 | 10.000 |
| | | 1.67 | 0.017 | 0.10 | 100.000 |
| | | | | 0.01 | 1.000.000 |
| | | | | 0.10 | 10.000.000 |
| | | | | 1.00 | 100.000.000 |

في هذا الجدول $10^{-6} \mu s = 1 \text{ ثانية}$ و $10^{-3} \text{ ثانية} = 1 \text{ ms}$.

بقية هذا القسم يقوم بعمل تدوين لكيفية نمو الدالة $f(n)$ مع زيادة n دون حدود. هذا يعني أن القسم يقوم بتدوين ما هو مفيد في وصف سلوك الحلول الحسابية ويعطينا أفيد المعلومات عن الحلول الحسابية. أولاً نقوم بتعريف مصطلح "مقارب".

تعريف: لتكن f دالة n . المصطلح "مقارب" يعني دراسة الدالة f حيث تصبح n أكبر وأكبر دون حد. انظر الى الدوال $g(n) = n^2$ و $f(n) = n^2 + 4n + 20$. ان الدالة g لا تحتوي على أي عنصر خطي وهذا يعني أن معامل n في g يساوي صفر. انظر الى جدول 1-4.

جدول 1-4: معدل نمو n^2 و $n^2 + 4n + 20$:

| $f(n) = n^2 + 4n + 20$ | $g(n) = n^2$ | n |
|------------------------|--------------|-------------|
| 10 | 100 | 360 |
| 50 | 2500 | 2720 |
| 100 | 10.000 | 10.420 |
| 1.000 | 1.000.000 | 1.004.020 |
| 10.000 | 100.000.000 | 100.040.020 |

من جدول 4-1 ينتج أن مع زيادة n يصبح الحد $4n+20$ في $f(n)$ غير مؤثر والحد n^2 يصبح الحد السائد. بالنسبة الى القيم الكبيرة من n يمكننا توقع سلوك $f(n)$ بالنظر الى سلوك $g(n)$. في تحليل الحل الحسابي اذا كان يمكن وصف تعقيد الدالة بواسطة تعقيد دالة تربيعية دون حد خطي نقول أن الدالة من $O(n^2)$ تسمى "Big-O of n^2 ".

لتكن f و g دوال ذات قيمة حقيقية افترض أن f و g ليست سالبة. تعريف: نقول أن $f(n)$ عبارة عن $Big-O$ of $g(n)$ ويتم كتابتها $f(n) = O(g(n))$ اذا كان يوجد ثوابت ايجابية c و n_0 مثل: $f(n) \leq cg(n)$ for all $n \geq n_0$

يوضح جدول 5-1 بعض الدوال المشتركة من Big-O التي تظهر في تحليل الحل الحسابي. لتكن $f(n) = O(g(n))$ حيث n هو حجم المشكلة.

جدول 5-1: بعض دالات Big-O التي تظهر في تحليل الحل الحسابي:

| الدالة $g(n)$ | معدل نمو $f(n)$ |
|---------------|-----------------|
| | |
| | |
| | |
| | |
| | |

باستخدام التدوينات السابقة يمكننا استنتاج أن المعادلة (1-1) من $O(n)$ والمعادلة (2-1) من $O(n^2)$ فضلاً عن ذلك يكون الحل الحسابي في مثال 1-1 من $O(1)$ والحل الحسابي في مثال 2 من $O(n)$.

$$O(1) < O(\log_2 n) < O(n) < O(n) < O(n \cdot \log_2 n) < O(n^2) < O(2^n)$$

يمكن للقارئ أن يتخطى هذا القسم اذا كان يعرف كيفية تطبيق الفئات في C++. تذكر أنه في التصميم القائم على الموضوع تكون الخطوة الأولى هي تحديد المكونات التي تعرف بالموضوعات والموضوع يقوم بتغليف أو دمج البيانات والعمليات الواقعة على تلك البيانات في وحدة واحدة. في C++ تكون الآلية التي تسمح لك بدمج البيانات والعمليات الواقعة على تلك البيانات في وحدة واحدة مسماة بـ "فئة" وهذا القسم يصف كيفية استخدام الفئات في C++.

الفئة عبارة عن مجموعة من عدد ثابت من المكونات ويطلق على مكونات الفئة أعضاء الفئة. التركيب العام لتعريف الفئة هو:

```
class classIdentifier
{
    classMemberList
};
```

حيث تتكون classMemberList من بيانات و/أو دوال متغيرة. وهذا يعني أن عضو الفئة قد يكون اما متغير (لتخزين البيانات) أو دالة.

- اذا كان عضو الفئة عبارة عن متغير فانك تعلنه كما هو مثل أي متغير آخر. فضلاً عن ذلك في تعريف الفئة لا يمكنك بدء متغير هند اعلانك اياه.
- اذا كان عضو الفئة عبارة عن دالة فأنت تستخدم نموذج الدالة لتعريف هذا العضو.
- اذا كان عضو الفئة عبارة عن دالة فيمكنه (مباشرةً) الوصول الى أي عضو في الفئة – أعضاء البيانات وأعضاء الدالة. هذا يعني أنه عندما تكتب تعريف دالة العضو يمكنك تناول بيانات أي عضو من الفئة دون تمريره كعامل ثابت. الشرط الوحيد الواضح هو أنه يجب عليك اعلان معرف قبل أن تستخدمه.

في C++ تكون الفئة كلمة محفوظة وتحدد فقط نوع البيانات ولا يتم تخصيص أي ذاكرة. انها تعلن بيان الفئة فضلاً عن هذا لاحظ الفاصلة المنقوطة (;) بعد الزوج الأيمن. الفاصلة المنقوطة جزء من التركيب ولهذا تتسبب الفاصلة المنقوطة المفقودة في خطأ في التركيب.

يتم تصنيف أعضاء الفئة الى ثلاث فئات: خاصة وعامة ومصونة ويطلق عليها محددات وصول العضو. هذا الفصل يناقش النوعين الأوليين بشكل أساسي – العام والخاص. فيما يلي بعض الحقائق عن أعضاء الفئة العامة والخاصة:

- افتراضياً يكون جميع أعضاء أي فئة عامين.
- اذا كان عضو الفئة خاص لا يمكنك الوصول اليه خارج الفئة.
- العضو العام يمكن الوصول اليه خارج الفئة.
- لجعل عضو الفئة عام تستخدم محدد تناول العضو خاص بفاصلة.

في C++، تعتبر كلمات خاصة ومصونة وعامة كلمات محفوظة.

مثال 3-1:

افترض أننا نريد تعريف فئة clockType لتطبيق الزمن في البرنامج. فضلاً عن هذا افترض أن الوقت يتم تمثيله كمجموعة من ثلاثة أعداد صحيحة: عدد يمثل الساعات وعدد يمثل الدقائق وعدد يمثل الثواني. اننا نريد أيضاً أداء العمليات التالية على الزمن:

- 1- ضبط الزمن.
- 2- اعادة الزمن.
- 3- طباعة الزمن.
- 4- زيادة الزمن ثانية واحدة.
- 5- زيادة الزمن دقيقة واحدة.
- 6- زيادة الزمن ساعة واحدة.

7- مقارنة الوقتين من أجل التساوي.

من هذه المناقشة من الواضح أن فئة clockType بها 10 أعضاء: ثلاثة أعضاء بيانات وسبعة أعضاء عمليات.

بعض عناصر الفئة clockType عناصر خاصة. ان اتخاذ قرار بالعناصر التي تكون خاصة والتي تكون عامة يعتمد على طبيعة العنصر. القاعدة العامة هي أن أي عنصر يحتاج الى أن يتم تناوله خارج الفئة يتم اعلانه عام وأي عنصر لا يجب تناوله بشكل مباشر بواسطة المستخدم يجب اعلانه كعنصر خاص. على سبيل المثال يجب أن يكون المستخدم قادراً على ضبط الوقت وطباعته. لهذا فان العناصر التي تضبط وتطبع الوقت يجب أن يتم اعلانها كعناصر عامة.

بالمثل يجب اعلان عناصر زيادة الزمن ومقارنة الزمن من أجل التساوي كعناصر عامة. على الجانب الآخر من أجل التحكم في الاحتكار المباشر لعناصر البيانات: الساعات والدقائق والثواني سوف نعلن أن عناصر البيانات هذه خاصة. فضلاً عن هذا لاحظ أنه اذا كان المستخدم يمتلك وصول مباشر الى عناصر البيانات فانه لا يكون هناك حاجة الى دوال عناصر مثل تحديد الزمن (setTime).

البيانات التالية تقوم بتعريف فئة clockType (نوع الساعة):

فئة نوع الساعة:

عامة:

ابطال تحديد الزمن (ساعات، دقائق، ثواني)،

// دالة ضبط الزمن.

//يتم ضبط الزمن وفقاً للعوامل.

// شرط مسبق: س = ساعات، د = دقائق، ث = ثواني.

// تتأكد الدالة مما اذا كانت قيم الساعات والدقائق والثواني صحيحة أم لا واذا كانت هناك قيمة غير

صحيحة يتم تحديد القيمة الافتراضية صفر.

ابطال الزمن (ساعات ودقائق وثواني)،

// دالة استرجاع الزمن.

// شرط مسبق: ساعات = س، دقائق = د، ثواني = ث.

ابطال طباعة الزمن ()،

// دالة طباعة الزمن.

// شرط مسبق: يتم طباعة الزمن في صيغة ساعات: دقائق:ثواني.

ابطال زيادة الثواني ()،

// دالة لزيادة الزمن ثانية واحدة.

// شرط مسبق: يتم زيادة الزمن ثانية واحدة.

// اذا كان الزمن قبل الزيادة 23:59:59 فان الزمن تتم اعادة ضبطه الى 00:00:53

إبطال زيادة الساعات ()،

// دالة لزيادة الزمن ساعة واحدة.

// شرط مسبق: يتم زيادة الزمن ساعة واحدة.

// إذا كان الزمن قبل الزيادة 23:45:53 فإن الزمن تتم إعادة ضبطه الى 00:45:53

Bool زمن متساوي (const clockType& otherClock)،

// دالة لمقارنة الزمنين.

// شرط مسبق: تنتج صحيح إذا كان الزمن مساوي لساعة أخرى بخلاف هذا تنتج خطأ.

خاصة:

Int hr // استعادة الساعات.

Int min // استعادة الدقائق.

Int sec // استعادة الثواني.

لاحظ ما يلي في تعريف فئة نوع الساعة clockType:

- فئة نوع الساعة تتضمن سبعة عناصر دالة: تحديد الزمن، والحصول على الزمن، وطباعة الزمن، وزيادة الثواني، وزيادة الدقائق، وزيادة الساعات، والزمن المتساوي. وتتضمن ثلاثة عناصر بيانات وهي س، د، ث.
 - عناصر البيانات الثلاثة وهي س، د، ث خاصة على الفئة ولا يمكن تناولها خارج الفئة.
 - عناصر الدالة السبعة وهي تحديد الزمن، والحصول على الزمن، وطباعة الزمن، وزيادة الثواني، وزيادة الدقائق، وزيادة الساعات، والزمن المتساوي يمكن أن تتناول عناصر البيانات مباشرة. بمعنى آخر لا يمكننا امرار عناصر البيانات كعوامل لدوال العنصر.
 - في دالة الزمن المتساوي equalTime يكون العامل otherClock عامل اشارة ثابت وهذا يعني أنه في استدعاء الدالة equalTime يتلقى العامل otherClock عنوان العامل الحقيقي ولكن لا يمكن للعامل otherTime أن يقوم بتعديل قيمة العامل الحقيقي. يمكنك اعلان OtherClock كعامل قيمة ولكن هذا قد يتطلب من OtherClock نسخ قيمة العامل الحقيقي الذي قد يتسبب في أداء ضعيف. (من أجل التفسير انظر قسم "عوامل الاشارة وموضوعات الفئة (المتغيرات)" الموجود لاحقاً في هذا الفصل).
 - الكلمة const في نهاية دوال العضو printTime وequalTime تحدد أن تلك الدوال لا يمكنها تعديل عناصر البيانات الخاصة بمتغير من نوع clockType.
- (ترتيب العناصر العامة والخاصة من الفئة) ++C ليس لها ترتيب معين تعلن به الآن العناصر عامة أو خاصة ويمكنك اعلانها في أي ترتيب. الشيء الوحيد الذي تحتاجه اليه هو تذكر أن جميع عناصر الفئة تكون خاصة افتراضياً. يجب أن تستخدم محدد تناول العنصر (عام) public لكي تجعل العنصر متاح للتناول العام. إذا قررت اعلات العناصر الخاصة private بعد العناصر

الخاصة public (كما حدث في حالة clockType) يجب أن تستخدم محدد تناول العنصر (خاص) لبدء اعلان العناصر الخاصة.

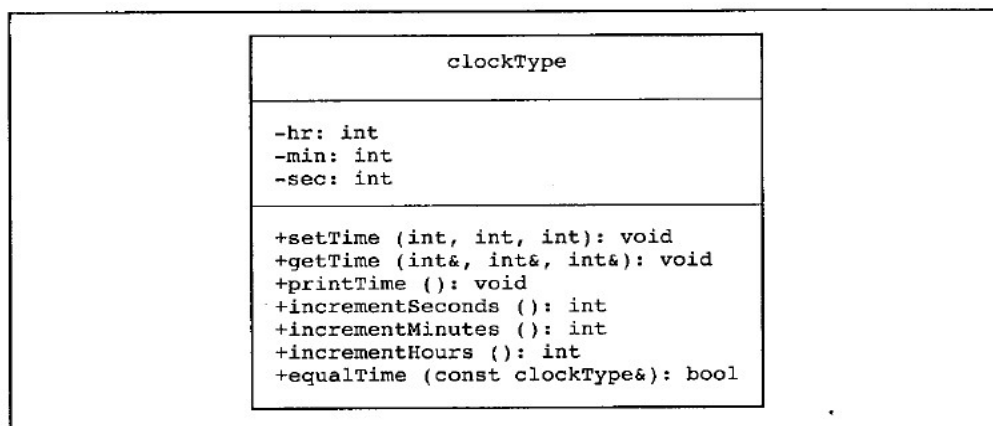
في تعريف الفئة clockType تكون جميع عناصر البيانات private وجميع عناصر الدالة public. بالرغم من هذا يمكن أن يكون عضو الدالة private كذلك. على سبيل المثال اذا تم استخدام دالة العنصر فقط من أجل تطبيق دالات عنصر آخر من الفئة ولا يحتاج المستخدم الى تناول هذه الدالة فانك تجعلها خاصة private. بالمثل يمكن أن يكون عنصر البيانات من فئة ما عام public.

لاحظ أننا لم نقم بعد بكتابة تعريفات عناصر دالة الفئة clockType وسوف نتعلم كيفية كتابتها لاحقاً .

دالة setTime تضبط عناصر البيانات الثلاثة – hr, min, & sec – لقيم معينة. يتم تمرير القيم المعطاة كعوامل لدالة setTime. وتقوم دالة printTime بطباعة الزمن أي قيمة hr, min, & sec. تقوم دالة زيادة الثواني (incrementSeconds) بزيادة الزمن ثانية واحدة، و تقوم دالة زيادة الدقائق (incrementMinutes) بزيادة الزمن دقيقة واحدة، و تقوم دالة زيادة الساعات (incrementHours) بزيادة الزمن ساعة واحدة، ودالة الزمن المتساوي (equalTime) بالمقارنة بين زمنين من أجل التساوي.

الرسوم البيانية للغة التشكيل الموحدة:

يمكن وصف الفئة وعناصرها بيانياً باستخدام مجموعة رموز تعرف باسم رموز لغة التشكيل الموحدة. على سبيل المثال يوضح شكل 1-5 الرسم البياني للغة التشكيل الموحدة للدالة clockType.



شكل 1-5: الرسم البياني للغة التشكيل الموحدة للفئة clockType.

الصندوق العلوي يحتوي على اسم الفئة والصندوق الأوسط يحتوي على عناصر البيانات وأنواع بياناتها والصندوق الأخير يحتوي على اسم دالة العنصر وقائمة العوامل ونوع نتيجة الدالة. تشير علامة زائد (+) أمام العنصر إلى أن هذا العنصر عبارة عن عنصر عام وعلامة ناقص (-) تشير إلى أن العنصر خاص والرمز (#) قبل اسم العنصر يشير إلى أن العنصر عبارة عن نصر محمي.

اعلان المتغير (الموضوع):

بمجرد تعريف الفئة، يمكنك لعلان متغيرات هذا النوع. في مصطلحات C++ يسمى متغير الفئة **موضوع الفئة** أو **مثال الفئة**. لمساعدتك على أن تصبح على علم بهذه المصطلحات سوف نبدأ من الآن باستخدام مصطلح **موضوع الفئة** أو **الموضوع** للتعبير عن متغير الفئة. ان تركيب اعلان الفئة هو نفسه تركيب اعلان أي متغير آخر. البيانات التالية تعلن موضوعين من نوع clockType:

```
clockType    myClock;  
clockType    yourClock;
```

كل موضوع به 10 عناصر: سبعة عناصر دالة وثلاثة عناصر بيانات وكل موضوع له ذاكرة منفصلة مخصصة من أجل الساعات والدقائق والثواني hr, min, & sec. على سبيل المثال وفي وقت محدد قد تحتوي الساعات والدقائق والثواني الخاصة بـ myClock على 5، و12، و30 على التوالي وقد تحتوي الساعات والدقائق وثواني الخاصة بـ yourClock على 12، و35، و45 على التوالي. في الحقيقة يتم تخصيص الذاكرة فقط من أجل عناصر بيانات كل موضوع فئة. المجمع C++ يقوم فقط بتوليد نسخة مادية واحدة فقط من عنصر دالة الفئة وكل موضوع في الفئة يقوم بتنفيذ نفس النسخة من دالة العضو.

تناول عناصر الفئة:

بمجرد اعلان الموضوع يمكنه تناول عناصر الفئة العامة ويكون التركيب العام لتناول عناصر الفئة هو:

classVariableName.memberName

تذكر أنه في C++، تكون النقطة (.) عبارة عن عامل يسمى **عامل تناول العنصر**. المثال 1-4 يوضح كيفية تناول عناصر الفئة.

مثال 1-4:

انظر الى البيانات التالية:

```
myClock.setTime (5, 2, 30);  
myClock.printTime ();  
yourClock.setTime (x, y, z);
```

افترض أن x, y, z متغيرات من نوع int
إذا كانت (myClock.equalTime (yourClock))

.

.

.

هذه البيانات قانونية أي صحيحة من الناحية التركيبية.

في البيان الأول (5, 2, 30) myClock.setTime يتم تطبيق عنصر الدالة setTime. يتم تمرير القيم 5، و2، و30 كعوامل للدالة setTime وتستخدم الدالة تلك القيم لتحديد قيم عناصر البيانات الثلاثة الساعات والدقائق والثواني hr, min, & sec من myClock الى 5، و2، و30 على التوالي. بالمثل تنفذ البيان الثاني دالة العنصر printTime وتخرج محتويات عناصر البيانات الثلاثة من myClock. في البيان الثالث يتم استخدام قيم المتغيرات x, y, z لتحديد قيم عناصر البيانات الثلاثة من yourClock.

في البيان الرابع، تقوم دالة العنصر equalTime بتنفيذ ومقارنة عناصر البيانات الثلاثة من myClock مع عناصر البيانات المتوافقة من yourClock.

لأن في هذا البيان تكون equalTime عنصر من الموضوع myClock فان لها تناول مباشر لعناصر البيانات الثلاثة من myClock. لهذا فهي تحتاج الى موضوع آخر الذي يكون في هذه الحالة yourClock الذي تتم المقارنة معه. هذا يفسر سبب امتلاك الدالة equalTime لعامل واحد فقط. موضوع الفئة يمكنه تناول عناصر الفئة العامة فقط ولهذا تكون البيانات التالية غير قانونية لأن الساعات والدقائق والثواني (hr, min, & sec) عبارة عن عناصر خاصة من الفئة clockType ولهذا لا يمكن تناولها بالموضوع myClock:

```
myClock.hr = 10;           //illegal
myClock.min = yourClock.min; //illegal
```

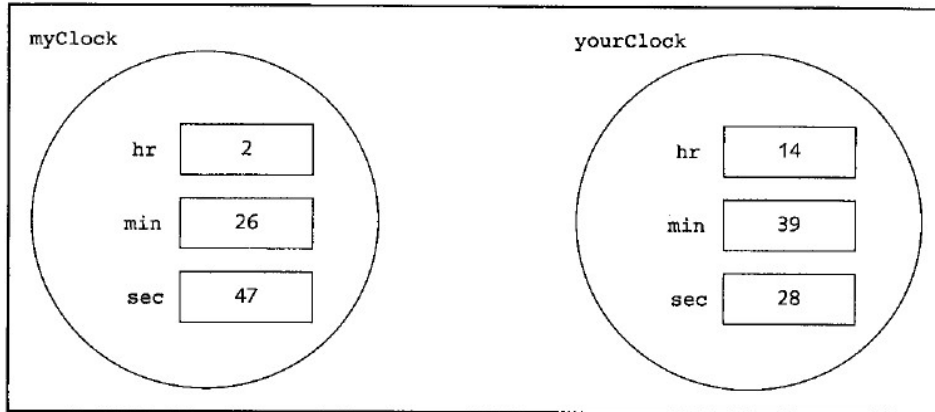
العمليات المدمجة التي تتم على الفئات:

غالبية العمليات المدمجة الخاصة ب C++ لا تنطبق على الفئات ولا يمكنك استخدام العمليات الحسابية لأداء عمليات حسابية على موضوعات الفئة (الا اذا كانت محملة، انظر الفصل رقم 2). على سبيل المثال لا يمكنك استخدام العامل (+) لاضافة موضوعين فئة للنوع clockType على سبيل المثال فانه لن يمكنك كذلك استخدام العوامل المرتبطة للمقارنة بين موضوعي فئة من أجل التساوي (الا اذا كانت محملة، انظر شكل رقم 2).

العمليتان المدمجتان الصالحتان لموضوعات الفئة هي تناول العنصر (.) وتحديد (=). لقد رأيت كيفية تناول عنصر فردي للفئة باستخدام اسم موضوع الفئة ثم نقطة ثم اسم العنصر. القسم التالي يوضح كيفية عمل بيان التحديد بمساعدة مثال.

عامل وفئات التحديد:

افترض أن myClock و yourClock متغيرات من النوع clockType كما تم التعريف مسبقاً. فضلاً عن هذا افترض أن قيم myClock و yourClock كما هي موضحة في شكل 6-1.



شكل 6-1: موضوعات myClock و yourClock
البيان:

```
myClock = yourClock; // Line 1
```

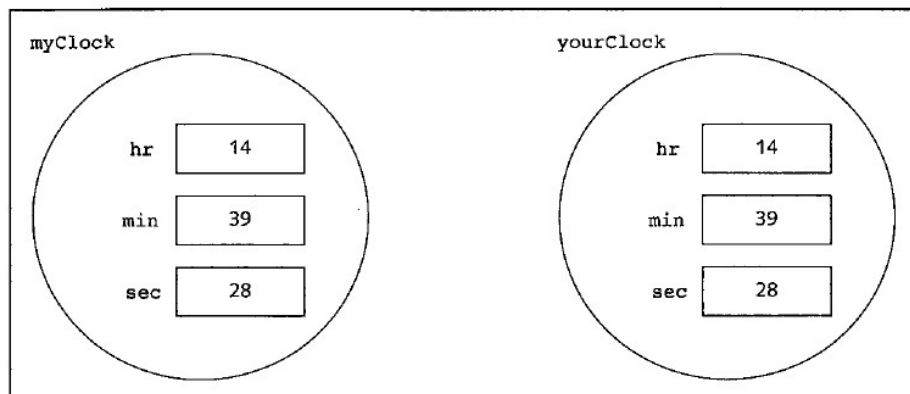
ينسخ قيمة yourClock في myClock وهذا يعني:

1. يتم نسخ قيمة yourClock.hr في myClock.hr.

2. يتم نسخ قيمة yourClock.min في myClock.min.

3. يتم نسخ قيمة yourClock.sec في myClock.sec.

بمعنى آخر، يتم نسخ قيم عناصر البيانات الثلاثة من yourClock في عناصر البيانات المتوافقة من myClock. لهذا فإن بيان التحديد يقوم بتنفيذ نسخة من العنصر. بعد تنفيذ البيان في الصف 1، تكون قيم myClock و yourClock كما هي موضحة في شكل 7-1.



شكل 1-7: موضوعات myClock و yourClock بعد تنفيذ بيان التحديد
myClock = yourClock

مجال الفئة:

كما مع متغيرات الأنواع البسيطة، يمكن أن يكون موضوع الفئة اما آلي (هذا يعني أنه يتم عمله في كل وقت يصل فيه التحكم الى اعلانه وتحطيمه عند خروج التحكم من القالب المحيط) أو ساكن (يتم عمله مرة واحدة عندما يصل التحكم الى اعلانه وتحطيمه عندما يتوقف البرنامج). يمكنك كذلك اعلان نظام موضوعات الفئة. موضوع الفئة له نفس مجال المتغيرات الأخرى. عنصر الفئة يكون مقتصر على الفئة. يمكنك الوصول الى عنصر فئة عام خارج الفئة باستخدام اسم موضوع الفئة وعامل تناول العنصر (.).

الدوال والفئات:

القواعد التالية تصف العلاقة بين الدوال والفئات:

- يمكن تمرير موضوعات الفئة كعوامل للدوال وانتاجها كقيم للدالة.
- مثل العوامل للدوال يمكن تمرير الفئات اما بالقيمة أو بالإشارة.
- اذا تم تمرير موضوع الفئة بالقيمة فانه يتم نسخ محتويات عناصر بيانات العامل الحقيقي داخل عناصر البيانات المتوافقة من العامل الرسمي.

عوامل الاشارة وموضوعات الفئة (متغيرات):

تذكر أنه عندما يتم تمرير المتغير بالقيمة يقوم العامل الرسمي بنسخ قيمة العامل الحقيقي وهذا يعني أن يتم تخصيص الذاكرة لنسخ قيمة العامل الحقيقي للعامل الرسمي. ان موضوع الفئة مثل العامل يمكن تمريره بالقيمة.

افترض أن الفئة بها العديد من عناصر البيانات التي تحتاج قدر كبير من الذاكرة لتخزين البيانات وأنت تحتاج الى تمرير المتغير بالقيمة. عند هذا يستقبل العامل الرسمي المتوافق نسخة من بيانات المتغير. هذا يعني أن المجمع يجب أن يخصص الذاكرة للعامل الرسمي وكذلك لنسخ قيمة عناصر بيانات العامل الحقيقي. هذه العملية قد تتطلب بالإضافة الى القدر الكبير من مكان التخزين كمية معقولة من زمن الحاسب الآلي لكي ينسخ قيمة العامل الحقيقي في العامل الرسمي.

على الجانب الآخر اذا تم تمرير متغير ما بالإشارة فان العامل الرسمي يستقبل فقط عنوان العامل الحقيقي ولهذا هناك طريقة فعالة لتمرير المتغير كعامل وهو بالإشارة. اذا تم تمرير المتغير بالإشارة اذن فعندما يتغير العامل الرسمي يتغير العامل الحقيقي كذلك. بالرغم من هذا فانك في بعض الأحيان لا تريد أن تكون الدالة قادرة على تغيير قيم عناصر البيانات. في C++ يمكنك تمرير متغير ما بالإشارة ولايزال يمنع الدالة من تغيير قيمتها باستخدام كلمة const في اعلان العامل الرسمي. كمثال انظر الى تعريف الدالة التالي:

```
void testTime(const clockType& otherTime)
{
    clockType dTime;
    ...
}
```

تحتوي دالة testTime على عامل ساره وهو otherTime. يتم اعرض العامل otherTime باستخدام الكلمة الرئيسية const. لهذا في استدعاء الدالة testTime يستقبل العامل الرسمي otherTime عنوان العامل الحقيقي ولكن لا يمكن للعامل otherTime تعديل محتويات العامل الحقيقي. على سبيل المثال بعد تنفيذ البيان التالي لا يتم تغيير قيمة myClock:

```
testTime (myClock);
```

بوجه عام اذا كنت تريد اعلان موضوع فئة كعامل قيمة فانك تعلنه كعامل اشارة باستخدام الكلمة الرئيسية const كما هو موضح من قبل.

تذكر أنه اذا كان العامل الرسمي عامل قيمة فيمكنك ضمن تعريف الدالة أن تقوم بتغيير قيمة العامل الرسمي أي يمكنك استخدام بيان التحديد لتغيير قيمة العامل الرسمي (الذي قد لا يكون له تأثير بالطبع على العامل الحقيقي). بالرغم من هذا اذا كان العامل الرسمي عامل اشارة ثابت فانه لا يمكنك استخدام بيان تحديد لتغيير قيمته في الدالة ولا يمكنك استخدام أي دالة أخرى لتغيير قيمته. لهذا وضمن تعريف دالة testTime لا يمكنك تغيير قيمة otherTime. على سبيل المثال سوف يكون ما يلي غير قانوني في تعريف دالة testTime:

```
otherTime.setTime(5, 34, 56); //illegal
otherTime = dTime;           //illegal
```

تطبيق دالات العناصر:

عندما تم تعريف الفئة clockType لعناصر الدالة تم تضمين نماذج الدالة فقط. من أجل أن تعمل تلك الدالات بشكل صحيح يجب أن نكتب الحلول الحسابية المرتبطة بها. هناك طريقة لتطبيق تلك الدالات وهي تقديم تعريف الدالة بدلاً من نموذج الدالة في الفئة ذاتها. لسوء الحظ قد يكون تعريف الفئة طويل للغاية وصعب الفهم. سبب آخر لتقديم نماذج الدالة بدلاً من تعريفات الدالة يعود الى **اخفاء المعلومات** أي أننا نريد اخفاء تفاصيل العمليات المؤداة على البيانات.

بعد هذا لنقم بكتابة تعريفات عناصر دالة الفئة clockType أي أننا سوف نكتب تعريفات الدالات setTime، getTime، printTime، وincrementTime، وequlaTime، وهكذا. المحددات setTime، وprintTime وهلم جرا مقتصرين على الفئة ولهذا لا يمكننا الاشارة اليهم (بشكل مباشر) خارج الفئة. من أجل مرجعية تلك المحددات نستخدم **عامل حل المجال** الذي هو عبارة عن نقطتان فوق بعضهما البعض يتم تكرارهما مرتين. في عنوان تعريف الدالة يكون اسم الدالة هو اسم الفئة ويليه عامل حل المجال ثم اسم الدالة. على سبيل المثال يكون تعريف الدالة setTime كما يلي:

```

void clockType::setTime(int hours, int minutes, int seconds)
{
    if(0 <= hours && hours < 24)
        hr = hours;
    else
        hr = 0;

    if(0 <= minutes && minutes < 60)
        min = minutes;
    else
        min = 0;

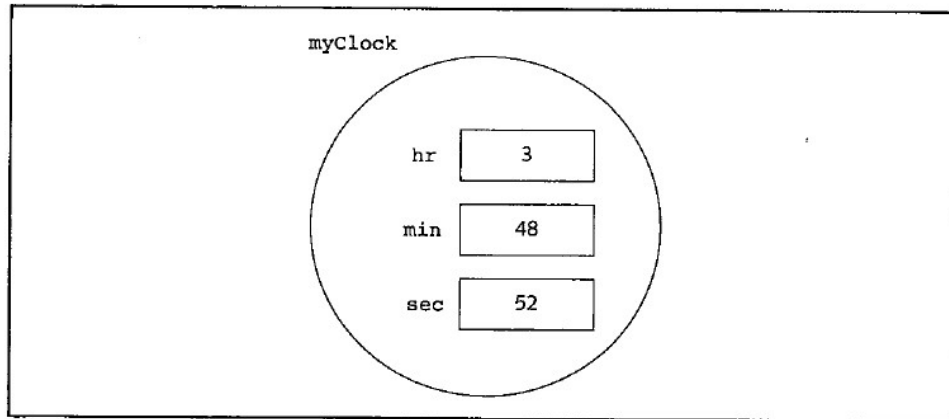
    if(0 <= seconds && seconds < 60)
        sec = seconds;
    else
        sec = 0;
}

```

لاحظ أن تعريف الدالة setTime يتحقق من القيم الصحيحة للساعات والدقائق والثواني. إذا كانت قيمة عامل ما خارج النطاق فإن عنصر البيانات المتوافقة يتم تهيئته الى صفر. على سبيل المثال إذا كانت قيمة الساعات خارج النطاق يتم تهيئة عنصر البيانات hr (الساعات) الى صفر.

افترض أن myClock موضوع للنوع clockType (كما هو معلن من قبل) وانظر الى البيان التالي:
myClock.setTime (3, 48, 52);

بعد تنفيذ هذا البيان يكون موضوع myClock كما هو موضح في شكل 8-1.



شكل 8-1: موضوع myClock بعد تنفيذ البيان myClock.setTime.

بعد هذا نعطي تعريفات عناصر الدالة الأخرى من الفئة clockType ويكون اتباع تعريفات هذه الدالات بسيط وسهل:

```

void clockType::getTime(int& hours, int& minutes, int& seconds
{
    hours = hr;
    minutes = min;
    seconds = sec;
}

```

```

void clockType::printTime() const
{
    if(hr < 10)
        cout<<"0";
    cout<<hr<<":";

    if(min < 10)
        cout<<"0";
    cout<<min<<":";

    if(sec < 10)
        cout<<"0";
    cout<<sec;
}

```

```

void clockType::incrementHours()
{
    hr++;
    if(hr > 23)
        hr = 0;
}

```

```

void clockType::incrementMinutes()
{
    min++;
    if(min > 59)
    {
        min = 0;
        incrementHours(); //increment the hours
    }
}

```

```

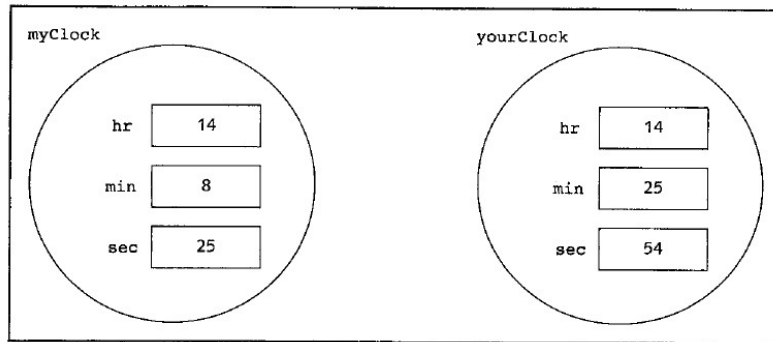
void clockType::incrementSeconds()
{
    sec++;
    if(sec > 59)
    {
        sec = 0;
        incrementMinutes(); //increment the minutes
    }
}

```

من تعريفات دالات incrementMinutes و incrementSeconds من الواضح أن دالة العنصر يمكن أن تستدعي دالات عنصر أخرى.
دالة equalTime لها التعريف التالي:

```
bool clockType::equalTime(const clockType& otherClock) const
{
    return(hr == otherClock.hr
           && min == otherClock.min
           && sec == otherClock.sec);
}
```

لنرى كيفية عمل دالة العنصر equalTime.
افترض أن myClock و yourClock موضوعات للنوع clockType كما هو معلن من قبل.بالإضافة الى هذا افترض أننا لدينا myClock و yourClock كما هو موضح في شكل 9-1.



شكل 9-1: موضوعات myClock و yourClock.

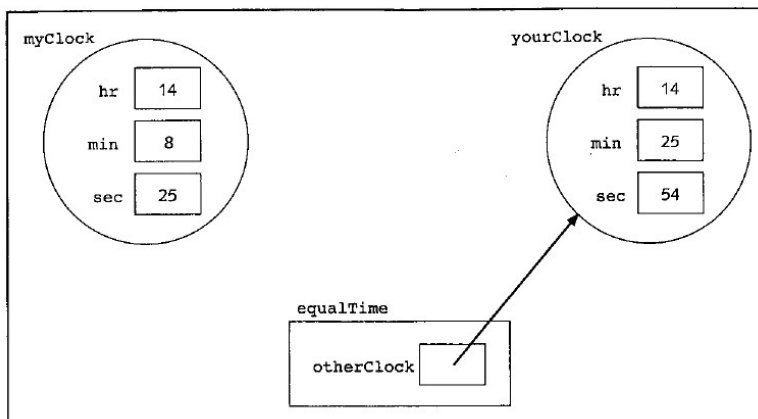
انظر الى البيان التالي:

If (myClock.equalTime (yourClock))

في التعبير:

myClock.equalTime (yourClock)

الموضوع myClock يتناول دالة العنصر equalTime. يتم تمرير عنوان العامل yourClock الى العامل الرسمي otherClock كما هو موضح في شكل 10-1.



شكل 1-10: الموضوع myClock والعامل otherClock.

عناصر البيانات hr, min, & sec (الساعات والدقائق والثواني) تمتلك القيم البالغة 14، و25، و54 على التوالي. بمعنى آخر عندما يجري هيكل الدالة equalTime تكون قيمة otherClock.hr 14، وقيمة otherClock.min تكون 25، وقيمة otherClock.sec تكون 54. ان الدالة equalTime عنصر من الدالة myClock. عندما تجري الدالة equalTime، تكون المتغيرات hr, min, & sec (الساعات والدقائق والثواني) في هيكل الدالة equalTime هي عناصر بيانات المتغير myClock. بناءً على هذا تتم مقارنة العنصر hr من myClock مع otherClock.hr، ومقارنة العنصر min من myClock مع otherClock.min، ومقارنة العنصر sec من myClock مع otherClock.sec.

يتضح مرة أخرى من تعريف الدالة equalTime سبب امتلاكها لعامل واحد فقط.

المقومات:

انظر الى البيانات التالية:

```
clockType myClock;           //Line 1
myClock.setTime(6,27,46);     //Line 2
myClock.printTime();          //Line 3
```

البيان في الصف رقم 2 يحدد قيم عناصر البيانات hr, min, & sec (الساعات والدقائق والثواني) الخاصة ب myClock عند 6 و27 و46 على التوالي والبيان في الصف رقم 3 يتسبب في المحصلة التالية:

06:27:46

بالرغم من هذا اذا قام برنامج ما بتنفيذ البيان الموجود في الصف رقم 3 دون تحديد الزمن بوقت صحيح فان البرنامج يمكنه انتاج بعض الأرقام الغريبة وهذا يرجع الى حقيقة أن C++ لا تقوم بتهيئة المتغيرات بشكل آلي. عناصر الفئة الخاصة لا يمكن تناولها خارج الفئة (في حالتنا عناصر البيانات) ولهذا فان المستخدم اذا نسي تهيئة تلك المتغيرات عن طريق استدعاء الدالة setTime فان البرنامج يقدم نتائج خاطئة.

من أجل ضمان تهيئة عناصر بيانات الفئة تقوم باستخدام المقومات وهناك نوعان من المقومات: بعوامل وبدون عوامل. المقوم الذي يكون بدون عامل يطلق عليه: **مقوم افتراضي**.

المقومات تتميز بالخصائص التالية:

- اسم المقوم هو نفسه اسم الفئة.
- ان المقوم بالرغم من أنه دالة لا يكون له نوع أي أنه لا يكون دالة منتجة لقيمة ما ولا دالة باطلة.

- قد يكون في الفئة الواحدة أكثر من مقوم وبالرغم من هذا فإن جميع مقومات الفئة الواحدة يكون لها نفس الاسم.
 - إذا كانت الفئة بها أكثر من مقوم سواء كان لها عدد مختلف من العوامل الرسمية أم سواء كان عدد العوامل الرسمية ثابت فإن أنواع بيانات العوامل الرسمية يجب عند تحديدها أن تختلف في موضع واحد على الأقل.
 - تحديد المقوم الذي يقوم بالتنفيذ يعتمد على نوع القيم التي تم تمريرها الى موضوع الفئة عندما تم اعلان موضوع الفئة،
 - يتم تنفيذ المقومات بشكل آلي عندما يدخل موضوع الفئة في مجاله ولا يمكن استدعائها مثل الدالات الأخرى لأنها لا تحتوي على أنواع.
- دعونا نتوسع في تعريف الفئة clockType عن طريق تضمين مقومين.

فئة clockType

}

عامة:

Void setTime (int hours, int minutes, int seconds);

// دالة لضبط الوقت.

// يتم ضبط الوقت وفقاً للعوامل.

// شرط مسبق: hr = ساعات، min = دقائق، sec = ثواني.

// تتحقق الدالة مما إذا كانت قيم

// الساعات والدقائق والثواني صحيحة ام لا. اذا

// كانت هناك قيمة غير صحيحة، يتم تحديد القيمة الافتراضية صفر.

Void getTime (int hours, int minutes, int seconds);

// دالة لتقديم الوقت.

// شرط مسبق: hr = ساعات، min = دقائق، sec = ثواني.

Void printTime () const;

// دالة لطبع الوقت.

// شرط تالي: تتم طباعة الوقت في صيغة

.hh:mm:ss //

Void incrementSeconds ();

// دالة لزيادة الوقت ثانية واحدة.

// شرط تالي: تتم زيادة الوقت ثانية واحدة.

// اذا كان الوقت قبل الزيادة 23:59:59،

```

// تتم اعادة ضبط الوقت الى 00:00:00
Void incrementMinutes ( );
// دالة لزيادة الوقت دقيقة واحدة.
// شرط تالي: تتم زيادة الوقت دقيقة واحدة.
// اذا كان الوقت قبل الزيادة 23:59:53،
// تتم اعادة ضبط الوقت الى 00:00:53
Void incrementHours ( );
// دالة لزيادة الوقت ساعة واحدة.
// شرط تالي: تتم زيادة الوقت ساعة واحدة.
// اذا كان الوقت قبل الزيادة 23:45:53،
// تتم اعادة ضبط الوقت الى 00:45:53
bool equalTime (const clockType& otherClock) const;
// دالة لمقارنة الزمنين.
// شرط تالي: تنتج صحيح اذا كان هذا الوقت متساوي
// مع otherClock وبخلاف ذلك
// تنتج خطأ.
clockType (int hours, int minutes, int seconds);
// مقوم ذات عوامل.
// يتم ضبط الوقت وفقاً للعوامل.
// شرط تالي: hr = ساعات، min = دقائق، و sec = ثواني
// تحقق المقوم مما اذا كانت قيم
// الساعات والدقائق والثواني صحيحة ام لا. اذا
// كانت هناك قيمة غير صحيحة، يتم تحديد القيمة الافتراضية صفر.
clockType ( );
// مقوم افتراضي.
// يتم ضبط الوقت على القيم الافتراضية.
// شرط تالي: hr = 0، min = 0، و sec = 0

```

خاصة:

Int hr // تخزين الساعات.

Int min // تخزين الدقائق.

Int sec // تخزين الثواني.

};

هذا التعريف للفئة clockType يتضمن مقومين: مقوم منهما له ثلاثة عوامل والآخر لا يوجد به عوامل. لنقم بكتابة تعريفات تلك المقومات.

```
clockType::clockType(int hours, int minutes, int seconds)
{
    if(0 <= hours && hours < 24)
        hr = hours;
    else
        hr = 0;

    if(0 <= minutes && minutes < 60)
        min = minutes;
    else
        min = 0;

    if(0 <= seconds && seconds < 60)
        sec = seconds;
    else
        sec = 0;
}

clockType::clockType() //default constructor
{
    hr = 0;
    min = 0;
    sec = 0;
}
```

من تعريفات تلك المقومات ينتج أن المقوم الافتراضي يضبط عناصر البيانات الثلاثة وهي hr, min, & sec (الساعات والدقائق والساعات) عند صفر. كما يقوم المقوم ذو العوامل بضبط عناصر البيانات عند أي قيم مهما كانت محددة للعوامل الرسمية. فضلاً عن هذا يمكنك كتابة تعريفات تلك المقومات عن طريق استدعاء دالة setTime كما يلي:

```
clockType::clockType(int hours, int minutes, int seconds)
{
    setTime(hours, minutes, seconds);
}

clockType::clockType()
{
    setTime(0, 0, 0);
}
```

هذه التعريفات للمقومات تجعل التصحيح أكثر سهولة لأنه يجب التحقق من كود واحد فقط للدالة setTime.

استحضار المقوم:

تذكر أنه عند اعلان موضوع الفئة، يتم تنفيذ المقوم بشكل آلي. قد تحتوي الفئة على أكثر من مقوم واحد بما فيها المقوم الافتراضي ولهذا يناقش القسم التالي كيفية استحضار مقوم محدد.

استحضار المقوم الافتراضي: افترض أن هناك فئة تحتوي على المقوم الافتراضي. يكون التركيب لاستحضار المقوم الافتراضي هو:

| | |
|-----------|-------------------|
| ClassName | classVariableName |
|-----------|-------------------|

على سبيل المثال، البيان التالي:

```
clockType yourClock;
```

يعلن أن yourClock موضوع من النوع clockType في هذه الحالة يتم تطبيق المقوم الافتراضي ويتم تهيئة عناصر بيانات yourClock الى صفر.

إذا قمت باعلان موضوع ما وأردت أن يتم تنفيذ المقوم الافتراضي فان القوسين الذين يأتيان بعد اسم الموضوع ليسوا مطلوبين في بيان اعلان الموضوع. في الحقيقة إذا قمت بادخال القوسين بدون قصد يقوم المجمع بتوليد رسالة خطأ تركيب. على سبيل المثال يعتبر البيان التالي غير صحيح:

```
clockType yourClock ( ); // illegal
```

استحضار مقوم ذو عوامل: افترض أن هناك فئة تحتوي على مقومات ذات عوامل. يكون التركيب لاستحضار المقوم ذي العوامل هو:

| | |
|-----------|--|
| ClassName | classVariableName (argument1, argument2, ...); |
|-----------|--|

 حيث

1 argument، argument2، وهكذا اما عبارة عن متغير أو تعبير.

لاحظ ما يلي:

- يجب أن يتوافق عدد البراهين ونوعها مع العوامل الرسمية (بالترتيب المعطى) لأحد المقومات.
- إذا لم يكن نوع البراهين متوافق مع العوامل الرسمية لأي مقوم (بالترتيب المعطى) فان C++ تستخدم تحويل النوع وتبحث عن أفضل توافق. على سبيل المثال قد يتم تحويل قيمة عدد صحيح الى قيمة قائمة النقطة بها جزء عشري صفري وأي غموض يتسبب في خطأ في تجميع الوقت.

انظر الى البيان التالي:

```
clockType myClock (5, 12, 40);
```

هذا البيان يعلن الموضوع myClock للنوع clockType. هنا نقوم بتمرير ثلاثة قيم من النوع int الذي يناسب نوع العوامل الرسمية للمقوم ذو العوامل. لهذا يجري المقوم ذو العوامل من الفئة clockType ويتم ضبط عناصر البيانات الثلاثة – الساعات والدقائق والثواني – للموضوع myClock عند 5، و12، و40 على التوالي.

المقومات والعوامل الافتراضية:

يمكن أن يحتوي المقوم كذلك على عوامل افتراضية وفي هذه الحالة تكون قواعد اعلان العوامل الرسمية هي نفسها قواعد اعلان القواعد الرسمية في الدالة. فضلاً عن هذا، يتم تمرير العوامل الحقيقية لمقوم ذو عوامل افتراضية وفقاً لقواعد الدالات ذات العوامل الافتراضية. باستخدام قواعد تعريف العوامل الافتراضية يمكنك في تعريف الفئة clockType أن تستبدل كلا المقومين باستخدام البيان التالي. (لاحظ أن في نموذج الدالة يكون اسم العامل الرسمي اختياري).

```
clockType clockType(int = 0, int = 0, int = 0); //Line 1
```

في ملف التطبيق، يكون تعريف هذا المقوم هو نفسه تعريف المقوم ذو العوامل. اذا استبدلت مقومات الفئة clockType بالمقوم في الصف رقم 1 (المقوم ذو العوامل الافتراضية) يمكنك هند هذا اعلان موضوعات clockType ذات 0، أو 1، أو 2، أو 3 براهين كما يلي:

```
clockType clock1; //Line 2
clockType clock2(5); //Line 3
clockType clock3(12, 30); //Line 4
clockType clock4(7, 34, 18); //Line 5
```

يتم تهيئة عناصر بيانات clock1 الى صفر وتهيئة عنصر البيانات hr من clock2 الى 5، وتهيئة عناصر البيانات min وsec من clock2 الى صفر. يتم تهيئة عنصر البيانات hr من clock3 الى 12 وتهيئة عنصر البيانات min من clock3 الى 30، وتهيئة عنصر البيانات sec من clock3 الى صفر. يتم تهيئة عنصر البيانات hr من clock4 الى 7 وتهيئة عنصر البيانات min من clock4 الى 34، وتهيئة عنصر البيانات sec من clock4 الى 18.

باستخدام هذه التقاليد يمكننا قول أن المقوم الذي ليس به عوامل أو جميع عوامله افتراضية يسمى **مقوم افتراضي**.

المدمرات:

المدمرات مثلها مثل المقومات عبارة كذلك عن دالات. فضلاً عن هذا لا تمتلك المدمرات مثل المقومات نوع فهي ليست دالة منتجة لقيمة ما ولا دالة باطلة. بالرغم من هذا يمكن للفئة أن يكون بها

مدمر واحد ويمكن للمدمر ألا يكون به أية عوامل. اسم المدمر هو الرمز (~) يليه اسم الفئة (الرمز (~) يسمى تلمذة). على سبيل المثال يكون اسم مدمر الفئة clockType هو: ~ clockType ();

يجري المدمر تلقائياً عندما يخرج موضوع الفئة من المجال. لهذا لا يمكن استدعاء المدمرات بوضوح مثلها مثل المقومات.

البنيات:

البنيات هي نوع خاص من الفئات فرضياً جميع عناصر الفئة تكون عناصر خاصة بينما فرضياً جميع عناصر البنية تكون عناصر عامة. في C++ تقوم بتعريف البنيات باستخدام بنية الكلمة المحفوظة. اذا كانت جميع عناصر الفئة عامة يفضل المبرمجون استخدام بنية لتجميع العناصر كما سوف تفعل في هذا الكتاب. يتم تعريف البنية مثل الفئة.

تجريد البيانات، والفئات، وأنواع البيانات المجردة:

يمكن للقارئ تخطي هذا القسم اذا كان القارئ على علم بهذه الموضوعات.

هل تسائلت أبداً عن كيفية عمل محرك سيارتك؟ غالبية الأفراد يريدون أن يعرفوا كيفية تشغيل وقيادة السيارة فقط ولا يكونوا مهتمين بمدى تعقيد عمل المحرك. بفصل تفاصيل تصميم محرك السيارة عن الاستخدام يساعد المنتج السائق على التركيز على كيفية قيادة السيارة. حياتنا اليومية بها أمثلة أخرى مماثلة. في الغالب يهتم الناس فقط بكيفية استخدام عناصر معينة أكثر من معرفة كيفية عملها.

ان فصل تفاصيل التطبيق (أي كيفية عمل محرك السيارة) عن استخدام العنصر يسمى التجريد. بمعنى آخر يقوم التجريد بالتركيز على ما يفعله المحرك وليس على كيفية عمله. لهذا فان التجريد هو عملية فصل الخصائص المنطقية عن تفاصيل التطبيق. قيادة السيارة خاصية منطقية وبناء المحرك يشكل تفاصيل التطبيق. انك تمتلك رؤية مجردة عما يقوم به المحرك ولكنك غير مهتم بالتطبيق الفعلي للمحرك.

يمكن تطبيق التجريد على البيانات أيضاً. الأقسام السابقة من هذا الفصل قامت بتعريف نوع البيانات clockType الذي لديه ثلاثة عناصر بيانات والعمليات الأساسية التالية:

1. ضبط الوقت.
2. استرجاع الوقت.
3. طباعة الوقت.
4. زيادة الوقت ثانية واحدة.
5. زيادة الوقت دقيقة واحدة.

6. زيادة الوقت ساعة واحدة.

7. مقارنة الوقتين لرؤية ما اذا كانا متساويين.

لقد تم تأجيل التطبيق الفعلي للعمليات المؤداة على clockType. يتم تعريف تجريد البيانات بأنه عملية فصل الخصائص المنطقية للبيانات عن تطبيقها. ان تعريف clockType وعملياتها الأساسية هي الخصائص المنطقية وتخزين clockType في الحاسب الآلي، والحلول الحسابية لأداء تلك العمليات هي تفاصيل تطبق clockType.

نوع البيانات المجرد: نوع من البيانات يحدد الخصائص المنطقية بدون تفاصيل التطبيق.

يهتم نوع البيانات المجرد بتفاصيل المواصفات (ماذا) وليس تفاصيل التطبيق (كيف).

ان نوع البيانات المجرد مثله مثل أي نوع آخر من البيانات به ثلاثة أشياء مرتبطة به: اسم نوع البيانات المجرد الذي يسمى **اسم النوع**، ومجموعة القيم التي تنتمي الى نوع البيانات المجرد والتي تسمى **المجال**، ومجموعة **العمليات** المؤداة على البيانات. بعد هذه التقاليد يمكننا تعريف نوع البيانات المجرد clockType كما يلي:

اسم النوع

clockType

المجال

كل قيمة من clockType عبارة عن وقت من اليوم في صيغة

ساعات ودقائق وثواني.

العمليات:

ضبط الوقت.

استرجاع الوقت.

طباعة الوقت.

زيادة الوقت ثانية واحدة.

زيادة الوقت دقيقة واحدة.

زيادة الوقت ساعة واحدة.

اختبار الوقتين لرؤية ما اذا كانا متساويين.

مثال 1-5:

القائمة تعرف بأنها مجموعة من القيم من نفس النوع. جميع القيم الموجودة في القائمة من نفس النوع ولهذا تكون الطريقة المناسبة لتمثيل ومعالجة القائمة هي استخدام المصفوف. يمكنك تعريف القائمة كنوع بيانات مجرد كما يلي:

اسم النوع

listType

المجال

كل عنصر من نوع listType عبارة عن مجموعة من 1000 عدد مثلاً .

العمليات

الفحص لرؤية ما اذا كانت القائمة خالية.

الفحص لرؤية ما اذا كانت القائمة ممتلئة.

البحث عن عنصر محدد في القائمة.

حذف عنصر ما من القائمة.

ادخال عنصر ما في القائمة.

تنظيم القائمة.

تدمير القائمة.

طباعة القائمة.

السؤال التالي الواضح هو كيفية تطبيق نوع البيانات المجرد في برنامج ما. من أجل تطبيق نوع بيانات مجرد يجب أن تقوم بتمثيل البيانات وكتابة الحلول الحسابية لأداء العمليات اللازمة.

لقد قام القسم السابق باستخدام الفئات لتجميع البيانات والدالات معاً. فضلاً عن ذلك تكون تعريفنا للفئة من مواصفات العمليات فقط وتمت كتابة دالات تطبيق العمليات بشكل منفصل. لهذا ترى أن الفئات عبارة عن طريقة مناسبة لتطبيق نوع البيانات المجرد. في الحقيقة لقد تم تصميم فئات C++ خصيصاً للتعامل مع نوع البيانات المجرد.

الفئة التالية تقوم بتعريف القائمة كنوع بيانات مجرد (نظر أيضاً شكل 1-11). لكي تكون محدد افترض أن القائمة عبارة عن مجموعة من عناصر النوع int.

```
Class intListType
```

```
{
```

عامة:

```
Bool isEmpty ( );
```

```
// دالة لتحديد ما اذا كانت القائمة خالية.
```

```
// شرط مسبق: يجب أن تتواجد القائمة.
```

```
شرط تالي: تنتج صحيح اذا كانت القائمة خالية
```

```
وخطأ بخلاف هذا.
```

```
Bool isFull ( );
```

```
// دالة لتحديد ما اذا كانت القائمة ممتلئة.
```

```
// شرط مسبق: يجب أن تتواجد القائمة.
```

```
شرط تالي: تنتج صحيح اذا كانت القائمة ممتلئة
```

وخطأ بخلاف هذا.

Int search (int searchItem);

// دالة لتحديد ما اذا كان عنصر البحث موجود

// في القائمة.

// شرط تالي: اذا كان عنصر البحث في القائمة

// تقدم مؤشر الى

// مكانه في القائمة،

// بخلاف هذا تقدم -1.

Void insert (int newItem);

// دالة لادخال عنصر جديد في القائمة.

// شرط مسبق: يجب أن تتواجد القائمة وألا تكون ممتلئة.

// شرط تالي: يتم ادخال العنصر الجديد في القائمة و

يزيد الطول بمقدار عنصر واحد.

Void remove (int removeItem);

// دالة لحذف عنصر من القائمة.

// شرط مسبق: يجب أن تتواجد القائمة وألا تكون خالية.

// شرط تالي: اذا وجد العنصر يتم حذفه من القائمة و

// يقل الطول بمقدار عنصر واحد، وبخلاف هذا،

// يتم طبع رسالة مناسبة.

Void destroy ();

// دالة لحذف جميع عناصر القائمة.

// شرط مسبق: يجب أن تتواجد القائمة.

// شرط تالي: يتم ضبط الطول عند صفر.

Void printLisy ();

// دالة لايخراج عناصر القائمة.

// شرط مسبق: يجب أن تتواجد القائمة.

// شرط تالي: يتم طبع عناصر القائمة

// على جهاز الاخراج القياسي.

inListType ();

// مقوم افتراضي

// شرط تالي: الطول = صفر.

خاصة:

Int list [1000];

Int length;

};

| inListType |
|---|
| -list: int [] -length: int |
| + isEmpty () : bool + isFull () : bool + search (int) : int + insert (int) : void + remove (int) : void + destroy () : void + printList () : void + inListType () |

شكل 1-11: الرسم البياني للغة التشكيل الموحدة للفئة inListType.

اخفاء المعلومات:

قام القسم السابق بتعريف الفئة clockType لتطبيق الوقت في برنامج ما. لاحظ أنه تم اعلان أن عناصر البيانات خاصة. فضلاً عن هذا لكي تستخدم الفئة clockType في برنامج يجب على المستخدم أن يهتم فقط بما تفعله كل دالة وليس كيف تفعله. هذا يعني أننا يجب أن نقوم باخفاء تفاصيل تطبيق كل دالة. في C++ يتم تحقيق اخفاء المعلومات عن طريق اعلان أن عناصر البيانات خاصة وكذلك اخفاء تفاصيل تطبيق دالات العنصر.

ان تعريف الفئة clockType يحتوي على عناصر البيانات ونماذج الدالة ووصف لما تفعله كل دالة فقط. من أجل تنفيذ اخفاء المعلومات يجب أن نفصل تعريف الفئة عن تعريفات دالات العنصر ونمد المستخدم بتعريف الفئة فقط. لكي يكون المستخدم قادراً على استخدام الدالات يتم تزويده بالكود المجمع من الدالات.

هذا القسم يناقش كيفية تنفيذ اخفاء المعلومات وسوف نستخدم الفئة clockType لأغراض توضيحية. لتطبيق الفئة clockType في برنامج ما يجب على المستخدم أن يعلن موضوعات من النوع clockType وأن يعرف ما هي العمليات المسموحة وما تفعله العمليات. لذلك يجب أن يصل المستخدم الى تفاصيل المواصفات. المستخدم لا يكون مهتم بتفاصيل التطبيق ولهذا يجب أن تضع تلك التفاصيل في ملف منفصل يسمى **ملف التطبيق**. وبما أن تفاصيل المواصفات قد تكون طويلة للغاية

يجب أن تقوم باعفاء المستخدم من وجوب وضعها في البرنامج بشكل مباشر. بالرغم من هذا يجب أن يكون المستخدم قادراً على النظر الى تفاصيل المواصفات حتى يمكنه استدعاء الدالات بشكل صحيح وهكذا. لهذا يجب أن تضع تفاصيل المواصفات في ملف منفصل. الملف الذي يحتوي على تفاصيل المواصفات يطلق عليه **الملف الرئيسي (أو الملف البيني)**.

يتضمن ملف التطبيق تعريفات الدالات لتطبيق عمليات موضوع ما. هذا الملف يتضمن من ضمن أشياء أخرى (مثل تعليمات ما قبل المعالجة) بيانات C++. برنامج C++ يمكنه الاحتواء على دالة واحدة فقط ولهذا لا يحتوي ملف التطبيق على الدالة الأساسية main. ان برنامج المستخدم فقط يحتوي على الدالة الأساسية main. لأن ملف التطبيق لا يشتمل على الدالة Main، لا يمكن انتاج الكود التنفيذي من هذا الملف. في الحقيقة أنك تقدم ما يسمى كود الموضوع من ملف التطبيق. بعد هذا يقوم المستخدم بربط كود الموضوع المنتج بواسطة ملف التطبيق بكود موضوع البرنامج الذي يستخدم الفئة لعمل الكود التنفيذي النهائي.

أخيراً يكون امتداد الملف الرئيسي h بينما يكون امتداد ملف التطبيق cpp. افترض أن تفاصيل مواصفات الفئة clockType موجودة في ملف يسمى clockType فان الاسم الكامل للملف الرئيسي يجب أن يكون clockType.h. اذا كانت تفاصيل تطبيق الفئة clockType موجودة في ملف يسمى مثلاً clockTypeImp فان اسم هذا الملف يجب أن يكون clockTypeImp.cpp.

الملف clockTypeImp.cpp يحتوي فقط على تعريفات الدالات وليس تعريف الفئة. لهذا ومن أجل حل مسألة محدد غير مصرح به (مثل أسماء الدالة وأسماء المتغيرات) تقوم بتضمين الملف الرئيسي clockType.h في الملف clockTypeImp.cpp بمساعدة بيان التضمين.

بيان التضمين التالي مطلوب من أي برنامج يستخدم الفئة clockType وكذلك من ملف التطبيق الذي يقوم بتعريف العمليات الخاصة بالفئة clockType:

```
# include "clockType.h"
```

لاحظ أن الملف الرئيسي clockType.h محصور في علامتين تنصيص وليس في أقواس عادية. الملف الرئيسي clockType.h يطلق عليه **الملف الرئيسي المعرف بواسطة المستخدم**. يتم وضع جميع الملفات الرئيسية المعروفة بواسطة المستخدم بين علامتين تنصيص بينما يتم وضع الملفات الرئيسية المقدمة بواسطة النظام (مثل iostream) بين أقواس ذات زاوية.

فيما يلي ملفات المواصفات والتطبيق الخاصة بالفئة clockType:

```
clockType.h // ملف مواصفات الفئة clockType
```

```
الفئة clockType
```

```
}
```

عامة:

Void setTime (int hours, int minutes, int seconds);
// دالة لضبط الوقت.

// يتم ضبط الوقت وفقاً للعوامل.

// شرط مسبق: hr = ساعات، min = دقائق، و sec = ثواني.

// تتحقق الدالة مما اذا كانت قيم

// الساعات والدقائق والثواني صحيحة ام لا. اذا

// كانت هناك قيمة غير صحيحة، يتم تحديد القيمة الافتراضية صفر.

Void getTime (int& hours, int& minutes, int& seconds);
// دالة لتقديم الوقت.

// شرط مسبق: hr = ساعات، min = دقائق، و sec = ثواني.

Void printTime () const;

// دالة لطبع الوقت.

// شرط مسبق: تتم طباعة الوقت في صيغ

// .hh:mm:ss

Void incrementSeconds ();

// دالة لزيادة الوقت ثانية واحدة.

// شرط تالي: تتم زيادة الوقت ثانية واحدة.

// اذا كان الوقت قبل الزيادة 23:59:59،

// تتم اعادة ضبط الوقت الى 00:00:00

Void incrementMinutes ();

// دالة لزيادة الوقت دقيقة واحدة.

// شرط تالي: تتم زيادة الوقت دقيقة واحدة.

// اذا كان الوقت قبل الزيادة 23:59:53،

// تتم اعادة ضبط الوقت الى 00:00:53

Void incrementHours ();

// دالة لزيادة الوقت ساعة واحدة.

// شرط تالي: تتم زيادة الوقت ساعة واحدة.

// اذا كان الوقت قبل الزيادة 23:45:53،
// تتم اعادة ضبط الوقت الى 00:45:53

```
bool equalTime (const clockType& otherClock) const;  
// دالة لمقارنة الزمنين.  
// شرط تالي: تنتج صحيح اذا كان هذا الوقت متساوي  
// مع otherClock وبخلاف ذلك  
// تنتج خطأ.
```

```
clockType (int hours, int minutes, int seconds);  
// مقوم ذات عوامل.  
// يتم ضبط الوقت وفقاً للعوامل.  
// شرط تالي: hr = ساعات، min = دقائق، و sec = ثواني  
// تحقق المقوم مما اذا كانت قيم  
// الساعات والدقائق والثواني صحيحة ام لا. اذا  
// كانت هناك قيمة غير صحيحة، يتم تحديد القيمة الافتراضية صفر.
```

```
clockType ();  
// مقوم افتراضي.  
// يتم ضبط الوقت على القيم الافتراضية.  
// شرط تالي: hr = 0، min = 0، و sec = 0
```

خاصة:

```
Int hr // تخزين الساعات.  
Int min // تخزين الدقائق.  
Int sec // تخزين الثواني.  
};
```

```
// clockTypeImp.cpp, the implementation file  
# include <iostream>  
# include "clockType.h"  
Using namespace std;
```

// تعريفات دالات العنصر clockType تذهب هنا.

بعد هذا نصف ملف المستخدم الذي يحتوي على البرنامج الذي يستخدم الفئة clockType.

// testClock.cpp

// برنامج المستخدم الذي يستخدم الفئة clockType.

include <iostream>

include "clockType.h"

Using namespace std;

// ضع تعريفات الدالة الأساسية main والدالات الأخرى المعرفة بواسطة المستخدم هنا.

المثال 6-1 يوضح كيفية تصميم وتطبيق الفئات. الفئة personType المصممة في المثال 6-1 مفيدة للغاية وسوف نستخدمها في فصول تالية.

مثال 6-1:

مميزات الشخص الأكثر شيوعاً هي اسم الشخص الأول واسمه الأخير. العمليات التقليدية المؤداة على اسم الشخص هي ضبط الاسم وطباعته. البيانات التالية تقوم بتعريف الفئة personType بتلك الخصائص (انظر شكل 12-1 أيضاً).

الفئة personType

}

Void print () const;

// دالة لاجراج الاسم الأول والاسم الأخير.

// في صيغة الاسم الأول الاسم الأخير.

Void setName (string first, string last);

//دالة لتحديد الاسم الأول والاسم الأخير وفقاً

// الى العوامل.

// شرط تالي: الاسم الأول = first والاسم الأخير = last.

Void getName (string first, string& last);

//دالة لاسترجاع الاسم الأول والاسم الأخير عبر

// العوامل.

// شرط تالي: first = الاسم الأول و last = الاسم الأخير.

personType (string first = " ", string last = " ");

// مقوم

//يحدد الاسم الأول والاسم الأخير وفقاً الى

// العوامل.

// القيم الافتراضية للعوامل خيوط خالية.

// شرط تالي: الاسم الأول = first والاسم الأخير = last.

خاصة:

String firstName; // يقوم بتخزين الاسم الأول.

String lastName; // يقوم بتخزين الاسم الأخير.

{

| personType |
|--|
| -firstName: string -lastName: string |
| +print (): void +setName (string, string): void +getName (string, string): void +personType (string = " ". string = " ") |

شكل 1-12، الرسم البياني للغة التشكيل الموحدة للفئة personType
اننا الآن نعطي تعريفات عناصر دالة الفئة personType.

```
void personType::print() const
{
    cout<<firstName<<" "<<lastName;
}

void personType::setName(string first, string last)
{
    firstName = first;
    lastName = last;
}

void personType::getName(string& first, string& last)
{
    first = firstName;
    last = lastName;
}

//constructor
personType::personType(string first, string last)
{
    firstName = first;
    lastName = last;
}
```

مثال على البرمجة: ماحيه الحوى:

المكان الشائع لشراء الحلوى منه هو ماكينة الحلوى ولقد تم شراء ماكينة حلوى جديدة من أجل
الجمنازيوم ولكنها لم تكن تعمل جيداً . تقوم الماكينة ببيع الحلويات، وشرائح البطاطس المقلية، ولبان،
وكعك. تم توجيه طلب اليك بكتابة برنامج من أجل ماكينة الحلوى هذه حتى يمكن تشغيلها.
يجب أن يقوم البرنامج بما يلي:

1. عرض المنتجات المختلفة التي يتم بيعها بواسطة ماكينة الحلوى للزبون.

2. ترك الزبون يقوم بالاختيار.

3. اظهار تكلفة العنصر المختار للزبون.

4. قبول النقود من الزبون.

5. اخراج العنصر.

المدخلات: اختيار العنصر وتكلفة العنصر.

المخرجات: العنصر الذي يتم اختياره.

تحليل المشكلة وتصميم الحل الحسابي:

ماكينة الحلوى بها مكونان أساسيان وهما سجل نقدية مدمج، وعدة أوعية لامساك واخراج المنتجات.

سجل النقدية:

دعونا نقوم أولاً بمناقشة خصائص سجل النقدية. السجل به بعض النقدية فيه ويقبل المبالغ من الزبائن. اذا كان المبلغ الذي يتم ادخاله أكثر من تكلفة العنصر فان الماكينة – اذا أمكنها – تعيد الباقي. من أجل البساطة افترض أن المستخدم يقوم بادخال المبلغ المضبوط للمنتج. سوف يكون سجل النقدية قادراً كذلك على اظهار مبلغ النقود الموجود في السجل في أي وقت لصاحب ماكينة الحلوى. الفئة التالية تقوم بتعريف خصائص سجل النقدية (انظر كذلك شكل 1-13).

فئة cashRegister

}

عامة:

Int currentBalance ();

// دالة لاطهار المبلغ الحالي في سجل النقدية.

// شرط تالي: يتم انتاج قيمة النقدية المتناولة.

Void acceptAmount (int amountIn);

// دالة لاستقبال المبلغ الذي يقوم بايداعه

// الزبون وتحديث المبلغ في السجل.

// الشرط التالي: النقدية المتناولة = النقدية المتناولة + المبلغ الداخل.

cashRegister (int cashIn = 500);

// مقوم لضبط النقدية في السجل عند

// مبلغ محدد.

// شرط تالي: النقدية المتناولة = النقدية الداخلة.

// اذا لم يتم تحديد قيمة عند اعلان الموضوع

// فان القيمة الافتراضية التي يتم تحديدها للنقدية المتناولة

// هي 500.

خاصة:

Int cashOnHand; // متغير لتخزين النقدية

// في السجل.

{

| |
|---------------------------|
| cashRegister |
| -cashOnHand: int |
| +currentBalance () : int |

```
+ acceptAmount (int) : void
+cashRegister (int = 500)
```

شكل 1-13: السرم البياني للغة التشكيل الموحدة للفئة cashRegister
بعد هذا نعطي تعريفات الدالات لتنفيذ عمليات الفئة cashRegister. تعريفات تلك الدالات شديدة البساطة والسهولة في اتباعها.

تقوم الدالة currentBalance باظهار المبلغ الحالي في سجل النقدية وتقوم بتقديم قيمة عنصر البيانات الخاص cashOnHand. لذلك يكون تعريفها هو:

```
Int cashRegister : : currentBalance ( )
}
```

```
Return cashOnHand;
{
```

تقوم الدالة acceptAmount بقبول المبلغ الذي يقوم الزبون بادخاله وتقوم بتحديث النقدية في السجل عن طريق اضافة المبلغ الذي يدخله الزبون الى المبلغ السابق الموجود في سجل النقدية. جوهرياً تعريف هذه الدالة يكون:

```
Void cashRegister : : acceptAmount (int amountIn)
}
```

```
casgOnHand += amountIn;
{
```

في تعريف الفئة cashRegister، يتم اعلان المقوم مع قيمة افتراضية ولهذا اذا لم يقم المستخدم بتحديد قيمة عند اعلان الموضوع فانه يتم استخدام القيمة الافتراضية لتهيئة عنصر البيانات cashOnHand.

لقد قمنا بتحديد القيمة الافتراضية لعامل المقوم في تعريف الفئة ولهذا فاننا في تعريف المقوم لا نقوم بتحديد القيمة الافتراضية في العنوان. تعريف المقوم يكون كما يلي:

```
cashRegister : : cashRegister (int cashIn)
}
```

```
If (cashIn >= 0)
cashOnHand = cashIn;
else
cashOnHand = 500;
{
```

لاحظ أن تعريف المقوم يتحقق من القيم الصحيحة للعامل cashIn. اذا كانت قيمة cashIn أقل من صفر تكون القيمة المحددة لعنصر البيانات cashOnHand هي 500.

الوعاء:

يقوم الوعاء باخراج العنصر الذي يتم اختياره اذا لم يكن فارغاً . كما يجب أن يظهر عدد العناصر الموجودة في الوعاء وتكلفة العنصر الواحد. تقوم الفئة التالية بتعريف خصائص الوعاء. لنقم بتسميتها الفئة dispenserType. (انظر شكل 14-1).

الفئة dispenserType

}

عامة:

Int count ();

// دالة لاطهار عدد العناصر في الماكينة.

// شرط تالي: يتم تقديم قيمة عنصر البيانات numberOfItems (عدد العناصر).

Int productCost ();

// دالة لاطهار تكلفة العنصر.

// يتم تقديم قيمة عنصر البيانات cost (التكلفة).

// شرط تالي: يتم تقديم قيمة التكلفة.

Void makeSale ();

// دالة لتقليل عدد العناصر بمقدار 1.

// شرط تالي: numberOfItems— (عدد العناصر--).

dispenserType (int setNoOfItems = 50, int setCost = 50);

// مقوم لتحديد تكلفة وعدد العناصر في الوعاء محددين بواسطة المستخدم.

NumberOfItems = setNoOfItems; // شرط تالي:

cost = setCost //

// اذا لم يتم تحديد قيمة للعامل فانه يتم تحديد قيمته الافتراضية

// مع عنصر البيانات المتوافق.

خاصة:

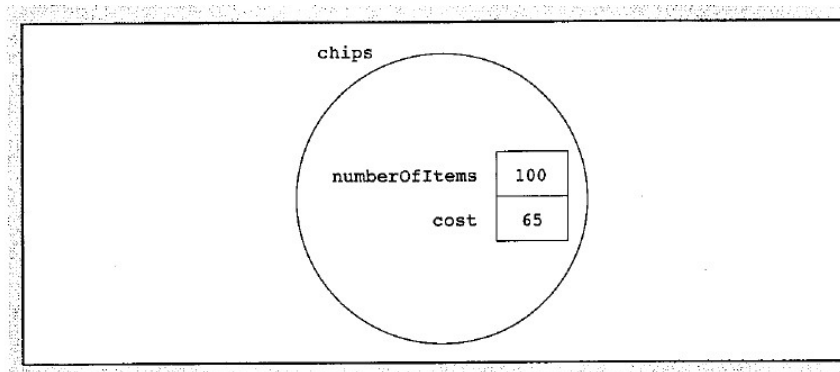
```
Int numberOfItems;
// متغير لتخزين عدد العناصر في الوعاء.
Int cost;
// متغير لتخزين تكلفة العنصر الواحد.
{;
```

| dispenserType |
|-------------------------------------|
| -numberOfItems : int |
| -cost : int |
| +count () : int |
| +productCost () : int |
| +makeSale () : void |
| +dispenserType (int = 50, int = 50) |

شكل 1-14: الرسم البياني للغة التشكيل الموحدة للفئة dispenserType.
لأن ماكينة الحلوى تباع أربعة أنواع من العناصر فأنت تقوم بإعلان أربعة موضوعات من النوع dispenserType على سبيل المثال يقوم البيان التالي:

```
dispenserType chips (100, 65);
```

بإعلان أن رقائيق البطاطس المقليّة موضوع من النوع dispenserType ويضبط عدد أكياس البطاطس في الوعاء عند العدد 100 وتكلفة كل كيس بطاطس عند 65 سنت. (انظر شكل 1-15).



شكل 1-15: موضوع رقائيق البطاطس المقليّة.

بعد هذا نناقش تعريفات الدالات لتطبيق عمليات الفئة dispenserType.
تقوم الدالة count بتقديم عدد العناصر من منتج معين. يتم تخزين عدد العناصر الموجودة حالياً في الوعاء في عنصر البيانات الخاص numberOfItems، ولهذا فإنها تقدم قيمة عنصر البيانات الخاص numberOfItems.

تعريف هذه الدالة يكون:

```
Int dispenserType : : count ( )
```

```

    }
    Return numberOfItems;
}

الدالة productCost تقدم تكلفة المنتج ولأن تكلفة المنتج مخزنة في عنصر البيانات الخاص cost
(تكلفة) فانها تقدم قيمة عنصر البيانات الخاص cost. تعريف هذه الدالة يكون:
Int dispenserType : : productCost ( )
}

Return cost;
{

عندما يتم بيع منتج ما يتم انقاص عدد العناصر الموجودة في هذا الوعاء بمقدار 1 ولهذا تقوم الدالة
makeSale بتقليل عدد العناصر في الوعاء بمقدار 1. هذا يعني أنها تقلل قيمة عنصر البيانات
الخاص numberOfItems بمقدار 1 ويكون تعريف هذه الدالة هو:
Void dispenserType : : makeSale ( )
}

numberOfItems--;
{

يتحقق تعريف المقوم من القيم الصحيحة من العوامل واذا كانت تلك القيم أقل من صفر يتم تحديد القيم
الافتراضية لعناصر البيانات ويكون تعريف المقوم كما يلي:
// constructor
dispenserTyoe : : dispenserType (int setNoOfItems, int setCost)
{
    If (setNoOfItems >= 0)
        numberOfItems = setNoOfItems;
    else
        numberOfItems = 50;
    If (setCost >= 0)
        Cost = setCost;
    else
        cost = 50;
}

```

البرنامج الأساسي:

عندما يجري البرنامج يجب أن يفعل ما يلي:

1. اظهار المنتجات المختلفة التي يتم بيعها بواسطة ماكينة الحلوى.
2. اظهار كيفية اختيار منتج معين.
3. اظهار كيفية ايقاف البرنامج.

فضلاً عن هذا، يجب عرض تلك التعليمات بعد معالجة كل اختيار (فيما عدا الخروج من البرنامج) ولهذا لا يحتاج المستخدم الى تذكر ما يفعله اذا اراد شراء عنصرين أو أكثر. بمجرد أن يقوم المستخدم بالاختيار المناسب يجب أن تعمل ماكينة الحلوى بناءً على ذلك. اذا اختار المستخدم أن يشتري منتج ما واذا كان هذا المنتج متوفر يجب أن توضح ماكينة الحلوى تكلفة المنتج وتطلب من المستخدم أن يضع النقود. اذا كانت النقود الموضوعة هي تكلفة العنصر يجب أن تبيع ماكينة الحلوى العنصر وتعرض رسالة مناسبة.

تترجم تلك المناقشة الى الحل الحسابي التالي:

1. اظهار الاختيار للزبون.

2. الحصول على الاختيار.

3. اذا كان الاختيار موجود والوعاء المتوافق مع الاختيار غير فارغ يتم بيع المنتج.

اننا نقوم بتقسيم هذا البرنامج الى ثلاث دالات: عرض الاختيار، وبيع المنتج، والدالة الأساسية (showSelection, sellProduct, & main).

دالة عرض الاختيار showSelection:

تقوم هذه الدالة بعرض المعلومات اللازمة لمساعدة المستخدم على اختيار وشراء المنتج. جوهرياً تحتوي هذه الدالة على بيانات المخرجات التالية. (نفترض أن ماكينة الحلوى تبيع أربعة أنواع من المنتجات).

*** أهلاً بكم في متجر شيلي للحلوى ***

لاختيار عنصر، ادخل

1 للحلوى

2 لرقائق البطاطس

3 للبان

4 للكعك

9 للخروج

تعريف الدالة showSelection يكون:

Void showSelection ()

{

Cout <<"*** Welcome to Shelly's Candy Shop *** <<end1;

cout <<"To select an item, enter"<<end1;

cout <<"1 for Candy"<<end1;

cout <<"2 for Chips"<<end1;

cout <<"3 for Gum"<<end1;

```

cout <<"4 for Cookies"<<endl;
cout <<"9 for exit"<<endl;
} // end showSelection

```

دالة بيع المنتج sellProduct

هذه الدالة تحاول بيع المنتج الذي يتم اختياره من جانب الزبون. لهذا يجب أن تصل الى الوعاء المحتوي على المنتج. الشئ الأول الذي تقوم به هذه الدالة هو التأكد مما اذا كان الوعاء المحتوي على المنتج فارغ أم لا. اذا كان الوعاء فارغاً تبلغ الدالة الزبون أن المنتج قد نفذ واذا كان الوعاء غير فارغ تخبر المستخدم بوضع المبلغ اللازم لشراء المنتج.

اذا لم يتم المستخدم بايداع المال الكافي لشراء المنتج تخبره دالة بيع المنتج sellProduct المبلغ الاضافي الذي يجب عليه ايداعه واذا فشل المستخدم في ايداع المال الكافي في محاولتين (انزر تمرين البرمجة رقم 9) لشراء المنتج تقوم الدالة بارجاع المال ببساطة. اذا كان المال الذي أودعه المستخدم كافياً تقبله الماكينة وتبيع المنتج. ان بيع المنتج يعني تقليل عدد العناصر الموجودة في الوعاء بمقدار 1 وتحديث المال الموجود في سجل النقدية عن طريق اضافة تكلفة المنتج. (لأن هذا البرنامج لا يعيد المال الزائد الذي يودعه المستخدم يتم تحديث سجل النقدية باضافة المال الذي يقوم المستخدم بادخاله). يتضح من هذه المناقشة أن دالة بيع المنتج sellProduct يجب أن تصل الى كل من الوعاء المحتوي على المنتج (لتقليل عدد العناصر في الوعاء بمقدار 1 وعرض تكلفة العنصر) وسجل النقدية (لتحديث النقدية). لهذا يوجد لهذه الدالة عاملين: عامل يقابل الوعاء والآخر يقابل سجل النقدية. بالاضافة الى هذا يجب الاشارة الى كل من العاملين.

الحل الحسابي لهذه الدالة هو:

أ. اذا كان الوعاء غير فارغ:

1. اظهار وحث الزبون على ادخال تكلفة العنصر.

2. الحصول على المبلغ الذي يدخله الزبون.

3. اذا كان المبلغ الذي يدخله الزبون أقل من تكلفة المنتج:

أ. اظهار وحث الزبون على ادخال المبلغ الاضافي.

ب. حساب المبلغ الكلي الذي أدخله الزبون.

4. اذا كان المبلغ الذي أدخله الزبون يساوي تكلفة المنتج:

أ. تحديث المبلغ في سجل النقدية.

ب. بيع المنتج أي تقليل عدد العناصر في الوعاء بمقدار 1.

ج. عرض رسالة مناسبة.

5. اذا كان المبلغ الذي أدخله المستخدم أقل من تكلفة العنصر اعادة المبلغ.

ب. اذا كان الوعاء فارغ ابلاغ المستخدم بأن المنتج قد نفذ.

هذا التعريف لدالة بيع المنتج sellProduct هو:

```
void sellProduct(dispenserType& product, cashRegister& pCounter)
{
    int amount; //variable to hold the amount entered
    int amount2; //variable to hold the extra amount needed

    if(product.count() > 0) //Step a
    {
        cout<<"Please deposit "<<product.productCost()
            <<" cents"<<endl; //Step a.i
        cin>>amount; //Step a.ii

        if(amount < product.productCost()) //Step a.iii
        {
            cout<<"Please deposit another "
                <<product.productCost() - amount //Step a.iii.1
                <<" cents"<<endl;
            cin>>amount2; //Step a.iii.2
            amount = amount + amount2; //Step a.iii.3
        }

        if(amount >= product.productCost()) //Step a.iv
        {
            pCounter.acceptAmount(amount); //Step a.iv.1
            product.makeSale(); //Step a.iv.2
            cout<<"Collect your item at the bottom"
                <<" and enjoy."<<endl; //Step a.iv.3
        }
        else
        {
            cout<<"The amount is not enough. "
                <<"Collect what you deposited."<<endl; //Step a.v
            cout<<"*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*"
                <<endl<<endl;
        }
        else
        {
            cout<<"Sorry this item is sold out."<<endl; //Step b
        }
    }
} //end sellProduct
```

الآن وبعد أن تم وصف الدالتين showSelection و sellProduct (عرض الاختيار وبيع المنتج)

يتم بعد هذا وصف الدالة الأساسية main.

الدالة main: الحل الحسابي للدالة main كما يلي:

1. عمل سجل النقدية – أي اعلان متغير ما من النوع cashRegister.
2. عمل أربعة أوعية – أي اعلان أربع موضوعات للنوع dispenserType وتهيئة تلك الموضوعات. على سبيل المثال يقوم البيان:
dispenserType candy (100, 50);
بعمل موضوع الوعاء وهو الحلوى لامسك الحلويات. عدد العناصر في الوعاء 100 و تكلفة العنصر الواحد 50 سنت.
3. اعلان متغيرات اضافية حسب الضرورة.

4. اظهار الاختيار واستدعاء دالة اظهار الاختيار shoeSelection.

5. الحصول على الاختيار.

6. عندما لا يتم هذا (الاختيار رقم 9 يخرج من البرنامج):

أ. بيع المنتج واستدعاء دالة بيع المنتج sellProduct.

ب. اظهار الاختيار واستدعاء دالة عرض الاختيار showSelection.

ت. الحصول على الاختيار.

تعريف الدالة main يكون كما يلي:

```
int main()
{
    cashRegister counter; //Step 1
    dispenserType candy(100,50); //Step 2
    dispenserType chips(100,65); //Step 2
    dispenserType gum(75,45); //Step 2
    dispenserType cookies(100,85); //Step 2

    int choice; //Step 3

    showSelection(); //Step 4
    cin>>choice; //Step 5

    while(choice != 9) //Step 6
    {
        switch(choice) //Step 6a
        {
            case 1: sellProduct(candy, counter);
                    break;
            case 2: sellProduct(chips, counter);
                    break;
            case 3: sellProduct(gum, counter);
                    break;
            case 4: sellProduct(cookies, counter);
                    break;
            default: cout<<"Bad Selection"<<endl;
        } //end switch

        showSelection(); //Step 6b
        cin>>choice; //Step 6c
    } //end while

    return 0;
} //end main
```

قائمة البرنامج:

```
//Main Program
#include <iostream>

#include "cashRegister.h"
#include "dispenserType.h"

using namespace std;

void showSelection();
void sellProduct(dispenserType& product, cashRegister& pCounter);

//Place the definition of the function main here.
//Place the definition of the function showSelection here.
//Place the definition of the function sellProduct here.

// ضع تعريف الدالة main هنا.
// ضع تعريف الدالة showSelection هنا.
// ضع تعريف الدالة sellProduct هنا.
```

تشغيل تجريبي: في هذا التشغيل التجريبي تكون مدخلات المستخدم مظللة.

*** أهلاً بكم في متجر شيلي للحلوى ***

لاختيار عنصر، ادخل

1 للحلوى

2 لرقائق البطاطس

3 للبان

4 للكعك

9 للخروج

1

يرجى ادخال 50 سنت

50

خذ ما طلبت من أسفل واستمتع

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*

*** أهلاً بكم في متجر شيلي للحلوى ***

لاختيار عنصر، ادخل

1 للحلوى

2 لرقائق البطاطس

3 للبان

من قائمة الأفعال المحددة في وصف المشكلة اختر قائمة بالعمليات الممكنة التي يمكن لموضوع من هذه الفئة القيام بها أو قام بها على نفسه. على سبيل المثال من قائمة الأفعال الخاصة بوصف مشكلة الأسطوانة - كتابة، وإدخال، وحساب، وطباعة - تكون العمليات الممكنة لموضوع الأسطوانة هي الإدخال، والحساب، والطباعة.

بالنسبة الى الفئة cylinderType تقوم الأبعاد بتمثيل البيانات ويكون مركز القاعدة، وقطر القاعدة، وارتفاع الأسطوانة هي خصائص الأبعاد. يمكنك ادخال البيانات الى الموضوع اما بواسطة مقوم أو بواسطة دالة.

الفعل حساب ينطبق على تحديد الحجم والمساحة الكلية. من هذا يمكنك استنتاج العمليات: cylinderVolume و cylinderSurfacaArea (حجم الأسطوانة، ومساحة سطح الأسطوانة). بالمثل ينطبق الفعل طباعة على عرض الحجم والمساحة الكلية على جهاز اخراج.

ان تحديد الفئات عبر الأسماء والأفعال من أوصاف المشكلات ليس هو الأسلوب الوحيد الممكن فهناك عدة أساليب أخرى للتصميم القائم على الموضوع يتم استخدامها في الأدب وبالرغم من هذا فان هذا الأسلوب يعتبر كافياً لتمارين البرمجة في هذا الكتاب.

مراجعة سريعة:

1. البرمجيات عبارة عن برامج يتم تشغيلها بواسطة الحاسب الآلي.
2. يمر البرنامج خلال العديد من المراحل من الوقت الذي يتم فيه تصوره أول مرة وحتى الوقت الذي يتم فيه رفعه من الخدمة وهذه الفترة تسمى بدورة حياة البرنامج.
3. المراحل الأساسية الثلاثة التي يمر بها أي برنامج هي التطوير والاستخدام والصيانة.
4. يتم عمل البرنامج الجديد في مرحلة تطوير البرمجيات.
5. في عملية صيانة البرمجيات يتم تعديل البرنامج لاصلاح المشكلات (التي يتم تحديدها) و/أو تعزيزها.
6. يتم رفع البرنامج من الخدمة اذا لم يكن يتم اصدار نسخة جديدة منه.
7. مراحل تطوير البرمجيات هي: التحليل والتطبيق والاختبار والتصحيح.
8. خلال مرحلة التصميم يتم تحديد حلول حسابية لحل المشكلة.
9. الحل الحسابي عبارة عن حل المشكلة خطوة بخطوة حتى يتم الوصول الى الحل في فترة زمنية محدودة.
10. يتم بناء أسلوبيين معروفين من التصميم وهما التصميم التركيبي والتصميم القائم على الموضوع.
11. في التصميم التركيبي يتم تقسيم المشكلة الى مشكلات فرعية أصغر. يتم حل كل مشكلة ثم يتم دمج حلول جميع المشكلات الفرعية لحل المشكلة.

12. في التصميم القائم على الموضوع يكون البرنامج عبارة عن مجموعة من الموضوعات المتفاعلة.
13. الموضوع يتكون من بيانات وعمليات مؤداة على البيانات.
14. المبادئ الأساسية الثلاثة للتصميم القائم على الموضوع هي: التغليف والتوريث وتعدد الأشكال.
15. في مرحلة التطبيق تقوم بكتابة وجمع كود البرمجة لتطبيق الفئات والدالات التي تم اكتشافها في مرحلة التصميم.
16. الشرط المسبق عبارة عن بيان يحدد الشروط التي يجب أن تكون فعلية قبل استدعاء الدالة.
17. الشرط التالي عبارة عن بيان يحدد ما هو فعلي بعد أن يتم الانتهاء من استدعاء الدالة.
18. خلال مرحلة الاختبار يتم اختبار البرنامج من أجل تصحيحه وهذا يعني التأكد من أن البرنامج يؤدي ما مفترض منه أن يؤديه.
19. التصحيح هو عمليو ايجاد الأخطاء وتصحيحها اذا وجدت.
20. للعثور على أي مشكلات في البرنامج يتم تشغيله خلال سلسلة من حالات الاختبار.
21. حالة الاختبار تتكون من مجموعة من المدخلات، أو تصرفات المستخدم، أو شروط أولية أخرى، والمخرجات المتوقعة.
22. هناك نوعان من الاختبار – اختبار الصندوق الأسود واختبار الصندوق الأبيض.
23. أثناء تحليل حل حسابي معين تقوم عادةً بإحصاء عدد العمليات المؤداة بواسطة الحل الحسابي.
24. لنكن f هي دالة n . يشير المصطلح "مقارب" الى دراسة الدالة f حيث تصبح n أكبر وأكبر بدون حد.
25. الفئة عبارة عن مجموعة من عدد ثابت من المكونات.
26. يطلق على مكونات الفئة عناصر الفئة.
27. يتم تناول عناصر الفئة بالاسم.
28. في C++ تكون الفئة كلمة محفوظة.
29. يتم تصنيف عناصر الفئة في واحدة من ثلاث فئات: خاصة ومحمية وعامة.
30. عناصر الفئة الخاصة لا يمكن تناولها خارج الفئة.
31. عناصر الفئة العامة يمكن تناولها خارج الفئة.
32. افتراضياً جميع عناصر الفئة تكون خاصة.
33. يتم اعلان العناصر الخاصة باستخدام محدد تناول العنصر private (خاص).
34. يتم اعلان العناصر العامة باستخدام محدد تناول العنصر Public (عام).
35. قد يكون عنصر الفئة دالة أو متغير (أي بيانات).

36. إذا كان أي عنصر من الفئة عبارة عن دالة فانك تقوم عادةً باستخدام نموذج الدالة لإعلانها.
37. إذا كان أي عنصر من الفئة عبارة عن متغير يتم إعلانه مثل أي متغير آخر.
38. في تعريف الفئة لا يمكنك تهيئة المتغير عندما تقوم بإعلانه.
39. في رسم لغة التشكيل الموحدة لفئة ما يحتوي الصندوق الأعلى على اسم الفئة ويحتوي الصندوق الأوسط على عناصر البيانات وأنواع بياناتها والصندوق الأخير يحتوي على اسم دالة العنصر وقائمة العوامل ونوع ناتج الدالة. تشير علامة زائد (+) أمام العنصر إلى أن هذا العنصر عنصر عام تشير علامة ناقص (-) أمام العنصر إلى أن هذا العنصر عنصر خاص. كما يشير الرمز (#) قبل اسم الهنصر إلى أن العنصر عنصر محمي.
40. في C++ الفئة عبارة عن تعريف. لا يتم تخصيص ذاكرة حيث يتم تخصيص الذاكرة من أجل متغيرات الفئة عندما تعلنها.
41. في C++ يطلق على متغيرات الفئة موضوعات الفئة أو موضوعات فقط.
42. يتم تناول عنصر الفئة باستخدام اسم متغير الفئة يليه نقطة (0) ثم يليها اسم العنصر.
43. العمليات الوحيدة المدمجة والتي يتم تأديتها على الفئات هي التحديد واختيار العنصر.
44. إن الفئات مثلها مثل العوامل بالنسبة إلى الدالات يمكن تمريرها إما بالقيمة أو بالإشارة.
45. يمكن أن تقوم الدالة بإنتاج قيمة من فئة النوع.
46. تضمن المقومات أنه يتم تهيئة عناصر البيانات عندما يتم إعلان موضوع ما.
47. اسم المقوم هو اسم الفئة.
48. يمكن أن يكون لدى الفئة الواحدة أكثر من مقوم.
49. المقوم الذي ليس له عوامل يسمى مقوم افتراضي.
50. المقوم يجري ألياً عندما يدخل موضوع الفئة في مجاله.
51. المدمرات تجري ألياً عندما يخرج موضوع الفئة من المجال.
52. يمكن أن يكون للفئة الواحدة مدمر واحد فقط والمدمر ليس له عوامل.
53. اسم المدمر عبارة عن تلمذة (~) يليها اسم الفئة (لا مسافات بينها).
54. المقومات والمدمرات عبارة عن دالات بدون أي نوع فهم ليسوا دالات منتجة لقيمة ولا دالات باطلة. كنتيجة لهذا لا يمكنك استدعائها مثل الدالات الأخرى.
55. نوع البيانات الذي يحدد الخصائص المنطقة بدون تفاصيل التطبيق يسمى نوع البيانات المجرد.
56. لتطبيق التصميم القائم على الموضوع يجب أن تقوم بتمثيل البيانات وكتابة الحلول الحسابية المرتبطة بها لتطبيق العمليات.

57. هناك طريقة سهلة لتحديد الفئات والموضوعات والعمليات وهي وصف المشكلة باللغة الانجليزية ثم تحديد جميع الأسماء والأفعال. اختر فئاتك (الموضوعات) من قائمة الأسماء والعمليات من قائمة الأفعال.

تمارين:

1. أياً من البيانات التالية صحيحة وأيها خطأ:
 - أ. دورة حياة البرمجيات تشير الى المراحل من النقطة التي تم عندها تصور البرمجيات وحتى رفعه من الخدمة.
 - ب. المراحل الأساسية الثلاثة للبرمجيات هي التطوير والاستخدام والنبد.
 - ت. التعبير $4n + 2n^2 + 5$ هو $O(n)$.
 - ث. يجب أن يكون عناصر بيانات الفئة الواحدة من نفس النوع.
 - ج. يجب أن تكون عناصر الدالة للفئة عامة.
 - ح. يمكن ان يكون للفئة الواحدة أكثر من مقوم.
 - خ. يمكن أن يكون للفئة الواحدة أكثر من مدمر.
 - د. يمكن لكل من المقومات والمدمرات امتلاك عوامل.
2. انظر الى نموذج الدالة التالي الذي يقدم الجذر التربيعي لعدد حقيقي:

Double sqrt (double x);

ما هي الشروط المسبقة والشروط التالية لهذه الدالة؟

3. كل من التعبيرات التالية تمثل عدد العمليات لحل حسابي معين. ما هو ترتيب كل من تلك التعبيرات؟

- أ. $n^2 + 6n + 4$
- ب. $5n^3 + 2n + 8$
- ج. $(n^2 + 1)(3n + 5)$
- د. $5(6n + 4)$

4. انظر الى الدالة التالية:

```
Void funcExercise4 (int x, int y)
```

```
{
```

```
    Int z;
```

```
    z = x + y;
```

```
    x = y;
```

```
    y = z;
```

```
    z = x;
```

```
    cout<<"x = "<<x<<" , y = "<<y<<" , z = "<<z<<endl;
```

```
}
```

جد عدد العمليات التي يتم تنفيذها بواسطة الدالة funcExercise4.
5. انظر الى الدالة التالية:

```
Int funcExercise5 (int list [ ], int size)
{
    int sum = 0;
    for (int index = 0; index < size; index++)
        Sum = sum + list [index];
    return sum;
}
```

أ. جد عدد العمليات التي يتم تنفيذها بواسطة الدالة funcExercise5 اذا كانت قيمة الحجم هي 10.

ب. جد عدد العمليات التي يتم تنفيذها بواسطة الدالة funcExercise5 اذا كانت قيمة الحجم هي n.

ت. ما هو ترتيب الدالة funcExercise5؟

6. انظر الى نموذج الدالة التالي:

```
Int funcExercise6 (int x);
```

الدالة funcExercise6 تنتج القيمة كما يلي: اذا كانت $x \geq 0$ ، تنتج القيمة $2x$ واذا

كانت $-50 \leq x < 0$ ، تنتج القيمة -999. ما هي قيم الحدود المنطقية للدالة funcExercise6؟

7. ما هو البيان الذي يوقف جميع بيانات التأكيد assert في البرنامج.

8. اكتب دالة تستخدم حلقة لايجاد مجموع مربعات جميع الأعداد الصحيحة بين 1 و n. ما هو ترتيب دالتك؟

9. ما هو اختبار الصندوق الأسود؟

10. ما هو اختبار الصندوق الأبيض؟

11. جد الأخطاء التركيبية في تعريفات الفئات التالية:

a.

```
class AA
{
public:
    void print();
    int sum();
    AA();
    int AA(int, int);
private:
    int x ;
    int y ;
};
```

b.

```
class BB
{
    int one ;
    int two;
public:
    bool equal();
    print();
    BB(int, int);
}
```

c.

```
class CC
{
public:
    void set(int, int);
    void print();
    CC();
    CC(int, int);
    bool CC(int, int);
private:
    int u;
    int v;
};
```

12. انظر الى الاعلانات التالية:

```
class xClass
{
public:
    void func();
    void print() const;

    xClass ();
    xClass (int, double);
private:
    int u;
    double w;
};

xClass x;
```

أ. كم عدد العناصر التي تمتلكها الفئة xClass؟

ب. كم عدد العناصر الخاصة التي تمتلكها الدالة xClass؟

ت. كم عدد المقومات التي تمتلكها الدالة xClass؟

ث. اكتب تعريف دالة العنصر func حتى يتم ضبط u عند 10 و w عند 15.3.

- ج. اكتب تعريف دالة العنصر print التي تقوم بطباعة محتويات u و w.
- ح. اكتب تعريف المقوم الافتراضي للفئة xClass حتى يتم تهيئة عناصر البيانات الخاصة عند صفر.
- خ. اكتب بيان C++ يطبع قيم عناصر بيانات الموضوع x.
- د. اكتب بيان C++ يعلن موضوع النوع xClass ويهيئ عناصر البيانات الخاصة بـ t عند 20 و 35.0 على التوالي.
13. انظر الى تعريف الفئة التالية:

```
class CC
{
public:
    CC(); //Line 1
    CC(int); //Line 2
    CC(int, int) //Line 3
    CC(double, int) //Line 4
private:
    int u;
    double v;
};
```

- أ. قم باعطاء رقم الصف المحتوي على المقوم الذي يتم تنفيذه في كل من الاعلانات التالية:
1. CC one
 2. CC two (5, 6)
 3. CC three (3, 5, 8)
- ب. اكتب تعريف المقوم في الصف 1 حتى يتم تهيئة عناصر البيانات الخاصة عند صفر.
- ت. اكتب تعريف المقوم في الصف 2 حتى يتم تهيئة عناصر البيانات الخاصة u وفقاً لقيمة العامل وعنصر البيانات الخاصة v عند صفر.
- ث. اكتب تعريف المقومات في الصفوف 3 و 4 حتى يتم تهيئة عناصر البيانات الخاصة وفقاً لقيم العوامل.

14. انظر الى تعريف الفئة التالية:

```
class testClass
{
public:
    int sum();
    //Postcondition: Returns the sum of the
    //                private data members.
    void print() const;
    //Prints the values of the private data
    //members.
    testClass();
    //default constructor
    //Postcondition: x = 0; y = 0
    testClass(int a, int b);
    //constructor with parameters
    //Initializes the private data members to the
    //values specified by the parameters.
    //Postcondition: x = a; y = b
private:
    int x;
    int y;
};
```

أ. اكتب تعريف دالات العنصر كما يتم وصفها في تعريف الفئة testClass.

ب. اكتب برنامج اختبار لاختبار العمليات المتنوعة للفئة testClass.

تمارين برمجة:

1. اكتب برنامج لاختبار العمليات المتنوعة للفئة clockType.

2. اكتب برنامج يقوم بتحويل عدد يتم ادخاله بالأرقام الرومانية الى عدد عشري. يجب أن يتكون

برنامجك من فئة ولنقل romanType ويجب أن يقوم موضوع النوع romanType بالقيام

بما يلي:

أ. تخزين الرقم كرقم روماني.

ب. تحويل وتخزين الرقم الى رقم عشري.

ت. طباعة الرقم كرقم روماني أو رقم عشري حسب طلب المستخدم.

القيم العشرية للأرقام الرومانية هي:

| | |
|------|---|
| 1000 | M |
| 500 | D |
| 100 | C |
| 50 | L |
| 10 | X |
| 5 | V |
| 1 | I |

ث. قم باختبار برنامجك باستخدام الأرقام الرومانية التالية: MCXIV، وCCCLIX، وMDCLXVI.

3. قم بتصميم وتطبيق الفئة dayType التي تقوم باستخدام يوم من أيام الأسبوع في برنامج. يجب أن تقوم الفئة dayType بتخزين اليوم مثل الأحد من أجل يوم الأحد. يجب أن يكون البرنامج قادراً على أداء جميع العمليات التالية على موضوع النوع dayType:

- أ. ضبط اليوم.
- ب. طباعة اليوم.
- ت. اخراج اليوم.
- ث. اخراج اليوم التالي.
- ج. اخراج اليوم السابق.
- ح. تحديث اليوم المخزن في موضوع DayType عن طريق اضافة أيام معينة اليه. على سبيل المثال اذا كان اليوم هو يوم الاثنين وأضفت 4 أيام فان اليوم الذي يتم تحديثه هو يوم الجمعة. بالمثل اذا كان اليوم هو يوم الثلاثاء وأضفت 13 يوم فان اليوم الذي يتم تحديثه هو يوم الاثنين.
- خ. اصف المقومات المناسبة.

4. اكتب تعريفات الدالات لتنفيذ عمليات الفئة dayType كما هي معرفة في تمرين البرمجة رقم 3 وقم كذلك بكتابة برنامج لاختبار العمليات المتنوعة للفئة dayType.

5. قام المثال 1-6 بتعريف الفئة personType لتخزين اسم الشخص ودالات العنصر التي ضمنها كانت طباعة وتحديد اسم الشخص. قم باعادة تعريف الفئة personType حتى يمكنك أيضاً :

- أ- تحديد الاسم الأخير فقط.
- ب- تحديد الاسم الأول فقط.
- ت- تخزين وتحديد الاسم الأوسط.
- ث- التحقق مما اذا كان الاسم الأخير المعطى هو نفس الاسم الأخير لهذا الشخص.
- ج- التحقق مما اذا كان الاسم الأول المعطى هو نفس الاسم الأول لهذا الشخص.
- ح- اكتب تعريفات دالات العنصر لتنفيذ العمليات لتلك الفئة.

6.

أ. بعض من مميزات هذا الكتاب ما يلي: العنوان، والمؤلف، والناشر، وISBN، والسعر، وعام النشر. قم بتصميم الفئة bookType التي تعرف الكتاب كنوع بيانات مجردة.

يمكن لكل موضوع من الفئة bookType أن يحمل المعلومات التالية عن كتاب: العنوان، وحتى أربعة مؤلفين، الناشر، وISBN، والسعر، وعدد النسخ في المخزن. للحفاظ على تتبع عدد المؤلفين قم بإضافة عنصر بيانات آخر.

قم بتضمين دالات العنصر لكي تؤدي العمليات المتنوعة على موضوعات bookType. على سبيل المثال العمليات التقليدية التي يمكن أدائها على العنوان هي اظهار العنوان، والتحقق مما اذا كان العنوان هو نفس الاسم الفعلي للكتاب. بالمثل تكون العمليات التقليدية التي يمكن أدائها على عدد النسخ في المخزن هي اظهار عدد النسخ في المخزن، وضبط عدد النسخ في المخزن، وتحديث عدد النسخ في المخزن، وتقديم عدد النسخ في المخزن. أضف عمليات مماثلة بالنسبة الى الناشر، وISBN، وسعر الكتاب، والمؤلفين. أضف مقومات مناسبة ومدمر مناسب (إذا كان هناك حاجة الى واحد).

ب. اكتب تعريفات دالات عنصر الفئة bookType.

ج. كتابة برنامج يستخدم الفئة bookType ويختبر العمليات المتنوعة على موضوعات الفئة bookType. قم باعلان مجال مكون من 100 مكون من النوع bookType. بعض من العمليات الي يجب عليك أدائها هي البحث عن كتاب بواسطة عنوانه أو البحث بواسطة ISBN، وتحديث عدد النسخ في المخزن.

7. في هذا التمرين سوف تقوم بتصميم فئة memberType.

أ- يمكن لكل موضوع من memberType أن يحمل اسم شخص، وهوية العضو، وعدد الكتب المشتراة، والمبلغ المنفق.

ب- قم بتضمين دالات العنصر لأداء العمليات المتنوعة على موضوعات memberType – على سبيل المثال قم بتحديد و اظهار اسم الشخص وبالمثال قم بتحديث وتعديل و اظهار عدد الكتب المشتراة والمبلغ المنفق.

ت- أضف المقومات المناسبة والمدمر المناسب (إذا كان المدمر لازماً).

ث- اكتب تعريفات دالات العنصر memberType.

8. باستخدام الفئات المحددة في تمارين البرمجة 6 و 7 اكتب برنامج لمحاكاة متجر بيع الكتب. متجر الكتب يه نوعان من العملاء: هؤلاء الذين يكونون أعضاء في المتجر وهؤلاء الذين يشترون كتب من المتجر من حين لآخر فقط. على كل عضو دفع 10 دولار كرسوم عضوية سنوية ويتلقى 5% خصم على كل كتاب يتم شراؤه.

بالنسبة الى كل عضو يقوم متجر بيع الكتب بتتبع عدد الكتب المشتراة والمبلغ الكلي الذي يتم انفاقه. لكل احدى عشر كتاباً يشتريها العضو يأخذ متجر الكتب متوسط المبلغ الكلي للعشرة كتب الأخيرة التي تم شرائها ويقوم بتطبيق هذا المبلغ كخصم ثم تعيد ضبط المبلغ الكلي المنفق عند القيمة صفر.

اكتب برنامج يمكنه معالجة ما يصل الى 1000 عنوان كتاب و500 عضو. يجب أن يشتمل برنامجك على قائمة تعطي المستخدم خيارات مختلفة لتشغيل البرنامج بفاعلية أي يجب أن يتم تشغيل برنامجك بالقائمة.

9. الدالة sellProduct لمثال البرمجة ماكينة الحلوى تعطي المستخدم فرصتين لادخال النقود لشراء المنتج. أعد كتابة تعريف الدالة sellProduct حتى يمكنها الحفاظ على حث المستخدم على ادخال النقود طالما لم يتم المستخدم بادخال النقود الكافية لشراء المنتج. كذلك اكتب برنامج لاختبار دالتك.

7.

أ. بعض من مميزات هذا الكتاب ما يلي: العنوان، والمؤلف، والناشر، وISBN، والسعر، وعام النشر. قم بتصميم الفئة bookType التي تعرف الكتاب كنوع بيانات مجردة. يمكن لكل موضوع من الفئة bookType أن يحمل المعلومات التالية عن كتاب: العنوان، وحتى أربعة مؤلفين، والناشر، وISBN، والسعر، وعدد النسخ في المخزن. للحفاظ على تتبع عدد المؤلفين قم بإضافة عنصر بيانات آخر.

قم بتضمين دالات العنصر لكي تؤدي العمليات المتنوعة على موضوعات bookType. على سبيل المثال العمليات التقليدية التي يمكن أدائها على العنوان هي اظهار العنوان، والتحقق مما اذا كان العنوان هو نفس الاسم الفعلي للكتاب. بالمثل تكون العمليات التقليدية التي يمكن أدائها على عدد النسخ في المخزن هي اظهار عدد النسخ في المخزن، وضبط عدد النسخ في المخزن، وتحديث عدد النسخ في المخزن، وتقديم عدد النسخ في المخزن. أضف عمليات مماثلة بالنسبة الى الناشر، وISBN، وسعر الكتاب، والمؤلفين. أضف مقومات مناسبة ومدمر مناسب (إذا كان هناك حاجة الى واحد).

ب. اكتب تعريفات دالات عنصر الفئة bookType.

ج. كتابة برنامج يستخدم الفئة bookType ويختبر العمليات المتنوعة على موضوعات الفئة bookType. قم باعلان مجال مكون من 100 مكون من النوع bookType. بعض من العمليات الي يجب عليك أدائها هي البحث عن كتاب بواسطة عنوانه أو البحث بواسطة ISBN، وتحديث عدد النسخ في المخزن.

7. في هذا التمرين سوف تقوم بتصميم فئة memberType.

ج- يمكن لكل موضوع من memberType أن يحمل اسم شخص، وهوية العضو، وعدد الكتب المشتراة، والمبلغ المنفق.

ح- قم بتضمين دالات العنصر لأداء العمليات المتنوعة على موضوعات memberType – على سبيل المثال قم بتحديد وإظهار اسم الشخص وبالمثال قم بتحديث وتعديل وإظهار عدد الكتب المشتراة والمبلغ المنفق.

خ- أضف المقومات المناسبة والمدمر المناسب (إذا كان المدمر لازماً).

د- اكتب تعريفات دالات العنصر memberType.

8. باستخدام الفئات المحددة في تمارين البرمجة 6 و 7 اكتب برنامج لمحاكاة متجر بيع الكتب. متجر الكتب به نوعان من العملاء: هؤلاء الذين يكونون أعضاء في المتجر وهؤلاء الذين يشترون كتب من المتجر من حين لآخر فقط. على كل عضو دفع 10 دولار كرسوم عضوية سنوية ويتلقى 5% خصم على كل كتاب يتم شراؤه.

بالنسبة الى كل عضو يقوم متجر بيع الكتب بتتبع عدد الكتب المشتراة والمبلغ الكلي الذي يتم انفاقه. لكل احدى عشر كتاباً يشتريها العضو يأخذ متجر الكتب متوسط المبلغ الكلي للعشرة كتب الأخيرة التي تم شرائها ويقوم بتطبيق هذا المبلغ كخصم ثم تعيد ضبط المبلغ الكلي المنفق عند القيمة صفر. اكتب برنامج يمكنه معالجة ما يصل الى 1000 عنوان كتاب و 500 عضو. يجب أن يشتمل برنامجك على قائمة تعطي المستخدم خيارات مختلفة لتشغيل البرنامج بفاعلية أي يجب أن يتم تشغيل برنامجك بالقائمة.

9. الدالة sellProduct لمثال البرمجة ماكينة الحلوى تعطي المستخدم فرصتين لادخال النقود لشراء المنتج. أعد كتابة تعريف الدالة sellProduct حتى يمكنها الحفاظ على حث المستخدم على ادخال النقود طالما لم يتم المستخدم بادخال النقود الكافية لشراء المنتج. كذلك اكتب برنامج لاختبار دالتك.

الفصل

2

التصميم القائم على الموضوع

C++ و

في هذا الفصل سوف:

قام الفصل الأول بتقديم الفئات، وأنواع البيانات المجردة، وطرق تطبيق نوع البيانات المجردة في C++. باستخدام الفئات يمكنك دمج البيانات والعمليات في وحدة واحدة. لهذا يصبح الموضوع هوية مكثفية ذاتياً. يمكن أن نتناول العمليات البيانات بشكل مباشر ولكن لا يمكن التلاعب في الحالة الداخلية للموضوع بشكل مباشر.

بالإضافة الى تطبيق نوع البيانات المجردة تمتلك الفئات سمات أخرى. على سبيل المثال يمكنك خلق فئات جديدة من فئات موجودة بالفعل وهذه الميزة الهامة تشجع على إعادة استخدام الكود.

التوريث:

افترض أنك تريد تصميم فئة `partTimeEmployee` لتطبيق ومعالجة خصائص موظف العمل الجزئي. السمات الأساسية المرتبطة بموظف العمل الجزئي هي الاسم، ومعدل الأجر، وعدد الساعات التي يتم العمل فيها. في المثال 1-6 (في الفصل 1) قمنا بتصميم فئة لتطبيق اسم الشخص وكل موظف عمل جزئي عبارة عن شخص. لهذا بدلاً من تصميم الفئة `partTimeEmployee` من الصفر نريد أن نكون قادرين على مد تعريف الفئة `personType` (من المثال 1-6) عن طريق إضافة عناصر إضافية (بيانات و/أو دالات).

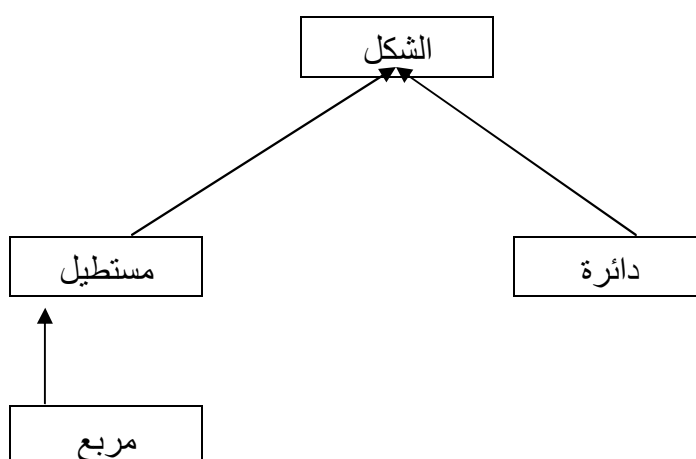
اننا بالطبع لا نريد عمل التغييرات اللازمة مباشرةً على فئة `personType` - أي تعديل الفئة `personType` عن طريق إضافة أو حذف عناصر. في الحقيقة اننا نريد عمل الفئة `partTimeEmployee` دون عمل أي تغييرات مادية في الفئة `personType` بإضافة العناصر الضرورية فقط. على سبيل المثال تمتلك الفئة `personType` بالفعل عناصر بيانات لتخزين الاسم الأول والاسم الأخير ولهذا لن ندخل أي عناصر كهذه في الفئة `partTimeEmployee`. في الحقيقة سوف يتم توريث عناصر البيانات هذه من الفئة `personType` (سوف نقوم بتصميم هذه الفئة في المثال 2-2).

في الفصل الأول قمنا بدراسة وتصميم الفئة clockType على نحو واسع لتطبيق الوقت في برنامج ما. الفئة clockType لها ثلاثة عناصر بيانات لتخزين الساعات والدقائق والثواني. بالإضافة الى الساعات والدقائق والثواني قد تحتاج تطبيقات معينة منا أن نقوم كذلك بتخزين المنطقة الزمنية. في هذه الحالة سوف نحب مد تعريف الفئة clockType وعمل فئة extClockType لاستضافة هذه المعلومات الجديدة. هذا يعني أننا نريد أن نخرج الفئة extClockType عن طريق اضافة عنصر بيانات – لنقل المنطقة الزمنية – وعناصر الدالة اللازمة الى clockType لاحتكار الوقت. في C++ تكون الآلية التي تسمح لنا بانجاز هذه المهمة هو مبدأ التوريث. التوريث عبارة عن علاقة "is-a" على سبيل المثال "كل موظف شخص".

التوريث يدعنا نقوم لعمل فئات جديدة من فئات موجودة بالفعل. الفئة الموجودة تعرف **بالفئة القاعدة** والفئة الجديدة التي نعملها من فئة موجودة تسمى **فئة مستمدة**. الفئة المستمدة ترث خصائص القاعدة الأساسية. بدلاً من عمل فئات جديدة تماماً من الصفر يمكننا الاستفادة من التوريث وتقليل تعقيد البرمجيات.

كل فئة مستمدة تصبح بدورها فئة أساسية لفئة مستمدة مستقبلية. قد يكون التوريث توريث فردي أو توريث متعدد. في **التوريث الفردي** تكون الفئة المستمدة مستمدة من فئة أساسية واحدة وفي **التوريث المتعدد** تكون الفئة المستمدة مستمدة من أكثر من فئة أساسية. يقوم هذا الفصل بالتركيز على التوريث الفردي.

يمكن عرض التوريث في هيئة شجرة أو تركيب تسلسلي حيث يتم اظهار الفئة الأساسية مع فئاتها المستمدة. انظر الى رسم الشجرة الموجود في الشكل 1-2 الذي يوضح العلاقة بين أنواع مختلفة من الأشكال.



شكل 1-2 تسلسل التوريث.

في هذا الشكل الشكل هو القاعدة الأساسية وفئات الدائرة والمستطيل مستمدتين من الشكل وفئة المربع مستمدة من المستطيل. كل دائرة وكل مستطيل عبارة عن شكل وكل مربع عبارة عن مستطيل. التركيب العام لتعريف الفئة الممتدة هو:

```
Class className : memberAccessSpecifier baseClassName
{
    Member list
};
```

اسم الفئة : محدد تناول العنصر اسم الفئة الأساسية

قائمة العنصر

حيث يكون محدد تناول العنصر عام أو محمي أو خاص. عندما لا يتم تحديد محدد تناول العنصر من المفترض أن يكون التوريث فردي. (اننا نناقش التوريث المحمي لاحقاً في هذا الفصل).

تذكر الحقائق التالية عن الفئات الأساسية والمستمدة:

- العناصر الخاصة للفئة الأساسية خاصة بالفئة الأساسية ومن هنا لا يمكن لعناصر الفئة المستمدة تناولها بشكل مباشر. عندما تقوم بكتابة تعريفات دالات عنصر الفئة المستمدة لا يمكنك تناول العناصر الخاصة للفئة الأساسية بشكل مباشر.
- يمكن توريث العناصر العامة للفئة الأساسية اما كعناصر عامة أو كعناصر خاصة بواسطة الفئة المستمدة. هذا يعني أن العناصر العامة للفئة الأساسية يمكنها أن تصبح عناصر عامة أو خاصة للفئة المستمدة.
- يمكن للفئة المستمدة أن تتضمن بيانات اضافية و/أو عناصر دالة اضافية.
- الفئة المستمدة يمكنها اعادة تعريف دالات العنصر العام للفئة الأساسية أي أنك في الفئة المستمدة يمكنك الحصول على عنصر دالة بنفس الاسم والعدد وأنواع العوامل مثل دالة في الفئة الأساسية. بالرغم من هذا ينطبق اعادة التعريف هذا على موضوعات الفئة المستمدة فقط وليس على موضوعات الفئة الأساسية.
- جميع عناصر بيانات الفئة الأساسية تعتبر عناصر بيانات الفئة المستمدة. بالمثل تكون دالات عنصر الفئة الأساسية (الا اذا تم اعادة تعريفها) هي كذلك دالات عنصر الفئة المستمدة. (تذكر أول عنصر منقوط عند تناول عنصر لفئة أساسية في الفئة المستمدة).

مثال 1-2:

افترض أننا لدينا الفئة shape.

(أ) البيانات التالية تحدد أن الفئة Circle (دائرة) مستمدة من shape (شكل) وهذا توريث عام:

```
class circle: public shape
{
    .
    .
    .
};
```

(ب) انظر الى التعريف التالي للفئة circle:

```
class circle: private shape
{
    .
    .
    .
};
```

(ت) هذه البيانات تحدد أن أن الفئة Circle (دائرة) مستمدة من الفئة shape (شكل) وهذا

توريث خاص. تصبح العناصر العامة للفئة shape عناصر خاصة للفئة circle. لهذا أي

موضوع من النوع circle لا يمكنه تناول تلك العناصر بشكل مباشر. التعريف السابق للدائرة

circle متكافئ مع

```
class circle: shape
{
    .
    .
    .
};
```

القسمان التاليان يقومان بوصف موضوعين هامين مرتبطين بالتوريث. الموضوع الأول هو إعادة تعريف دالات عنصر الفئة الأساسية في الفئة المستمدة. أثناء مناقشة هذا الموضوع سوف نناقش كذلك كيفية تناول العناصر الخاصة للفئة الأساسية في الفئة المستمدة. موضوع التوريث الثاني مرتبط بالمقوم. لا يمكن لمقوم الفئة المستمدة أن يتناول عناصر البيانات الخاصة للفئة الأساسية مباشرةً ولهذا نحتاج الى التأكد من أن عناصر البيانات الخاصة التي يتم توريثها من الفئة الأساسية يتم تهيئتها عند يجري مقوم الفئة المستمدة.

إعادة تعريف دالات عنصر الفئة الأساسية:

افترض أن الفئة derivedClass مستمدة من الفئة baseClass وفضلاً عن هذا افترض أن كلا من derivedClass و baseClass لها بعض عناصر بيانات. اذن ينتج أن عناصر بيانات الفئة derivedClass هي عناصر البيانات الخاصة بها مع عناصر بيانات baseClass. افترض أن الفئة

baseClass تحتوي على دالة طباعة print تقوم بطبع قيم عناصر بيانات الفئة baseClass. الآن تحتوي فئة derivedClass على عناصر بيانات أخرى بالإضافة الى عناصر البيانات التي ورثتها من فئة baseClass. افترض أنك تريد ادخال دالة تقوم بطبع عناصر بيانات الفئة derivedClass. يمكنك اعطاء أي اسم لهذه الدالة. بالرغم من هذا يمكنك في الفئة derivedClass تسمية هذه الدالة بالاسم print كذلك (نفس الاسم المستخدم بواسطة baseClass). هذا هو ما يسمى **اعادة تعريف** (يسمى أيضاً سيطرة) دالة عنصر الفئة الأساسية. بعد هذا نوضح كيفية اعادة تعريف دالات العنصر العامة للفئة الأساسية.

لاعادة تعريف دالة عنصر عام للفئة الأساسية في الفئة المستمدة يجب أن يكون للدالة المتوافقة في الفئة المستمدة نفس اسم وعدد وأنواع العوامل. بمعنى آخر اسم الدالة التي يتم اعادة تعريفها في الفئة المستمدة يجب أن يكون له نفس قائمة العامل الرسمي. اذا كانت الدالات المتوافقة في الفئة الأساسية والفئة المستمدة لها نفس الاسم ولكن قائمة عامل رسمي مختلف فان هذه الدالة تنقل في الفئة المستمدة وهذا مسموح به كذلك.

انظر الى تعريف الفئة التالية:

```
class baseClass
{
public:
    void print() const;

private:
    int u;
    int v;
    char ch;
};
```

الفئة baseClass بها أربعة عناصر. افترض أن تعريف دالة العنصر print (الطباعة) للفئة baseClass هو:

```
void baseClass::print() const
{
    cout<<"Base Class: u = "<<u<<" , v = "<<v
        <<" , ch = "<<ch<<endl;
}
```

الآن انظر الى تعريف الفئة التالية:

```
class derivedClass: public baseClass
{
public:
    void print() const;

private:
    int first;
    double second;
};
```

من تعريف الفئة derivedClass من الواضح أن الفئة derivedClass مستمدة من الفئة baseClass وأنه توريث عام. لهذا فان جميع العناصر العامة للفئة baseClass يتم توريثها كعناصر عامة للفئة derivedClass. كما تقوم الفئة derivedClass أيضاً بكتابة دالة الطباعة print.

بعد هذا دعونا نكتب تعريف دالة العنصر print للفئة derived Class.

الفئة derivedClass لها خمسة عناصر بيانات وهي u و v و ch و first و second. دالة الطباعة print للفئة derivedClass تقوم بطبع قيم عناصر البيانات هذه. لكتابة تعريف دالة العنصر print للفئة derivedClass تذكر ما يلي:

- عناصر البيانات u و v و ch عناصر خاصة للفئة baseClass ولهذا لا يمكن تناولها مباشرة في فئة derivedClass. لهذا عند كتابة تعريف الدالة print للفئة derivedClass لا يمكننا تناول u و v و ch بشكل مباشر.

- عناصر البيانات u و v و ch للفئة baseClass يمكن تناولهم في الفئة derivedClass خلال العناصر العامة للفئة baseClass.

لهذا عند كتابة تعريف الدالة print (طباعة) للفئة derivedClass نقوم أولاً باستدعاء دالة العنصر print للفئة baseClass لطباعة قيم كل من u و v و ch. بعد طباعة القيم u و v و ch نقوم باخراج قيم الأول والثاني first & second.

لاستدعاء الدالة print من الفئة baseClass في تعريف الدالة print للفئة derivedClass يجب علينا استخدام البيان التالي:

```
baseClass::print();
```

يضمن هذا البيان أننا نستدعي الدالة print من الفئة baseClass وليس من الفئة derivedClass. تعريف دالة العنصر print للفئة derivedClass هو:

```
void derivedClass::print() const
{
    baseClass::print();
    cout<<"Derived Class: first = "<<first
        <<" , second = "<<second<<endl;
    cout<<"_____ "
        <<"_____"<<endl;
}
```

مقومات الفئات المستمدة والأساسية:

الفئة المستمدة لها عناصر بياناتها الخاصة ولهذا يمكن للفئة المستمدة أن يكون لها مقوماتها الخاصة. من المعتاد أن يعمل المقوم على تهيئة عناصر البيانات. عندما نعلن موضوع فئة مستمدة يرث هذا الموضوع عناصر الفئة الأساسية وبالرغم من هذا لا يمكن لموضوع الفئة المستمدة تناول العناصر الخاصة للفئة الأساسية مباشرة. نفس الأمر حقيقي بالنسبة الى دالات عنصر الفئة المستمدة. هذا يعني أن دالات عنصر الفئة المستمدة لا يمكنها تناول العناصر الخاصة للفئة الأساسية مباشرة.

بالتالي يمكن لمقومات الفئة المستمدة أن تهيئ عناصر البيانات الخاصة فقط للفئة المستمدة (مباشرة) ولهذا عند اعلان موضوع فئة مستمدة يجب أن تقوم أوتوماتيكياً بتنفيذ واحد من مقومات القاعدة الأساسية. لا يمكن استدعاء المقومات مثل الدالات الأخرى ولهذا فان تنفيذ مقوم فئة مستمدة يجب أن يسبب تنفيذ واحد من مقومات القاعدة الأساسية وهذا هو ما يحدث في الحقيقة. فضلاً عن هذا يتم تحديد استدعاء مقوم الفئة الأساسية في عنوان تعريف مقوم الفئة المستمدة.

لنقم بتوضيح هذا المفهوم بمساعدة مثال. نقوم أولاً بتعريف فئة أساسية ثم فئة مستمدة ولكل من الفئتين مقوماتها الخاصة بها.

انظر الى تعريف الفئة التالية (نظر أيضاً شكل 2-2):

```
class baseClass
{
public:
    void print();                //baseClass Line 1

    baseClass();                //baseClass Line 2
    baseClass(int x, int y);     //baseClass Line 3
    baseClass(int x, int y, char w); //baseClass Line 4

private:
    int u;
    int v;
    char ch;
};
```

| baseClass |
|---|
| -u : int -v : int -ch : char |
| +print () : void +baseClass () +baseClass (int, int) +baseClass (int, int, char) |

شكل 2-2: رسم لغة التشكيل الموحدة للفئة baseClass.

الفئة baseClass لها ثلاث مقومات، ودالة عنصر print، وثلاثة عناصر بيانات. افترض أن تعريفات دالة العنصر ومقومات الفئة baseClass كما يلي:

```
void baseClass::print()
{
    cout<<"Base Class: u = "<<u<<" , v = "<<v
    <<" , ch = "<<ch<<endl;
}

//default constructor; baseClass Line 2
baseClass::baseClass()
{
    u = 0;
    v = 0;
    ch = '*';
}

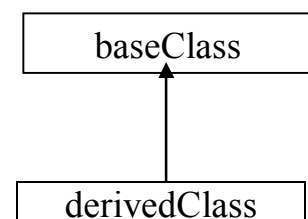
//constructor; baseClass Line 3
baseClass::baseClass(int x, int y)
{
    u = x;
    v = y;
    ch = '*';
}

//constructor; baseClass Line 4
baseClass::baseClass(int x, int y, char w)
{
    u = x;
    v = y;
    ch = w;
}
```

انظر الآن الى تعريف الفئة التالية (نظر أيضاً شكل 2-3):

```
class derivedClass: public baseClass
{
public:
    void print(); //derivedClass Line 1
    derivedClass(); //derivedClass Line 2
    derivedClass(int x, int y,
        int one, double two); //derivedClass Line 3
    derivedClass(int x, int y, char w,
        int one, double two); //derivedClass Line 4
private:
    int first;
    double second;
};
```

| DerivedClass |
|--|
| -first : int -second : double |
| +print () : void +derivedClass () +derivedClass (int, int, int, double) +derivedClass (int, int, char, int, double) |



شكل 2-3: رسم لغة التشكيل الموحدة للفئة derivedClass وتسلسل التوريث.

الفئة derivedClass مستمدة من الفئة baseClass وهذا توريث عام. الفئة derivedClass لها خمسة عناصر بيانات هي: u و v و ch و first و second. يتم توريث عناصر البيانات u و v و ch و first و second من الفئة baseClass. لنقم أولاً بكتابة تعريف دالة العنصر print للفئة derivedClass.

```
void derivedClass::print()
{
    baseClass::print();
    cout<<"Derived Class: first = "<<first
        <<" , second = "<<second<<endl;
    cout<<"_____ "
        <<"_____ "<<endl;
}
```

سوف نقوم الآن بكتابة تعريفات مقومات الفئة derivedClass. تذكر أن استدعاء مقوم الفئة الأساسية محدد في عنوان تعريف مقوم الفئة المستمدة.

أولاً نقم بكتابة تعريف المقوم الافتراضي للفئة derivedClass. تذكر أنه إذا احتوت الفئة على المقوم الافتراضي ولم يتم تحديد قيم أثناء اعلان الموضوع فإن المقوم الافتراضي يجري ويهيئ الموضوع. الفئة baseClass تحتوي على المقوم الافتراضي ولهذا عندما يتم كتابة تعريف المقوم الافتراضي للفئة derivedClass لا نقوم بتحديد أي مقوم للفئة الأساسية.

```
derivedClass::derivedClass() //default constructor
{
    first = 0;
    second = 0;
}
```

بعد هذا نناقش كيفية كتابة تعريفات المقومات ذات العوامل. للتسبب في تنفيذ مقوم ما (ذو عوامل) للفئة الأساسية نقوم بتحديد اسم مقوم الفئة الأساسية مع العوامل في عنوان تعريف مقوم الفئة المستمدة. انظر الى التعريف التالي لمقوم الفئة المستمدة derivedClass (المعلنة في الصف رقم 3 :derivedClass)

```
derivedClass::derivedClass(int x, int y, int one, double two)
    : baseClass(x,y)
{
    first = one;
    second = two;
}
```

في هذا التعريف نقوم بتحديد مقوم الفئة الأساسية (baseClass) بعاملين (معلنان في الصف رقم 3 (baseClass). عندما يجري هذا المقوم للفئة المستمدة (derivedClass) يتسبب في تنفيذ المقوم بعاملين int (معلنان في الصف رقم 3 (baseClass). بعد هذا انظر الى التعريف التالي لمقوم الفئة المستمدة (derivedClass) المعلن في السف رقم 4 من الفئة المستمدة (derviedClass):

```
derivedClass::derivedClass(int x, int y, char w,
                           int one, double two)
    :baseClass(x,y,w)
{
    first = one;
    second = two;
}
```

هذا التعريف يحدد مقوم الفئة الأساسية (baseClass) بثلاثة عوامل (معلنة في الصف الرابع المسمى (baseClass). عندما يجري هذا المقوم الخاص بالفئة المستمدة (derivedClass) يتسبب في تنفيذ مقوم ذو ثلاثة عناصر بالترتيب int و int و char (المعلنة في الصف الرابع المسمى (baseClass).

مثال 2-2:

افترض أنك تريد تعريف فئة لتجميع صفات الموظفين. هناك موظفين يعملون وقت كلي وموظفين يعملون وقت جزئي. موظفون الوقت الجزئي يتم دفع راتبهم على أساس عدد الساعات التي يعملونها وبالساعة. افترض أنك تريد تعريف فئة لتتبع معلومات موظفين الوقت الجزئي مثل الاسم (name) ومعدل الأجر (pay rate) وساعات العمل (hours worked). اذن يمكنك طباعة اسم الموظف مع أجوره. كل موظف عبارة عن شخص والمثال رقم 1-6 (الفصل 1) قام بتعريف فئة نوع الشخص (personType) لتخزين الاسم الأول والاسم الأخير معاً مع العمليات الضرورية على الاسم ولهذا نستطيع تعريف فئة موظف الوقت الجزئي (partTimeEmployee) بناءً على فئة نوع الشخص (personType). (انظر شكل 2-4) كما يمكنك اعادة تعريف دالة الطباعة (print) من أجل طباعة المعلومات المناسبة.

```

class partTimeEmployee: public personType
{
public:
    void print();
        //Function to output the first name, last name,
        //and the wages.
        //Postcondition: Outputs:
        //      firstName lastName wages are $$$$.$$

    double calculatePay();
        //Function to calculate and return the wages.
        //Postcondition: The pay is calculated and returned.

    void setNameRateHours(string first, string last,
                           double rate, double hours);
        //Function to set the first name, last name,
        //payRate, and hoursWorked according to the
        //parameters.
        //Postcondition: firstName = first; lastName = last;
        //      payRate = rate; hoursWorked = hours

    partTimeEmployee(string first = "", string last = "",
                      double rate = 0, double hours = 0);
        //constructor with parameters
        //Sets the first name, last name, pay rate, and
        //hours worked according to the parameters. If
        //no value is specified, the default values are
        //assumed.
        //Postcondition: firstName = first;
        //      lastName = last; payRate = rate;
        //      hoursWorked = hours

```

Class partTimeEmployee : public personType

}

عامّة:

Void print ();

// دالة لاجراج الاسم الأول والاسم الأخير

// والأجور.

// شرط تالي: مخرجات:

// الاسم الأول الاسم الأخير الأجور هي \$ \$ \$ \$ \$ \$

Double calculatePay ();

// دالة لحساب وانتاج الأجور.

// شرط تالي: يتم حساب وانتاج الأجر.

Void setNameRateHours (string first, string last,

double rate, double hours);

// دالة لتحديد الاسم الأول والاسم الأخير

// ومعدل الأجر وساعات العمل وفقاً الى

// العوامل.

// شرط تالي: الاسم الأول = أول، الاسم الأخير = أخير،

معدل الأجر = معدل، ساعات العمل = ساعات

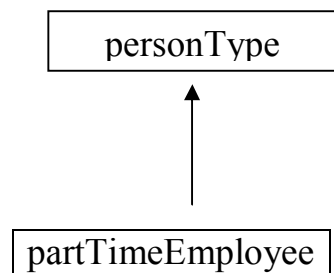
```
partTimeEmployee (string first = " ", string last = " ",
double rate = 0, double hours = 0);
// مقوم ذو عوامل
// يحدد الاسم الأول والاسم الأخير ومعدل الأجر و
// ساعات العمل وفقاً للعوامل. اذا
// لم يتم تحديد أي قيمة يتم افتراض القيم الافتراضية.
// شرط تالي: الاسم الأول = أول، الاسم الأخير = أخير،
معدل الأجر = معدل، ساعات العمل = ساعات
```

```
private:
    double payRate;        //stores the pay rate
    double hoursWorked;    //stores the hours worked
};
```

خاصة:

```
Double payRate;
Double hoursWorked;
{
    // تخزين معدل الأجر.
    // تخزين ساعات العمل.
```

| partTimeEmployee |
|--|
| -payRate : double -hoursWorked : double |
| +print () : void +calculatePay () : double +setNameRateHours (string, string, Double, double) : void +partTimeEmployee (string = " ", string = " ", Double = 0, double = 0) |



شكل 2-4: رسم لغة التشكيل الموحدة للفئة partTimeEmployee وتسلسل التوريث.

تعريفات دالات العنصر للفئة partTimeEmployee كما تلي:

```
void partTimeEmployee::print()
{
    personType::print();//print the name of the
                        //employee
    cout<<" wages are: "<<calculatePay()<<endl;
}

double partTimeEmployee::calculatePay()
{
    return (payRate * hoursWorked);
}

void partTimeEmployee::setNameRateHours(string first, string last,
double rate, double hours)
{
    personType::setName(first, last);
    payRate = rate;
    hoursWorked = hours;
}
```

```

partTimeEmployee::partTimeEmployee(string first, string last,
                                   double rate, double hours)
    : personType(first, last) //constructor
{
    payRate = rate;
    hoursWorked = hours;
}

```

ملف الفئة المستمدة:

قام القسم السابق بتوضيح كيفية اشتقاق فئات جديدة من الفئات التي يتم تعريفها من قبل. لتعريف فئات جديدة تقوم بعمل ملفات جديدة. الفئات الأساسية معرفة بالفعل وتعريفاتها موجودة في الملفات. لعمل فئات جديدة مبنية على الفئات المعرفة من قبل تحتوي ملفات الفئات الجديدة على أوامر تخبر الحاسب الآلي بالمكان الذي يبحث فيه عن تعريفات الفئات الأساسية.

افترض أن تعريفات الفئة personType موضوعة في الملف الرئيسي من personType.h. لعمل تعريف الفئة partTimeEmployee يجب أن يحتوي الملف الرئيسي

ptEmployee.h – على تعليمات ما قبل المعالجة

include "personType.h"

قبل تعريف الفئة partTimeEmployee. لكي يكون الملف الرئيسي ptEmployee.h محدد يكون:

```

//Header file ptEmployee.h
#include "personType.h"

class partTimeEmployee: public personType
{
public:
    void print() const;
    double calculatePay();
    void setNameRateHours(string first, string last,
                        double rate, double hours)
    partTimeEmployee(string first = "", string last = "",
                    double rate = 0, double hours = 0);

private:
    double payRate;
    double hoursWorked;
};

```

يمكن وضع تعريفات دالات العنصر في ملف منفصل. تذكر أنه من أجل ادخال ملف رئيسي مقدم من النظام مثل iostream في برنامج المستخدم يجب أن ترفق الملف الرئيسي بين أقواس الزاوية ولادخال ملف رئيسي معرف بواسطة المستخدم في برنامج ما يجب أن ترفق الملف الرئيسي بين علامتي تنصيص مزدوجة.

ادراجات متعددة للملف الرئيسي:

ناقش القسم التالي كيفية عمل الملف الرئيسي لفئة مستمدة. لادخال ملف رئيسي في برنامج ما تستخدم أمر ما قبل المعالجة. تذكر أنه قبل تجميع البرنامج يقوم المعالج أولاً بمعالجة البرنامج. انظر الى الملف الرئيسي التالي:

```
//Header file test.h
const int ONE = 1;
const int TWO = 2;
```

افترض أن الملف الرئيسي testA.h يحتوي على الملف test.h من أجل استخدام المعرفات ONE وTWO. لكي تكون محدداً افترض أن الملف الرئيسي testA.h يشبه ما يلي:

```
//Header file testA.h
#include "test.h"
.
.
.
```

انظر الآن الى كود البرنامج التالي:

```
//Program headerTest.cpp
#include "test.h"
#include "testA.h"
.
.
.
```

عندما يتم تجميع البرنامج headerTest.cpp تتم معالجته أولاً بواسطة المعالج السابق. المعالج الأول يتضمن الملف الرئيسي test.h ثم الملف الرئيسي testA.h. عندما يتم تضمين الملف الرئيسي testA.h لأنه يحتوي على موجه المعالج الأول # include "test.h" يتم ادخال الملف الرئيسي test.h مرتين في البرنامج. الادراج الثاني للملف الرئيسي test.h يتسبب في أخطاء تجميع الوقت مثلما تم اعلان المعرف ONE. تحدث هذه المشكلة لأن الادراج الأول للملف الرئيسي test.h قام بالفعل بتعريف المتغيرات ONE وTWO. من أجل تجنب الادراج المتعددة لملف ما في البرنامج نستخدم أوار معالجة أولية معينة في الملف الرئيسي. لنقم أولاً باعادة كتابة الملف الرئيسي test.h باستخدام أوامر المعالجة الأولية هذه ثم تفسير معنى تلك الأوامر.

```
//Header file test.h

#ifndef H_test
#define H_test
const int ONE = 1;
const int TWO = 2;
#endif
```

- `#ifndef H-test` تعني "إذا كانت غير معرفة H-test".
- `#define H-test` تعني "تعريف H-test".
- `#endif` تعني "تنتهي اذا".

تأثير هذه الأوامر كما يلي: إذا لم يكن المعرف `H_test` غير معرف قم بتعريف المعرف `H_test` واجعل البيانات المتبقية بين `#define` و `#endif` تمر خلال المجمع. إذا تم تضمين الملف الرئيسي `test.h` مرة ثانية في البرنامج فإن البيان `#ifndef` يفشل ويتم تخطي جميع البيانات حتى `#endif`. في الحقيقة يتم كتابة جميع الملفات الرئيسية باستخدام أوامر معالجة أولية مماثلة.

العناصر المحمية من الفئة:

العناصر الخاصة بالفئة عناصر خاصة بالفئة ولا يمكن تناولها مباشرة خارج الفئة فقط يمكن لدالات العناصر فقط تناول العناصر الخاصة. كما نوقش من قبل لا يمكن لفئة مستمدة أن تتناول العناصر الخاصة بالفئة الأساسية بطريقة مباشرة. بالرغم من هذا يكون من اللازم في بعض الأحيان بالنسبة الى الفئة المستمدة أن تتناول العنصر الخاص من الفئة الأساسية. إذا جعلت عنصر خاص يصبح عنصر عام اذن فان أي شخص يمكنه تناول هذا العنصر. تذكر أن عناصر الفئة مصنفة الى ثلاث فئات: عامة، وخاصة، ومحمية. لذلك فانه بالنسبة الى فئة أساسية تمنح العنصر الوصول الى فئتها المستمدة ولا تزال تمنع تناولها المباشر خارج الفئة يجب عليك اعلان أن هذا العنصر الواقع تحت محدد تناول العنصر يعتبر عنصر محمي. لهذا تكون امكانية الوصول الى عنصر محمي من الفئة بين عام وخاص. الفئة المستمدة يمكنها تناول العنصر المحمي للفئة الأساسية مباشرة.

للايجاز، إذا احتاجت فئة مستمدة الى تناول عنصر من الفئة الأساسية والى منع التناول المباشر للعنصر خارج الفئة الأساسية يتم اعلان هذا العنصر الخاص بالفئة الأساسية تحت محدد تناول العضو المحمي.

التوريث كتوريث عام أو محمي أو خاص:

افترض أن الفئة B مستمدة من الفئة A اذن لا يمكن للفئة B تناول العناصر الخاصة للفئة A بشكل مباشر. ماذا عن العناصر العامة والمحمية للفئة A؟ يقوم هذا القسم باعطاء القواعد التي يتم تطبيقها بوجه عام عند تناول عناصر فئة أساسية.

انظر الى البيان التالي:

```
class B: memberAccessSpecifier A
{
    .
    .
    .
};
```

في هذا البيان، يكون محدد تناول العنصر (memberAccessSpecifier) اما عام أو محمي أو خاص.

1. اذا كان محدد تناول العنصر عام بالتالي يكون التوريث عام.
 - أ- العناصر العامة للفئة A عبارة عن عناصر عامة للفئة B ويمكن تناولها بشكل مباشر في الفئة B.
 - ب- العناصر المحمية للفئة A عبارة عن عناصر محمية للفئة B ويمكن تناولها بشكل مباشر بواسطة دالات العنصر (الدالات الصديق) للفئة B.
 - ت- لا يمكن تناول العناصر الخاصة للفئة A بشكل مباشر في الفئة B ولكن يمكن تناولها بواسطة دالات عنصر (والدالة الصديقة) للفئة B فقط عن طريق العناصر العامة أو المحمية للفئة A.
 2. اذا كان محدد تناول العنصر محمي بالتالي يكون التوريث محمي.
 - أ- العناصر العامة للفئة A عبارة عن عناصر محمية للفئة B ويمكن تناولها بشكل مباشر بواسطة دالات عنصر (أو الدالات الصديقة) للفئة B.
 - ب- العناصر المحمية للفئة A عبارة عن عناصر محمية للفئة B ويمكن تناولها بشكل مباشر بواسطة دالات العنصر (الدالات الصديق) للفئة B.
 - ت- لا يمكن تناول العناصر الخاصة للفئة A بشكل مباشر في الفئة B ولكن يمكن تناولها بواسطة دالات عنصر (والدالة الصديقة) للفئة B فقط عن طريق العناصر العامة أو المحمية للفئة A.
 3. اذا كان محدد تناول العنصر خاص بالتالي يكون التوريث خاص.
 - أ- العناصر العامة للفئة A عبارة عن عناصر خاصة للفئة B ويمكن تناولها بشكل مباشر بواسطة دالات عنصر (أو الدالات الصديقة) للفئة B.
 - ب- العناصر المحمية للفئة A عبارة عن عناصر خاصة للفئة B ويمكن تناولها بشكل مباشر بواسطة دالات العنصر (الدالات الصديق) للفئة B.
 - ت- لا يمكن تناول العناصر الخاصة للفئة A بشكل مباشر في الفئة B ولكن يمكن تناولها بواسطة دالات عنصر (والدالة الصديقة) للفئة B فقط عن طريق العناصر العامة أو المحمية للفئة A.
- قسم "الدالات الصديقة للفئات" (الموجود لاحقاً في هذا الفصل) يصف الدالات الصديقة.

التركيب:

التركيب عبارة عن طريقة أخرى للربط بين فئتين. في التركيب يكون عنصر أو أكثر من عنصر للفئة موضوعات لنوع فئة آخر. التركيب عبارة عن علاقة "has-a" على سبيل المثال "كل شخص له تاريخ ميلاد".

المثال 1-6 في الفصل الأول قام بتعريف فئة تسمى نوع الشخص (personType) التي تقوم بتخزين اسم الشخص الأول واسمه الأخير. افترض أننا نريد تتبع معلومات اضافية للشخص مثل هويته

الشخصية (على سبيل المثال رقم التأمين الاجتماعي) وتاريخ الميلاد. بما أن كل شخص له هوية شخصية وتاريخ ميلاد فانه يمكننا تعريف فئة جديدة تسمى معلومات شخصية (personalInfo) التي يكون فيها أحد العناصر موضوع للنوع personType. يمكننا اعلان عناصر اضافية لتخزين الهوية الشخصية وتاريخ الميلاد للفئة personalInfo.

نقوم أولاً بتعريف فئة أخرى وهي dateType (نوع التاريخ) لتخزين تاريخ ميلاد الشخص ثم نقوم ببناء الفئة personalInfo (معلومات شخصية) من الفئتين personType (نوع الشخص) و dateType (نوع التاريخ). يمكننا بهذه الطريقة توضيح كيفية تعريف فئة جديدة باستخدام فئتين. من أجل تعريف الفئة dateType (نوع التاريخ) نحتاج الى ثلاث عناصر بيانات لتخزين الشهر ورقم اليوم والعام. بعض من العمليات التي يلزم أداؤها على التاريخ هي تحديد التاريخ وطباعة التاريخ. البيانات التالية تعرف الفئة dateType (انظر الشكل 2-5 كذلك):

```
class dateType
{
public:
    void setDate(int month, int day, int year);
        //Function to set the date.
        //Data members dMonth, dDay, and dYear are set
        //according to the parameters.
        //Postcondition: dMonth = month; dDay = day;
        //                    dYear = year

    void getDate(int& month, int& day, int& year);
        //Function to return the date.
        //Postcondition: month = dMonth; day = dDay;
        //                    year = dYear

    void printDate() const;
        //Function to output the date in the form
        //mm-dd-yyyy.

    dateType(int month = 1, int day = 1, int year = 1900);
        //constructor to set the date
        //Data members dMonth, dDay, and dYear are set
        //according to the parameters.
        //Postcondition: dMonth = month; dDay = day;
        //                    dYear = year
        //If no values are specified, the default values are
        //used to initialize the data members.

private:
    int dMonth;           //variable to store the month
    int dDay;             //variable to store the day
    int dYear;            //variable to store the year
};
```

الفئة dateType
 }
 عامة:

```

Void setDate (int month, int day, int year);
// دالة لتحديد التاريخ.
// يتم تحديد عناصر البيانات dMonth و dDay و dYear
// وفقاً للعوامل.
// شرط تالي: dMonth = شهر، dDay = يوم، و dYear = عام.

```

```

Void getDate (int& month, int& day, int& year);
// دالة لاستنتاج التاريخ.
// شرط تالي: dMonth = شهر، dDay = يوم، و dYear = عام.

```

```

Void printDate ( ) const;
// دالة لخراج التاريخ في الصيغة
// شهر-يوم-عام.

```

```

dateType (int month = 1, int day = 1, int year = 1900);
// مقوم لتحديد التاريخ
// وفقاً للعوامل.
// شرط تالي: dMonth = شهر، dDay = يوم، و dYear = عام
// اذا لم يتم تحديد قيم سوف يتم استخدام
// القيم الافتراضية لتهيئة عناصر البيانات.

```

خاصة:

```

Int dMonth;
Int dDay;
Int dYear;
// متغير لتخزين الشهر
// متغير لتخزين اليوم
// متغير لتخزين العام
{

```

| dateType |
|--|
| -dMonth: int -dDay: int -dYear: int |
| +setDate (int, int, int) : void +getDate (int&, int&, int&) : void +printDate () const : void +dateType (int = 1, int = 1, int = 1900) |

شكل 2-5: رسم لغة التشكيل الموحدة للفئة dateType.
تعريفات دالات عنصر الفئة dateType كما يلي:

```

void dateType::setDate(int month, int day, int year)
{
    dMonth = month;
    dDay = day;
    dYear = year;
}

```

هذا التعريف للدالة setDate قبل تخزين التاريخ في عناصر البيانات لا يتحقق مما اذا كان التاريخ صحيح أم لا وهذا يعني أنه لا يؤكد ما اذا كان الشهر بين 1 و 12 والعام أكبر من صفر واليوم صحيح (على سبيل المثال بالنسبة الى شهر يناير يجب أن يكون اليوم بين 1 و 31) في تمرين البرمجة رقم 2 في نهاية هذا الفصل يطلب منك اعادة كتابة تعريف الدالة setDate حتى يتم تصحيح التاريخ قبل تخزينه في عناصر البيانات.

تعريفات دالات العناصر المتبقية تكون كما يلي:

```
void dateType::getDate(int& month, int& day, int& year)
{
    month = dMonth;
    day = dDay;
    year = dYear;
}

void dateType::printDate() const
{
    cout<<dMonth<<"-"<<dDay<<"-"<<dYear;
}

//constructor

dateType:: dateType(int month, int day, int year)
{
    setDate(month, day, year);
}
```

لأن المقوم يستخدم الدالة setDate قبل تخزين التاريخ داخل عناصر البيانات فان المقوم كذلك لا يتأكد مما اذا كان التاريخ صحيح. في تمرين البرمجة رقم 2 في نهاية هذا الفصل عندما تعيد كتابة تعريف الدالة setDate لتصحيح التاريخ والمقوم يستخدم الدالة setDate فان التاريخ الذي يتم تحديده بواسطة المقوم سوف يتم تصحيحه كذلك.

بعد هذا نعطي تعريف الفئة personalInfoType (انظر شكل 2-6 أيضاً):

```
Class personalInfoType
{
```

عامة:

```
Void setpersonalInfo (string first, string last,
                      Int month, int day,
                      Int year, int ID);
```

// دالة لتحديد المعلومات الشخصية.

//يتم تحديد عناصر البيانات وفقاً الى

```
// العوامل.
// شرط تالي: الاسم الأول = أول، الاسم الأخير = أخير، dMonth = شهر،
// dDay = يوم، dYear = عام، personID = الهوية
Void printpersonalInfo ( ) const;
// دالة لطباعة المعلومات الشخصية.
personalInfoType (string first = " ", string last = " ",
int month = 1, int day = 1,
int year = 1900, int ID = 0);
// مقوم
// يتم تحديد عناصر البيانات وفقاً الى
// العوامل.
// شرط تالي: الاسم الأول = أول، الاسم الأخير = أخير، dMonth = شهر،
// dDay = يوم، dYear = عام، personID = الهوية
// خاصة:
personType name;
dateType bDay;
int personID;
{
```

| personalInfoType |
|--|
| -name: personType -bDay: dateType -personID: int |
| setpersonalInfo (string, string, int, int, int, int) : void printpersonalInfo () const : void personalInfoType (string = " ", string = " ", int = 1, int = 1, int = 1900, int = 0) |

شكل 2-6: رسم لغة التشكيل الموحدة للفئة personalInfoType.
قبل أن نعطي تعريف دالات عنصر الفئة personalInfoType لنقم بمناقشة كيفية تحفيز مقومات الموضوعات bDay و name.
تذكر أن مقوم الفئة يقوم أوتوماتيكياً بالتنفيذ عندما يدخل موضوع الفئة في نطاقه.

افترض أن لدينا البيان التالي:

personalInfoType student;

عندما يدخل الموضوع student (طالب) في نطاقه تدخل كذلك الموضوعات bDay و name وهي عناصر للطالب في نطاقها. كنتيجة لذلك يتم تنفيذ واحد من مقوماتهم. لهذا نحن في حاجة الى معرفة تمرير البراهين الى مقومات موضوعات العنصر (وهي bDay و name). تذكر أن المقومات ليس لها نوع ولذلك لا يمكن استدعائها مثل الدالات الأخرى. يتم تحديد براهين مقوم موضوع العنصر (مثل bDay) في الجزء الرئيسي من مقوم الفئة. البيانات التالية توضح كيفية تمرير البراهين الى مقومات موضوعات العنصر:

```
personalInfoType::personalInfoType(string first, string last,
                                     int month, int day, int year, int ID)
    : name(first,last), bDay(month,day,year)
{
    .
    .
    .
}
```

يتم بناء موضوعات عنصر الفئة بالترتيب الذي يتم اعلانها (وليس بالترتيب الذي تم ذكرها به في قائمة تهيئة عنصر المقوم) وقبل أن يتم بناء موضوعات الفئة المرفقة. لهذا في هذه الحالة يتم تهيئة الموضوع name (الاسم) أولاً ثم bDay وأخيراً student (الطلاب).

فيما يلي تعريفات دالات عنصر الفئة personalInfo:

```
void personalInfoType::setpersonalInfo(string first, string last
                                       int month, int day,
                                       int year, int ID)
{
    name.setName(first, last);
    bDay.setDate(month, day, year);
    personID = ID;
}

void personalInfoType::printpersonalInfo () const
{
    name.print();
    cout<<"'s date of birth is ";
    bDay.printDate();
    cout<<endl;
    cout<<"and personal ID is "<<personID;
}

personalInfoType::personalInfoType(string first, string last,
                                     int month, int day,
                                     int year, int ID)
    :name(first, last), bDay(month, day, year)
{
    personID = ID;
}
```

في حالة التوريث استخدم اسم الفئة لاستدعاء مقوم الفئة الأساسية. في حالة التركيب استخدم اسم موضوع العنصر لاستدعاء مقومه الخاص.

تعدد الأشكال: انقال العامل والدالة:

في الفصل الأول تعلمت كيف يتم استخدام الفئات في C++ لدمج البيانات والعمليات المؤداة على تلك البيانات في هوية واحدة. ان القدرة على دمج البيانات والعمليات المؤداة على تلك البيانات تعرف **بالتغليف**. انه المبدأ الأول للتصميم القائم على الموضوع. قام الفصل الأول بتعريف نوع البيانات المجرد ووصف كيفية تنفيذ الفئات في C++ لنوع البيانات المجرد. القسم الأول من هذا الفصل ناقش كيف يمكن اشتقاق فئات جديدة من فئات متواجدة عن طريق آلية التوريث. التوريث وهو المبدأ الثاني للتصميم القائم على الموضوع يشجع على اعادة استخدام الكود.

بقية هذا الفصل تناقش المبدأ الثالث للتصميم القائم على الموضوع وهو تعدد الأشكال. أولاً نقوم بمناقشة تعدد الأشكال عبر **انقال العامل** ثم بعد ذلك عبر **النماذج**. النماذج تمكن المبرمج من كتابة أكواد شاملة للدالات والفئات المرتبطة. سوف نقوم بتبسيط انقال الدالة عن طريق استخدام النماذج المسماة **نماذج الدالة**. النوع الثالث من تعدد الأشكال عبر الدالات الافتراضية تتم مناقشته في الملحق و.

انقال العامل:

يقوم هذا القسم بتوضيح كيفية انقال العوامل في C++ ولكن لنقم أولاً برؤية سببرغبتك في انقال العوامل.

سبب الحاجة الى انقال العامل:

قام الفصل الأول بتعريف وتطبيق الفئة `clockType` كما أوضح أنه يمكنك استخدام الفئة `clockType` لتمثيل الوقت في برنامج ما. دعونا نستعرض بعض من سمات الفئة `clockType`. انظر الى البيانات التالية:

```
clockType myClock (8, 23, 34);
```

```
clockType yourClock (4, 5, 30);
```

البيان الأول يعلن أن `myClock` موضوع للنوع `clockType` يقوم بتهيئة عناصر البيانات `hr` و `min` و `sec` ل `myClock` عند 8 و 23 و 34 على التوالي. البيان الثاني يعلن `yourClock` موضوع للنوع `clockType` يقوم بتهيئة عناصر البيانات `hr` و `min` و `sec` ل `myClock` عند 4 و 5 و 30 على التوالي.

انظر الآن الى البيانات التالية:

```
myClock.printTime();
myClock.incrementSeconds();
if(myClock.equalTime(yourClock))
{
    .
    .
    .
}
```

البيان الأول يطبع قيمة myClock في هيئة hr : min : sec (ساعات : دقائق : ثواني). والبيان الثاني يزيد قيمة myClock بمقدار ثانية واحدة والبيان الثالث يتحقق مما اذا كانت قيمة myClock هي نفسها قيمة yourClock.

تلك البيانات تقوم بوظيفتها وبالرغم من هذا اذا كان يمكننا استخدام عامل الادراج >> لايخراج قيمة myClock وعامل الزيادة ++ لزيادة قيمة myClock ثانية واحدة والعوامل ذات العلاقة من أجل المقارنة يمكننا دعم مرونة C++ بشكل معقول بشكل أكثر تحديداً بفضل استخدام البيانات التالية بدلاً من البيانات السابقة:

```
cout<<myClock;
myClock++;
if(myClock == yourClock)
{
    .
    .
    .
}
```

تذكر أن العمليات المدمجة فقط التي تتم تأديتها على الفئات هي عامل التحديد وعامل اختيار العنصر. لهذا لا يمكن تطبيق العوامل الأخرى بشكل مباشر على موضوعات الفئة. بالرغم من هذا تسمح C++ للمبرمج بمد تعريفات غالبية العوامل حتي يمكن تطبيق تلك العوامل – مثل العوامل ذات الصلة، والعوامل الحسابية، وعوامل الادراج لايخراج البيانات، وعوامل الاستخراج بالنسبة الى مدخلات البيانات – على الفئات. في مصطلحات C++ يطلق على هذا **اثقال العامل**. بالاضافة الى اثقال العامل يقوم هذا الفصل بمناقشة اثقال الدالة.

اثقال العامل:

تذكر كيف يعمل العامل الحسابي. اذا كان كل من طرفي العمليات الرياضية عدد صحيح فان النتيجة تكون عدد صحيح وبخلاف هذا تكون النتيجة عدد ذو نقطة عائمة. بالمثل يكون عامل ادخال التدفق >> وعامل استخلاص التدفق << مثقلان. يتم استخدام العامل >> ككل من عامل ادخال تدفق وعامل نقل أيسر. ويتم استخدام العامل << ككل من عامل استخلاص تدفق وعامل نقل أيمن. فيما يلي أمثلة على اثقال العامل.

الأمثلة الأخرى على العوامل المثقلة هي + و - . نتائج + و - مختلفة بالنسبة الى علم حساب الأعداد الصحيحة وعلم حساب النقطة العائمة.

تسمح C++ للمستخدم باثقال معظم العوامل حتى يمكن للعوامل العمل بفاعلية في تطبيق محدد. انها لا تسمح للمستخدم بعمل عمليات جديدة. يمكن اثقال غالبية العمليات المتواجدة للسيطرة على موضوعات الفئة.

من أجل ائصال عامل ما يجب أن تكتب تعريف دالة (أي العنوان والهيكل). اسم الدالة التي تثقل العامل عبارة عن عامل كلمة محفوظة يليه العامل لكي يتم ائصاله. على سبيل المثال اسم الدالة التي تثقل العامل = هو:

`>= Operator (عامل <=)`

دالة العامل: الدالة التي تثقل العامل.

تركيب دالات العامل:

نتيجة أي عملية عبارة عن قيمة ولهذا فان دالة العنصر عبارة عن دالة منتجة لقيمة.
تركيب عنوان دالة العامل هو:

| | | |
|------------|----------|----------------------------|
| returnType | operator | operatorSymbol (arguments) |
|------------|----------|----------------------------|

في ++C، العامل يكون عبارة عن كلمة محفوظة.

تذكر أن العمليات المدمجة الوحيدة المؤداة على الفئات هي التحديد (=) واختيار العنصر. لاستخدام عوامل أخرى على موضوعات الفئة يجب أن يتم ائصالها بشكل واضح. ان ائصال العامل يقدم نفس التعبيرات الدقيقة بالنسبة الى أنواع البيانات التي يحددها المستخدم كما يفعل بالنسبة الى أنواع البيانات المدمجة.

لائصال العامل للفئة:

1. قم بادراج البيان لاعلان أن الدالة لائصال العامل (وهذا يعني دالة العامل) في تعريف الفئة.

2. اكتب تعريف دالة العامل.

يجب اتباع قواعد معينة عند ادراجك لدالة عنصر في تعريف الفئة. يتم توضيح هذه النتائج فيما بعد في هذا الفصل وفي هذا القسم المسمى "دالات العامل كدالات مستخدم ودالات غير المستخدم".

ائصال عامل: بعض القيود:

عند ائصال عامل ما تذكر ما يلي:

- لا يمكنك تغيير أسبقية عامل.
- لا يمكن تغيير الترابط. (على سبيل المثال فان ترابط العامل الحسابي + يكون من اليسار الى اليمين ولا يمكن تغييره).
- لا يمكنك استخدام معطيات افتراضية ذات عامل مثقل.
- لا يمكنك تغيير عدد المعطيات التي يتخذها العامل.
- لا يمكنك عمل عوامل جديدة ويكن ائصال العوامل المتواجدة فقط. العوامل التي لا يمكن ائصالها

هي:

sizeof ?::.*.

- معنى كيفية عمل العامل ذو الأنواع المدمجة مثل int يبقى كما هو.
- يمكن اثقال العوامل اما بالنسبة الى أهداف من النوع الذي يحدده المستخدم أو بالنسبة الى اندماج من أهداف نوع يحدده المستخدم وأهداف من النوع المدمج.

المؤشر this:

يمكن لدالة مستخدم الفئة أن تتناول عنصر البيانات لهذه الفئة بشكل مباشر بالنسبة الى هدف معطى. في بعض الأحيان يكون من الضروري لمستخدم الدالة أن يشير الى الهدف ككل بدلاً من عناصر البيانات الفردية للهدف. كيف تشير الى الهدف ككل (أي كوحدة واحدة) في تعريف دالة المستخدم خاصة لا يتم تمرير الهدف كمعامل؟ كل هدف للفئة يحتفظ بمؤشر (خفي) لنفسه ويكون اسم هذا المؤشر هو this. في C++، تكون this كلمة محفوظة وهو مؤشر متاح لك لكي تستخدمه. عندما يستدعي الهدف دالة المستخدم تقوم دالة المستخدم بحالة المؤشر this للهدف. على سبيل المثال افترض أن هناك فئة تسمى test ولها دالة مستخدم تسمى funcOne وافترض كذلك أن تعريف الدالة funcOne يبدو كما يلي:

```
test test::funcOne()  
{  
    .  
    .  
    .  
    return *this;  
}
```

إذا كانت x و y هدفان من النوع test فان البيان التالي:

Y = x.funcOne ();

ينسخ قيمة الهدف x في الهدف y وهذا يعني أنه يتم نسخ عناصر المعطيات x في عناصر المعطيات الموافقة لها من y. عندما يقوم الهدف x باستدعاء الدالة funcOne يكون المؤشر this الموجود في تعريف دالة المستخدم funcOne يشير الى الهدف x ولهذا فان this تعني عنوان x و *this تعني قيمة x.

المثال التالي يوضح كيفية عمل المؤشر this:

مثال 2-3:

في مثال 1-6 (في الفصل الأول) قمنا بتصميم فئة ما لتطبيق اسم الشخص في برنامج ما وهنا نقوم بالتوسع في تعريف الدالة personType لتحديد الاسم الأول والاسم الأخير للشخص بشكل فردي ثم استعادة الهدف بأكمله. التعريف الموسع للفئة personType هو:

الفئة personType
}

عامة:

Void print () const;

// دالة لايخرج الاسم الأول والاسم الأخير.

// شرط تالي: تتم طباعة الاسم في صيغة

// الاسم الأول الاسم الأخير

Void setName (string first, string last);

// دالة لتحديد الاسم الأول والاسم الأخير وفقاً

// الى العوامل.

// شرط تالي: الاسم الأول = first والاسم الأخير = last.

personType& setFirstName (string first);

// دالة لتحديد الاسم الأول.

// شرط تالي: الاسم الأول = أول

// بعد تحديد الاسم الأول يتم استعادة اشارة

// الى الهدف وهو عنوان الهدف.

personType& setLastName (string first);

// دالة لتحديد الاسم الأخير.

// شرط تالي: الاسم الأخير = أخير

// بعد تحديد الاسم الأخير يتم استعادة اشارة

// الى الهدف وهو عنوان الهدف.

Void getName (string first, string& last);

// دالة لاسترجاع الاسم الأول والاسم الأخير عبر

// العوامل.

// شرط تالي: first = الاسم الأول و last = الاسم الأخير.

personType (string first = " ", string last = " ");

// مقوم
 //يحدد الاسم الأول والاسم الأخير وفقاً الى
 // العوامل.
 // القيم الافتراضية للعوامل خيوط خالية.
 // شرط تالي: الاسم الأول = first والاسم الأخير = last.

خاصة:

String firstName; // يقوم بتخزين الاسم الأول.
 String lastName; // يقوم بتخزين الاسم الأخير.
 {

ان تعريف الدالات print و setTime و getName والمقوم هم نفس ما سبق (انظر مثال 1-6).
 وتعريفات الدالتين setName و setLastName كما يلي:

```
personType& personType::setLastName(string last)
{
    lastName = last;

    return *this;
}

personType& personType::setFirstName(string first)
{
    firstName = first;

    return *this;
}
```

Consider the following function main:

```
int main()
{
    personType student1("Angela", "Clodfelter");           //Line 1

    personType student2;                                     //Line 2

    personType student3;                                     //Line 3

    cout<<"Line 4 -- Student 1: ";                          //Line 4
    student1.print();                                        //Line 5
    cout<<endl;                                              //Line 6

    student2.setFirstName("Shelly").setLastName("Malik"); //Line 7
}
```

```

cout<<"Line 8 -- Student 2: ";           //Line 8
student2.print();                         //Line 9
cout<<endl;                              //Line 10

student3.setFirstName("Chelsea");        //Line 11

cout<<"Line 12 -- Student 3: ";           //Line 12
student3.print();                         //Line 13
cout<<endl;                              //Line 14

student3.setLastName("Tomek");            //Line 15

cout<<"Line 16 -- Student 3: ";           //Line 16
student3.print();                         //Line 17
cout<<endl;                              //Line 18

return 0;
}

```

المخرجات:

الصف 4 -- الطالب 1: أنجيلا كلودفلتر

الصف 8 -- الطالب 2: شيلي ماليك

الصف 12 -- الطالب 3: تشيلسي

الصف 16 -- الطالب 3: تشيلسي توميك

البيانات في الصفوف 1 و 2 و 3 تعلن وتتهيئ الموضوعات الطالب 1، والطالب 2، والطالب 3 على التوالي. يتم تهيئة الهدفين طالب 2 وطالب 3 لافراغ Strings. البيان في الصف رقم 5 يخرج قيمة طالب 1) انظر الصف 4 في المخرجات الذي يحتوي على مخرجات الصفوف 4 و 5 و 6). يعمل البيان في الصف رقم 7 كما يلي. في البيان:

Student 2.setFirstName ("Shelly"). setLastName ("Malik");

يتم أولاً تنفيذ التعبير:

Student 2.setFirstName ("Shelly")

لأن ارتباط عامل النقطة **dot point** يكون من اليسار الى اليمين. هذا التعبير يحدد الاسم الأول "Shelly" ويعيد اشارة الى الهدف وهو الطالب 2.

لهذا يكون التعبير التالي الذي يتم تنفيذه هو:

Student2.setLastName ("Malik")

الذي يحدد الاسم الأخير للطالب 2 "Malik". البيان في الصف رقم 9 يخرج قيمة الطالب 2 والبيان في الصف رقم 11 يحدد الاسم الأول للهدف الطالب 3 "Chelsea" والبيان في الصف رقم 13 يخرج قيمة الطالب 3. لاحظ المخرجات في الصف 12 حيث أنها توضح الاسم الأول فقط وليس الاسم الأخير لأننا لم نقوم بعد بتحديد الاسم الأخير للطالب 3. لا يزال الاسم الأخير للطالب 3 فارغاً وقد تم

تحديده بواسطة البيان في الصف 3 عندما تم اعلان الطالب 3. بعد هذا يقوم البيان في الصف 15 بتحديد الاسم الأخير للطالب 3 ويقوم البيان في الصف 16 باخراج قيمة الطالب 3.

الدالات الصديقة للفئات:

الدالة التي يتم تعريفها خارج نطاق الفئة تسمى دالة صديقة للفئة. الدالة الصديقة عبارة عن دالة لغير المستخدم للفئة ولكننا نتناول عناصر المعطيات الخاصة للفئة. لجعل الدالة صديقة للفئة تسبق الكلمة المحفوظة friend نموذج الدالة (في تعريف الفئة). (الكلمة friend تظهر فقط في نموذج الدالة في تعريف الفئة وليس في تعريف الدالة الصديقة).
انظر الى البيانات التالية:

```
class classIllusFriend
{
    friend void friendFunc(...);
    .
    .
    .
};
```

في تعريف الفئة classIllusFriend يتم الاعلان عن friendFunc كدالة صديقة للفئة classIllusFriend أي أنها غير لغير المستخدم للفئة classIllusFriend. عندما تقوم بكتابة تعريف الدالة friendFunc فان أي هدف من النوع classIllusFriend الذي يكون اما متغير محلي للدالة friendFunc أم معامل رسمي للدالة friendFunc يمكنه تناول عناصرها الخاصة ضمن تعريف الدالة friendFunc. (المثال يوضح 2-4 يوضح هذا المفهوم). بالاضافة الى هذا فان كون الدالة الصديقة ليست عنصر للفئة يجعل من الممكن وضع اعلانها ضمن الجزء الخاص أو المحمي أو العام من الفئة.

تعريف دالة صديقة:

عند كتابة تعريف دالة صديقة لا يسبق اسم الفئة وعامل عزم المجال اسم الدالة الصديقة في عنوان الدالة. تذكر أيضاً أن الكلمة friend لا تظهر في عنوان تعريف الدالة الصديقة ولهذا فان تعريف الدالة friendFunc في الفئة السابقة classIllusFriend يكون:

```
void friendFunc(...)
{
    .
    .
    .
}
```

سوف نقوم بالطبع بوضع تعريف الدالة الصديقة في ملف التطبيق. القسم التالي يوضح الفرق بين دالة المستخدم ودالة غير المستخدم (الدالة الصديقة) عندما ننقل بعض من العوامل لفئة معينة. المثال 4-2 يوضح كيفية تناول الدالة الصديقة للعناصر الخاصة للفئة.

مثال 4-2:

انظر الى الفئة التالية:

```
class classIllusFriend
{
    friend void friendFunc(classIllusFriend cIFObject);

public:
    void print();
    void setx(int a);

private:
    int x;
};
```

في تعريف الدالة classIllusFriend يتم الاعلان عن الدالة friendFunc كدالة صديقة. افترض أن تعريفات دالات المستخدم للفئة classIllusFriend كما يلي:

```
void classIllusFriend::print()
{
    cout<<"In class classIllusFriend: x = "<<x<<endl;
}

void classIllusFriend::setx(int a)
{
    x = a;
}
```

الآن انظر الى التعريف التالي للدالة friendFunc:

```
#include <iostream>
#include "classIllusFriend.h"

using namespace std;

void classIllusFriend::print()
{
    cout<<"In class classIllusFriend: x = "<<x<<endl;
}

void classIllusFriend::setx(int a)
{
    x = a;
}

void friendFunc(classIllusFriend cIFObject) //Line 1
{
    classIllusFriend localObject;           //Line 2
```

المخر

~~*~*~* اختبار الدالة الصديقة *~*~*~*~*

الصف 6: في الدالة الصديقة التي تتناول عنصر البيانات الخاص 45 x

الصف 10: في الدالة friendFunc التي تتناول عنصر البيانات الخاص x 88

دالات العنصر كدالات مستخدم ودالات غير المستخدم:

غالبية دالات العامل اما أن تكون دالات مستخدم أو دالات غير مستخدم – أي دالة صديقة للفئة. من أجل جعل دالة العامل دالة مستخدم أو دالة غير مستخدم للفئة تذكر ما يلي:

- يجب الاعلان عن الدالة التي تتقبل أي من العوامل () أو [] أو - أو = لفئة ما كعنصر للفئة.
- افترض أن العامل op مثقل للفئة لنقل مثلاً opOverClass.
- إذا كان المعامل في أقصى يسار op هدف من نوع مختلف (أي ليس من النوع opOverClass) فإن الدالة التي تتقبل المعامل op للفئة opOverClass يجب أن تكون دالة غير مستخدم – أي صديقة للفئة opOverClass.
- إذا كانت دالة العامل التي تتقبل المعامل op للفئة opOverClass عنصر من الفئة opOverClass اذن عند تطبيق op على أهداف النوع opOverClass يجب أن يكون المعامل الواقع على أقصى يسار op من النوع opOverClass.

يجب أن تتبع هذه القواعد عند ادخال دالة عامل في تعريف فئة ما. سوف ترى لاحقاً في هذا الفصل أن الدالات التي تثقل عامل الادخال << وعامل الاستخلاص >> لفئة ما يجب أن تكون دالات لغير المستخدم – أي دالات صديقة للفئة. باستثناء عوامل محددة مذكورة سابقاً يمكن اثقال العوامل اما كدالات مستخدم أو كدالات غير مستخدم. الجزء التالي يوضح الفرق بين هذين النوعين من الدالات. لتسهيل هذه المناقشة نقوم باستخدام الفئة التالية لتوضيح اثقال المعامل:

```
class opOverClass
{
    .
    .
    .
private:
    int a;
    int b;
};
```

الفئة opOverClass لها عنصري بيانات خاصين وهما a و b من النوع int. اننا نقوم باضافة دالات المعامل للفئة opOverClass مع اثقالنا للمعاملات. افترض كذلك أن لديك البيانات التالية:

```
opOverClass x;
opOverClass y;
opOverClass z;
```

هذه البيانات تعلن أن x و y و z أهداف من النوع opOverClass.

اثقال معاملات ثنائية:

تتكون C++ من معاملات أحادية وثنائية كما يوجد بها معاملات ثلاثية لا يمكن اثقالها. يقوم هذا القسم والأقسام القليلة التالية بمناقشة كيفية اثقال معاملات ثنائية وأحادية متنوعة ونبدأ بوصف كيفية اثقال المعاملات الثنائية.

افترض أن # تمثل معامل ثنائي (حسابي أو متعلق) سوف يتم اثقاله للفئة opOverClass. يمكن اثقال هذا المعامل اما كدالة مستخدم للفئة أو كدالة صديقة. سوف نقوم بوصف كلتا الطريقتين لاثقال هذا المعامل.

اثقال معاملات ثنائية (حسابية أو متعلقة) كدالات مستخدم:

افترض أن # مثقل كدالة مستخدم للفئة opOverClass. اسم الدالة التي تثقل # للفئة opOverClass يكون:

Operator #

بما أن x و y أهداف من النوع opOverClass يمكنك أداء العملية

x # y

يقوم المصرف بترجمة هذا التعبير الى التعبير التالي:

x.operator # (y)

في هذا التعبير يمكنك رؤية أن الدالة operator# لها معطى واحد فقط وهو y.

بما أن operator# عنصر للدالة opOverClass و x هدف من النوع opOverClass في المثال السابق فان operator# له تناول مباشر الى العناصر الخاصة من الهدف x. لهذا يكون المعطى الأول لل operator# هدف يقوم باستدعاء الدالة operator# ويتم تمرير المعطى الثاني كمعامل لهذه الدالة.

التركيب العام لاثقال المعاملات الثنائية (حسابية أم متعلقة) كدالات مستخدم:

هذا القسم يصف الصيغة العامة للدالات لاثقال معاملات ثنائية كدالات مستخدم لفئة ما.

نموذج الدالة (التي يتم ادخالها في تعريف الفئة):

```
returnType operator op(const className&) const;
```

حيث op تمثل المعامل الثنائي سواء كان حسابي أو متعلق الذي يتم اثقاله و returnType هو نوع القيمة التي تنتجها الدالة و className هو اسم الفئة الذي يتم اثقال المعامل من أجلها.

تعريف الدالة:

```
returnType className::operator op
                        (const className& otherObject) const
{
    //algorithm to perform the operation

    return (value);
}
```

مثال 2-5:

لنقم باثقال + و == للفئة opOverClass. يتم اثقال هذه العمليات كدالات مستخدم.

```
class opOverClass
{
public:
    void print() const;

    opOverClass operator+(const opOverClass&) const;
```

```

        //Overloads the operator +

        bool operator==(const opOverClass&) const;
        //Overloads the operator ==

        opOverClass(int i = 0, int j = 0);

    private:
        int a;
        int b;
};

```

تعريفات الدالات print و operator+ و operator== والمقومات هي:

```

void opOverClass::print() const
{
    cout<<"("<<a<<", "<<b<<")";
}

opOverClass::opOverClass(int i, int j)
{
    a = i;
    b = j;
}

opOverClass opOverClass::operator+
    (const opOverClass& right) const
{
    opOverClass temp;

    temp.a = a + right.a;
    temp.b = b + right.b;

    return temp;
}

bool opOverClass::operator==(const opOverClass& right) const
{
    return(a == right.a && b == right.b);
}

```

انقال معاملات ثنائية (حسابية أو متعلقة) كدالات لغير المستخدم:

افترض أن # تمثل المعامل الثنائي (حسابي أم متعلق) الذي يتم انقاله كدالة لغير المستخدم للفئة

.opOverClass

افترض كذلك أنه سوف يتم أداء العملية التالية:

x # y

في هذه الحالة يتم تجميع التعبير كما يلي:

Operator # (x, y)

نرى هنا أن الدالة operator# لها معطيان ومن الواضح كذلك في هذا التعبير أن الدالة operator# ليست عنصر للهدف x وليست عنصر للهدف y. سوف يتم تمرير الأهداف التي تتم اضافتها كمعطيات للدالة operator#.

من أجل ادخال دالة المعامل operator# كغير عضو للفئة في تعريف الفئة فيجب أن تظهر الكلمة المحفوظة friend قبل عنوان الدالة. الدالة operator# يجب كذلك أن يكون لها معطيان.

التركيب العام لاثقال المعاملات الثنائية (حسابية أم متعلقة) كدالات لغير المستخدم:

هذا القسم يصف الصيغة العامة للدالات لاثقال معاملات ثنائية كدالات لغير المستخدم لفئة ما. نموذج الدالة (التي يتم ادخالها في تعريف الفئة):

```
friend returnType operator op(const className&,  
                             const className&);
```

حيث op تمثل المعامل الثنائي الذي يتم اثقاله و returnType هو نوع القيمة التي تنتجها الدالة و className هو اسم الفئة الذي يتم اثقال المعامل من أجلها. تعريف الدالة:

```
returnType operator op(const className& firstObject,  
                      const className& secondObject)  
{  
    //algorithm to perform the operation  
    return (value);  
}
```

معاملات ادخال تدفق الاثقال (<<) والاستخلاص (>>):

دالة المعامل التي تنقل معامل الادخال << أو معامل الاستخلاص >> للفئة يجب أن يكون غير عضو لهذه الفئة للسبب التالي:

انظر الى التعبير التالي:

Cout<<x;

في هذا التعبير يكون المعامل الواقع على أقصى اليسار من << (أي cout) هدف من النوع ostream وليس هدف من النوع opOverClass. لأن المعامل الواقع على أقصى اليسار << ليس هدف من النوع opOverClass فان دالة المعامل التي تنقل عامل الادخال للفئة opOverClass يجب ألا تكون عضو للفئة opOverClass.

بالمثل يجب ألا تكون دالة المعامل التي تثقل معامل استخلاص التدفق للفئة opOverClass دالة غير عضو للفئة opOverClass.

اثقال معامل ادخال التدفق (<<):

هذا القسم يصف التركيب العام لاثقال عامل ادخال التدفق << لفئة ما.

نموذج الدالة (التي يتم ادخالها في تعريف الفئة):

Friend ostream& operator<< (ostream&, const className&);

تعريف الدالة:

```
ostream& operator<<(ostream& osObject, const className& object)
{
    //local declaration if any
    //Output the members of the object
    //osObject<<. . .

    //Return the ostream object
    return osObject;
}
```

في تعريف الدالة السابقة:

- كلا المعاملان عبارة عن معاملات مرجعية.
- المعامل الأول وهو osObject عبارة عن اشارة الى الهدف ostream.
- عادةً يكون المعامل الثاني اشارة const الى فئة معينة لأن (تذكر الفصل 1) الطريقة الأكثر فاعلية لتمرير هدف ما كمعامل للفئة هي الاشارة. في هذه الحالة لا يحتاج المعامل الرسمي الى نسخ عناصر بيانات العامل الحقيقي. في التعريف السابق تظهر كلمة "const" قبل اسم الفئة لأننا نريد فقط أن نطبع عناصر بيانات الهدف. وهذا يعني أن الدالة يجب ألا تقوم بتعديل عناصر بيانات الهدف.

- نوع انتاج الدالة عبارة عن مرجعية الى الهدف ostream.

لاحظ أن نوع انتاج الدالة التي تثقل معامل ادخال التدفق << عبارة عن مرجعية ولهذا يمكن تنفيذ بيانات مثل التالية:

Cout<<x<<y;

وهذا يعني أنه يمكن استخدام معامل ادخال التدفق في صيغة متعاقبة.

اثقال معامل استخلاص التدفق (>>):

هذا القسم يصف التركيب العام لاثقال عامل استخلاص التدفق >> لفئة ما.

نموذج الدالة (التي يتم ادخالها في تعريف الفئة):

Friend istream& operator>> (istream&, className&);

تعريف الدالة:

```
istream& operator>>(istream& isObject, className& object)
{
    //local declaration if any
    //Read the data into the object
    //isObject>>. . .

    //Return the istream object
    return isObject;
}
```

في تعريف الدالة السابقة:

- كلا المعاملان عبارة عن معاملات مرجعية.
- المعامل الأول وهو isObject عبارة عن اشارة الى الهدف istream.
- عادةً يكون المعامل الثاني اشارة الى فئة معينة والبيانات التي تتم قراءتها سوف يتم تخزينها في الهدف.
- نوع انتاج الدالة عبارة عن مرجعية الى الهدف istream.

لاحظ أن نوع انتاج الدالة التي تتقل معامل ادخال التدفق >> عبارة عن مرجعية ولهذا يمكن تنفيذ بيانات مثل التالية:

```
Cin>>x>>y;
```

وهذا يعني أنه يمكن استخدام معامل استخلاص التدفق في صيغة متعاقبة.

مثال 2-6:

انظر الى التعريف التالي للفئة opOverClass والتعريفات التالية لدالات المعامل:

```
class opOverClass
{
    //Overload the stream insertion and
    //extraction operators.
    friend ostream& operator<<(ostream&, const opOverClass&);
    friend istream& operator>>(istream&, opOverClass&);

public:
    //Overload + and ==
    opOverClass operator+(const opOverClass&) const;
    bool operator==(const opOverClass&) const;

    opOverClass(int i = 0, int j = 0);
}
```

```

private:
    int a;
    int b;
};
    //The definitions of the functions operator+,
    //operator==, and the constructor are the same
    //as in Example 2-5.

ostream& operator<<(ostream& osObject, const opOverClass& right)
{
    osObject<<("<<right.a<<", "<<right.b<<");

    return osObject;
}

istream& operator>>(istream& isObject, opOverClass& right)
{
    isObject>>right.a>>right.b;

    return isObject;
}

```

انظر الى الدالة التالية main:

```

int main()
{
    opOverClass u(23, 45);                //Line 1
    opOverClass v;                        //Line 2

    cout<<"Line 3: u : "<<u<<endl;        //Line 3

    cout<<"Line 4: Enter two integers: ";  //Line 4
    cin>>v;                               //Line 5
    cout<<endl;                           //Line 6
    cout<<"Line 7: v : "<<v<<endl;        //Line 7

    cout<<"Line 8: u + v : "<<u + v<<endl; //Line 8

    return 0;
}

```

تنفيذ العينة: في تنفيذ العينة هذا يتم تضليل مدخلات المستخدم.

الصف 3: u = (23 و 45)

الصف 4: ادخل عددين صحيحان: 5 و 6

الصف 7: v = (5 و 6)

الصف 8: v + u = (28 و 51)

تقوم البيانات في الصفوف 1 و 2 باعلان وتهيئة u و v لكي تكون أهداف من النوع opOverClass.

البيان في الصف 3 يخرج قيمة u باستخدام cout وعامل الادخال.

لبيّن في الصف رقم 5 يدخل البيانات في v باستخدام cin وعامل الاستخلاص. البيان في الصف رقم 7 يخرج قيمة v باستخدام cout وعامل الإخراج. يقوم بيان cout في الصف رقم 8 بإضافة u و v وإخراج الناتج. المخرجات تظهر أن كل من معاملي إدخال التدفق وإخراج التدفق تم إثقالتها بنجاح.

إثقال معاملات أحادية:

عملية إثقال المعاملات الأحادية مماثلة لعملية إثقال المعاملات الثنائية والفرق الوحيد هو أنه في حالة المعامل الأحادي يكون للمعامل عامل واحد فقط وفي حالة المعامل الثنائي يكون للمعامل عاملين. لهذا من أجل إثقال معامل أحادي لفئة ما:

- إذا كانت دالة المعامل عنصر من الفئة لا يكون لها معاملات.
- إذا لم تكن دالة المعامل عنصر أي دالة صديقة للفئة يكون لها معامل واحد.

إثقال المعامل: العنصر في مقابل غير العنصر:

قامت الأقسام السابقة بمناقشة وتوضيح كيفية إثقال المعاملات. هناك معاملات معينة يجب إثقالتها كدالات عناصر للفئة والبعض الآخر يجب إثقاله كدالات لغير العناصر (صديقة). ماذا عن المعاملات التي يمكن إثقالتها إما طدالات عناصر أو دالات غير عناصر؟ على سبيل المثال يمكن إثقال المعامل الحسابي الثنائي + كدالة عنصر أو كدالة لغير العنصر. إذا قمت بإثقال + كدالة عنصر يكون للمعامل + تناول مباشر لعناصر بيانات واحد من الأهداف وتحتاج إلى تمرير هدف واحد فقط كمعامل. على الجهة الأخرى إذا قمت بإثقال + كدالة غير عنصر يجب أن تقوم بتمرير كلا الهدفين كمعاملات. لهذا قد يحتاج إثقال + كغير عنصر ذاكرة إضافية وزمن إضافي من الحاسب الآلي لعمل نسخة محلية من البيانات. لهذا ولأغراض الكفاءة يجب أن تقوم متى أمكنك بإثقال العوامل كدالات عنصر.

مثال برمجة: أعداد مركبة:

العدد المكون من الصيغة $a + ib$ ، a و b أعداد حقيقية يسمى عدد مركب. اننا نطلق على a الجزء الحقيقي وعلى b الجزء التخيلي من $a + ib$. يمكن كذلك تمثيل الأعداد المركبة كأزواج مرتبة (a, b) . يتم تعريف جمع وضرب الأعداد المركبة بواسطة القواعد التالية:

$$(a + ib) + (c + id) = (a + c) + i(b + d)$$

$$(a + ib) + (c + id) = (ac - bd) + i(ad + bc)$$

باستخدام الزوج المرتب يتم كتابة هذه القواعد كما يلي:

$$(a, b) + (c, d) = ((a + c), (b + d))$$

$$(a, b) * (c, d) = ((ac - bd), (ad + bc))$$

في هذا المثال نقوم بتكوين نوع بيانات يسمى `complexNumber` يمكن استخدامه لمعالجة الأعداد المركبة. اننا نقوم باثقال عوامل ادخال التدفق واستخلاص التدفق من أجل مدخلات ومخرجات سهلة. كما نقوم باثقال المعاملات $+$ و $*$ لأداء جمع وضرب الأعداد المركبة. اذا كان x و y أعداد مركبة يمكننا تقييم تقييمات مثل $x + y$ و $x * y$. ان اثقال المعاملات $-$ و $/$ متروك لك كتمرين: انظر تمرين البرمجة رقم 12 في نهاية هذا الفصل.

انظر الى التعريف التالي للدالة `complexType` (نظر أيضاً شكل 2-7):

```
//Specification file complexType.h
#ifndef H_complexNumber
#define H_complexNumber

#include <iostream>
using namespace std;

class complexType
{
    //Overload the stream insertion and extraction operators.
    friend ostream& operator<< (ostream&, const complexType&);
    friend istream& operator>> (istream&, complexType&);

public:
    void setComplex(const double& real, const double& imag);
    //Function to set the complex numbers according to
    //the parameters.
    //Postcondition: realPart = real; imaginaryPart = imag

    complexType(double real = 0, double imag = 0);
    //constructor
    //Initializes the complex numbers according to
    //the parameters.
    //Postcondition: realPart = real; imaginaryPart = imag

    complexType operator+(const complexType& otherComplex) const;
    //Overload the operator +

    complexType operator*(const complexType& otherComplex) const;
    //Overload the operator *

    bool operator==(const complexType& otherComplex) const;
    //Overload the operator ==

private:
    double realPart;        //variable to store the real part
    double imaginaryPart;   //variable to store the imaginary part
};
#endif
```

| complexType |
|---|
| -realPart : double -imaginaryPart : double |
| +operator<<(ostream&, const complexType&) : ostream& +operator>>(istream&, complexType&) : istream7 +setComplex (const double&, const double&) : void +complexType (double = 0, double = 0) +operator+ (const complexType&) const : complexType +operator* (const complexType&) const : complexType +operator== (const complexType&) const : bool |

شكل 2-7: رسم لغة التشكيل الموحدة للفئة complexType
نقوم الآن بكتابة تعريفات الدالات لتطبيق عمليات متنوعة للفئة complexType.
ان تعريفات غالبية الدالات تكون شديدة البساطة والوضوح ونحن نقوم فقط بمناقشة تعريفات الدالات
لاثقال معامل ادخال التدفق << ومعامل استخلاص التدفق >>.
لاخراج عدد مركب في الصيغة
(a, b)

حيث a هو الجزء الحقيقي و b هو الجزء التخيلي فان الحلول الحسابية تكون:

- أ. اخراج قوس أيسر).
- ب. اخراج الجزء الحقيقي.
- ت. اخراج الفاصلة.
- ث. اخراج الجزء التخيلي.
- ج. اخراج القوس الأيمن).

لهذا يكون تعريف معامل الدالة << هو:

```
ostream& operator<<(ostream& osObject, const complexType& complex)
{
    osObject<<"(";           //Step a
    osObject<<complex.realPart; //Step b

    osObject<<", ";          //Step c
    osObject<<complex.imaginaryPart; //Step d
    osObject<<")";           //Step e

    return osObject;
}
```

بعد هذا نناقش تعريف الدالة لاثقال معامل استخلاص التدفق >>.

المدخلات صيغتها كما يلي:

(3, 5)

في هذه المدخلات يكون الجزء الحقيقي من العدد المركب هو 3 والجزء التخيلي هو 5. الحل الحسابي لقراءة العدد المركب هو:

أ. قراءة وطرح القوس الأيسر.

ب. قراءة وتخزين الجزء الحقيقي.

ت. قراءة وطرح الفاصلة.

ث. قراءة وتخزين الجزء التخيلي.

ج. قراءة وطرح القوس الأيمن.

باتباع هذه الخطوات يكون تعريف معامل الدالة >> هو:

```
istream& operator>> (istream& isObject, complexType& complex)
{
    char ch;

    isObject>>ch;                //Step a
    isObject>>complex.realPart;   //Step b
    isObject>>ch;                //Step c
    isObject>>complex.imaginaryPart; //Step d
    isObject>>ch;                //Step e

    return isObject;
}
```

تعريفات الدالات الأخرى تكون كما يلي:

```
bool complexType::operator==(const complexType& otherComplex) const
{
    return(realPart == otherComplex.realPart &&
           imaginaryPart == otherComplex.imaginaryPart);
}

void complexType::setComplex(const double& real,
                             const double& imag)
{
    realPart = real;
    imaginaryPart = imag;
}

//constructor
complexType::complexType(double real, double imag)
{
    setComplex(real, imag);
}

//Overload the operator +
complexType complexType::operator+
(const complexType& otherComplex) const
{
    complexType temp;

    temp.realPart = realPart + otherComplex.realPart;
    temp.imaginaryPart = imaginaryPart
        + otherComplex.imaginaryPart;

    return temp;
}

//Overload the operator *
complexType complexType::operator*
(const complexType& otherComplex) const
{
    complexType temp;

    temp.realPart = (realPart * otherComplex.realPart) -
        (imaginaryPart * otherComplex.imaginaryPart);
    temp.imaginaryPart = (realPart * otherComplex.imaginaryPart)
        + (imaginaryPart * otherComplex.realPart);
    return temp;
}
```

البرنامج التالي يوضح استخدام الفئة complexType:

```
//Program that uses the class complexType

#include <iostream>
#include "complexType.h"

using namespace std;

int main()
{
    complexType num1(23,34);           //Line 1
    complexType num2;                  //Line 2
    complexType num3;                  //Line 3

    cout<<"Line 4: Num1 = "<<num1<<endl; //Line 4
    cout<<"Line 5: Num2 = "<<num2<<endl; //Line 5

    cout<<"Line 6: Enter the complex number "
        <<"in the form (a,b) ";          //Line 6
    cin>>num2;                          //Line 7
    cout<<endl;                          //Line 8

    cout<<"Line 9: New value of num2 = "
        <<num2<<endl;                    //Line 9

    num3 = num1 + num2;                 //Line 10

    cout<<"Line 11: Num3 = "<<num3<<endl; //Line 11
    cout<<"Line 12: "<<num1<<" + "<<num2
        <<" = "<<num1 + num2<<endl;      //Line 12
    cout<<"Line 13: "<<num1<<" * "<<num2
        <<" = "<<num1 * num2<<endl;      //Line 13

    return 0;
}
```

تنفيذ العينة: في تنفيذ العينة هذا يتم تضليل مدخلات المستخدم.

الصف 4: Num1 = (23 و 34)

الصف 5: Num2 = (0 و 0)

الصف 6: ادخل العدد المركب في الصيغة (a, b) (3, 4)

الصف 9: قيمة جديدة ل num2 = (3, 4)

الصف 11: Num3 = (26, 38)

الصف 12: (26, 38) = (3, 4) + (23, 34)

الصف 13: (-67, 194) = (3, 4) * (23, 34)

اثقال الدالة:

القسم السابق ناقش اثقال المعامل. ان اثقال المعامل يزود المبرمج بنفس المدون الدقيق لأنواع البيانات التي يحددها المستخدم مثل المدون الذي يمتلكه المعامل مع الأنواع المدمجة. ان أنواع المعطيات المستخدمة مع معامل ما تحدد التصرف الواجب اتخاذه.

بالتشابه مع اثقال المعامل تسمح C++ للمبرمج باثقال اسم الدالة. تذكر أن الفئة قد يكون لديها أكثر من مقوم واحد ولكن جميع مقومات الفئة لها نفس الاسم وهو اسم الفئة وهذه الحالة مثال على اثقال الدالة. يشير اثقال الدالة الى خلق عدة دالات بنفس الاسم. ومع هذا اذا كان هناك عدة دالات لها نفس الاسم فيجب أن يكون لكل دالة قائمة عوامل رسمية مختلفة، أنواع العوامل تحدد نوع الدالة التي يتم تنفيذها. افترض أنك في حاجة الى كتابة دالة ما تقوم بتحديد العنصر الأكبر من بين عنصرين. قد يكون العنصران عددين صحيحين أو أعداد ذات فاصلة عائمة أو رموز أو مقطعين. يمكنك كتابة العديد من الدالات كما يلي:

```
int largerInt(int x, int y);
char largerChar(char first, char second);
double largerDouble(double u, double v);
string largerString(string first, string second);
```

تقوم دالة largerInt بتحديد العدد الأكبر من بين عددين صحيحين والدالة largerChar تحدد الرمز الأكبر من بين رمزين وهكذا. تقوم جميع هذه العمليات بأداء العمليات ذاتها. بدلاً من اعطاء أسماء مختلفة لتلك الدالات يمكنك استخدام نفس الاسم – على سبيل المثال larger – لكل دالة أي أنه اذا أمكنك اثقال الدالة larger يمكنك اذن كتابة نماذج الدالة السابقة ببساطة كما يلي:

```
int larger(int x, int y);
char larger(char first, char second);
double larger(double u, double v);
string larger(string first, string second);
```

على سبيل المثال اذا كان الاستدعاء (5, 3) larger يتم تنفيذ الدالة الأولى واذا كان الاستدعاء larger ('A', '9') فانه يتم تنفيذ الدالة الثانية وهكذا.

بالنسبة الى جعل اثقال الدالة يعمل يجب أن نعطي تعريف كل دالة والأقسام التالية تعلمنا كيفية اثقال الدالات بجزء أحادي الكود وترك مهمة توليد الكود لدالت منفصلة الى المصنف.

القوالب:

القوالب عبارة عن خصائص شديدة القوة من C++ وباستخدام القوالب يمكنك كتابة جزء أحادي الكود لمجموعة من الدالات المترابطة يطلق عليه **قالب الدالة** وبالنسبة الى الفئات المترابطة يطلق عليه **قالب الفئة**. التركيب الذي نستخدمه للقوالب هو:

```
template<class Type>
declaration;
```

حيث Type هو نوع البيانات و declaration اما لاعلان الدالة أو اعلان الفئة. في C++ تكون template كلمة محفوظة وتشير كلمة class في العنوان الى أي نوع يحدده المستخدم أو أي نوع مدمج. يتم الاشارة الى type كعامل رسمي للقالب. كما تكون المتغيرات معاملات للدالات تكون الأنواع (أي أنواع البيانات) معاملات للقوالب.

قوالب الدالة:

في هذا القسم "انقال الدالة" (الموجود سابقاً في هذا الفصل) عندما تم تقديم انقال الدالة تم انقال الدالة larger لايجاد العنصر الأكبر من بين عددين صحيحين أو رمزين أو عددين ذات فاصلة عائمة أو مقطعين. لتطبيق الدالة larger نحتاج الى كتابة أربع تعريفات دالة لنوع البيانات: تعريف من أجل int، وتعريف من أجل Char، وتعريف من أجل double، وتعريف من أجل string. بالرغم من هذا يكون هيكل كل دالة متشابه. تقوم C++ بتبسيط عملية انقال الدالات عن طريق توفير قوالب للدالة. تركيب قالب الدالة يكون كما يلي:

```
template<class Type>
function definition;
```

حيث تتم الاشارة الى Type كمعامل رسمي للقالب ويتم استخدامه لتحديد نوع المعاملات الخاصة بالدالة ونوع ناتج الدالة ومن أجل اعلان المتغيرات داخل الدالة.

البيانات:

```
template<class Type>
Type larger(Type x, Type y)
{
    if(x >= y)
        return x;
    else
        return y;
}
```

تقوم بتعريف نموذج الدالة larger الذي قدم ناتج العنصر الأكبر من بين عنصرين. في عنوان الدالة يكون نوع المعاملات الرسمية x و y هو Type وسوف يتم تحديده بواسطة نوع المعاملات الرسمية عندما يتم استدعاء الدالة. على سبيل المثال البيان:

```
cout<<larger(5,6)<<endl;
```

يكون استدعاء قالب الدالة larger. بما أن 5 و 6 من النوع int (أعداد صحيحة) يتم استبدال نوع البيانات int للدالة Type ويقوم المصرف بتوليد الكود المناسب. إذا قمت بحذف هيكل الدالة في تعريف قالب نموذج الدالة فان نموذج الدالة يكون هو عادةً النموذج. المثال التالي يوضح استخدام قوالب الدالة.

مثال 2-7:

هذا المثال يستخدم قالب الدالة larger لتحديد الأكبر من بين عنصرين.

```
#include <iostream>
#include <string>

using namespace std;

template<class Type>
Type larger(Type x, Type y);

int main()
{
    cout<<"Line 1: Larger of 5 and 6 = "
        <<larger(5, 6)<<endl;           //Line 1

    cout<<"Line 2: Larger of A and B = "
        <<larger('A', 'B')<<endl;      //Line 2

    cout<<"Line 3: Larger of 5.6 and 3.2 = "
        <<larger(5.6, 3.2)<<endl;       //Line 3

    string str1 = "Hello";             //Line 4
    string str2 = "Happy";             //Line 5

    cout<<"Line 6: Larger of "<<str1<<" and "
        <<str2<<" = "<<larger(str1, str2)
        <<endl;                         //Line 6

    return 0;
}

template<class Type>
Type larger(Type x, Type y)
{
    if(x >= y)
        return x;
    else
        return y;
}
```

المخرجات:

الصف 1: الأكبر من 5 و 6 = 6

الصف 2: الأكبر من A و B = B

الصف 3: الأكبر من 5.6 و 3.2 = 5.6

الصف 6: الأكبر من Hello و Happy = Hello

قوالب الفئة:

ان قوالب الفئة مثلها مثل قوالب الدالة يتم استخدامها لكتابة جزء أحادي الكود لمجموعة من الفئات المرتبطة. على سبيل المثال قمنا في الفصل 1 بتعريف قائمة ما بأنها نوع بيانات مجرد وكان نوع عامل قائمتنا هناك `int`. اذا تغير نوع عامل القائمة من `int` الى `char` أو `double` أو `string` تحتاج الى كتابة فئات منفصلة لكل نوع عامل. في الغالبية تبقى العمليات المؤداة على القائمة والحلول الحسابية لتطبيق تلك العمليات ثابتة. باستخدام قوالب الفئة يمكنك عمل فئة منتجة `listType` ويمكن للمصرف توليد كود المصدر المناسب من أجل تطبيق خاص. التركيب الذي نستخدمه لقالب الفئة هو:

```
template< class Type>
class declaration
```

يطلق على قوالب الفئة أنواع محددة المعامل لأن توليد فئة معينة يكون بناءً على نوع المعامل. على سبيل المثال اذا كان نوع معامل القالب هو `int` يمكنك توليد قائمة لمعالجة الأعداد الصحيحة واذا كان نوع المعامل هو `string` يمكنك توليد قائمة لمعالجة المقاطع. يتم تعريف قالب الفئة لنوع قائمة من نوع البيانات المجرد كما يلي:

```
template<class elemType>
class listType
{
public:
    bool isEmpty();
        //Function to determine whether the list is empty.
        //Postcondition: Returns true if the list is empty;
        // otherwise, returns false.

    bool isFull();
        //Function to determine whether the list is full.
        //Postcondition: Returns true if the list is full;
        // otherwise, returns false.

    void search(const elemType& searchItem, bool& found);
        //Function to search the list for searchItem.
        //Postcondition: found is set to true if the
        // searchItem is found in the list; otherwise,
        // found is set to false.
```

```

Template <class elemType>
Class listType
}
عامة:
bool isEmpty ();
// دالة لتحديد ما اذا كانت القائمة خالية.
// شرط تالي: تقدم true اذا كانت القائمة خالية وبخلاف هذا تقدم false.
bool isFull ();
// دالة لتحديد ما اذا كانت القائمة ممتلئة.
// شرط تالي: تقدم true اذا كانت القائمة ممتلئة وبخلاف هذا تقدم false.
Void search (const elemType& searchItem, bool& found);
// دالة للبحث في القائمة عن عنصر البحث
// شرط تالي: يتم تحديد أن found true اذا تم العثور على عنصر البحث
// وبخلاف هذا يتحم تحديد أنه false.

void insert(const elemType& newElement);
//Funtion to insert newElement in the list.
//Precondition: Prior to insertion, the list must
// not be full.
//Postcondition: The list is the old list plus the
// newElement.

void remove(const elemType& removeElement);
//Funtion to remove an element from the list.
//Postcondition: If the list is empty, it outputs
// the message, "Cannot delete from the empty
// list".
// If the list is nonempty, then if
// removeElement is found, the list is the
// old list minus removeElement; otherwise,
// the list is the same as the old list.

void destroyList();
//Funtion to destroy the list.
//Postcondition: length = 0
void printList();
//Funtion to output the elements of the list.

listType();
//default constructor
//Sets the length of the list to 0.
//Postcondition: length = 0

private:
elemType list[100]; //array to hold the list elements
int length; //variable to store the number
//of elements in the list
};

```

Void insert (const elemType& newElement);

```

// دالة لادخال عنصر جديد في القائمة.
// شرط مسبق: قبل الادخال يجب ألا تكون القائمة ممتلئة.
// شرط تالي: القائمة هي القائمة الجديدة بالاضافة الى العنصر الجديد.
Void remove (const elemType& removeElement);
// دالة لازالة عنصر من القائمة.
// شرط تالي: اذا كانت القائمة خالية فانها تقدم
// الرسالة "لا يمكن الازالة من القائمة الخالية".
// اذا لم تكن القائمة خالية واذا تم العثور على العنصر المزال
// تكون القائمة هي القائمة القديم ناقص منها العنصر المزال وبخلاف هذا
// تكون القائمة كما كانت القائمة القديمة.
Void destroyList ( );
// دالة لتدمير القائمة.
// شرط تالي: الطول = صفر.
Void printList ( );
// دالة لاجراج عناصر القائمة.
listType ( );
// مقوم افتراضي
// يحدد طول القائمة = صفر.
// شرط تالي: الطول = صفر
خاصة:
elemType list [100]; // مجال لحمل عناصر القائمة.
int length; // متغير لتخزين عدد العناصر في القائمة.
};

هذا التعريف لنموذج الفئة listType عبارة عن تعريف عام ويتضمن فقط العمليات الأساسية المؤداة
على القائمة. لاشتقاق قائمة محددة من هذه القائمة واطافة أو اعادة كتابة العمليات نعلن المجال
المحتوي على عناصر القائمة وطول القائمة على أنها محمية.
بعد هذا نقوم بوصف قائمة محددة. افترض أنك تريد عمل قائمة لمعالجة بيانات عدد صحيح.
البيان:
ListType<int> intList; // Line 1
يعلن أن intList قائمة مكونة من 100 عنصر مع كون كل عنصر من النوع int وبالمثال يقوم البيان:
ListType<string> stringList; // Line 2

```

يعلن أن stringList قائمة مكونة من 100 عنصر مع كون كل عنصر من النوع string.

في البيانات الموجودة في الصفوف 1 و 2 تتم الإشارة الى listType<int> و listType<string> على أنها **تجسيديات القالب** أو **تجسيديات لقالب الفئة** listType<elemType> حيث يكون elemType هو معامل الفئة في عنوان القالب. يمكن عمل تجسيد القالب اما بنوع مدمج أو بنوع يحدده المستخدم.

يتم اعتبار عناصر دالة قالب الفئة قوالب دالة ولهذا عند اعطاء تعريفات لعناصر دالة قالب الفئة يجب أن تتبع تعريف قالب الدالة. على سبيل المثال يكون تعريف العنصر insert للفئة listType هو:

```
template<class elemType>
void listType<elemType>::insert(const elemType& newElement)
{
    .
    .
    .
}
```

في عنوان تعريف دالة العنصر يقوم elemType بتحديد نوع بيانات عناصر القائمة.

ملف العنوان وملف تطبيق قالب الفئة:

لقد قمنا حتى الآن بوضع تعريف الفئة (في ملف التحديد) وتعريفات دالات المستخدم (في ملف التطبيق) في ملفات منفصلة، تم توليد كود الهدف من ملف التطبيق (بشكل مستقل عن أي كود عميل) وتم ربطها بكود العميل. هذه الاستراتيجية لا تعمل مع قوالب الفئة. ان تمرير المعاملات الى الدالة له تأثير في زمن التطبيق بينما يكون لتمرير معامل الى قالب الفئة له تأثير في زمن التجميع. بما أن المعاملات الفعلية محددة في كود العميل وبما أن المجمع لا يمكنه تجسيد قالب الدالة بدون المعاملات الفعلية للقالب اذن لم يعد يمكننا تجميع ملف التطبيق بشكل مستقل عن كود العميل.

هذه المشكلة لها العديد من الحلول الممكنة. يمكننا وضع تعريف الفئة وتعريفات قوالب الدالة بشكل مباشر في كود العميل أو يمكننا أن نضع تعريف الفئة وتعريفات قوالب الدالة معاً في نفس الملف الرئيسي. هناك بديل آخر وهو وضع تعريف الفئة وتعريفات الدالات في ملفات منفصلة (كما هو معتاد) ولكنها تتضمن موجه الى ملف التطبيق في نهاية الملف الرئيسي (أي ملف التحديد). في أي من الحالتين يتم تجميع تعريفات الدالة وكود العميل معاً. لأغراض توضيحية سوف نقوم بوضع تعريف الفئة وتعريفات الدالة في نفس الملف الرئيسي.

مراجعة سريعة:

1. التوريث والتكوين طرق مفيدة للربط بين فئتين أو أكثر.

2. التوريث عبارة عن علاقة "is-a".
3. التكوين عبارة عن علاقة "has-a".
4. في التوريث الفردي يتم اشتقاق الفئة المستمدة من فئة واحدة موجودة فقط تسمى القاعدة الأساسية.
5. في التوريث المتعدد يتم اشتقاق الفئة المستمدة من أكثر من قاعدة أساسية واحدة.
6. العناصر الخاصة لفئة أساسية ما تكون خاصة بالفئة الأساسية ولا يمكن للفئة المستمدة تناولها بشكل مباشر.
7. العناصر العامة لفئة أساسية ما يمكن توريثها اما كعناصر عامة أو محمية أو خاصة بواسطة الفئة المستمدة.
8. يمكن أن تقوم الفئة المستمدة باعادة تعريف عناصر دالة القاعدة الأساسية ولكن هذا التعريف الجديد ينطبق فقط على أهداف الفئة المستمدة.
9. يتم تحديد الاستدعاء الى مقوم الفئة الأساسية في عنوان تعريف مقوم الفئة المستمدة.
10. عند تهيئة هدف الفئة المستمدة يتم تنفيذ مقوم الفئة الأساسية أولاً.
11. راجع قواعد التوريث المعطاة في هذا الفصل.
12. في التكوين يكون عنصر الفئة عبارة عن هدف لفئة أخرى.
13. في التكوين يتم تحديد الاستدعاء الى مقوم أهداف العنصر في عنوان تعريف مقوم الفئة.
14. المبادئ الثلاثة الأساسية ل OOD هي التغليف والتوريث وتعدد الأشكال.
15. المعامل الذي لديه معاني مختلفة مع أنواع بيانات مختلفة يقال عنه أنه مثقل.
16. في C++ يتم استخدام << كمعامل ادخال تدفق وكمعامل انتقال أيسر. بالمثل يتم استخدام >> كمعامل استخلاص تدفق وكمعامل انتقال أيمن وكلاهما مثال على ائقال المعامل.
17. الدالة التي تنقل المعامل تسمى دالة معامل.
18. تركيب عنوان دالة المعاملة هو

| |
|---|
| returnType operator operatorSymbol (parameters) |
|---|

19. في C++ يكون المعامل عبارة عن كلمة محفوظة.
20. دالات المعامل عبارة عن دالات منتجة لقيمة.
21. باستثناء معامل التحديد ومعامل اختيار العنصر يجب ان يتم ائقال المعامل لكي يتم استخدامه على أهداف الفئة. يؤدي معامل التحديد نسخة افتراضية.

22. يقيم اثنال المعامل نفس المدون الدقيق لأنواع البيانات التي يحددها المستخدم كما هو متاح مع أنواع البيانات المدمجة.

23. عندما يتم اثنال معامل لا يمكن تغيير أسببته ولا يمكن تغيير علاقته ولا يمكن استخدام معطياته الافتراضية ولا يمكن تغيير عدد المعطيات التي يأخذها المعامل ويبقى معنى كيفية عمل المعامل مع أنواع البيانات المدمجة كما هو.

24. من غير الممكن خلق عمليات جديدة ويمكن اثنال العمليات المتواجدة فقط.

25. غالبية معاملات C++ يمكن اثنالها.

26. العمليات التي لا يمكن اثنالها هي - و * و :: و ؟: و sizeof.

27. المؤشر this يشير الى الهدف ككل.

28. دالة المعامل التي تنقل المعاملات () و [] و > - و = يجب أن تكون عناصر للفئة.

29. الدالة الصديقة ليست عنصر في الفئة.

30. عنوان الدالة الصديقة تسبقه كلمة Friend.

31. في C++ تكون friend كلمة محفوظة.

32. اذا كانت دالة المعامل عضو في الفئة يجب أن يكون المعامل الموجود أقصى يسار

العامل هدف الفئة (أو اشارة الى هدف الفئة) لفئة هذا العامل.

33. عندما تكون دالة المعامل الثنائي عضو في فئة ما يكون لها معامل واحد فقط وعندما

لا تكون عضو بالفئة يكون لها معاملان.

34. دالات العامل التي تنقل عامل ادخال التدفق << و عامل استخلاص التدفق >> لفئة ما

يجب أن تكون دالات صديقة لهذه الفئة.

35. في C++ يمكن اثنال اسم الدالة.

36. كل مثال على دالة مثقلة له قائمة معامل رسمي مختلفة.

37. في C++ تكون template كلمة محفوظة.

38. باستخدام القوالب يمكنك كتابة جزء أحادي الكود لمجموعة من الدالات المترابطة

يسمى قالب الدالة.

39. باستخدام القوالب يمكنك كتابة جزء أحادي الكود لمجموعة من الفئات المترابطة

يسمى قالب الفئة.

40. تركيب قالب يكون:

```
Template<class elemType>  
Declaration;
```

حيث elemType محدد بواسطة المستخدم ويتم استخدامه لتمرير الأنواع (أي أنواع البيانات) كمعاملات و declaration اما أن يكون دالة أو فئة. تشير كلمة class في العنوان الى أي نوع بيانات يحدده المستخدم أو نوع بيانات مدمجة.

41. قوالب الفئة يطلق عليها أنواع محددة العوامل.

42. في قالب الفئة يقوم المعامل elemType بتحديد كيفية ضبط قالب الفئة العام لتشكيل فئة محددة.

43. المعامل elemType مذكور في كل عنوان فئة وكل تعريف لدالة المستخدم.

44. افترض أن cType قالب فئة وأن func دالة مستخدم للفئة cType. يكون عنوان تعريف الدالة func هو:

Template<class elemType>

funcType cType<elemType>: : func(formal parameters)

حيث funcType هو نوع الدالة مثل void.

45. افترض أن cType قالب فئة يمكنه أخذ int كمعامل. البيان x; cType<int> يعلن أن x هدف من النوع cType والنوع الذي تم تمريره الى الفئة cType هو int.

تمارين:

1- ضع علامة صح أو خطأ أمام الجمل التالية:

- أ- مقوم الفئة المستمدة يحدد الاستدعاء الى مقوم الفئة الأساسية في عنوان تعريف الدالة.
- ب- مقوم الفئة المستمدة يحدد الاستدعاء الى مقوم الفئة الأساسية باستخدام اسم الفئة.
- ت- افترض أن x و y فئتان وواحد من عناصر بيانات x هدف من النوع y وكلتا الفئتان لها مقومات. فان مقوم x يحدد استدعاء الى مقوم y باستخدام اسم الهدف من النوع y.
- ث- الفئة المستمدة يجب أن يكون لها مقوم.
- ج- في C++ يمكن ائصال جميع المعاملات بالنسبة الى أنواع البيانات التي يحددها المستخدم.
- ح- في C++ لا يمكن اعادة تعريف المعاملات بالنسبة الى الأنواع المدمجة.
- خ- الدالة التي تنقل المعامل تسمى دالة المعامل.
- د- C++ تسمح للمستخدمين بعمل معاملات خاصة بهم.
- ذ- لا يمكن تغيير أسبقية المعامل ولكن يمكن تغيير ارتباطه.
- ر- كل مثال على دالة مثقلة له نفس العدد من المعاملات.
- ز- ليس من الضروري ائصال معاملات مترابطة للفئات التي لديها عناصر بيانات int فقط.
- س- دالة المستخدم الخاصة بقالب فئة عبارة عن قالب دالة.
- ش- عند كتابة تعريف دالة صديقة يجب أن تظهر كلمة friend في عنوان الدالة.

ص- عنوان دالة المعامل لا ثقال معامل ما قبل الزيادة (++) ومعامل ما بعد الزيادة (++) لا يتغير لأن كلا المعاملين لهما نفس الرموز.

2- ارسـم تسلسـل الفئـة الـذي يـتم به اشتقاق فئات متعددة من فئة أساسية واحدة.

3- افترض أن الفئة employeeType مستمدة من الفئة personType (انظر مثال 1-6 في الفصل الأول). اعطي أمثلة عن عناصر البيانات والدالة التي يمكن اضافتها الى الفئة employeeType.

4- اشرح الفرق بين عناصر الفئة الخاصة والمحمية.

5- انظر الى تعريف الفئة التالي:

```
class aClass
{
public:
    void print() const;
    void set(int, int);
    aClass();
    aClass(int, int);

private:
    int u;
    int v;
};
```

ما الخطأ في تعريفات الفئات التالية؟
أ-

```
class bClass public aClass
{
public:
    void print()
    void set(int, int, int);
private:
    int z;
}
```

ب-

```
class cClass: public aClass
{
public:
    void print();
    int sum();
    cClass();
    cClass(int)
}
```

6- انظر الى البيانات التالية:

```
class yClass
{
public:
    void one();
    void two(int, int);
    yClass();
private:
    int a;
    int b;
};
```

- أ. العناصر الخاصة للفئة yClass تعتبر عناصر عامة للفئة xClass. صح أم خطأ؟
ب. ضع علامة صحيح أم غير صحيح أمام البيانات التالية وإذا كان البيان غير صحيح وضح السبب.

(i)

```
void yClass::one()
{
    cout<<a+b<<endl;
}
```

(ii)

```
y.a = 15;
x.b = 30;
```

(iii)

```
void xClass::one()
{
    a = 10;
    b = 15;
    z = 30;
    cout<<a+b+z<<endl;
}
```

(iv)

```
cout<<y.a<<" "<<y.b<<" "<<x.z<<endl;
```

- 7- افترض اعلان
أ. اكتب تعريف المقوم الافتراضي للفئة yClass حتى يتم تهيئة عناصر البيانات الخاصة بالفئة yClass عند صفر.
ب. اكتب تعريف المقوم الافتراضي للفئة xClass حتى يتم تهيئة عناصر البيانات الخاصة بالفئة xClass عند صفر.

- ج. اكتب تعريف دالة المستخدم two of yClass حتى تتم تهيئة عنصر البيانات الخاص a عند قيمة المعامل الأول من two وتهيئة عنصر البيانات الخاص b عند قيمة المعامل الثاني من two.

8- ما الخطأ في الكود التالي؟

```
class classA
{
protected:
    void setX(int a);                //Line 1
    //Postcondition: x = a          //Line 2
private:                             //Line 3
    int x;                          //Line 4
};
.
.
.
int main()
{
    classA aObject;                //Line 5

    aObject.setX(4);                //Line 6
    return 0;                       //Line 7
}
```

9- انظر الى الكود التالي:

```
class one
{
public:
    void print() const;
    //Outputs the values of x and y.
protected:
    void setData(int u, int v);
    //Postcondition: x = u; y = v
private:
    int x;
    int y;
};

class two: public one
{
public:
    void setData(int a, int b, int c);
    //Postcondition: x = a; y = b; z = c
    void print() const;
    //Outputs the values of x, y, and z.
private:
    int z;
};
```

- أ. اكتب تعريف الدالة setData للفئة two.
ب. اكتب تعريف الدالة print للفئة two.

10- ما هي مخرجات برنامج C++ التالي؟

```

#include <iostream>
#include <string>

using namespace std;

class baseClass
{
public:
    void print()const;

    baseClass(string s = " ", int a = 0);
    //Postcondition: str = s; x = a
protected:
    int x;

private:
    string str;
};

class derivedClass: public baseClass
{
public:
    void print()const;

    derivedClass(string s = "", int a = 0, int b = 0)
        //Postcondition: str = s; x = a; y = b

private:
    int y;
};

int main()
{
    baseClass baseObject("This is base class", 2);
    derivedClass derivedObject("DDDDDD", 3, 7);

    baseObject.print();
    derivedObject.print();

    return 0;
}

void baseClass::print() const
{
    cout<<x<<" "<<str<<endl;
}

baseClass::baseClass(string s, int a)
{
    str = s;
    x = a;
}

void derivedClass::print()const
{
    cout<<"Derived class: "<<y<<endl;
    baseClass::print();
}

derivedClass::derivedClass(string s, int a,
                           int b)
    :baseClass("Hello Base", a + b)
{
    y = b;
}

```

11- ما هي مخرجات البرنامج التالي؟

```
#include <iostream>

using namespace std;

class baseClass
{
public:
    void print()const;

    int getX();

    baseClass(int a = 0);

protected:
    int x;
};

class derivedClass: public baseClass
{
public:
    void print()const;

    int getResult();

    derivedClass(int a = 0, int b = 0);

private:
    int y;
```

```
};
```

```
int main()
```

```
{
```

```
    baseClass baseObject(7);
```

```
    derivedClass derivedObject(3,8);
```

```
    baseObject.print();
```

```
    derivedObject.print();
```

```
    cout<<"***** "<<baseObject.getX()<<endl;
```

```
    cout<<"#####"<<derivedObject.getResult()<<endl;
```

```
    return 0;
```

```
}
```

```
void baseClass::print()const
```

```
{
```

```
    cout<<"In base: x = "<<x<<endl;
```

```
}
```

```
baseClass::baseClass(int a)
```

```
{
```

```
    x = a;
```

```
}
```

```
int baseClass::getX()
```

```
{
```

```
    return x;
```

```
}
```

```
void derivedClass::print()const
```

```
{
```

```
    cout<<"In derived: x = "<<x<<"", y = "<<y  
    <<"", x + y = "<<x + y<<endl;
```

```
}
```

```
int derivedClass::getResult()
```

```
{
```

```
    return x + y;
```

```
}
```

```
derivedClass::derivedClass(int a, int b)
```

```
    :baseClass(a)
```

```
{
```

```
    y = b;
```

```
}
```

12- ما هي الدالة الصديقة؟

13- افترض أن المعامل << سوف يتم اثقاله للفئة التي يحددها المستخدم المسماة mystery. لماذا يجب اثقال << كدالة صديقة؟

14- افترض أن المعامل الثنائي + يتم اثقاله كدالة مستخدم للفئة strange. كم عدد العوامل التي تمتلكها الدالة +operator؟

15- انظر الى البيان التالي:

```
class strange
{
    .
    .
    .
};
```

أ. اكتب بيان يوضح الاعلان في الفئة strange لاثقال المعامل >>.

ب. اكتب بيان يوضح الاعلان في الفئة strange لاثقال المعامل الثنائي + كدالة مستخدم.

ت. اكتب بيان يوضح الاعلان في الفئة strange لاثقال المعامل == كدالة مستخدم.

ث. اكتب بيان يوضح الاعلان في الفئة strange لاثقال معامل ما بعد الزيادة ++ كدالة مستخدم.

16- افترض اعلان التمرين رقم 15.

أ. اكتب بيان يوضح الاعلان في الفئة strange لاثقال المعامل الثنائي + كدالة صديقة.

ب. اكتب بيان يوضح الاعلان في الفئة strange لاثقال المعامل == كدالة صديقة.

ج. اكتب بيان يوضح الاعلان في الفئة strange لاثقال معامل ما بعد الزيادة ++ كدالة صديقة.

17- جد الأخطاء في الكود التالي:

```
class mystery //Line 1
{
    ...
    bool operator <= (mystery); //Line 2
    ...
};

bool mystery::<=(mystery rightObj) //Line 3
{
    ...
}
```

18- جد الأخطاء في الكود التالي:

```
class mystery //Line 1
{
    ...
    bool operator <= (mystery, mystery); //Line 2
    ...
};
```

19- جد الأخطاء في الكود التالي:

```
class mystery                                //Line 1
{
    ...
    friend operator+ (mystery);              //Line 2
    //Overload the binary operator +
    ...
};
```

20- كم عدد المعاملات اللازمة لاثقال عامل ما قبل الزيادة لفئة ما كدالة مستخدم؟

21- كم عدد المعاملات اللازمة لاثقال عامل ما قبل الزيادة لفئة ما كدالة صديقة؟

22- كم عدد المعاملات اللازمة لاثقال عامل ما بعد الزيادة لفئة ما كدالة مستخدم؟

23- كم عدد المعاملات اللازمة لاثقال عامل ما بعد الزيادة لفئة ما كدالة صديقة؟

24- جد الأخطاء في الكود التالي:

```
template<class type>                        //Line 1
class strange                              //Line 2
{
    ...
};

strange<int> s1;                           //Line 3
strange<type> s2;                          //Line 4
```

25- انظر الى الاعلان التالي:

```
template<class Type>
class strange
{
    ...
private:
    Type a;
    Type b;
};
```

أ- اكتب بيان يعلن أن sObj هدف من النوع strange مثلما عناصر البيانات الخاصة a و b من النوع int.

ب- اكتب بيان يوضح أن الاعلان في الفئة strange يثقل المعامل == كدالة مستخدم.

ت- افترض أن الهدفين من النوع strange متساويان اذا كانت عناصر بياناتهم متوافقة. اكتب تعريف معادل الدالة == للفئة strange الذي يتم اثقاله كدالة مستخدم.

26- انظر الى تعريف قالب الدالة التالية:

```
template<class Type>
Type surprise(Type x, Type y)
{
    return x + y ;
}
```

ما هي مخرجات البيانات التالية؟

```
cout<<surprise(5, 7)<<endl;      أ-  
  
string str1 = "Sunny";           ب-  
string str2 = " Day";  
cout<<surprise(str1, str2)<<endl;
```

27- انظر الى تعريف قالب الدالة التالية:

```
template<class Type>  
Type funcExp(Type list[], int size)  
{  
    int j;  
    Type x = list[0];  
    Type y = list[size - 1];  
  
    for(j = 1; j < (size - 1)/2; j++)  
    {  
        if(x < list[j])  
            x = list[j];  
        if(y > list[size - 1 - j])  
            y = list[size - 1 - j];  
    }  
  
    return x + y;  
}
```

افترض كذلك أن لديك الاعداد السابيه:

```
int list[10] = {5, 3, 2, 10, 4, 19, 45, 13, 61, 11};  
string strList[] = {"One", "Hello", "Four", "Three", "How",  
                    "Six"};
```

ما هي مخرجات البيانات التالية؟

```
cout<<funExp(list, 10);           أ.  
cout<<funExp(strList, 6)<<endl;  ب.
```

28- اكتب تعريف قالب الدالة الذي يبادل محتويات متغيرين.

تمارين برمجة

1. في الفصل الأول تم تصميم الفئة clockType لتطبيق الزمن في برنامج ما. هناك تطبيقات معينة بالاضافة الى الساعات والدقائق والثواني قد تحتاج منك تخزين المنطقة الزمنية. قم باشتقاق الفئة extClockType من الفئة clockType عن طريق اضافة عنصر بيانات لتخزين المنطقة الزمنية. أضف دالات العناصر والمقومات اللازمة لجعل الفئة عاملة. و قم أيضاً بكتابة تعريفات دالات العنصر والمقومات وأخيراً اكتب برنامج اختبار لاختبار فنتك.

2. في هذا الفصل تم تصميم الفئة `dateType` لتطبيق التاريخ في برنامج ما ولكن دالة العنصر `setDate` والمقوم لا يتحقق مما اذا كان التاريخ صحيح قبل تخزينه في عناصر البيانات أم لا. أعد كتابة تعريفات الدالة `setDate` والمقوم حتى يتم التحقق من قيم الشهر واليوم والعام قبل تخزين التاريخ داخل عناصر البيانات. أضف عنصر الدالة `isLeapYear` للتحقق مما اذا كانت السنة سنة كبيسة وبالإضافة الى هذا أكتب برنامج اختبار لاختبار فئتك.

3. يتم تمثيل نقطة في المستوى س – ص بواسطة نظير س ونظير ص. قم بتصميم فئة تسمى `pointType` يمكنها تخزين ومعالجة نقطة في المستوى س – ص. يجب أن تؤدي عمليات على النقطة مثل توضيح النقطة، وتحديد نظائر النقطة، وطباعة نظائر النقطة، وانتاج نظير س، وانتاج نظير ص وقم كذلك بكتابة برنامج اختبار من أجل اختبار عمليات متعددة على النقطة.

4. كل دائرة لها مركز ونصف قطر. باعطاء نصف القطر يمكنك تحديد مساحة ومحيط الدائرة وباعطاء المركز يمكنك تحديد موقعه في المستوى س – ص. مركز الدائرة هو نقطة في المستوى س – ص. قم بتصميم فئة تسمى `circleType` يمكنها تخزين نصف قطر الدائرة ومركزها. بما أن المركز عبارة عن نقطة في المستوى س – ص وبما أنك صممت فئة لايجاد خصائص نقطة ما في تمرين البرمجة رقم 3 اذن يجب عليك اشتقاق الدالة `circleType` من الفئة `pointType`. يجب أن تكون قادراً على أداء العمليات المعتادة على الدائرة مثل تحديد نصف القطر، وطباعة نصف القطر، وحساب وطباعة المساحة والمحيط، وتنفيذ العمليات المعتادة على المركز.

5. كل أسطوانة لها قاعدة وارتفاع حيث تكون القاعدة عبارة عن دائرة. قم بتصميم فئة `cylinderType` يمكنها الاحتواء على خصائص الأسطوانة واجراء العمليات المعتادة عليها. قم باشتقاق هذه الفئة من الفئة `circleType` التي تم تصميمها في تمرين البرمجة رقم 4. من ضمن العمليات التي يمكن اجرائها على الأسطوانة ما يلي: حساب وطباعة الحجم، وحساب وطباعة المساحة السطحية، وتحديد الارتفاع، وتحديد نصف قطر القاعدة، وتحديد مركز القاعدة.

6. في تمرين البرمجة رقم 2 تم تصميم الفئة `dateType` وتطبيقها لتتبع التاريخ ولكن عملياتها كانت محدودة للغاية. أعد تعريف الفئة `dateType` حتى يمكنها أداء العمليات التالية على التاريخ بالإضافة الى العمليات المعرفة بالفعل:

أ. تحديد الشهر.

ب. تحديد اليوم.

ج. تحديد العام.

د. انتاج الشهر.

هـ. انتاج اليوم.

و. انتاج العام.

ز. اختبار ما اذا كانت السنة كبيسة.

ح. انتاج عدد الأيام في الشهر. على سبيل المثال اذا كان التاريخ 3-12-2003 فان عدد الأيام الذي يتم انتاجه هو 31 لأن هناك 31 يوم في شهر مارس.

ط. انتاج عدد الأيام التي مرت في العام. على سبيل المثال اذا كان التاريخ 3-18-2003 فان عدد الأيام التي مرت من العام هو 77 يوم. لاحظ أن عدد الأيام الناتج يشمل اليوم الحالة أيضاً .

ي. انتاج عدد الأيام المتبقية من العام. على سبيل المثال اذا كان التاريخ 3-18-2003 فان عدد الأيام المتبقية من العام يكون 288 يوم.

ك. حساب التاريخ الجديد عن طريق اضافة عدد ثابت من الأيام الى التاريخ. على سبيل المثال اذا كان التاريخ 3-18-2003 والأيام التي تتم اضافتها هي 25 يوم فان التاريخ الجديد يكون 4-12-2003

7- اكتب تعريف الدالات لتطبيق العمليات التي تم تعريفها في الفئة `dateType` في تمرين البرمجة رقم 6.

8- تقوم فئة `dateType` التي تم تعريفها في تمرين البرمجة رقم 6 بطباعة التاريخ بصيغة رقمية وهناك بعض التطبيقات التي تحتاج الى طباعة التاريخ بصيغة أخرى مثل 24 مارس 2003. قم باشتقاق الفئة `extDateType` حتى يمكن طباعة التاريخ بوحدة من الصيغتين.

أضف عنصر بيانات الى الفئة `extDateType` حتى يمكن كذلك تخزين الشهر في صيغة مقطعية. أضف عنصر دالة لاجراج الشهر في صيغة مقطعية يليه العام مثل الصيغة مارس 2003.

أكتب تعريفات الدالات لتطبيق العمليات الخاصة بالفئة `extDateType`.

9- باستخدام الفئات `extDateType` (تمرين البرمجة 8)، و `dayType` (الفصل الأول، تمرين البرمجة 3) قم بتصميم الفئة `calendarType` التي يمكنك بوجود الشهر والعام أن تطبع تقويم هذا الشهر.

لطباعة تقويم شهري يجب أن تعرف اليوم الأول من الشهر، وعدد الأيام في هذا الشهر. لهذا يجب أن تقوم بتخزين اليوم الأول من الشهر الذي يكون من الصيغة `dayType`، وشهر وعام التقويم. يمكن تخزين الشهر والعام في هدف من الصيغة `extDateType` عن طريق تحديد مكون اليوم من التاريخ عند 1 والشهر والعام كما يحدده المستخدم. لهذا يكون للفئة `calendarType` عنصري بيانات هما: هدف من النوع `dayType` وهدف من النوع `extDateType`.

قم بتصميم الفئة `calendarType` حتى يمكن للبرنامج طباعة تقويم لأي شهر بدايةً من 1 يناير 1500. لاحظ أن اليوم الموافق للأول من يناير 1500 هو يوم الاثنين. من أجل حساب اليوم الأول من أي شهر يمكنك اضافة الأيام المناسبة الى يوم الاثنين من الأول من يناير 1500.

بالنسبة الى الفئة `calendarType`، قم بادخال العمليات التالية:

أ- تحديد اليوم الأول من الشهر الذي سوف يتم طباعة التقويم له وأطلق على هذه العملية

`.firstDayOfMonth`

ب- تحديد الشهر.

ت- تحديد العام.

ث- انتاج الشهر.

ج- انتاج العام.

ح- طباعة تقويم شهر محدد.

خ- اضافة المقومات المناسبة لتهيئة عناصر البيانات.

10. أ. اكتب تعريفات دالات العنصر للفئة `calendarType` (التي تم تصميمها في تمرين البرمجة 9) لتطبيق عمليات الفئة `calendarType`.

ب. اكتب برنامج اختبار لطباعة التقويم اما لشهر محدد أو عام محدد. على سبيل المثال التقويم الخاص بسبتمبر 2003 هو:

| سبتمبر 2003 | | | | | | |
|-------------|-------|--------|--------|------|------|-----|
| أحد | اثنين | ثلاثاء | أربعاء | خميس | جمعة | سبت |
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | | | | |

11. في الفصل 1 تم تصميم الفئة `clockTime` لتطبيق الزمن في برنامج ما وناقش هذا الفصل كيفية انقال معاملات مختلفة. أعد تصميم الفئة `clockTime` عن طريق انقال المعاملات التالية: معاملات ادخال التدفق << واستخلاص التدفق >> للمدخلات والمخرجات ومعاملات ما قبل وما بعد الزيادة لزيادة الزمن ثنائية واحدة، والمعاملات المرتبطة للمقارنة بين زمنين. اكتب كذلك برنامج اختبار لاختبار معاملات متنوعة من الفئة `clockTime`.

12. أ. قم بمد تعريف الفئة `complexType` (من مثال البرمجة على الأعداد المركبة) حتى تجري عمليات الطرح والقسمة. قم بانقال طرح وقسمة المعاملات لهذه الفئة كدالات عنصر.

إذا كانت (أ، ب) و (ج، د) أعداد مركبة فاذن

$$(أ، ب) - (ج، د) = (أ - ج، ب - د)$$

إذا كانت (ج، د) أعداد غير صفرية فان (أ، ب) / (ج، د) = ((أج + ب د) / (ج د + د د))،

$$(-أ د + ب ج) / (ج د + د د)$$

ب. اكتب تعريفات الدالات لانقال المعاملات - و / كما هي معرفة في الجزء أ.

ج. اكتب برنامج اختبار يختبر عمليات متنوعة على الفئة `complexType`.

قم بتكوين اجابتك مع منزلتين عشريتين.

13. أ. أعد كتابة تعريف الفئة complexType حتى يتم ائصال المعاملات الحسابية والمترابطة كدالات لغير العنصر.

ب. اكتب تعريفات دالات العنصر للفئة complexType كم هي مصممة في الجزء أ.
ج. اكتب برنامج اختبار يختبر عمليات متنوعة على الفئة complexType كما هو مصمم في الأجزاء أ، وب وقم بتكوين اجابتك مع منزلتين عشريتين.

14. ليكن $a + ib$ عدد مركب والمتقارن الخاص بـ $a + ib$ هو $a - ib$ والقيمة الحقيقية لـ $a + ib$ هي a . قم بالتوسع $\sqrt{a^2 + b^2}$ الفئة complexType لمثال البرمجة الخاص بالأعداد المركبة عن طريق ائصال المعاملات ~ و 1 كدالات عنصر حتى يقوم ~ باننتاج مقارن العدد المركب ويقوم 1 باننتاج القيمة الحقيقية. اكتب تعريفات دالات هذه المعاملات.

15. أعد عمل تمرين البرمجة رقم 14 حتى يتم ائصال المعاملات ~ و 1 كدالات لغير العنصر.
16. الكسور المنطقية من الصيغة a / b حيث a و b أعداد صحيحة و $b \neq 0$ صفر. في هذا التمرين الكسور تعني كسور منطقية. افترض أن a / b و c / d كسور. فان العمليات الحسابية المؤداة على الكسور يتم تعريفها بالقواعد التالية:

$$a / b + c / d = (ad + bc) / bd$$

$$a / b - c / d = (ad - bc) / bd$$

$$a / b \times c / d = ac / bd$$

$$(a / b) / (c / d) = ad / bc \text{ حيث } c / d \neq 0$$

تتم مقارنة الكسور كما يلي: $a / b \text{ op } c / d$ اذا $ad \text{ op } bc$ حيث op تكون أي معامل من المعاملات المترابطة. على سبيل المثال تكون $a / b < c / d$ اذا كان $ad < bc$.

أ. قم بتصميم فئة مثل fractionType تجري العمليات الحسابية والمترابطة على الكسور. قم بائصال المعاملات الحسابية والمترابطة حتى يمكن استخدام الرموز المناسبة لأداء تلك العمليات. قم أيضاً بائصال معاملات ادخال التدفق واستخلاص التدفق من أجل وجود مدخلات ومخرجات سهلة.

ب. اكتب برنامج C++ يقوم بأداء عمليات على الكسور باستخدام الفئة fractionType.

ت. من ضمن أشياء أخرى اختبر ما يلي: افترض أن x و y و z أهداف من النوع fraction. اذا كانت المدخلات 3/2 فان البيان

Cin>>x;

يجب أن يقوم بتخزين 3/2 في x والبيان

Cout<<x+y<<endl;

يجب أن يقوم باخراج قيمة x+y في صيغة كسرية. البيان

$$z = x + y$$

يجب أن يقوم بتخزين مجموع x و y في z في صورة كسرية. لا تحتاج اجابتك الى أن تكون في الصيغة الأقل.

17. أ. في تمرين البرمجة رقم 2 في الفصل 1 قمت بتعريف الفئة `romanType` لتطبيق الأرقام الرومانية في البرنامج. في هذا التمرين قمت كذلك بتطبيق دالة هي `romanToDecimal` لتحويل الرقم الروماني الى العدد العشري المساوي له.

قم بتعديل تعريف الفئة `romanType` حتى يتم اعلان عناصر البيانات بأنها محمية. استخدم الفئة `String` للتحكم في المقاطع. فضلاً عن هذا قم باثقال معاملات ادخال التدفق واستخلاص التدفق من أجل مدخلات ومخرجات سهلة. يقوم معامل ادخال التدفق بانتاج الرقم الروماني في الصيغة الرومانية. كذلك ادخل دالة العنصر `decimalToRoman` لتحويل الرقم العشري (يجب أن يكون العدد العشري عدد صحيح موجب) الى رقم روماني مساوي له. اكتب تعريف دالة العنصر `decimalToRoman`. من أجل البساطة، افترض أن الحرف `I` يمكنه فقط الظهور أمام حرف آخر وأنه يظهر فقط أمام الحروف `v` و `x`. على سبيل المثال العدد 4 يتم تمثيله بالعدد `IV` والعدد 9 يتم تمثيله بالعدد `IX` والعدد 39 يتم تمثيله بالعدد `XXXIX` والعدد 49 يتم تمثيله بالعدد `XXXIX`. كذلك يتم تمثيل العدد 40 بالرمز `XXXX` والعدد 190 بالرمز `CLXXXX` وهكذا.

ب. قم باشتقاق الفئة `extRomanType` من الفئة `romanType` للقيام بما يلي: في الفئة `extRomanType` قم باثقال المعاملات الحسابية `+` و `-` و `*` و `/` حتى يمكن اجراء عمليات حسابية على الأرقام الرومانية. وقم كذلك باثقال معاملات ما قبل الزيادة وما بعد الزيادة والنقص كدالات عنصر للفئة `extRomanType`.

لاضافة (طرح أو ضرب أو قسمة) أعداد رومانية قم باضافة (طرح أو ضرب أو قسمة على التوالي) ممثليها العشريين ثم قم بتحويلها الناتج الى صيغة الأعداد الرومانية. بالنسبة الى الطرح اذا كان العدد الأول أصغر من العدد الثاني قم باخراج رسالة تذكر أن "بما أن العدد الأول أصغر من العدد الثاني لا يمكن طرح العددين". بالمثل بالنسبة الى القسمة يجب أن يكون المقسوم أكبر من المقسوم عليه. استخدم علامات مماثلة لمعاملات الزيادة والنقص.

ج. اكتب تعريفات الدالات لاثقال المعاملات الموصوفة في الجزء ب.

د. اختبر فنتك extRomanType على البرنامج التالي. (ادخل الملفات الأساسية المناسبة).

```
int main()
{
    extRomanType num1("XXXIV");
    extRomanType num2("XV");
    extRomanType num3;

    cout<<"Num1 = "<<num1<<endl;
    cout<<"Num2 = "<<num2<<endl;
    cout<<"Num1 + Num2 = "<<num1+num2<<endl;
    cout<<"Num1 * Num2 = "<<num1*num2<<endl;

    cout<<"Enter two numbers in Roman format: ";
    cin>>num1>>num2;
    cout<<endl;

    cout<<"Num1 = "<<num1<<endl;
    cout<<"Num2 = "<<num2<<endl;

    num3 = num2 * num1;
    cout<<"Num3 = "<<num3<<endl;

    return 0;
}
```

الفصل

3

المؤشرات و

القوائم المبنية على المصفوفة

أنواع البيانات في C++ يتم تصنيفها الى ثلاث فئات: بسيطة، وتركيبية، ومؤشرات. حتى الآن قمنا بالعمل مع نوعي البيانات الأوليان فقط. هذا الفصل يناقش نوع البيانات الثالث: نوع بيانات المؤشرات. سوف نتعلم أولاً كيفية اعلان متغيرات المؤشرات (أو المؤشرات للايجاز) والسيطرة على البيانات التي تشير اليها. بعد هذا سوف نستخدم هذه المفاهيم عند دراسة المصفوفات الحركية والقوائم المتصلة. تتم مناقشة القوائم المتصلة في الفصل رقم 5.

أنواع بيانات المؤشرات ومتغيرات المؤشرات:

ان القيم التي تنتمي الى أنواع بيانات المؤشر هي عناوين ذاكرة الحاسب الآلي. كما يحدث في العديد من اللغات الأخرى لا يوجد اسم مرتبط بنوع بيانات المؤشر في C++. بما أن المجال – أي قيم نوع بيانات المؤشر – هو العناوين (مواقع الذاكرة) فان متغير المؤشر هو المتغير الذي يكون محتواه عنوان أي موقع ذاكرة.

متغير المؤشر: المتغير الذي يكون محتواه عنوان (موقع ذاكرة).

اعلان متغيرات المؤشر:

بما أنه لا يوجد اسم مرتبط بأنواع بيانات المؤشر فانه لا يتم اعلان متغيرات المؤشر مثل المتغيرات الأخرى. عندما تقوم باعلان متغير مؤشر ما تقوم كذلك بتحديد نوع بيانات القيمة التي يتم تخزينها في موقع الذاكرة المشار اليه بواسطة متغير المؤشر. في C++ تقوم باعلانات متغير مؤشر ما باستخدام رمز astersik (*) بين نوع البيانات واسم المتغير. التركيب العام للاعلان عن متغير مؤشر ما هو:

`data Type *identifier;`

كمثال انظر الى البيانات التالية:

`int *p;`

`char *ch;`

في هذه البيانات يكون كل من p و ch متغيرات مؤشر. محتوى p (عند تحديده بشكل جيد) يشير الى موقع الذاكرة من النوع int ويشير محتوى ch الى موقع الذاكرة من النوع char. عادةً يطلق على p متغير مؤشر من النوع int ويطلق على ch متغير مؤشر من النوع char. قبل مناقشة كيفية عمل المؤشرات لنقم بعمل الملاحظات التالية. البيان:

`int *p;`

يكون مساوي للبيان

`int* p;`

الذي يساوي البيان

`int * p;`

لهذا يمكن أن يظهر الرمز * في أي مكان بين اسم نوع البيانات واسم المتغير.

الآن انظر الى البيان التالي:

`int* p, q;`

في هذا البيان يكون p فقط هو متغير المؤشر وليس q. هناك تكون q متغير int. لتجنب الاختلاط
نفضل الحاق الرمز * باسم المتغير. لذلك تتم كتابة البيان السابق كما يلي:

```
int *p, q;
```

بالطبع يقوم البيان:

```
int *p, *q;
```

باعلان أن كلا من p و q متغيرات مؤشر من النوع int.

الآن أنت تعرف كيفية اعلان المؤشرات وبعد هذا نناقش كيفية جعل المؤشر يؤشر الى مكان الذاكرة
وكيفية السيطرة على البيانات المخزنة في مواقع الذاكرة هذه.

بما أن قيمة المؤشر هي عنوان الذاكرة فان المؤشر يمكنه تخزين عنوان مكان الذاكرة من النوع
المصمم. على سبيل المثال اذا كانت p مؤشر من النوع int فيمكن أن تقوم p بتخزين عنوان أي مكان
بالذاكرة من النوع int. تقوم C++ بتوفير معاملين – عنوان المعامل (&) ومعامل dereferencing
(*) – للعمل مع المؤشرات. يقوم القسمان التاليان بمناقشة هذه المعاملات.

عنوان المعامل (&):

في C++ يطلق على & ampersand عنوان المعامل وهو معامل أحادي ينتج عنوان معامله. على
سبيل المثال باعطاء هذه البيانات:

```
int x;
```

```
int *p;
```

فان البيان:

```
P = &x;
```

يحدد عنوان x عند p. هذا يعني x وقيمة p تشير الى مكان الذاكرة نفسه.

معامل ***** (*):

لقد قمت حتى الآن باستخدام الرمز asterisk (*) كمعامل ضرب ثنائي. تقوم C++ كذلك باستخدام *
كمعامل أحادي. عند استخدام * كمعامل أحادي والذي تتم الاشارة اليه عادةً كمعامل ***** أو معامل
*** فانهم يقوموا بالاشارة الى الهدف الذي يشير اليه معامله. على سبيل المثال باعطاء هذه البيانات:

```
int x = 25;
```

```
int *p;
```

```
p = &x; //store the address of x in p
```

يقوم البيان:

```
cout<<*p<<endl;
```

بطباعة القيمة المخزنة في مكان الذاكرة المشار اليه بواسطة p وهي قيمة x. كما أن البيان:

```
*p = 55;
```

يقوم بتخزين 55 في مكان الذاكرة المشار اليه بواسطة p أي في x.

لننظر الى البيانات التالية:

```
int *p;
```

```
int num;
```

في هذه البيانات يكون p متغير مؤشر من النوع int ويكون num متغير من النوع int. انظر الشكل 1-3.

الذاكرة الأساسية

| |
|---|
| |
| |
| . |
| . |
| . |
| |
| . |
| . |
| . |
| |
| . |
| . |
| . |

P 1200

num 1800

شكل 1-3: الذاكرة الأساسية p و num.

لنقم بافتراض أن موقع الذاكرة 1200 مخصص من أجل p وموقع الذاكرة 1800 مخصص من أجل num. فان البيان:

```
Num = 78;
```

يقوم بتخزين 78 في num – أي في موقع الذاكرة 1800. انظر الشكل 2-3.

الذاكرة الأساسية

| |
|----|
| |
| |
| . |
| . |
| . |
| |
| . |
| . |
| . |
| 78 |
| . |
| . |
| . |

P 1200

num 1800

شكل 3-2: num بعد تنفيذ البيان $\text{num} = 78$

البيان

$P = \&\text{num};$

يقوم بتخزين عنوان num – أي 1800 – داخل p. بعد تنفيذ هذا البيان يشير كل من *p و num الى محتوى موقع الذاكرة 1800 – أي num. انظر شكل 3-3.
الذاكرة الأساسية

| |
|------|
| |
| |
| . |
| . |
| . |
| 1800 |
| . |
| . |
| . |
| 78 |
| . |
| . |
| . |

P 1200

num 1800

شكل 3-3: p بعد تنفيذ البيان $p = \&\text{num}$

يقوم بيان التحديد

$$*p = 24;$$

بتغيير محتوى موقع الذاكرة 1800 وبهذا يغير كذلك محتوى num. انظر شكل 4-3. الذاكرة الأساسية

| | |
|------|----------|
| | |
| | |
| . | |
| . | |
| . | |
| 1800 | P 1200 |
| . | |
| . | |
| . | |
| 24 | num 1800 |
| . | |
| . | |
| . | |

شكل 4-3: p و num بعد تنفيذ البيان $p = 24$.

لنقم بتلخيص المناقشة السابقة:

- كل من $&p$ ، p ، و $*p$ لهم معاني مختلفة.
- $&p$ تعني عنوان p – أي 1200 (في شكل 4-3).
- P تعني محتوى p (1800 في شكل 4-3).
- $*p$ تعني محتوى (24 في شكل 4-3) موقع الذاكرة (1800 في شكل 4-3) المشار اليه بواسطة p (أي المشار اليه بواسطة محتوى موقع الذاكرة 1200).

مثال 1-3:

انظر الى البيانات التالية:

```
int    *p;
```

```
int    *x;
```

افترض أن لدينا مخصص الذاكرة من أجل p و x كما هو موضح في الشكل 5-3.

الذاكرة الأساسية

| |
|---|
| |
| |
| . |
| . |
| . |
| |
| . |
| . |
| . |
| |
| . |
| . |
| . |

P 1400

X 1750

شكل 3-5: الذاكرة الأساسية p و x.

تكون قيم &p، و p، و *p، و &x، و x كما يلي:

| القيمة | |
|--------|----|
| ***** | &p |
| ***** | p |
| **** | *p |
| **** | &x |
| ***** | x |

افترض أن البيانات التالية يتم تنفيذها بالترتيب المعطى:

x = 50;

p = &x;

*p = 38;

قيم &p، و p، و *p، و &x، و x موضحة بعد تنفيذ كل من تلك البيانات. بعد تنفيذ البيان:

X = 50;

تكون قيم $\&p$ ، p ، و $*p$ ، و $\&x$ ، و x كما يلي:

| القيمة | |
|----------------------|-------|
| 1400 | $\&p$ |
| ؟؟؟ (مجهول) | p |
| لا تتواجد (غير معرف) | $*p$ |
| 1750 | $\&x$ |
| 50 | x |

بعد تنفيذ البيان:

$P = \&x;$

تكون قيم $\&p$ ، p ، و $*p$ ، و $\&x$ ، و x كما يلي:

| القيمة | |
|--------|-------|
| 1400 | $\&p$ |
| 1750 | p |
| 50 | $*p$ |
| 1750 | $\&x$ |
| 50 | x |

لاحظ أن بعد تنفيذ البيان $p = \&x$ تحتوي p على عنوان x ، وبهذا تشير $*p$ و x الى نفس موقع الذاكرة الذي يكون x . لهذا تكون قيمة $*p$ هي 50.

بعد تنفيذ البيان:

$*p = 38;$

تكون قيم $\&p$ ، p ، و $*p$ ، و $\&x$ ، و x كما يلي. (بما أن $*p$ و x تشير الى موقع الذاكرة ذاته فان قيمة x تتغير الى 38).

| القيمة | |
|--------|-------|
| 1400 | $\&p$ |
| 1750 | p |
| 38 | $*p$ |
| 1750 | $\&x$ |
| 38 | x |

لنقم بملاحظة ما يلي من المثال 3-1:

1. اعلان مثل:

```
int *p;
```

يخصص ذاكرة من أجل p فقط وليس *p. لاحقاً سوف نتعلم كيفية تخصيص

ذاكرة من أجل *p وما الذي يعنيه تخصيص ذاكرة من أجل *p.

2. افترض ما يلي:

```
int *p;
```

```
int x;
```

اذن:

أ. p عبارة عن متغير مؤشر.

ب. يشير محتوى p فقط الى موقع الذاكرة من النوع int.

ت. موقع الذاكرة x موجود وهو من النوع int. لهذا يكون بيان التحديد:

```
p = &x
```

صحيح. بعد تنفيذ بيان التحديد هذا تكون *p صحيحة وذات معنى.

مثال 3-2:

البرنامج التالي يوضح كيفية عمل متغيرات المؤشر.

```
//Chapter 3: Example 3-2
#include <iostream>

using namespace std;

int main()
{
    int *p;
    int x = 37;

    cout<<"Line 1: x = "<<x<<endl;           //Line 1
    p = &x;                                   //Line 2
    cout<<"Line 3: *p = "<<*p
        <<" , x = "<<x<<endl;                 //Line 3
    *p = 58;                                  //Line 4
    cout<<"Line 5: *p = "<<*p
        <<" , x = "<<x<<endl;                 //Line 5
    cout<<"Line 6: Address of p = "<<&p<<endl; //Line 6
    cout<<"Line 7: Value of p = "<<p<<endl;   //Line 7
    cout<<"Line 8: Value of the memory location "
        <<"pointed to by *p = "<<*p<<endl;   //Line 8
    cout<<"Line 9: Address of x = "<<&x<<endl; //Line 9
    cout<<"Line 10: Value of x = "<<x<<endl;  //Line 10

    return 0;
}
```

تنفيذ العينة:

الصف 1: $x = 37$

الصف 3: $x = 37, *p = 37$

الصف 5: $x = 58, *p = 58$

الصف 6: عنوان $p = 006BFDF4$

الصف 7: قيمة $p = 006BFDF0$

الصف 8: قيمة موقع الذاكرة المشار اليه بواسطة $*p = 58$

الصف 9: عنوان $x = 006BFDF0$

الصف 10: قيمة $x = 58$

البرنامج السابق يعمل كما يلي. البيان في الصف رقم 1 يخرج قيمة x والبيان في الصف رقم 2 يقوم بتخزين عنوان x داخل p . البيان في الصف رقم 3 يخرج قيم كل من $*p$ و x . بما أن p تحتوي على عنوان x فان قيم كل من $*p$ و x تكون هي نفسها كما هو موضح بواسطة مخرجات الصف 3. البيان في الصف رقم 4 يغير قيمة $*p$ الى 58 والبيان في الصف رقم 5 يخرج قيم كل من $*p$ و x وهي مرة أخرى متماثلة. البيانات بين الصف 6 والصف 10 تخرج عنوان p ، وقيمة p ، وقيمة $*p$ ، وعنوان x ، وقيمة x . لاحظ أن قيمة p وعنوان x متماثلين لأن عنوان x مخزن في p بواسطة البيان في الصف 2. (لاحظ أن عنوان p وقيمة p وعنوان x كما هي موضحة في مخرجات الصفوف 6 و 7 و 9 على التوالي معتمدة على الآلة. فضلاً عن هذا تكون هذه لقيم في صيغة عشرية سداسية. عندما تقوم بتشغيل هذا البرنامج على حاسبك الآلي من المحتمل أن تحصل على قيم مختلفة.)

الفئات، والبنيات، ومتغيرات المؤشر:

في القسم السابق تعلمت كيفية اعلان واحتكار المؤشرات الى عناصر البيانات البسيطة مثل `int` و `char`. كما يمكنك اعلان مؤشرات الى أنواع أخرى من البيانات مثل الفئات. سوف نتعلم الآن كيفية اعلان واحتكار المؤشرات الى الفئات والبنيات. (تذكر أن كلا من الفئات والبنيات لها نفس الامكانيات والفرق الوحيد هو أن جميع عناصر الفئة من الناحية الافتراضية تكون خاصة وجميع عناصر البنية عامة. لهذا تنطبق المناقشة التالية على الاثنين.)

انظر اعلان البنية التالي:

```
struct studentType
{
    char name[27];
    double gpa;
    int sID;
    char grade;
};

studentType    student;
studentType*   studentPtr;
```

في هذا الاعلان يكون student هدف من النوع studentType وstudentPtr عبارة عن متغير مؤشر من النوع studentType. البيان التالي يقوم بتخزين العنوان student في studentPtr:

```
studentPtr = &student;
```

البيان التالي يقوم بتخزين 3.9 في المكون gpa للهدف student:

```
(*studentPtr).gpa = 3.9;
```

التعبير gpa. (*studentPtr) عبارة عن خليط من Dereferencing مؤشر واختيار محتوى الفئة. في C++ معامل النقطة له أسبقية أعلى من عامل dereferencing. بسبب هذه الحقيقة تكون الأقواس ضرورية. لتبسيط تناول محتويات الفئة عبر مؤشر ما توفر C++ معامل آخر يطلق عليه سهم معامل تناول عنصر ->. المعامل -> يتكون من رمزين متتاليين: hyphen ورمز "أكبر من". التركيب الخاص بتناول عنصر فئة (بنية) ما باستخدام -> هو:

`pointerVariableName -> classMemberName`

لهذا يكون البيان:

```
(*studentPtr).gpa = 3.9;
```

مكافئ للبيان:

```
studentPtr->gpa = 3.9;
```

ان تناول مكونات الفئة (البنية) عبر المؤشرات التي تستخدم المعامل -> يقضي على استخدام كلا القوسين ومعامل dereferencing. بما أن الأخطاء المطبعية لا يمكن تجنبها والأقواس المفقودة قد تتسبب في إيقاف غير طبيعي للبرنامج أو في نتائج مغلوطة فان هذا الكتاب عند تناوله لمكونات الفئة عبر المؤشرات يستخدم رمز السهم.

المثال 3-3 يوضح كيفية عمل المؤشرات مع دالات عنصر الفئة.

مثال 3-3:

انظر الى الفئة التالية:

```
class classExample
{
public:
    void setX(int a);
        //Function to set the value of the data member x.
        //Postcondition: x = a
    void print() const;
        //Function to output the value of x.

private:
    int x;
};
```

تعريف دالة العنصر يكون كما يلي:

```
void classExample::setX(int a)
{
    x = a;
}

void classExample::print() const
{
    cout<<"x = "<<x<<endl;
}
```

انظر الى الدالة التالية main:

```
int main()
{
    classExample *cExpPtr;           //Line 1
    classExample cExpObject;         //Line 2

    cExpPtr = &cExpObject;           //Line 3

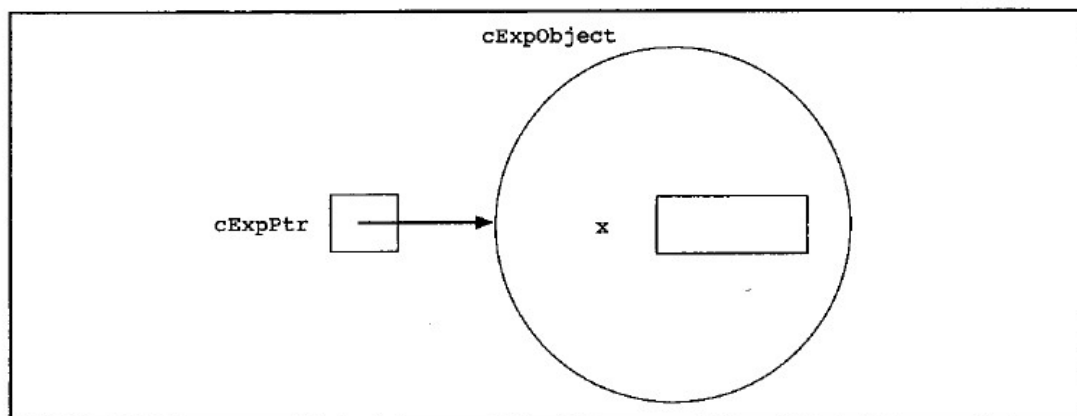
    cExpPtr->setX(5);                 //Line 4
    cExpPtr->print();                  //Line 5

    return 0;
}
```

المخرجات:

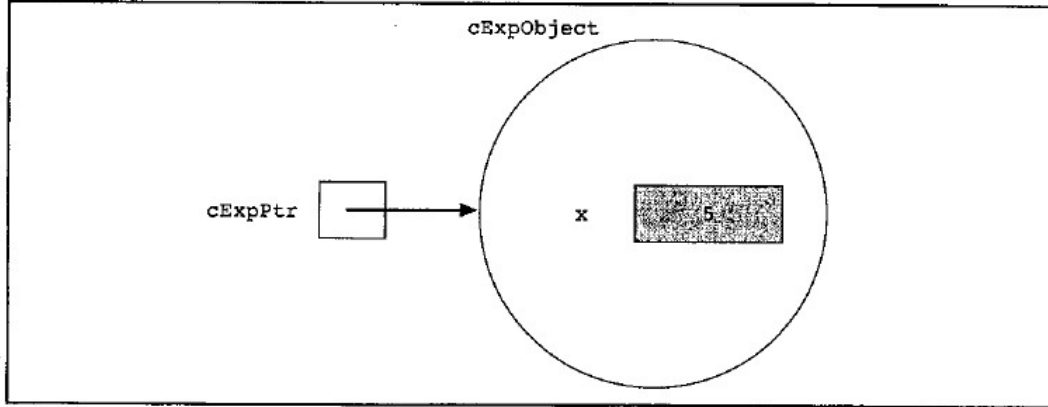
5 = x

في الدالة main، يقوم البيان في الصف 1 باعلان أن cExpPtr مؤشر من النوع classExample والبيان في الصف 2 يعلن أن cExpObject هدف من النوع classExample. البيان في الصف رقم 3 يقوم بتخزين عنوان cExpObject داخل cExpPtr. انظر شكل 6-3.



شكل 6-3: cExpObject و cExpPtr بعد تنفيذ البيان &cExpObject = cExpPtr

في البيان في الصف رقم 4 يتناول المؤشر cExpPtr دالة العنصر setX لتحديد قيمة عنصر البيانات x. انظر الشكل 7-3.



شكل 7-3: cExpPtr و cExpObject بعد تنفيذ البيان (5) setX -> cExpPtr

في البيان في الصف 5 يتناول المؤشر cExpPtr دالة العنصر print لطباعة قيمة x كما هو موضح في المخرجات.

تهيئة متغيرات المؤشر:

بما أن C++ لا تقوم تلقائياً بتهيئة المتغيرات يجب تهيئة متغيرات المؤشر اذا لم تكن تريدها تشير الى أي شيء. يتم تهيئة متغيرات المؤشر باستخدام القيمة الثابتة صفر التي يطلق عليها **مؤشر للا شيء**. لهذا يقوم البيان $p = 0$ بتخزين مؤشر اللا شيء في p أي أن p تشير الى لا شيء. يقوم بعض المبرمجين باستخدام الثابت المسمى NULL لتهيئة متغيرات المؤشر. البيانان التاليان متكافئان:

$p = \text{NULL};$

$p = 0;$

العدد صفر هو العدد الوحيد الذي يمكن تحديده بشكل مباشر لمتغير المؤشر.

متغيرات حركية:

في الأقسام السابقة تعلمت كيفية اعلان متغيرات المؤشر، وكيفية تخزين عنوان المتغير داخل متغير مؤشر من نفس نوع المتغير، وكيفية التعامل في البيانات المستخدمة للمؤشرات. بالرغم من هذا تعلمت كيفية استخدام المؤشرات للتلاعب في البيانات فقط داخل مساحات الذاكرة التي تم عملها باستخدام متغيرات أخرى. بمعنى آخر تقوم المؤشرات بالتلاعب في البيانات داخل مساحات الذاكرة. اذن ما هي المشكلة الكبيرة بشأن استخدام المؤشرات؟ يمكنك تناول مساحات الذاكرة هذه عن طريق العمل مع المتغيرات التي تم استخدامها لعملها.

في هذا القسم سوف نتعلم القوة التي تكون وراء المؤشرات وسوف نتعلم بوجه خاص كيفية تخصيص الذاكرة والغاءها أثناء تنفيذ البرنامج باستخدام المؤشرات.

ان المتغيرات التي يتم خلقها أثناء تنفيذ البرنامج يطلق عليها **متغيرات حركية**. بمساعدة المؤشرات تقوم C++ بخلق متغيرات حركية كما تقوم C++ بتوفير معاملين new و delete – لعمل وتدمير متغيرات حركية على التوالي. عندما يتطلب برنامج ما متغير حركي جديد يتم استخدام المعامل New وعندما يصبح البرنامج في غير حاجة الى متغير حركي يتم استخدام المعامل delete.

في C++ تكون كل من new و delete كلمات محفوظة. المعامل new له صيغتان: صيغة لتخصيص متغير واحد، وصيغة لتخصيص مصفوفة من المتغيرات ويكون تركيب استخدام المعامل new هو:

| | |
|------------------------|--------------------------|
| New dataType; | // لتخصيص متغير واحد |
| New dataType [intExp]; | // لتخصيص مصفوفة متغيرات |

حيث intExp عبارة عن أي تعبير يتم تقييمه بعدد صحيح موجب. المعامل new يخصص ذاكرة (متغير) من النوع المخصص وينتج مؤشر اليه – أي عنوان هذه الذاكرة المخصصة. فضلاً عن هذا لا يتم تهيئة الذاكرة المخصصة.

انظر الى الاعلان التالي:

```
int    *p;
char    *q;
int     x;
البيان:
p = &x;
```

يقوم بتخزين عنوان x في p. ومع هذا لا يتم تخصيص ذاكرة جديدة. على الجانب الآخر انظر الى الاعلان التالي:

```
p = new int;
```

يقوم هذا البيان بتخصيص مكان من الذاكرة من النوع int أثناء تنفيذ البرنامج في مكان ما في الذاكرة ويقوم بتخزين الذاكرة المخصصة في p. يتم تناول الذاكرة المخصصة عبر المؤشر المسمى *p. بالمثل يقوم البيان:

```
q = new char [16];
```

بخلق مصفوفة من 16 مكون من النوع char وتخزين العنوان الأساسي للمصفوفة في q.

بما أن المتغير الحركي غير مسمى لا يمكن تناوله بشكل مباشر بل يتم تناوله بشكل غير مباشر بواسطة المؤشر الناتج من new. البيان التالي يوضح هذه الفكرة:

```
int *p;           //p is a pointer of the type int
char *name;       //name is a pointer of the type char

string *str;      //str is a pointer of the type
                  //string

p = new int;      //Allocates memory of the type
                  //int and stores the address of
                  //the allocated memory in p.
*p = 28;          //Stores 28 in the allocated
                  //memory.

name = new char[5]; //Allocates memory for an array
                  //of five components of the type
                  //char and stores the base
                  //address of the array in name.
strcpy(name, "John"); //Stores John in name.

str = new string;  //Allocates memory of the type
                  //string and stores the address
                  //of the allocated memory in
                  //str.
*str = "Sunny Day"; //Stores the string "Sunny Day"
                  //in the memory pointed to by
                  //str.
```

| | |
|-----------------------|---|
| int *p; | // p مؤشر من النوع int |
| char *name; | // name مؤشر من النوع char |
| string *str; | // str مؤشر من النوع string |
| p = new int; | // تخصص ذاكرة من النوع int |
| *p = 28; | // وتخزن عنوان الذاكرة المخصصة في p |
| name = new char[5]; | // يخزن الرقم في الذاكرة المخصصة |
| strcpy(name, "John"); | // تخصص ذاكرة لمصفوفة من |
| str = new string; | // خمس مكونات من النوع char |
| *str = "Sunny Day"; | // ويخزن العنوان الأساسي للمصفوفة في name |
| | // يقوم بتخزين John في name (الاسم) |
| | // تخصص ذاكرة من النوع string |
| | // ويخزن عنوان الذاكرة المخصصة في str |
| | // يخزن المقطع "Sunny Day" في الذاكرة |
| | // المشار إليها بواسطة str. |

عندما لا يعود هناك حاجة الى المتغير الحركي يمكن تدميره أي أنه يتم ازالة تخصيص ذاكرته. يتم استخدام المعامل delete لتدمير المتغيرات الحركية. تركيب استخدام المعامل delete له صيغتان هنا:

| | |
|---------------------|----------------------------------|
| delete pointer; | // لتدمير متغير حركي واحد |
| delete [] pointer; | // لتدمير مصفوفة تم عملها حركياً |

لهذا تقوم البيانات:

delete p;

delete [] name;

بازالة تخصيص الذاكرة المشار اليها بواسطة المؤشرات p و name.

عمليات على متغيرات المؤشر:

العمليات التي يتم السماح باجرائها على متغيرات المؤشر هي التحديد والعمليات المرتبطة بها وكذلك بعض العمليات الحسابية المحدودة. يمكن تحديد قيمة متغير مؤشر واحد عند متغير مؤشر واحد من نفس النوع. يمكن المقارنة بين متغيرين مؤشر من هذا النوع من أجل التساوي وهكذا. يمكن جمع وطرح قيم صحيحة من متغير مؤشر ما ويمكن طرح قيمة متغير مؤشر واحد من متغير مؤشر آخر. على سبيل المثال افترض أن لدينا البيانات التالية:

int *p, *q;

يقوم البيان:

p = q;

بنسخ قيمة q داخل p. بعد تنفيذ هذا البيان يشير الى كل من p و q الى مكان الذاكرة ذاته. أي تغييرات يتم اجرائها على *p يقوم تلقائياً بتغيير قيمة *q والعكس صحيح. التعبير:

p == q

ينتج true اذا كان كل من p و q يملكان نفس القيمة – أي اذا كانوا يشيرون الى نفس مكان الذاكرة. بالمثل يقوم التعبير:

p ↓= q

بانتاج true اذا كان p و q يشيرون الى مواقع مختلفة من الذاكرة. العمليات الحسابية المسموح بها تختلف عن العمليات الحسابية على الأرقام. لنقم أولاً باستخدام البيانات التالية لشرح عمليات الزيادة والنقص على متغيرات المؤشر:

```
int *p;
double *q;
char *chPtr;
studentType *stdPtr; //studentType is as defined before
```

عادةً يكون حجم الذاكرة المخصص للمتغير `int` 4 بايت وحجم المتغير `double` يكون 8 بايت وحجم المتغير `Char` يكون 1 بايت. الذاكرة المخصصة للمتغير من النوع `studentType` يكون 40 بايت. البيان:

```
p++; أو p = p + 1;
```

يزيد قيمة `p` بمقدار 4 بايت لأن `p` مؤشر من النوع `int`. بالمثل تقوم البيانات:

```
q++;
```

```
chPtr++;
```

بزيادة قيمة `stdPtr` بمقدار 40 بايت.

يقوم معامل الزيادة بزيادة قيمة متغير مؤشر بحجم الذاكرة التي يشير إليها. بالمثل يقوم معامل النقص بتقليل قيمة متغير المؤشر بحجم الذاكرة التي يشير إليها.

فضلاً عن هذا يقوم البيان

```
p = p + 2;
```

بزيادة قيمة `p` بمقدار 8 بايت.

لهذا عندما تتم إضافة عدد صحيح إلى متغير مؤشر تزداد قيمة متغير المؤشر بعدد صحيح من حجم الذاكرة التي يشير إليها المؤشر. بالمثل عندما يتم طرح عدد صحيح من متغير مؤشر يتم تقليل قيمة متغير المؤشر بعدد صحيح من حجم الذاكرة التي يشير إليها المؤشر.

إذا لم تكن حريص قد يكون المؤشر شديد الخطورة. باستخدام المؤشر يمكن للبرنامج تناول مواقع الذاكرة من متغيرات أخرى ويغير محتوياتها بدون انذار تاركة المبرمج في حالة من الاختلاف ومحاولاً إيجاد الخطأ الذي حدث. إذا حاول متغير مؤشر ما تناول اما مواقع ذاكرة متغيرات أخرى أو مكان ذاكرة غير صحيح فان هناك بعض أنظمة توقف البرنامج برسالة خطأ مناسبة دائماً اعطي اهتمام زائد عند عمل المؤشرات.

مصفوفات حركية:

المصفوفات التي قمنا باستخدامها حتى الآن تسمى مصفوفات ساكنة لأنه تم تثبيت حجمها في زمن التجميع. من أحد قيود المصفوفات الساكنة أنه في كل مرة تستخدم فيها البرنامج يتم تثبيت حجم المصفوفة ولهذا قد يكون من غير الممكن استخدام نفس المصفوفة لمعالجة مجموعات بيانات مختلفة من نفس النوع. من أحد طرق التعامل مع هذا القيد هو اعلان أن المصفوفة كبيرة بشكل يكفي لمعالجة مجموعة متنوعة من مجموعات البيانات. بالرغم من هذا إذا كانت المصفوفة كبيرة للغاية ومجموعة البيانات صغيرة فقد يتسبب هذا الاعلان في ضياع الذاكرة. من الجانب الآخر قد يكون مفيد للغاية أثناء تنفيذ البرنامج أنه يمكنك حث المستخدم على ادخال حجم المصفوفة ثم عمل مصفوفة ذات حجم

مناسب. هذه الطريقة تكون مفيدة بوجه خاص اذا لم يمكنك حتى تخمين حجم المصفوفة أثناء كتابة البرنامج. في هذا القسم نتعلم كيفية عمل مصفوفات أثناء تنفيذ البرنامج ومعالجة تلك المصفوفات. المصفوفة التي يتم عملها أثناء تنفيذ البرنامج تسمى **مصفوفة حركية**. من أجل عمل مصفوفة حركية نقوم باستخدام الصيغة الثانية من المعامل new – المعامل الذي يخصص مصفوفة من المتغيرات.

البيان:

```
int *p;
```

يعلن أن p متغير مؤشر من النوع int. البيان:

```
p = new int [10];
```

يخصص 10 مواقع متتالية من الذاكرة كل منها من النوع int ويقوم بتخزين عنوان موقع الذاكرة الأولى داخل p. بمعنى آخر يصنع المعامل new مصفوفة من 10 مكونات من النوع int وينتج العنوان الأساسي للمصفوفة ويقوم معامل التحديد بتخزين العنوان الأساسي للمصفوفة في p. لهذا يقوم

البيان:

```
*p = 25;
```

بتخزين 25 داخل موقع الذاكرة الأول والبيانات:

```
p++; // يشير الى مكون المصفوفة التالي.
```

```
*p = 35;
```

تقوم بتخزين 35 داخل موقع الذاكرة الثاني. لهذا يمكنك باستخدام عمليات الزيادة والنقص تناول مكونات المصفوفة. بعد أداء القليل من عمليات الزيادة يكون من الممكن بالطبع فقدان تتبع مكون المصفوفة الأول. C++ يسمح لنا باستخدام تدوين المصفوفة لتناول مواقع الذاكرة هذه. على سبيل المثال تقوم البيانات:

```
p [0] = 25;
```

```
p [1] = 35;
```

بتخزين 25 و 35 داخل مكونات المصفوفة الأول والثاني على التوالي. هذا يعني أن p [0] تشير الى مكون المصفوفة الأول و p [1] تشير الى مكون المصفوفة الثاني وهكذا. بوجه عام يشير p[i] الى مكون المصفوفة (i + 1). بعد تنفيذ البيانات السابقة تظل p تشير الى مكون المصفوفة الأول. بالمثل تقوم الحلقة for بتهيئة كل مكون من المصفوفة عند صفر:

```
for(j = 0; j < 10; j++)  
    p[j] = 0;
```

حيث j عبارة عن متغير int.

عندما يتم استخدام تدوين المصفوفة لمعالجة المصفوفة المشار إليها بواسطة p تبقى p ثابتة عند موقع الذاكرة الأول. فضلاً عن هذا تكون p عبارة عن مصفوفة تم عملها أثناء تنفيذ البرنامج ويطلق عليها مصفوفة حركية.

البيان:

```
int list [10];
```

يعلن أن list عبارة عن مصفوفة مكونة من 10 مكونات. هناك مكان بالذاكرة يسمى list والقيمة التي يتم تخزينها في مكان الذاكرة هذا يكون العنوان الأساسي للمصفوفة وهو عنوان مكون المصفوفة الأول. بما أن مكان الذاكرة المسمى list يتضمن عنوان فان list يكون عبارة عن مؤشر. بالرغم من هذا لا يمكن تطبيق معاملات الزيادة والنقص على list لأننا دائماً نريد أن تشير list الى مكون المصفوفة الأول. أي محاولة لاستخدام عمليات الزيادة والنقص على list تتسبب في خطأ في زمن التجميع. فضلاً عن هذا اذا كان P متغير مؤشر ما من النوع int فان البيان:

```
p = list;
```

ينسخ قيمة list وهو العنوان الأساسي للمصفوفة داخل p. يجوز لنا أداء عمليات الزيادة والنقص على p.

ان أي اسم مصفوفة مثل list يكون عبارة عن مؤشر ثابت ويشير دائماً الى المكون الأول من المصفوفة. فضلاً عن هذا عندما تقوم باعلان مصفوفة مثل list فانها تشير دائماً الى نفس المصفوفة.

مثال 3-4:

البرنامج التالي يوضح كيفية الحصول على اجابة المستخدم للحصول على حجم المصفوفة وخلق مصفوفة حركية أثناء تنفيذ البرنامج. انظر الى البيانات التالية:

```
int *intList;           //Line 1
int arraySize;          //Line 2

cout<<"Enter the array size: "; //Line 3
cin>>arraySize;         //Line 4
cout<<endl;             //Line 5

intList = new int[arraySize]; //Line 6
```

البيان في الصف 1 يعلن أن intList مؤشر من النوع int، والبيان في الصف 2 يعلن أن arraySize متغير int. البيان في الصف 3 يحث المستخدم على ادخال حجم المصفوفة، والبيان في الصف 4 يدخل حجم المصفوفة بداخل المتغير arraySize. البيان في الصف 6 يخلق مصفوفة للحجم محددة بواسطة arraySize ويتم تخزين العنوان الأساسي للمصفوفة في intList. بدءاً من هذه النقطة يمكنك التعامل

مع `intList` كأى مصفوفة أخرى. على سبيل المثال يمكنك استخدام تدوين المصفوفة لمعالجة عناصر `intList` وتمرير `intList` كعامل للدالة.

الدالات والمؤشرات:

يمكن تمرير متغير مؤشر ما كعامل للدالة اما بالقيمة أو الاشارة. لاعلان كؤشر ما معامل قيمة في عنوان دالة ما يمكنك استخدام نفس الآلية التي تستخدمها لاعلان متغير ما. لجعل عامل رسمي عامل اشارة تقوم باستخدام العلامة (&) عندما تعلن العامل الرسمي في عنوان الدالة. لهذا يجب عليك استخدام العلامة & للاعلان عن عامل رسمي كعامل اشارة. بين اسم نوع البيانات واسم المحدد يجب أن تقوم بادخال كل من علامة النجمة (*) لجعل المحدد مؤشر والعلامة (&) لجعله عامل اشارة. السؤال الواضح هو: بأي ترتيب تظهر العلامات & و* بين اسم نوع البيانات والمحدد لاعلان المؤشر كعامل اشارة؟ في C++ لكي تجعل المؤشر عامل اشارة في عنوان دالة تظهر علامة النجمة * قبل العلامة & بين اسم نوع البيانات والمحدد. المثال التالي يوضح هذا المفهوم:

```
void example(int* &p, double *q)
{
    .
    .
    .
}
```

في هذا المثال يكون كل من p و q مؤشرين. العامل p عبارة عن عامل اشارة والعامل q عبارة عن عامل قيمة.

المؤشرات وقيم انتاج الدالة:

في C++ يمكن أن تنتج الدالة قيمة من النوع المؤشر. على سبيل المثال يكون نوع انتاج الدالة هو:

```
int* testExp(...)
{
    .
    .
    .
}
```

مؤشر من النوع `int`.

النسخ السطحي في مقابل النسخ العميق والمؤشرات:

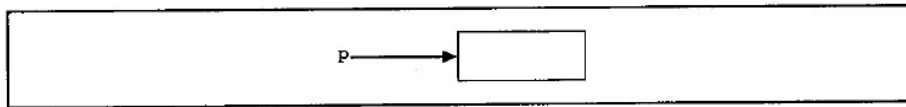
قسم "العمليات على متغيرات المؤشر" الموجود سابقاً في هذا الفصل ناقش المؤشرات ووضح أننا اذا لم نكن حذرين فقد يتناول مؤشر ما بيانات مؤشر آخر (غير مرتبط به كلياً). هذا قد يتسبب في نتائج

غير مشكوك بها أو نتائج مغلوبة. هنا نناقش سمة أخرى من سمات المؤشرات. لتسهيل هذه المناقشة سوف نستخدم رسومات مصورة لتوضيح المؤشرات والذاكرة المرتبطة بها. انظر الى البيانات التالية:

```
int *p;
```

```
p = new int;
```

البيان الأول يعلن أن p متغير مؤشر من النوع `int` والبيان الثاني يخصص ذاكرة من النوع `int` ويتم تخزين عنوان الذاكرة المخصصة في p . شكل 8-3 يوضح هذا الموقف.

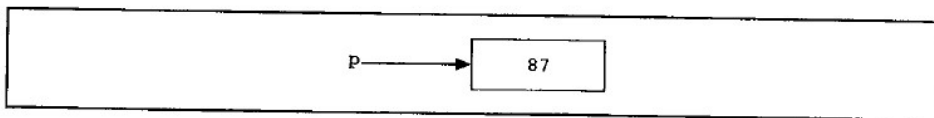


شكل 8-3: المؤشر p والذاكرة التي يشير إليها.

يشير الصندوق الى الذاكرة المخصصة (في هذه الحالة من النوع `int`). ان المؤشر p مع السهم تشير الى الذاكرة المخصصة. انظر الآن الى البيان التالي:

```
*p = 87;
```

هذا البيان يخزن 87 في الذاكرة المشار إليها بواسطة p . الشكل 9-3 يوضح هذا الموقف.



شكل 9-3: المؤشر p مع 87 في الذاكرة التي يشير إليها.

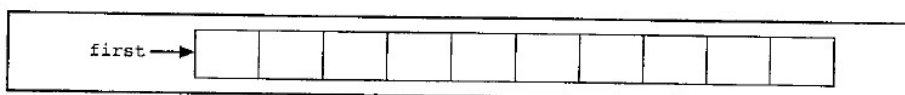
انظر الى البيانات التالية:

```
int *first;
```

```
int *second;
```

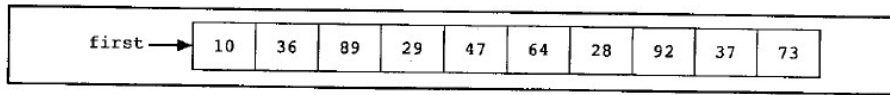
```
first = new int[10];
```

البيانان الأوليان يعلنان أن `first` و `second` متغيرات مؤشر من النوع `int`. البيان الثالث يخلق مصفوفة من 10 مكونات ويتم تخزين العنوان الأساسي للمصفوفة داخل `first`. انظر شكل 10-3.



شكل 10-3: المؤشر `first` والمصفوفة التي يشير إليها.

افترض أن بعض البيانات ذات المعنى مخزنة في مصفوفة مشار إليها بواسطة `first`. لكي تكون محدداً افترض أن هذه المصفوفة كما هو مشار إليها في الشكل 11-3.

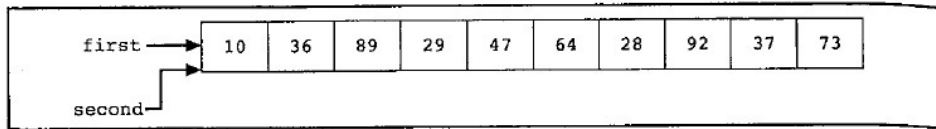


شكل 3-11: المؤشر first والمصفوفة التي يشير إليها.

بعد هذا انظر الى البيان التالي:

`second = first;` //Line A

هذا البيان ينسخ قيمة first داخل second. بعد تنفيذ هذا البيان يشير كل من first و second الى نفس المصفوفة كما هو موضح في شكل 3-12.



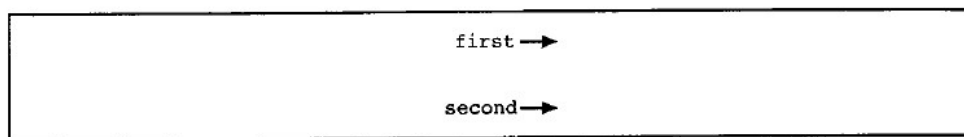
شكل 3-12: first و second بعد تنفيذ البيان `second = first`.

لاحظ أن البيان في الصف A جائز بالرغم من أن المؤشر first يشير الى مصفوفة. هذا لأن كل من first و second متغيرات مؤشر من نفس النوع. على الجانب الآخر اذا كانت list1 و list2 مصفوفات من نفس النوع اذن يكون list1 و list2 مؤشرات ثابتة وكذلك البيان `list1 = list2` غير جائز. في مثل هذه الحالة نقول كذلك أنه لا توجد عملية تحديد كلي للمصفوفات. العملية الكلية على المصفوفة هي العملية التي تتلاعب بالمصفوفة الكلية كوحدة واحدة.

لنقم بعد هذا بتنفيذ البيان التالي:

`delete [] second;`

بعد تنفيذ هذا البيان تتم ازالة المصفوفة المشار اليها بواسطة second. هذا التصرف يتسبب في الشكل 3-13.



شكل 3-13: first و second بعد تنفيذ البيان `delete [] second;`

بما أن first و second أشارت الى نفس المصفوفة فان بعد تنفيذ البيان:

`delete [] second;`

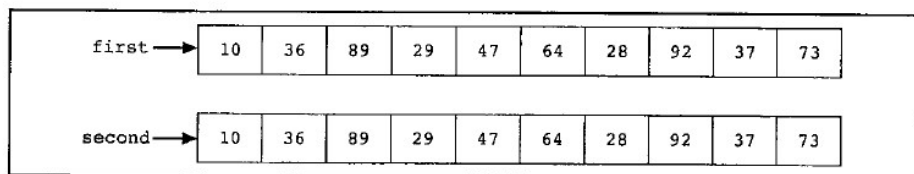
تصبح first غير صحيحة. ولهذا اذا حاول البرنامج بعد هذا أن يتناول الذاكرة المشار اليها بواسطة first فاما أن البرنامج قد يتناول الذاكرة الخطأ أو قد يقع في خطأ ما. هذه الحالة مثال على النسخ

السطحي. في النسخ السطحي يشير اثنان أو أكثر من المؤشرات من نفس النوع الى نفس الذاكرة أي أنهم يشيرون الى نفس البيانات كما هو موضح في شكل 3-12. من الجهة الأخرى افترض أن لدينا البيانات التالية بدلاً من البيان السابق; `second = first` (في الصف A):

```
second = new int[10];

for(int j = 0; j < 10; j++)
    second[j] = first[j];
```

البيان الأول يصنع مصفوفة من 10 مكونات من النوع `int` ويتم تخزين عنوان المصفوفة الأساسي في `second`. البيان الثاني ينسخ المصفوفة المشار إليها بواسطة `first` داخل المصفوفة المشار إليها بواسطة `second`. انظر شكل 3-14.



شكل 3-14: كل من `first` و `second` يشير الى البيانات الخاصة بها. الآن يشير كل من `first` و `second` الى بياناتهم الخاصة. اذا قامت `second` بحذف ذاكرتها لن يكون هناك تأثير على `first`. هذه الحالة مثال على **النسخ العميق**. في النسخ العميق يشير كل مؤشر الى بياناته الخاصة كما هو موضح في شكل 3-14 وليس نفس البيانات كما في شكل 3-12. من المناقشة السابقة ينتج أنك يجب أن تعلم الوقت الذي تستخدم فيه النسخ السطحي والوقت الذي تستخدم فيه النسخ العميق.

الفئات والمؤشرات: بعض الخصائص:

اذا كان متغير المؤشر من نوع الفئة فقد ناقشنا في القسم السابق كيفية تناول عنصر الفئة عبر المؤشر باستخدام تدوين السهم. بما أن الفئة يمكنها امتلاك عناصر بيانات مؤشر فان هذا القسم يصف بعض خصائص لهذه الفئات.

لتسهيل المناقشة نستخدم الفئة التالية:

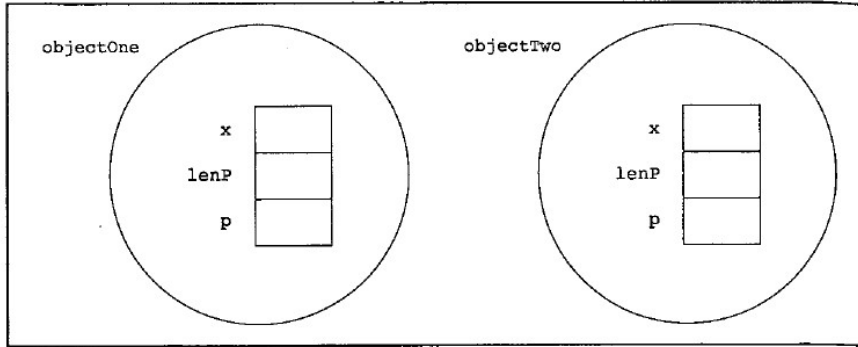
```
class pointerDataClass
{
public:
    ...

private:
    int x;
    int lenP;
    int *p;
};
```

انظر أيضاً الى البيانات التالية (انظر شكل 3-15):

```
pointerDataClass objectOne;
```

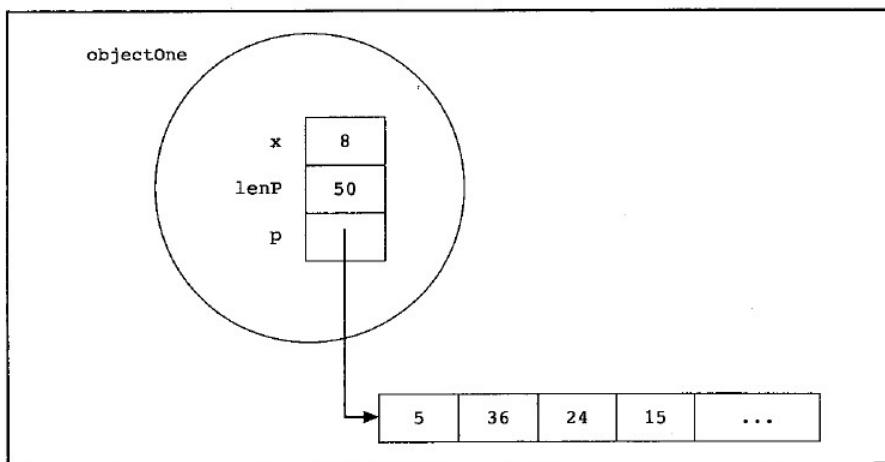
```
pointerDataClass objectTwo;
```



شكل 3-15: الأهداف objectOne و objectTwo

المدمر:

الهدف objectOne له عنصر بيانات مؤشر p. افترض أنه أثناء تنفيذ البرنامج يخلق المؤشر p مصفوفة حركية. عندما يخرج objectOne من المجال يتم تدمير جميع عناصر بيانات objectOne. بالرغم من هذا فقد خلقت p مصفوفة حركية ويجب ازالة تخصيص الذاكرة الحركية باستخدام المعامل delete. لهذا اذا لم يستخدم المؤشر p المعامل delete لازالة تخصيص المصفوفة الحركية فان مساحة ذاكرة المصفوفة الحركية سوف يبقى محدد كمساحة مخصصة حتى اذا لم يكن هناك ما يستطيع تناولها. هذا يعرف ب"تسرب الذاكرة". كيف نضمن أنه عندما يتم تدمير p يتم كذلك تدمير الذاكرة الحركية التي صنعها p؟ افترض أن objectOne يكون كما هو موضح في الشكل 3-16.



شكل 3-16: الهدف objectOne وبياناته.

تذكر أن اذا كانت الفئة لها مدمر فانه يجري تلقائياً في أي وقت يخرج فيه هدف الفئة عن المجال (انظر الفصل 1). لهذا يمكننا وضع الكود اللازم في المدمر لضمان أن عند خروج objectOne من المجال يتم ازالة تخصيص الذاكرة التي تم عملها بواسطة المؤشر p. هذا يعتبر واحد من الأهداف الرئيسية لادخال المدمر. على سبيل المثال يكون تعريف المقوم الخاص بالفئة pointerDataClass هو:

```
{
Delete [ ] p;
}
```

بالطبع يجب أن تدخل المدمر كعنصر للفئة في تعريفها. لنقم بالتوسع في تعريف الفئة pointerDataClass عن طريق ادخال المقوم. فضلاً عن هذا تفترض بقية هذا القسم أن تعريف المدمر كما هو معطى من قبل – أي يقوم المدمر بازالة تخصيص مساحة الذاكرة المشار اليها بواسطة p.

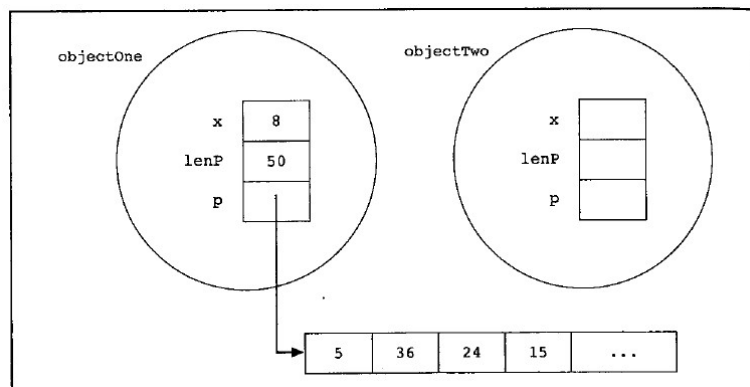
```
class pointerDataClass
{
public:
    ~pointerDataClass();
    ...

private:
    int x;
    int lenP;
    int *p;
};
```

لكي يعمل المدمر بشكل صحيح يجب أن يكون للمؤشر p قيمة صحيحة. اذا لم يتم تهيئة p بشكل صحيح (أي أن قيمة p مهمة) وتم تنفيذ المدمر فاما أن يقوم البرنامج بالتوقف مع ظهور رسالة خطأ أو يقوم المدمر بازالة تخصيص مساحة الذاكرة الغير مرتبطة. لهذا السبب يجب أن تمارس المزيد من الحذر عند العمل بالمؤشرات.

معامل الاسناد:

يقوم هذا القسم بوصف قيود معاملات الاسناد المدمجة للفئات ذات عناصر بيانات المؤشر. افترض أن objectOne و objectTwo كما هما موضحين في الشكل 3-17.

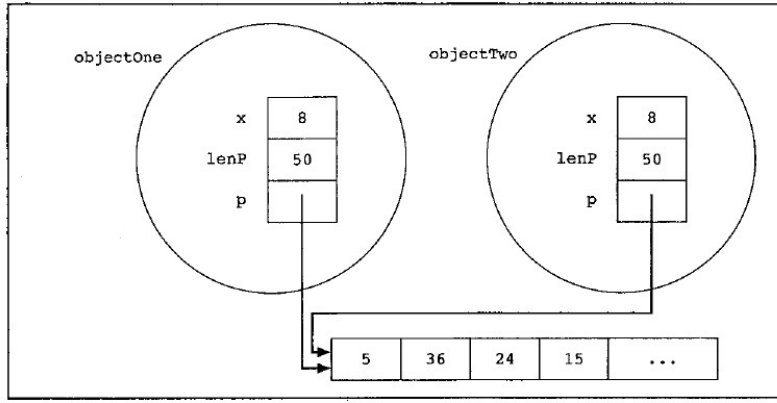


شكل 3-17: الأهداف objectOne و objectTwo.

تذكر أن واحدة من العمليات المدمجة المؤداة على الفئات هي معامل الاسناد. على سبيل المثال يقوم البيان:

objectTwo = objectOne;

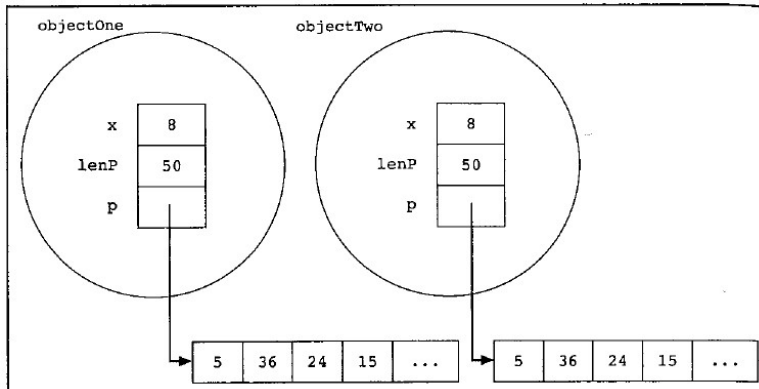
بنسخ عناصر بيانات objectOne بداخل objectTwo. هذا يعني أنه يتم نسخ قيمة objectOne.x بداخل objectTwo.x ويتم نسخ قيمة objectOne.lenP داخل objectTwo.lenP ويتم نسخ قيمة objectOne.p بداخل objectTwo.p. بما أن p مؤشر فان هذا النسخ الحكيم للبيانات سوف يؤدي الى نسخ سطحي للبيانات. أي أن كلا من objectTwo.p و objectOne.p سوف يشيران الى نفس مساحة الذاكرة كما هو موضح في شكل 3-18.



شكل 3-18: الأهداف objectOne و ObjectTwo بعد تنفيذ البيان

objectTwo = ObjectOne

الآن اذا قام objectTwo.p بازالة تخصيص مساحة الذاكرة التي يشير اليها فسوف يصبح objectOne.p غير صحيح. وهذا الموقف قد يحدث اذا كان للفئة pointerDataClass مدمر يقوم بازالة تخصيص مساحة الذاكرة المشار اليها بواسطة p عندما يخرج هدف من النوع pointerDataClass من المجال. هذا يقترح أنه يجب أن تكون هناك طريقة لتجنب هذا المأزق. لتجنب هذا النسخ السطحي للبيانات بالنسبة الى الفئات ذات عنصر بيانات المؤشر تسمح C++ للمبرمج بالتوسع في تعريف معامل الاسناد وتعرف هذه العملية باثقال معامل الاسناد. بمجرد اثقال معامل الاسناد بشكل صحيح يكون لكل من objectOne و objectTwo البيانات الخاصة بهما كما هو موضح في شكل 3-19.



شكل 3-19: الأهداف objectOne و objectTwo.

انقال معامل الاسناد:

بعد هذا نصف كيفية انقال معامل الاسناد.

التركيب العام لانقال معامل الاسناد (=) لفئة ما:

نموذج الدالة (التي يتم ادخالها في تعريف الفئة):

```
const className& operator = (const className&);
```

تعريف الدالة:

```
const className& className::operator=(const className& rightObject)
{
    //local declaration, if any

    if(this != &rightObject) //avoid self-assignment
    {
        //algorithm to copy rightObject into this object
    }

    //return the object assigned
    return *this;
}
```

في تعريف معامل الدالة =:

- يوجد عامل رسمي واحد فقط.
- العامل الرسمي يكون بوجه عام اشارة const الى فئة محددة.
- نوع انتاج الدالة يكون عبارة عن اشارة الى فئة محددة.

انظر الى البيان التالي:

```
x = x;
```

هنا نحاول نسخ قيمة x داخل x أي أن هذا البيان عبارة عن اسناد ذاتي. يجب أن نمنع مثل تلك البيانات لأنها تضيق وقت الحاسب الآلي.

هيكल معامل الدالة = يمنع مثل هذه الاسنادات. لنرى كيفية حدوث هذا.

قم بالتفكير في البيان if في هيكل معامل الدالة =:

```
if(this != &rightObject) //avoid self-assignment
{
    //algorithm to copy rightObject into this object
}
```

الآن يتم تجميع البيان:

```
x = x;
```

داخل البيان:

```
x.operator = (x);
```

بما أن معامل الدالة = يتم استدعاؤه بواسطة الهدف x فان المؤشر this في هيكل معامل الدالة = يشير الى الهدف x. فضلاً عن هذا وبما أن x عبارة كذلك عن عامل لمعامل الدالة = فان العامل الرسمي rightObject يشير أيضاً الى الهدف x. لهذا فانه في التعبير:

this 1 = &rightObject

this تعني عنوان x كما تعني &rightObject عنوان x أيضاً. لهذا سوف يقدم هذا التعبير false ولهذا سوف يتم اسقاط هيكل البيان if.

لاحظ أن نوع انتاج الدالة لاثقال معامل الاسناد يكون عبارة عن اشارة. هذا حتى يمكن تنفيذ بيانات مثل $x = y = z$; أي يمكن استخدام معامل الاسناد في صيغة متعاقبة.

المثال التالي يوضح كيفية اثقال معامل الاسناد.

مثال 5-3:

انظر الى الفئة التالية:

```
class cAssignmentOprOverload
{
public:
    const cAssignmentOprOverload& operator=
        (const cAssignmentOprOverload& otherList);
        //Overload the assignment operator.

    void print() const;
        //Function to print the list.

    void insertEnd(int item);
        //Function to insert an item at the end of the list.
        //Postcondition: If the list is not full, length++;
        //                  list[length] = item
        //                  If the list is full, outputs an
        //                  appropriate message.

    void destroyList();
        //Function to destroy the list.
        //Postcondition: length = 0; maxSize = 0;
        //                  list = NULL

    cAssignmentOprOverload(int size = 10);
        //constructor
        //Postcondition: length = 0; maxSize = size;
        //                  list is an array of size maxSize

private:
    int maxSize;
    int length;
    int *list;
};
```

تعريفات دالات العنصر للفئة cAssignmentOprOverload هي:

```

void cAssignmentOprOverload::print() const
{
    if(length == 0)
        cout<<"List is empty."<<endl;
    else
    {
        for(int i = 0; i < length; i++)
            cout<<list[i]<<" ";
        cout<<endl;
    }
}

void cAssignmentOprOverload::insertEnd(int item)
{
    if(length == maxSize)
        cout<<"List is full."<<endl;
    else
        list[length++] = item;
}

void cAssignmentOprOverload::destroyList()
{
    delete [] list;
    list = NULL;
    length = 0;
    maxSize = 0;
}

cAssignmentOprOverload::cAssignmentOprOverload(int size)
{
    length = 0;

    if(size <= 0)
        maxSize = 10;
    else
        maxSize = size;

    list = new int[maxSize];
    assert(list != NULL);
}

const cAssignmentOprOverload& cAssignmentOprOverload::operator=
    (const cAssignmentOprOverload& otherList)
{
    if(this != &otherList) //avoid self-assignment;    //Line 1
    {
        if(list != NULL)                                //Line 2
            destroyList();                               //Line 3
        maxSize = otherList.maxSize;                    //Line 4
        length = otherList.length;                     //Line 5

        if(maxSize != 0)                                //Line 6
        {
            list = new int[maxSize];                    //Line 7
            assert(list != NULL);                        //Line 8

            for(int i = 0; i < length; i++)              //Line 9
                list[i] = otherList.list[i];            //Line 10
        }
        else                                             //Line 11

            list = NULL;                                //Line 12
    }

    return *this;                                       //Line 13
}

```

الدالة الخاصة بانتقال معامل الاسناد تكون كما يلي. البيان في الصف 1 يتأكد مما اذا كان الهدف ينسخ نفسه والبيان في الصف 2 يتأكد مما اذا كانت list (القائمة) غير فارغة. اذا كانت غير فارغة يتم تدمير القائمة عن طريق ازالة تخصيص الذاكرة التي يحتلها list (القائمة). البيانات في الصفوف 4 و 5 تقوم بنسخ قيم عناصر البيانات maxSize و length of otherList داخل length و maxSize of otherList على التوالي. اذا لم تكن otherList (القائمة الأخرى) صفراً أو غير خالية فان البيانات بين الصفوف 6 و 10 تصنع قائمة المصفوفة وتنسخ otherList داخل list. اذا كانت otherList صفراً يتم تهيئة list عند صفر. لاحظ أن البرنامج اذا لم يكن قادراً على تخصيص ذاكرة للمصفوفة فان البيان الموجود في الصف 8 يوقف البرنامج.

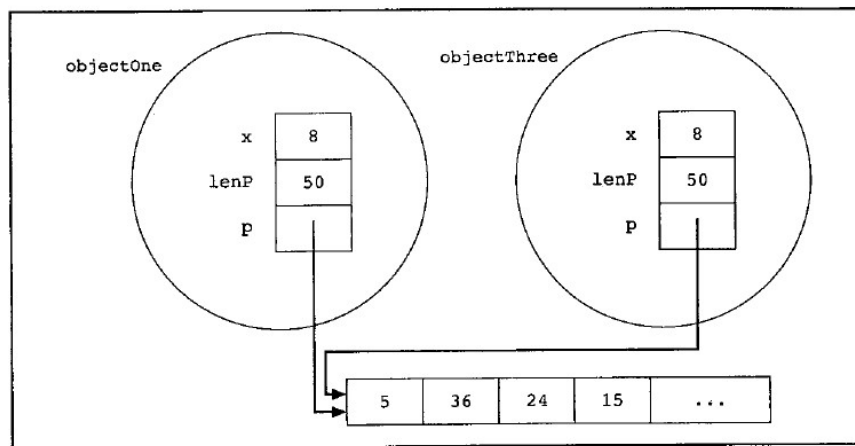
البيان في الصف 13 ينتج عنوان هذه القائمة لأن نوع انتاج معامل الدالة = عبارة عن نوع اشارة. اننا نتركه كتمرين لك لكي تكتب برنامج لاختبار معامل الاسناد للفئة c.AssignmentOprOverload

مقوم النسخ:

عند اعلان هدف فئة ما يمكنك تهيئته بواسطة استخدام قيمة هدف متواجد من نفس النوع. على سبيل المثال انظر الى البيان التالي:

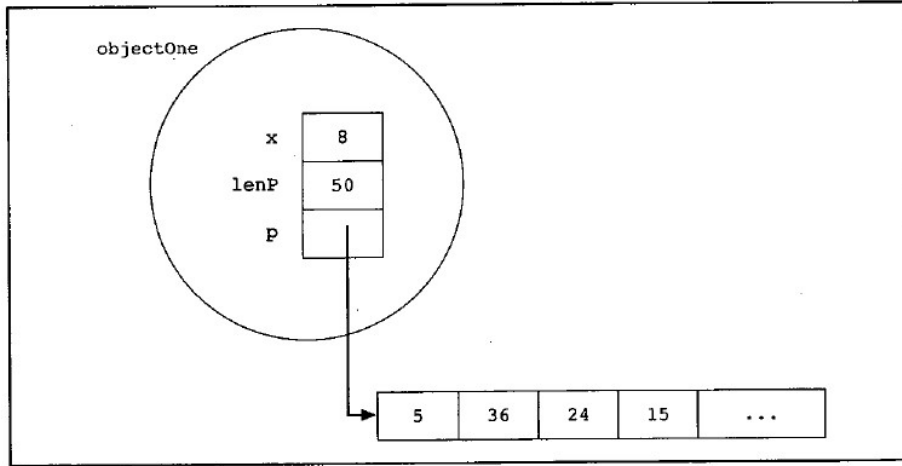
pointerDataClass objectThree (objectOne);

يتم اعلان الهدف objectThree ويتم تهيئته باستخدام قيمة objectOne. أي أنه يتم نسخ قيم عناصر بيانات objectOne داخل عناصر البيانات المتوافقة من objectThree. هذه التهيئة تسمى **تهيئة العنصر الافتراضية**. تهيئة العنصر الافتراضية بسبب المقوم تسمى **مقوم النسخ** (مقدم بواسطة المجمع). كما في حالة معامل الاسناد قد تؤدي هذه التهيئة الافتراضية الى وجود نسخ سطحي للبيانات كما هو موضح في شكل 3-20 لأن الفئة pointerDataClass لها عناصر بيانات مؤشر (افترض أن objectOne معطى كما سبق).



شكل 3-20: الاهداف objectOne و objectThree.

قبل وصف كيفية التغلب على هذا العيب لنقم بوصف موقف آخر الذي قد يقود كذلك الى حدوث نسخ سطحي للبيانات. الحل لكل من هاتين المشكلتين هو حل واحد. تذكر أن أهداف الفئة كمعاملات للدالة يمكن تمريرها اما بالاشارة أو بالقيمة. تذكر أن الفئة pointerDataClass لها مقوم يقوم بازالة تخصيص مساحة الذاكرة المشار اليها بواسطة p. افترض أن objectOne كما هو موضح في شكل 3-21.



شكل 3-21: الهدف objectOne.

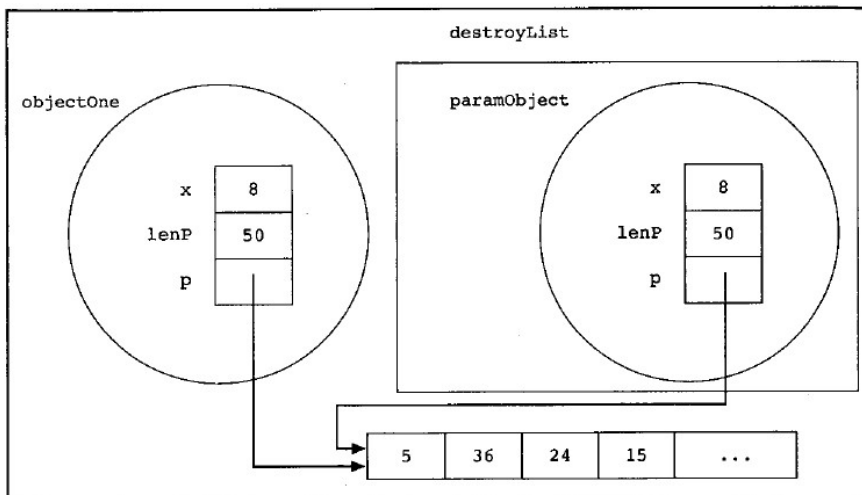
لنقم بالنظر الى نموذج الدالة التالي:

```
void destroyList (pointerDataClass paramObject);
```

الدالة destroyList لها عامل قيمة رسمي وهو paramObject. انظر الآن الى البيان التالي:

```
destroyList (objectOne);
```

في هذا البيان يتم تمرير objectOne كعامل للدالة destroyList. بما أن paramObject عبارة عن عامل قيمة فان مقوم النسخ يقوم بنسخ عناصر objectOne داخل عناصر البيانات المتوافقة في paramObject. كما في الحالة السابقة قد يشير كل من paramObject.p و objectOne.p الى نفس مساحة الذاكرة كما هو موضح في شكل 3-22.



شكل 3-22: عناصر بيانات المؤشر للأهداف objectOne و paramObject التي تشير الى نفس المصفوفة.

بما أن objectOne يتم تمريره بالقيمة فان عناصر بيانات paramObject يجب أن يكون لها نسختها الخاصة من البيانات. بوجه خاص يجب أن يمتلك paramObject مساحة ذاكرة خاصة بها لتخزين البيانات. كيف نضمن أن يكون هذا هو الحال في الحقيقة؟ اذا كانت الفئة لها عناصر بيانات مؤشر:

- أثناء اعلان الهدف سوف يؤدي تهيئة أحد العناصر باستخدام قيمة هدف آخر الى نسخ سطحي للبيانات اذا كان مسموح بعمل نسخ افتراضي حكيم للبيانات.
- اذا تم تمرير الهدف كمعامل بالقيمة وكان مسموح عمل نسخ حكيم للبيانات فان هذا سوف يقود الى نسخ سطحي للبيانات.

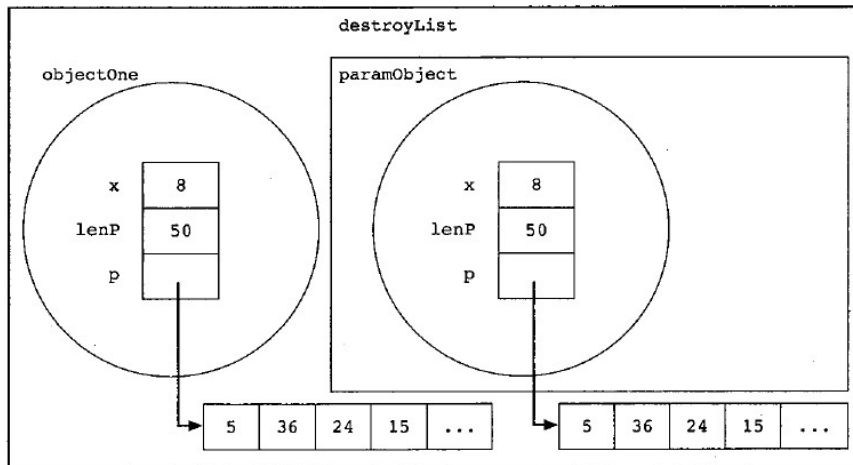
في تلك الحالات لاجبار كل هدف على امتلاكك نسخته الخاصة به من البيانات يجب أن نتجاهل تعريف مقوم النسخ المقدم من المجمع أي أننا يجب أن نوفر تعريفنا الخاص لمقوم النسخ. يتم عادةً عمل هذا عن طريق وضع بيان يتضمن مقوم النسخ في تعريف الفئة ثم كتابة تعريف مقوم النسخ. بعد هذا في أي وقت يحتاج فيه مقوم النسخ الى التنفيذ يقوم النظام بتنفيذ تعريف مقوم النسخ الذي نقدمه وليس التعريف المقدم بواسطة المجمع. لهذا بالنسبة الى الفئة pointerDataClass يمكننا التغلب على هذا النسخ السطحي للبيانات عن طريق ادخال مقوم النسخ في الفئة pointerDataClass والمثال 3-6 يوضح ذلك.

يتم تنفيذ مقوم النسخ تلقائياً في المواقع التالية (تم وصف الموقفان الأوليان من قبل):

- عندما يتم اعلان وتهيئة الهدف باستخدام قيمة هدف آخر.
- عندما يتم تمرير الهدف بالقيمة مثل المعامل.
- عندما تكون قيمة انتاج الدالة عبارة عن هدف.

لهذا بمجرد تعريف مقوم النسخ بشكل صحيح للفئة pointerDataClass سوف يكون لكل من objectOne.p و objectThree.p نسخاتهم الخاصة من البيانات. بالمثل سوف يكون لكل من objectOne.p و paramObject.p نسخاتهم الخاصة من البيانات كما هو موضح في شكل

3-23.



شكل 3-23: عناصر بيانات مؤشر الأهداف objectOne و paramObject ببياناتها الخاصة بها.

عندما تتواجد الدالة destroyList خرج العامل الرسمي paramObject من المجال ويقوم مدمر الهدف paramObject بإزالة تخصيص مساحة الذاكرة المشار إليها بواسطة paramObject. بالرغم من هذا إزالة التخصيص هذه ليس لها تأثير على objectOne. التركيب العام لإدخال مقوم النسخ في تعريف الفئة هو:

`className (const className& otherObject);`

المثال 3-6 يوضح كيفية إدخال مقوم النسخ في الفئة وكيفية عمله.

مثال 3-6:

انظر الى الفئة التالية:

```
class pointerDataClass
{
public:
    void print() const;
        //Function to output the value of x and
        //the value of the array p.
    void setData();
        //Function to input data into x and
        //into the array p.
    void destroyP();
        //Function to deallocate the memory space
        //occupied by the array p.

    pointerDataClass(int sizeP = 10);
        //constructor
        //Create an array of the size specified by the
        //parameter sizeP; the default array size is 10.

    ~pointerDataClass();
        //destructor
        //Deallocate the memory space occupied by the
        //array p.

    pointerDataClass (const pointerDataClass& otherObject);
        //copy constructor

private:
    int x;
    int lenP;
    int *p;        //pointer to an int array
};
```

افترض أن تعريفات دالات عنصر الفئة pointerDataClass كانت كما يلي:

```

void pointerDataClass::print() const
{
    cout<<"x = "<<x<<endl;

    cout<<"p = ";

    for(int i = 0; i < lenP; i++)
        cout<<p[i]<<" ";
    cout<<endl;
}

void pointerDataClass::setData()
{
    cout<<"Enter an integer for x: ";
    cin>>x;
    cout<<endl;

    cout<<"Enter "<<lenP<<" numbers: ";

    for(int i = 0; i < lenP; i++)
        cin>>p[i];

    cout<<endl;
}

void pointerDataClass::destroyP()
{
    lenP = 0;
    delete [] p;
    p = NULL;
}

pointerDataClass::pointerDataClass(int sizeP)
{
    x = 0;

    if(sizeP <= 0)
    {
        cout<<"Array size must be positive."<<endl;
        cout<<"Creating an array of size 10."<<endl;

        lenP = 10;
    }
    else
        lenP = sizeP;

    p = new int[lenP];
    assert(p != NULL);
}

```

```

pointerDataClass::~~pointerDataClass()
{
    delete [] p;
}

//copy constructor
pointerDataClass::pointerDataClass
    (const pointerDataClass& otherObject)
{
    x = otherObject.x;

    lenP = otherObject.lenP;
    p = new int[lenP];
    assert(p != NULL);

    for(int i = 0; i < lenP; i++)
        p[i] = otherObject.p[i];
}

```

اننا نتركه كتمرين لك لكي تكتب برنامج لاختبار مقوم النسخ للفئة pointerDataClass.

بالنسبة الى الفئات ذات عناصر بيانات المؤشر نفعل عادةً ثلاثة أشياء:

1. ادخال المدمر في الفئة.
2. ائصال معامل الاسناد للفئة.
3. ادخال مقوم النسخ.

بعد هذا نقوم بتعريف فئة مسماة newString واثقال معاملات الاسناد والمعاملات المرتبطة بها. هذا من أجل أن نكون قادرين عند اعلان متغير من النوع newString على استخدام معامل الاسناد لنسخ مقطع واحد داخل مقطع آخر وسوف نكون قادرين على معاملات مرتبطة للمقارنة بين المقطعين. قبل مناقشة الفئة newString ندرس ائصال المعامل []. تذكر أننا قمنا باستخدام المعامل [] لتنازل مكونات المصفوفة. لتناول رموز فردية في مقطع ما من النوع newString علينا ائصال المعامل [] للفئة newString.

اثقال معامل (تسجيل) مؤشر المصفوفة ([]):

تذكر أن دالة ائصال المعامل [] لفئة ما يجب أن تكون عنصر للفئة. بالإضافة الى هذا وبما أن المصفوفة يمكن اعلانها كثابت أو غير ثابت فاننا نحتاج الى ائصال المعامل [] لمعالجة هاتين الحالتين. تركيب اعلان معامل الدالة [] كعنصر للفئة بالنسبة الى المصفوفات الغير ثابتة يكون:

```
Type& operator [ ] (int index);
```

تركيب اعلان دالة المعامل [] كعنصر للفئة بالنسبة الى المصفوفات الثابتة يكون:

```
const Type& operator [ ] (int index) const;
```

حيث Type هو نوع بيانات عناصر المصفوفة.
افترض أن classTest عبارة عن فئة لها عنصر بيانات مصفوفة. تعريف classTest لاثقال المعامل
[] يكون:

```
class classTest
{
public:
    Type& operator[](int index);
    //Overload the operator for nonconstant arrays.
    const Type& operator[](int index) const;
    //Overload the operator for constant arrays.
    .
    .
    .
private:
    Type *list; //pointer to the array
    int arraySize;
};
```

حيث Type هو نوع بيانات عناصر المصفوفة.
تعريفات الدالات لاثقال المعامل [] للفئة classTest هي:

```
//Overload the operator [] for nonconstant arrays.
Type& classTest::operator[](int index)
{
    assert(0 <= index && index < arraySize);
    return(list[index]); //Return a pointer to the
                        //array component.
}

//Overload the operator [] for constant arrays.
const Type& classTest::operator[](int index) const
{
    assert(0 <= index && index < arraySize);
    return(list[index]); //Return a pointer to the
                        //array component.
}
```

مثال برمجة: newstring:

في C++:

- C-string عبارة عن تتابع من رمز أو عدة رموز.
- يتم ادخال C-strings في علامتي تنصيب.
- يتم تخزين C-string في مصفوفة حرفية.

في هذا المثال نقصد بالمقطع مقطع C.

العمليات الوحيدة المسموح باجرائها على المقاطع هي الادخال والاخراج. لاستخدام عمليات أخرى يحتاج المبرمج الى ادخال الملف الرئيسي cstring الذي يحتوي على تحدد عدة دالات لاحتكار المقطع.

أولياً لا تقدم C++ أي أنواع مدمجة من البيانات للتعامل مع المقاطع. بالرغم من هذا هناك نسخ حديثة من C++ تقدم فئة مقطع للتعامل مع المقاطع والعمليات المؤداة على المقاطع.

هدفنا في هذا المثال هو تعريف الفئة الخاصة بنا لاحتكار المقطع وفي نفس الوقت توضيح اثقال المعامل. بشكل أكثر خصوصية نقوم باثقال معامل الاسناد والمعاملات المرتبطة بهذا ومعاملات ادخال واستخلاص التدفق من أجل وجود ادخال واخراج سهل. لنطلق على هذه الفئة newString. أولاً نعطي تعريف الفئة newString. انظر شكل 3-24 من أجل رسم لغة التشكيل الموحدة.

```
//Header file newString.h
#ifndef H_newString
#define H_newString
#include <iostream>
using namespace std;

class newString
{
    //Overload the stream insertion and extraction operators.
    friend ostream& operator<<(ostream&, const newString&);
    friend istream& operator>>(istream&, newString&);

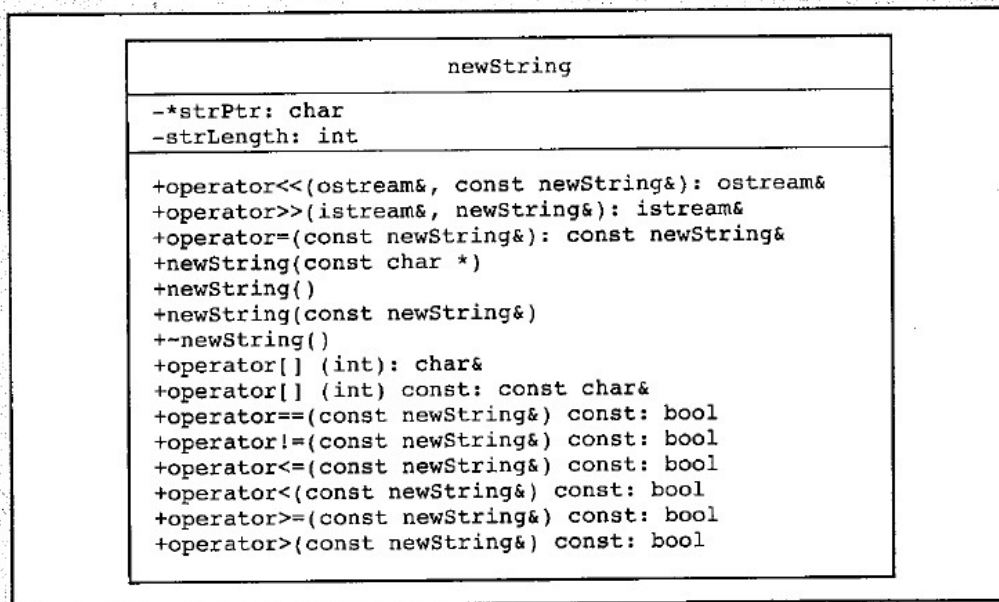
public:
    const newString& operator=(const newString&);
    //Overload the assignment operator.
    newString(const char *);
    //constructor; conversion from the char string
    newString();
    //default constructor to initialize the string to null
    newString(const newString&);
    //copy constructor
    ~newString();
    //destructor

    char& operator[] (int);
    const char& operator[] (int) const;
    //Overload the relational operators.
```

```

    bool operator==(const newString&) const;
    bool operator!=(const newString&) const;
    bool operator<=(const newString&) const;
    bool operator<(const newString&) const;
    bool operator>=(const newString&) const;
    bool operator>(const newString&) const;
private:
    char *strPtr;    //pointer to the char array
                    //that holds the string
    int strLength;  //data member to store the length
                    //of the string
};
#endif

```



شكل 3-24: رسم لغة التشكيل الموحدة للفئة newString.

الفئة newString لها عنصري بيانات خاصين: عنصر لتخزين المقطع، وعنصر لتخزين طول المقطع.

بعد هذا نعطي تعريفات الدالات لتطبيق عمليات newString. ملف التطبيق يتضمن الملف الرئيسي cassert لأننا نستخدم الدالة assert. لتفسير الدالة assert انظر الملف الرئيسي cassert في الملحق هـ.

```

//Implementation file newString.cpp
#include <iostream>
#include <iomanip>
#include <cstring>
#include <cassert>
#include "newString.h"

using namespace std;

//constructor; conversion from the char string to newString
newString::newString(const char *str)
{
    strLength = strlen(str);
    strPtr = new char[strLength + 1]; //allocate memory to store
                                     //the char string
    assert(strPtr != NULL);
    strcpy(strPtr, str); //copy the string into strPtr
}

//default constructor to store the null string
newString::newString()
{
    strLength = 0;
    strPtr = new char[1];
    assert(strPtr != NULL);
    strcpy(strPtr, "");
}

//copy constructor
newString::newString(const newString& rightStr)
{
    strLength = rightStr.strLength;
    strPtr = new char[strLength + 1];
    assert(strPtr != NULL);
    strcpy(strPtr, rightStr.strPtr);
}

newString::~newString() //destructor
{
    delete [] strPtr;
}

//Overload the assignment operator.
const newString& newString::operator=(const newString& rightStr)
{
    if(this != &rightStr) //avoid self-copy
    {

```

```

        delete [] strPtr;
        strLength = rightStr.strLength;
        strPtr = new char[strLength + 1];
        assert(strPtr != NULL);
        strcpy(strPtr, rightStr.strPtr);
    }
    return *this;
}

char& newString::operator[] (int index)
{
    assert(0 <= index && index < strLength);
    return strPtr[index];
}

const char& newString::operator[] (int index) const
{
    assert(0 <= index && index < strLength);
    return strPtr[index];
}

//Overload the relational operators.
bool newString::operator==(const newString& rightStr) const
{
    return(strcmp(strPtr, rightStr.strPtr) == 0);
}

bool newString::operator<(const newString& rightStr) const
{
    return(strcmp(strPtr, rightStr.strPtr) < 0);
}

bool newString::operator<=(const newString& rightStr) const
{
    return(strcmp(strPtr, rightStr.strPtr) <= 0);
}

bool newString::operator>(const newString& rightStr) const
{
    return(strcmp(strPtr, rightStr.strPtr) > 0);
}

bool newString::operator>=(const newString& rightStr) const
{
    return(strcmp(strPtr, rightStr.strPtr) >= 0);
}

bool newString::operator!=(const newString& rightStr) const

```

```

{
    return(strcmp(strPtr, rightStr.strPtr) != 0);
}

//Overload the stream insertion operator <<
ostream& operator<<(ostream& osObject, const newString& str)
{
    osObject<<str.strPtr;
    return osObject;
}

//Overload the stream extraction operator >>
istream& operator>>(istream& isObject, newString& str)
{
    char temp[81];

    isObject>>setw(81)>>temp;
    str = temp;
    return isObject;
}

```

انظر الى البيان:

```
isObject>> setw(81)>>temp;
```

في تعريف معامل الدالة >>. بما أنه يتم الاعلان عن temp كمصفوفة حجمها 81 اذن يكون طول المقطع الأكبر الذي يمكن تخزينه هو 80. المتلاعب setw في هذا البيان (أي في بيان الادخال) يضمن أنه لا يوجد أكثر من 80 رمز مقروء داخل temp.

مقوم التحويل عبارة عن دالة أحادية العامل تقوم بتحويل معطياتها الى هدف فئة المقوم. في حالتنا يقوم مقوم التحويل بتحويل المقطع الى هدف من النوع newString.

لاحظ أن معامل الاسناد يتم اثقاله بوضوح بالنسبة الى الأهداف من نوع newString. بالرغم من هذا يعمل معامل الاسناد المثقل كذلك اذا أردنا تخزين مقطع رمزي داخل الهدف newString. انظر الى الاعلان التالي:

```
newString str;
```

والبيان:

```
str = "Hello there";
```

يقوم المجمع بترجمة هذا البيان الى:

```
Str.operator= ("Hello there");
```

أولاً يقوم المجمع تلقائياً باستدعاء مقوم التحويل لعمل هدف من النوع newString لتخزين المقطع "Hello there" بشكل مؤقت.

ثانياً يستدعي المجمع معامل الاسناد المثقل لتحديد هدف newString المؤقت الى الهدف str.

من هنا ليس من اللازم اثقال معامل الاسناد بوضوح لتخزين مقطع رمزي داخل هدف من النوع
.newString

بعد هذا نكتب برنامج C++ يختبر بعض من عمليات الفئة newString.

```
//Test Program
#include <iostream>
#include <cstring>
#include "newString.h"

using namespace std;

int main()
{
    newString s1 = "Sunny";           //Initialize s1 using
                                     //the assignment operator
    const newString s2("Warm");       //Initialize s2 using
                                     //the conversion constructor
    newString s3;                     //Initialize s3 to null
    newString s4;                     //Initialize s4 to null

    cout<<"Line 1: "<<s1<<"    "<<s2<<"    ***"
         <<s3<<"###."<<endl;           //Line 1

    if(s1 <= s2)                      //Compare s1 and s2; Line 2
        cout<<"Line 3: "<<s1<<" is less than or equal to "<<s2
         <<endl;                       //Line 3
    else                               //Line 4
        cout<<"Line 5: "<<s2<<" is less than "<<s1
         <<endl;                       //Line 5

    cout<<"Line 6: Enter a string that has "
         <<"at least 7 characters --> "; //Line 6
    cin>>s1;                           //input s1; Line 7
    cout<<endl<<"Line 8: New value of s1 = "<<s1
         <<endl;                       //Line 8

    s4 = s3 = "Birth Day";             //Line 9
    cout<<"Line 10: s3 = "<<s3<<" , s4 = "<<s4<<endl; //Line 10

    s3 = s1;                           //Line 11
    cout<<"Line 12: The new value of s3 = "<<s3<<endl; //Line 12

    s1 = "Bright Sky";                 //Line 13

    s3[1] = s1[5];                     //Line 14
    cout<<"Line 15: After replacing the second character, s3 = "
         <<s3<<endl;                   //Line 15

    s3[2] = s2[3];                     //Line 16
    cout<<"Line 17: After replacing the third character, s3 = "
         <<s3<<endl;                   //Line 17

    s3[5] = 'g';                       //Line 18
    cout<<"Line 19: After replacing the sixth character, s3 = "
         <<s3<<endl;                   //Line 19

    return 0;
}
```

تنفيذ العينة: في تنفيذ العينة هذا يتم تظليل مدخلات المستخدم.

الصف 1: Sunny Warm ***###.

الصف 3: Sunny أقل من أو تساوي Warm.

123456789

الصف 6: ادخل مقطع به 7 حروف على الأقل -->

الصف 8: قيمة جديدة من s1 = 123456789

الصف 10: s3 = Birth Day, s4 = Birth Day

الصف 12: القيمة الجديدة لs3 = 123456789

الصف 15: بعد استبدال الرمز الثاني، s3 = 1t3456789

الصف 17: بعد استبدال الرمز الثالث، s3 = 1tw456789

الصف 19: بعد استبدال الرمز السادس، s3 = 1tw45g789

لاحظ أن في البيان (الصف 9):

S4 = s3 = "Birth Day";

s3 = بما أن ترابط معامل الاسناد يكون من اليمين الى اليسار اذن يتم تنفيذ البيان

"Birth Day" ثم البيان s4 = s3. البيان في الصف 11 ينسخ s1 داخل s3 باستخدام معامل الاسناد.

البيانات من الصف 14 وحتى الصف 19 توضح أن باستخدام معامل تسجيل المصفوفة [] يمكنك

التلاعب برموز المقطع الفردية.

القوائم القائمة على المصفوفة:

كل فرد على علم بالكلمة "قائمة" وقد يكون لديك قائمة مكونة من بيانات الموظفين، أو بيانات الطلاب،

أو بيانات المبيعات، أو قائمة بخصائص الايجار. هناك شئ مشترك في جميع القوائم وهو أن جميع

عناصر القائمة تكون من نفس النوع. بشكل رسمي أكثر يمكننا تعريف القائمة كما يلي:

القائمة: مجموعة من العناصر التي تكون من نفس النوع.

طول القائمة هو عدد العناصر الموجودة في القائمة.

فيما يلي بعض من العمليات التي يتم اجرائها على القائمة:

1. عمل القائمة. تتم تهيئة القائمة في وضع فارغ.

2. تحديد ما اذا كانت القائمة فارغة.

3. تحديد ما اذا كانت القائمة ممتلئة.

4. ايجاد حجم القائمة.

5. تدمير أو تفريغ القائمة.

6. تحديد ما اذا كان العنصر هو نفس عنصر القائمة المعطى.

7. ادخال عنصر في القائمة في موقع محدد.

8. ازالة عنصر من القائمة من موقع محدد.

9. استبدال عنصر في موقع محدد بعنصر آخر.

10. استرجاع عنصر من القائمة من موقع محدد.

11. البحث عن عنصر محدد في القائمة.

قبل مناقشة كيفية تطبيق هذه العمليات يجب أن نقرر أولاً كيفية تخزين القائمة في ذاكرة الحاسب الآلي. بما أن جميع عناصر القائمة تكون من نفس النوع فهناك طريقة فعالة وتقليدية لمعالجة القائمة وهي تخزينها في مصفوفة مبدئياً يكون حجم المصفوفة المحتوية على عناصر القائمة عادةً أكبر من عدد العناصر في القائمة حتى يمكن في مرحلة تالية أن تنمو القائمة الى حجم محدد. لهذا يجب أن نعرف مدى امتلاء المصفوفة أي أننا يجب أن نتتبع عدد عناصر القائمة المخزنة في المصفوفة. تسمح C++ للمبرمج بعمل مصفوفات حركية ولهذا سوف نترك للمستخدم تحديد حجم المصفوفة. يمكن تحديد حجم المصفوفة عندما يتم اعلان هدف بالقائمة. ينتج من هذا أنه من أجل الحفاظ على القائمة ومعالجتها في المصفوفة نحتاج الى المتغيرات الثلاث التالية:

- المصفوفة تحمل عناصر القائمة.
- متغير لتخزين طول القائمة (أي عدد عناصر القائمة الموجودة حالياً في المصفوفة).
- متغير لتخزين حجم المصفوفة (أي العدد الأقصى من العناصر التي يمكن تخزينها في المصفوفة).

افترض أن المتغير length يشير الى عدد العناصر في القائمة والمتغير maxsize يشير الى عدد العناصر الأقصى الذي يمكن تخزينه في القائمة. بهذا يكون كل من length و maxsize عددين صحيحين غير سالبين ولهذا يمكننا اعلانهما من النوع int. ماذا عن نوع المصفوفة أي نوع بيانات عناصر المصفوفة؟ اذا كان لدينا قائمة أرقام فاذن عناصر المصفوفة اما أن تكون من النوع int أو النوع double. اذا كان لدينا قائمة أسماء اذن عناصر المصفوفة تكون من النوع string. بالمثل اذا كان لدينا قائمة طلاب اذن تكون عناصر المصفوفة من النوع studentType (نوع بيانات يمكنك تعريفه) لهذا نرى أن هناك أنواع متنوعة من القوائم.

قائمة بيانات المبيعات أو قائمة بيانات الطلاب تكون فارغة اذا كان طولها صفر. لادخال عنصر في نهاية قائمة من أي نوع يتطلب هذا منا أن نضيف العنصر بعد العنصر الأخير ثم زيادة الطول بمقدار واحد. بالمثل يمكن رؤية أن في غالبية الحالات تتماثل الحلول الحسابية لتطبيق عمليات على قائمة أسماء، أو قائمة بيانات مبيعات، أو قائمة بيانات طلاب. اننا لا نريد قضاء الوقت والجهد في تنمية كود منفصل لكل نوه من القوائم التي نقابلها. بدلاً من هذا قد نحب تنمية كود عام يمكن استخدامه لتطبيق أي نوع من القوائم في برنامج ما.

بمعنى آخر أثناء تصميم الحلول الحسابية لا نريد أن نكون مهتمين بما اذا كنا نعالج قائمة أعداد، أو قائمة أسماء، أو بيانات الطلاب. بالرغم من هذا أثناء توضيح حل حسابي معين سوف ننظر الى نوع محدد من القوائم. لتطوير حلول حسابية عامة لتطبيق عمليات القوائم نستخدم قوالب الفئة. الآن أنت تعلم العمليات التي يتم عملها على القائمة وكيفية تخزين القائمة في ذاكرة الحاسب وبعد هذا نقوم بتعريف الفئة التي تطبق القائمة كنوع بيانات مجرد. الفئة التالية arrayListType تعرف القائمة كنوع بيانات مجرد:

```
template<class elemType>
class arrayListType
{
public:
    const arrayListType<elemType>&
        operator=(const arrayListType<elemType>&);
        //Overload the assignment operator.

    bool isEmpty();
        //Function to determine whether the list is empty.
        //Postcondition: Returns true if the list is empty;
        // otherwise, returns false.
    bool isFull();
        //Function to determine whether the list is full.
        //Postcondition: Returns true if the list is full;
        // otherwise, returns false.
    int listSize();
        //Function to determine the number of elements in
        //the list.
        //Postcondition: Returns the value of length.
    int maxListSize();
        //Function to determine the size of the list.
        //Postcondition: Returns the value of maxSize.
    void print() const;
        //Function to output the elements of the list.
        //Postcondition: The elements of the list are output
        // on the standard output device.
    bool isItemAtEqual(int location, const elemType& item);
        //Function to determine whether the item is the
        //same as the item in the list at the position
        //specified by location.
        //Postcondition: Returns true if the
        // list[location] is the same as
        // the item; otherwise, returns
        // false.
    void insertAt(int location, const elemType& insertItem);
        //Function to insert an item in the list at the
        //position specified by location. The item to be
        //inserted is passed as a parameter to the
```

template<class elemType>

class arrayListType
}

عامة:

```
const arrayListType<elemType>&
operator = (const arrayListType<elemType>&);
// انقال معامل الاسناد.
bool isEmpty ();
// دالة لتحديد ما اذا كانت القائمة فارغة.
// شرط تالي: تنتج true اذا كانت القائمة فارغة
// وبخلاف هذا تنتج False.
bool isFull ();
// دالة لتحديد ما اذا كانت القائمة ممتلئة.
// شرط تالي: تنتج true اذا كانت القائمة ممتلئة
// وبخلاف هذا تنتج False.
int listSize ();
// دالة لتحديد عدد العناصر في القائمة.
// شرط تالي: تنتج قيمة الطول.
int maxListSize ();
// دالة لتحديد حجم القائمة.
// شرط تالي: تنتج قيمة maxSize.
void print () const;
// دالة لايخراج عناصر القائمة.
// شرط تالي: عناصر القائمة مخرجات
// على جهاز الاخراج القياسي.
bool isItemAtEqual (int location, const elemType& item);
// دالة لتحديد ما اذا كان العنصر هو نفسه في القائمة
// في الموقع المحدد.
// شرط تالي: ينتج true اذا كان موقع القائمة هو نفس العنصر
// وبخلاف هذا تنتج false.
void insertAt (int location, const elemType& insertItem);
// دالة لادخال عنصر في القائمة في
// الموقع المحدد بالمكان. يتم تمرير العنصر الذي يتم ادخاله
// كمعامل الى الدالة.
```

```

//function.
//Postcondition: Starting at location, the
//                elements of the list are
//                shifted down,
//                list[location] = insertItem;,
//                and length++;
//    If the list is full or location is out of
//    range, an appropriate message is displayed.
void insertEnd(const elemType& insertItem);
//Function to insert an item at the end of the
//list.
//The parameter insertItem specifies the item to
//be inserted.
//Postcondition: list[length] = insertItem; and
//                length++;
//    If the list is full, an appropriate message
//    is displayed.
void removeAt(int location);
//Function to remove the item from the list at
//the position specified by location.
//Postcondition: The list element at
//                list[location] is removed
//                and length is decremented by 1.
//    If location is out of range, an appropriate
//    message is displayed.
void retrieveAt(int location, elemType& retItem);
//Function to retrieve the element from the list at
//the position specified by location.
//Postcondition: retItem = list[location]
//    If location is out of range, an appropriate
//    message is displayed.
void replaceAt(int location, const elemType& repItem);
//Function to replace the element in the list at
//the position specified by location. The item to be
//replaced is specified by the parameter repItem.
//Postcondition: list[location] = repItem
//    If location is out of range, an appropriate
//    message is displayed.
void clearList();
//Function to remove all the elements from the list.
//After this operation, the size of the list is
//zero.
//Postcondition: length = 0
int seqSearch(const elemType& item);
//Function to search the list for a given item.
//Postcondition: If the item is found, returns the
//                location in the array where the
//                item is found; otherwise,
//                returns -1.

```

```

// شرط تالي: بدءاً بالموقع يتم نقل عناصر القائمة الى أسفل
list[location] = insertItem;, and length++; //
// اذا كانت القائمة ممتلئة أو الموقع خارج المجال
// يتم عرض رسالة مناسبة.
void insertEnd (const elemType& insertItem)
// دالة لادخال عنصر في نهاية القائمة.
// المعامل insertItem تحدد العنصر الذي يتم ادخاله.
// شرط تالي: list[length] = insertItem;, and length++;
// اذا كانت القائمة ممتلئة يتم عرض رسالة مناسبة.
void removeAt (int location);
// دالة لازالة عنصر من القائمة بمكان محدد بالموقع.
// شرط تالي: تتم ازالة عنصر القائمة ويتم تقليل الطول بمقدار 1.
// اذا كان الموقع خارج المجال يتم عرض رسالة مناسبة.
void retrieveAt (int location, elemType& retItem);
// دالة لاسترجاع عنصر من القائمة بمكان محدد بالموقع.
// شرط تالي: retItem = list [location]
// اذا كان الموقع خارج المجال يتم عرض رسالة مناسبة.
void replaceAt (int location, const elemType& repItem);
// دالة لاستبدال عنصر في القائمة بمكان محدد بالموقع. العنصر الذي
// يتم استبداله بالمعامل repItem.
// شرط تالي: repItem = list [location]
// اذا كان الموقع خارج المجال يتم عرض رسالة مناسبة.
void clearList ( );
// دالة لازالة جميع العناصر من القائمة.
// بعد هذه العملية يكون حجم القائمة صفراً.
// شرط تالي: الطول = صفر.
int seqSearch (const elemType7 item);
// دالة للبحث عن عنصر محدد في القائمة.
// شرط تالي: اذا تم العثور على العنصر تنتج الموقع
// في المصفوفة حيث ووجد العنصر وبخلاف هذا تنتج -1.

```

```

void insert(const elemType& insertItem);
    //Function to insert the item specified by the
    //parameter insertItem at the end of the list.
    //However, first the list is searched to see whether
    //the item to be inserted is already in the list.
    //Postcondition: list[length] = insertItem; and
    //                length++
    //    If the item is already in the list or the
    //    list is full, an appropriate message is
    //    displayed.
void remove(const elemType& removeItem);
    //Function to remove an item from the list. The
    //parameter removeItem specifies the item to be
    //removed.
    //Postcondition: If removeItem is found in the
    //                list, it is removed from the list
    //                and length is decremented by one.

arrayListType(int size = 100);
    //constructor
    //Creates an array of the size specified by the
    //parameter size. The default array size is 100.
    //Postcondition: The list points to the array;
    //                length = 0; and maxSize = size

arrayListType(const arrayListType<elemType>& otherList);
    //copy constructor

~arrayListType();
    //destructor
    //Deallocates the memory occupied by the array.

protected:
    elemType *list;        //array to hold the list
                           //elements
    int length;            //variable to store the length of the
                           //list
    int maxSize;           //variable to store the maximum size of
                           //the list
};

```

void insert (const elemType& insertItem);

// دالة لادخال العنصر المحدد بالمعامل insertItem في نهاية القائمة.
// مع هذا يتم البحث أولاً في القائمة لرؤية ما اذا
// تم ادخال العنصر بالفعل في القائمة.
// شرط تالي: list[length] = insertItem; , and length++;
// اذا كان العنصر موجود بالفعل في القائمة أو كانت القائمة ممتلئة يتم
// يتم عرض رسالة مناسبة.

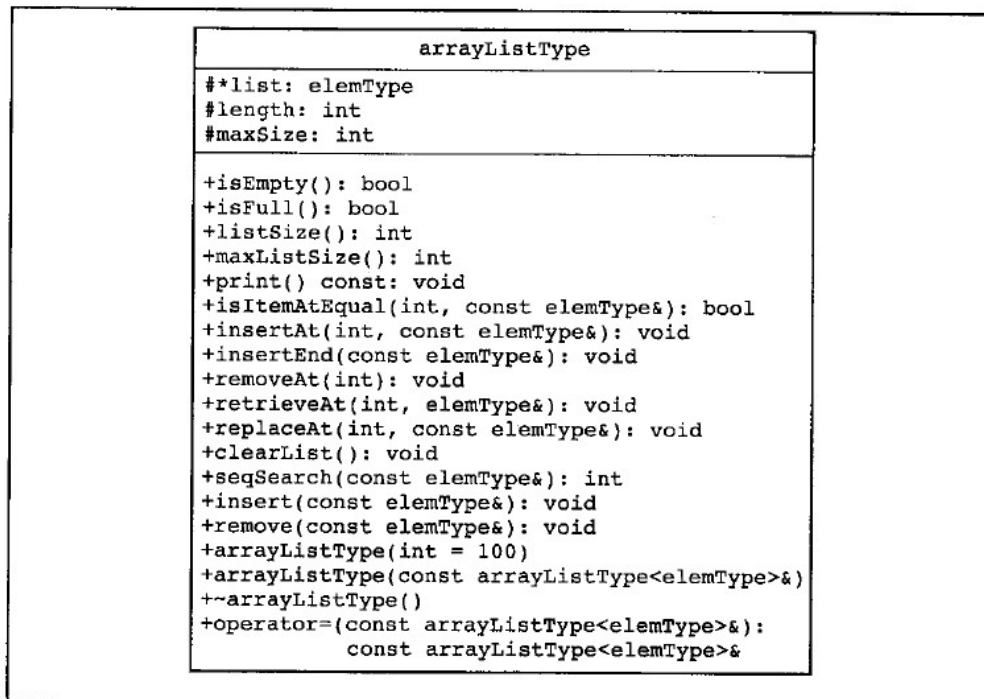
void remove (const elemType& removeitem);

// دالة لازالة عنصر من القائمة. المعامل removeItem يحدد

// العنصر الذي تتم ازالته.
 // شرط تالي: اذا تم العثور على removeItem في القائمة
 // تتم ازالته من القائمة ويتم تقليل الطول بمقدار واحد.
 arrayListType (int size = 100);
 // مقوم يخلق مصفوفة حجمها محدد بحجم المعامل.
 // حجم المصفوفة الافتراضي هو 100.
 // شرط تالي: القائمة تشير الى المصفوفة
 Length = 0; maxSize = size
 arrayListType (const arrayListType<elemType>& otherList);
 // مقوم نسخ.
 -arrayListType ();
 // مدمر يزيل تخصص الذاكرة التي تشغلها المصفوفة.

المحمية:

elemType *list; // مصفوفة لحمل عناصر القائمة.
 int length; // متغير لتخزين طول القائمة.
 int maxSize; // متغير لتخزين الحجم الأقصى للقائمة.
 {
 شكل 3-25 يوضح رسم لغة التشكيل الموحدة للفئة arrayListType.



شكل 3-25: رسم لغة التشكيل الموحدة للفئة arrayListType.

لاحظ أن عناصر بيانات الفئة `arrayListType` معلنة كعناصر محمية. هذا لأننا قد نحب اشتقاق فئات من هذه الفئة لتطبيق قوائم خاصة مثل قائمة مرتبة. بعد هذا نكتب تعريفات هذه الدالات. تكون القائمة فارغة إذا كان الطول `length` يساوي صفر وتكون ممتلئة إذا كان الطول `length` مساوي للحجم الأقصى `maxSize`. لهذا تكون تعريفات الدالات `isEmpty` و `isFull` كما يلي:

```
template<class elemType>
bool arrayListType<elemType>::isEmpty()
{
    return (length == 0);
}

template<class elemType>
bool arrayListType<elemType>::isFull()
{
    return (length == maxSize);
}
```

عنصر البيانات `length` للفئة يقوم بتخزين عدد العناصر الموجودة حالياً في القائمة. بالمثل بما أن حجم المصفوفة التي تضم عناصر القائمة مخزن في عنصر البيانات `maxSize` فإن `maxSize` يحدد الحجم الأقصى للقائمة. لهذا تكون تعريفات الدالات `listSize` و `maxListSize` هي:

```
template<class elemType>
int arrayListType<elemType>::listSize()
{
    return length;
}

template<class elemType>
int arrayListType<elemType>::maxListSize()
{
    return maxSize;
}
```

كل من الدالات `isEmpty` و `isFull`، و `listSize`، و `maxListSize` تحتوي على بيان واحد فقط قد يكون اما بيا م مقارنة أو بيان ينتج قيمة. ينتج من هذا أن كل من تلك الدالات تكون $O(1)$. دالة العنصر `print` تخرج عناصر القائمة. نفترض أن البيانات يتم ارسالها الى جهاز الاخراج القياسي.

```
template<class elemType>
void arrayListType<elemType>::print() const
{
    for(int i = 0; i < length; i++)
        cout<<list[i]<<" ";

    cout<<endl;
}
```

الدالة print تستخدم حلقة لايخراج عناصر القائمة. عدد المرات التي يتم بها تنفيذ الحلقة for يعتمد على عدد عناصر البيانات. اذا كانت القائمة بها 100 عنصر فان الحلقة for يتم تنفيذها 100 مرة. بوجه عام افترض أن عدد العناصر في القائمة n. تكون دالة print هي $O(n)$.

تعريف الدالة isItemAt معطى بعد هذا.

```
template<class elemType>
bool arrayListType<elemType>::isItemAtEqual
    (int location, const elemType& item)
{
    return(list[location] == item);
}
```

هيكل الدالة isItemAtEqual له بيان واحد فقط وهو بيان مقارنة. من السهل رؤية أن هذه الدالة هي $O(1)$.

الدالة insertAt تدخل عنصر في موقع محدد بالقائمة. العنصر الذي يتم ادخاله ومكان الادخال في المصفوفة يتم تمريرهما كمعاملات لهذه الدالة. من أجل ادخال العنصر في مكان ما بمنتصف القائمة يجب أن نوفر مكان للعنصر الجديد. هذا يعني أننا سوف نحتاج الى نقل عناصر محددة الى أسفل القائمة بمقطع واحد. الشكل 3-26 يوضح هذا المفهوم.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 35 | 24 | 45 | 17 | 26 | 78 | | | | |

شكل 3-26: قائمة المصفوفة.

عدد العناصر الموجودة حالياً في القائمة 6 وكذلك الطول 6. لهذا بعد ادخال عنصر جديد يصبح طول القائمة 7. اذا كان العنصر الذي يتم ادخاله يكون في الموقع 6 مثلاً فيمكننا بسهولة القيام بهذا عن طريق نسخ العنصر داخل القائمة [6] list. على الجهة الأخرى اذا كان العنصر سوف يتم ادخاله في الموقع 3 مثلاً فاننا نحتاج أولاً الى نقل العناصر [3] list، و [4] list، و [5] list مقطع واحد الى اليمين لتوفير مكان للعنصر الجديد. لهذا يجب أن نقوم أولاً بنسخ [5] list داخل [6] list، و [4] list داخل [5] list، و [3] list داخل [4] list بهذا الترتيب. بعد هذا يمكننا نسخ العنصر الجديد داخل list [3].

بالطبع هناك حالات خاصة مثل محاولة الادخال في قائمة ممثلة يجب التعامل معها على حدة. هناك دالات عناصر أخرى يمكن أن تتعامل مع البعض من هذه الحالات.

تعريف دالة insertAt يكون كما يلي:

```

template<class elemType>
void arrayListType<elemType>::insertAt
    (int location, const elemType& insertItem)
{
    if(location < 0 || location >= maxSize)
        cerr<<"The position of the item to be inserted "
            <<"is out of range."<<endl;
    else
        if(length >= maxSize) //list is full
            cout<<"Cannot insert in a full list."<<endl;
        else
        {
            for(int i = length; i > location; i--)
                list[i] = list[i - 1];        //move the elements
                                                //down

            list[location] = insertItem;      //insert the item
                                                //at the specified
                                                //position

            length++; //increment the length
        }
}
} //end insertAt

```

الدالة insertAt تستخدم الحلقة loop لتحويل عناصر القائمة. عدد المرات التي يتم بها تنفيذ الحلقة loop يعتمد على المكان الذي يتم ادخال العنصر فيه في القائمة. اذا كان العنصر سوف يتم ادخاله في الموضع الأول اذن يتم نقل جميع عناصر القائمة ويمكن توضيح أن هذه الدالة من النوع $O(n)$ بسهولة.

يمكن تطبيق الدالة insertEnd باستخدام الدالة insertAt. بالرغم من هذا لا تتطلب الدالة insertEnd تحويل العناصر. لهذا نقوم باعطاء تعريفه بشكل مباشر:

```

template<class elemType>
void arrayListType<elemType>::insertEnd(const elemType& insertItem)
{
    if(length >= maxSize) //the list is full
        cerr<<"Cannot insert in a full list."<<endl;
    else
    {
        list[length] = insertItem; //insert the item at the
                                    //end
        length++; //increment the length
    }
}
} //end insertEnd

```

عدد البيانات وكذلك عدد العمليات التي يتم تنفيذها في هيكل الدالة insertEnd ثابت ولهذا تكون هذه الدالة من النوع $O(1)$.

الدالة removeAt هي عكس الدالة insertAt. تقوم الدالة removeAt بإزالة عنصر من موقع معين بالقائمة ويتم تمرير الموقع الذي يتم إزالة العنصر منه كمعامل لهذه الدالة. بعد إزالة العنصر من القائمة

يتم تقليل طول القائمة بمقدار 1. اذا كان العنصر الذي يتم ازالته في مكان ما بمنتصف القائمة فاننا يجب أن نقوم بعد ازالة العنصر بنقل عناصر محددة مقطع واحد الى اليسار لأننا لا نريد ترك ثغوب في جانب المصفوفة المحتوية على القائمة. انظر الى القائمة الموجودة في الشكل 3-27.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 35 | 24 | 45 | 17 | 26 | 78 | | | | |

شكل 3-27: قائمة المصفوفة.

عدد العناصر الموجودة حالياً في القائمة 6 وكذلك الطول 6. لهذا بعد ازالة عنصر يصبح طول القائمة 5. افترض أن العنصر الذي يتم ازالته يكون في الموقع 3 مثلاً اذن يجب علينا بوضوح نقل العنصر [4] داخل [3] list و [5] list داخل [4] list بهذا الترتيب.

تعريف الدالة removeAt هو:

```
template<class elemType>
void arrayListType<elemType>::removeAt(int location)
{
    if(location < 0 || location >= length)
        cerr<<"The location of the item to be removed "
        <<"is out of range."<<endl;
    else
    {
        for(int i = location; i < length - 1; i++)
            list[i] = list[i+1];

        length--;
    }
} //end removeAt
```

مثل الدالة insertAt يمكن بسهولة رؤية أن الدالة removeAt من النوع $O(n)$. يتم اعطاء تعريف الدالة retrieveAt بعد ذلك. يتم تمرير مؤشر العنصر الذي يتم استرجاعه والموقع الذي يتم استرجاع العنصر منه كمعاملات لهذه الدالة.

```
template<class elemType>
void arrayListType<elemType>::retrieveAt
(int location, elemType& retItem)
{
    if(location < 0 || location >= length)
        cerr<<"The location of the item to be retrieved is "
        <<"out of range."<<endl;
    else
        retItem = list[location];
} //end retrieveAt
```

تعريف الدالة replaceAt يكون كما يلي:

```
template<class elemType>
void arrayListType<elemType>::replaceAt
    (int location, const elemType& repItem)
{
    if(location < 0 || location >= length)
        cerr<<"The location of the item to be replaced is "
            <<"out of range."<<endl;
    else
        list[location] = repItem;
} //end replaceAt
```

تقوم الدالة clearList بإزالة العناصر من القائمة تاركة إياها خالية. بما أن عنصر البيانات length يشير إلى عدد العناصر في القائمة فإن العناصر تتم إزالتها ببساطة عن طريق ضبط length عند صفر. لهذا يكون تعريف الدالة كما يلي:

```
template<class elemType>
void arrayListType<elemType>::clearList()
{
    length = 0;
} //end clearList
```

نناقش الآن تعريف المقوم والمدمر. المقوم يخلق مصفوفة حجمها محدد بواسطة المستخدم ويقوم بتهيئة طول القائمة عند صفر وتهيئة الحجم الأقصى عند حجم المصفوفة الذي يحدده المستخدم. يتم تمرير حجم المصفوفة كعامل للمقوم. حجم المصفوفة الافتراضي هو 100 والمدمر يقوم بإزالة تخصيص الذاكرة التي تشغلها المصفوفة التي تضم عناصر القائمة. تعريف المقوم والمدمر يكون كما يلي:

```
template<class elemType>
arrayListType<elemType>::arrayListType(int size)
{
    if(size < 0)
    {
        cerr<<"The array size must be positive. Creating "
            <<"an array of size 100."<<endl;

        maxSize = 100;
    }
    else
        maxSize = size;

    length = 0;

    list = new elemType[maxSize];
    assert(list != NULL);
}

template<class elemType>
arrayListType<elemType>::~~arrayListType()
{
    delete [] list;
}
```

كما سبق من السهل رؤية أن كل من الدالات `retrievAt`، `replaceAt`، و `clearList` وكذلك المقوم والمدر من النوع $O(1)$.

مقوم النسخ:

تذكر أن مقوم النسخ يتم استدعائه عندما يتم تمرير هدف ما كمعامل (قيمة) للدالة وعندما يتم استدعاء وتهيئة الهدف باستخدام قيمة هدف آخر من نفس النوع. انه يقوم بنسخ عناصر بيانات الهدف الحقيقي بداخل عناصر البيانات المتوافقة للمعامل الرسمي والهدف الذي يتم عمله. وتعريفه يكون كما يلي:

```
template<class elemType>
arrayListType<elemType>::arrayListType
    (const arrayListType<elemType>& otherList)
{
    maxSize = otherList.maxSize;
    length = otherList.length;
    list = new elemType[maxSize];    //create the array
    assert(list != NULL);            //terminate if unable to
                                    //allocate memory space

    for(int j = 0; j < length; j++) //copy otherList
        list[j] = otherList.list[j];
} //end copy constructor
```

اثقال معامل الاسناد:

بعد هذا نعطي تعريف قالب الدالة لاثقال معامل الاسناد لأننا ننقل معامل الاسناد للفئة `:arrayListType`

```
template<class elemType>
const arrayListType<elemType>&
    arrayListType<elemType>::operator=
    (const arrayListType<elemType>& otherList)
{
    if(this != &otherList)           //avoid self-assignment
    {
        delete [] list;
        maxSize = otherList.maxSize;
        length = otherList.length;

        list = new elemType[maxSize]; //create the array
        assert(list != NULL);          //if unable to allocate
                                        //memory space, terminate
                                        //the program

        for(int i = 0; i < length; i++)
            list[i] = otherList.list[i];
    }

    return *this;
}
```

مثل الدالة `print` يمكن رؤية أن كل من مقوم النسخ ودالة اثقال معامل الاسناد من النوع $O(n)$.

البحث:

الحل الحسابي للبحث الموصوف بعد هذا يطلق عليه بحث متتالي أو خطي.
انظر الى قائمة السبعة عناصر الموضحة في شكل 3-28.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 35 | 12 | 27 | 18 | 45 | 16 | 38 | ... |

شكل 3-28: قائمة السبعة عناصر.

افترض أنك تريد تحديد ما اذا كان 27 موجود في القائمة. سوف يعمل البحث المتتالي كما يلي: أولاً تقارن 27 مع list [0] – أي مقارنة 27 مع 35. بما أن $list [0] \neq 27$ فانك تقوم اذن بمقارنة 27 مع list [1] (أي مع 12 وهو العنصر الثاني في القائمة). بما أن $list [1] \neq 27$ فانك تقوم اذن بمقارنة 27 مع العنصر التالي في القائمة – أي مقارنة 27 مع list [2]. بما أن list [2] تساوي 27 اذن يتوقف البحث ويكون هذا بحث ناجح.

لنقم الآن بالبحث عن 10. كما سبق يبدأ البحث بالعنصر الأول في القائمة – أي list [0]. هذه المرة تتم مقارنة عنصر البحث 10 مع كل عنصر في القائمة. في النهاية لا يتم ترك بيانات في القائمة لمقارنتها مع عنصر البحث ويكون هذا بحث غير ناجح.

ينتج من هذا أنه بمجرد أن تجد عنصر في القائمة مساوي لعنصر البحث يجب أن تقوم بإيقاف البحث وكتابة "ناجح". (في هذه الحالة تقوم عادةً بقول الموقع الذي يتم فيه العثور على العنصر في القائمة). بخلاف هذا بعد مقارنة عنصر البحث مع كل عنصر في القائمة يجب أن تقوم بإيقاف البحث وكتابة "فاشل".

افترض أن اسم المصفوفة المحتوية على عناصر القائمة هو list. المناقشة التالية تتم ترجمتها الى الحل الحسابي التالي للبحث المتتالي:

```
found is set to false;

for(loc = 0; loc < length; loc++)
    if(list[loc] is equal to searchItem)
    {
        found is set to true
        exit loop
    }

if(found)
    return loc;
else
    return -1;
```

الدالة التالية تؤدي بحث متتالي بالقائمة:

```
template<class elemType>
int arrayListType<elemType>::seqSearch(const elemType& item)
{
    int loc;
    bool found = false;

    for(loc = 0; loc < length; loc++)
        if(list[loc] == item)
        {
            found = true;
            break;
        }

    if(found)
        return loc;
    else
        return -1;
} //end seqSearch
```

بما أننا نعرف الآن كيفية تطبيق حل البحث (المتتالي) يمكننا اعطاء تعريفات الدالات insert و remove. تذكر أن الدالة insert تقوم بادخال عنصر جديد في نهاية القائمة اذا لم يكن هذا العنصر موجود في القائمة واذا كانت القائمة غير ممتلئة. الدالة remove تقوم بازالة عنصر من القائمة اذا كانت القائمة غير فارغة واذا كان العنصر الذي يتم حذفه موجود في القائمة.

الفصل رقم 9 سوف يوضح أن الدالة seqSearch من النوع $O(n)$.

الادخال:

الدالة insert تقوم بادخال عنصر جديد في القائمة. بما أن المكررات غير مسموح بها فان هذه الدالة تقوم أولاً ببحث القائمة لتحديد ما اذا كان العنصر مدخل بالفعل في القائمة. لتحديد ما اذا كان العنصر مدخل أم لا في القائمة فان هذه الدالة تستدعي دالة العنصر seqSearch كما تم شرحه من قبل. اذا كان العنصر الذي يتم ادخاله غير موجود في القائمة اذن يتم ادخال العنصر الجديد في نهاية القائمة ويتم زيادة طول القائمة بمقدار 1. كذلك يتم تمرير العنصر الذي يتم ادخاله كعامل لهذه الدالة. تعريف هذه الدالة يكون:

```
template<class elemType>
void arrayListType<elemType>::insert(const elemType& insertItem)
{
    int loc;

    if(length == 0) //the list is empty
        list[length++] = insertItem; //insert the item and
                                        //increment the length
    else
        if(length == maxSize)
            cerr<<"Cannot insert in a full list."<<endl;
        else
```

```

{
    loc = seqSearch(insertItem);

    if(loc == -1)                //the item to be inserted
                                //does not exist in the list
        list[length++] = insertItem;
    else
        cerr<<"The item to be inserted is already in "
        <<"the list. No duplicates are allowed."<<endl;
}
} //end insert

```

الدالة insert تستخدم الدالة seqSearch لتحديد ما اذا كان عنصر الإدخال insertItem موجود بالفعل في القائمة. بما أن الدالة seqSearch من النوع $O(n)$ اذن تكون دالة الإدخال insert من النوع $O(n)$.

الازالة:

الدالة remove تقوم بازالة عنصر من القائمة. يتم تمرير العنصر الذي يتم ازالته كمعامل لهذه الدالة. من أجل ازالة العنصر تقوم الدالة باستدعاء دالة العنصر seqSearch لتحديد ما اذا كان العنصر الذي يتم ازالته موجود أم لا في القائمة. اذا تم ايجاد العنصر الذي تتم ازالته في القائمة فان العنصر تتم ازالته من القائمة ويتم تقليل طول القائمة بمقدار 1. اذا تم ايجاد العنصر الذي تتم ازالته في القائمة تنتج الدالة seqSearch مؤشر العنصر في القائمة الذي تتم ازالته. يمكننا الآن استخدام المؤشر الناتج من الدالة seqSearch واستخدام الدالة removeAt لازالة العنصر من القائمة. لهذا يكون تعريف الدالة remove هو:

```

template<class elemType>
void arrayListType<elemType>::remove(const elemType& removeItem)
{
    int loc;

    if(length == 0)
        cerr<<"Cannot delete from an empty list."<<endl;
    else
    {
        loc = seqSearch(removeItem);

        if(loc != -1)
            removeAt(loc);
        else
            cout<<"The item to be deleted is not in the list."
            <<endl;
    }
}

} //end remove

```

الدالة remove تستخدم الدالات seqSearch و removeAt لازالة عنصر من القائمة. بما أن كل من هذه الدالات من النوع $O(n)$ ويتم استدعاؤها بالتتابع اذن ينتج أن الدالة remove من النوع $O(n)$.

تعقيد وقت عمليات القائمة:

الجدول التالي يقوم بايجاز تعقيد زمن عمليات القائمة:

| الدالة | التعقيد الزمني |
|----------------------|----------------|
| isEmpty | O(1) |
| isFull | O(1) |
| listSize | O(1) |
| maxListSize | O(1) |
| print | ***** |
| isItemAtEqual | ***** |
| insertAt | |
| insertEnd | |
| removeAt | |
| retrieveAt | |
| replaceAt | |
| clearList | |
| constructor | |
| destructor | |
| copy constructor | |
| انتقال معامل الاسناد | |
| seqSearch | |
| insert | |
| remove | |

إذا استخدمت الفئة `arrayListType` لمعالجة القائمة يجب أن تضمن أن المعاملات المتصلة ومعاملات الاسناد معرفة بالنسبة إلى البيانات التي تعالجها.

البرنامج التالي يختبر عمليات متنوعة على القوائم القائمة على المصفوفة.

```
#include <iostream>

#include "newString.h"
#include "arrayListType.h"

using namespace std;
```

```

int main()
{
    arrayListType<int> intList(100);           //Line 1
    arrayListType<newString> stringList;       //Line 2

    int counter;                               //Line 3
    int number;                                //Line 4

    cout<<"Line 5: Processing the integer list"
        <<endl;                               //Line 5
    cout<<"Line 6: Enter 5 integers: ";         //Line 6

    for(counter = 0; counter < 5; counter++)   //Line 7
    {
        cin>>number;                           //Line 8
        intList.insertAt(counter, number);     //Line 9
    }

    cout<<endl;                               //Line 10
    cout<<"Line 11: The list you entered is: "; //Line 11
    intList.print();                           //Line 12
    cout<<endl;                               //Line 13

    cout<<"Line 14: Enter the item to be deleted: "; //Line 14
    cin>>number;                               //Line 15
    intList.remove(number);                    //Line 16
    cout<<"Line 17: After removing "<<number
        <<" , the list is:"<<endl;             //Line 17
    intList.print();                           //Line 18
    cout<<endl;                               //Line 19

    newString str;                             //Line 20

    cout<<"Line 21: Processing the string list"
        <<endl;                               //Line 21

    cout<<"Line 22: Enter 5 strings: ";         //Line 22

    for(counter = 0; counter < 5; counter++)   //Line 23
    {
        cin>>str;                               //Line 24
        stringList.insertAt(counter, str);     //Line 25
    }

    cout<<endl;                               //Line 26
    cout<<"Line 27: The list you entered is: "
        <<endl;                               //Line 27
    stringList.print();                       //Line 28
    cout<<endl;                               //Line 29
}

```

```

cout<<"Line 30: Enter the string to be deleted: ";    //Line 30

cin>>str;                                           //Line 31
stringList.remove(str);                             //Line 32
cout<<"Line 33: After removing "<<str
    <<" , the list is:"<<endl;                     //Line 33
stringList.print();                                 //Line 34
cout<<endl;                                         //Line 35

int intListSize;                                    //Line 36

cout<<"Line 37: Enter the size of the integer "
    <<"list: ";                                     //Line 37
cin>>intListSize;                                   //Line 38

arrayListType<int> intList2(intListSize);           //Line 39

cout<<"Line 40: Processing the integer list"
    <<endl;                                         //Line 40
cout<<"Line 41: Enter "<<intListSize
    <<" integers: ";                               //Line 41

for(counter = 0; counter < intListSize; counter++) //Line 42
{
    cin>>number;                                     //Line 43
    intList2.insertAt(counter, number);             //Line 44
}

cout<<endl;                                         //Line 45
cout<<"Line 46: The list you entered is: "<<endl;    //Line 46
intList2.print();                                   //Line 47
cout<<endl;                                         //Line 48

return 0;
}

```

تنفيذ العينة: في هذه العينة يتم بطيئ مدحارب المسخدم.

الصف 5: معالجة قائمة الأعداد الصحيحة.

الصف 6: ادخال خمسة أعداد صحيحة: 23، 78، 56، 12، 79

الصف 11: القائمة التي أدخلتها: 23، 78، 56، 12، 79

الصف 14: ادخل العنصر الذي يتم حذفه: 56

الصف 17: بعد ازالة 56 تكون القائمة: 23، 78، 12، 79

الصف 21: معالجة قائمة المقطع.

الصف 22: أدخل 5 مقاطع: hello sunny warm winter summer

الصف 27: القائمة التي تم ادخالها هي:

hello sunny warm winter summer

الصف 30: ادخل المقطع الذي يتم حذفه: hello

الصف 33: بعد ازالة hello تكون القائمة:

sunny warm winter summer

الصف 37: ادخل حجم قائمة الأعداد الصحيحة: 7

الصف 40: معالجة قائمة الأعداد الصحيحة.

الصف 41: ادخل 7 أعداد صحيحة: 23، 67، 77، 10، 12، 89، 34

الصف 46: القائمة التي أدخلتها هي: 23، 67، 77، 10، 12، 89، 34

البرنامج السابق يعمل كما يلي. البيان في الصف 1 يعلن أن `intList` هدف من النوع `arrayListType`. عنصر البيانات `List` الخاص بـ `intList` عبارة عن مصفوفة من 100 مكون ونوع المكون يكون `int`. البيان في الصف 2 يعلن أن `stringList` هدف من النوع `arrayListType`. عنصر البيانات `stringList` الخاص بـ `intList` عبارة عن مصفوفة من 100 مكون (الحجم الافتراضي) ونوع المكون يكون `newString`. البيان في الصف 6 يحدد المستخدم على إدخال 5 أعداد صحيحة. البيان في الصف 8 يحصل على العدد التالي من تدفق المدخلات. البيان في الصف 9 يستخدم دالة العنصر `insertAt` الخاص بالدالة `intList` لتخزين العدد داخل `intList`. البيان في الصف 12 يستخدم دالة العنصر `print` الخاصة بالدالة `intList` لإخراج عناصر `intList`. البيان في الصف 14 يحدد المستخدم على إدخال العدد الذي يتم حذفه من `intList` والبيان في الصف 15 يحصل على العدد الذي يتم حذفه من تدفق المدخلات. البيان في الصف 16 يستخدم دالة العنصر `remove` الخاصة بالدالة `intList` لإزالة العدد من `intList`.

البيانات من الصف 21 وحتى 35 تعمل بنفس الطريقة التي تعمل بها البيانات من الصف 5 وحتى 19. تقوم هذه البيانات بمعالجة قائمة من المقاطع.

البيان في الصف 37 يحدد المستخدم على إدخال حجم قائمة من الأعداد الصحيحة والبيان في الصف 38 يقوم بتخزين هذا الحجم داخل المتغير `intListSize`. البيان في الصف 39 يعلن أن `intList2` هدف من النوع `arrayListType` مثل أن عنصر البيانات `List` الخاص بالدالة `intListType` عبارة عن مصفوفة يكون فيها عدد المكونات `intListSize`. هذا يوضح أنه يمكنك تحديد حجم المصفوفة لتخزين البيانات أثناء تنفيذ البرنامج. معنى البيانات المتبقية واضح.

مثال برمجة: عمليات حدودية:

تعلمت في منهج الجبر أو التفاضل والتكامل بالجامعة أن الحدودية $p(x)$ في أحد المتغيرات يكون تعبير من الصيغة:

$$p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} + a_nx^n,$$

حيث a_i 's عبارة عن أعداد حقيقية (أو مركبة) و n عدد كامل غير سالب. إذا كانت $p(x) = a_0$

اذن يطلق على $p(x)$ تعدد حدود ثابت. اذا كان $p(x)$ حد ثابت متعدد غير صفري اذن يتم تعريف درجة $p(x)$ عند صفر.

مع هذا لا تكون درجة الحد الصفري معرفة وللهدف من هذا البرنامج سوف ننظر الى درجة تلك الحدود عند الصفر. اذا لم يكن $p(x)$ ثابت و كانت $a_n \neq 0$ سمى n درجة $p(x)$ أي درجة الحد الغير ثابت المعرف أن يكون أس القوة العظمى من x .
العمليات الأساسية التي يتم اجرائها على الحدود هي الاضافة والطرح والضرب والقسمة. يمكنك كذلك تقييم الحد عند نقطة معينة. على سبيل المثال افترض أن:

$$p(x) = 1 + 2x + 3x^2,$$

and

$$q(x) = 4 + x.$$

درجة $p(x)$ تكون 2 ودرجة $q(x)$ تكون 1. فضلاً عن هذا:

$$p(2) = 1 + 2 \cdot 2 + 3 \cdot 2^2 = 17$$

$$p(x) + q(x) = 5 + 3x + 3x^2$$

$$p(x) - q(x) = -3 + x + 3x^2$$

$$p(x) * q(x) = 4 + 9x + 14x^2 + 3x^3$$

الهدف من مثال البرمجة هذا هو تصميم وتطبيق الفئة `polynomialType` لأداء العمليات المتنوعة المتعددة الحدود في برنامج ما.

لكي تكون محدداً نقوم في هذا البرنامج بتطبيق العمليات التالية على الحدود المتعددة:

1. تقييم الحد المتعدد عند قيمة محددة.

2. اضافة حدود متعددة.

3. طرح حدود متعددة.

4. ضرب حدود متعددة.

بالاضافة الى هذا نفترض أن معاملات الحدود المتعددة عبارة عن أعداد حقيقية. سوف يطلب منك في تمرين البرمجة رقم 12 بتعميمها حتى تكون المعاملات أعداد مركبة كذلك.

لتخزين حد متعدد نستخدم المصفوفة الحركية كما يلي: افترض أن $p(x)$ حد متعدد من الدرجة $n \geq 0$. لتكن القائمة مصفوفة حجمها $n + 1$. يتم تخزين المعامل a_1 من x^1 في `list[i]`. انظر

شكل 3-29.

| | | | | | | | |
|--------|-------|-------|-----|-------|-----|-----------|-------|
| | [0] | [1] | | [i] | | [n-1] | [n] |
| $p(x)$ | a_0 | a_1 | ... | a_i | ... | a_{n-1} | a_n |

شكل 3-29: الحد المتعدد $p(x)$
 شكل 3-29 يوضح أنه إذا $p(x)$ حد متعدد من الدرجة n فإننا نحتاج إلى مصفوفة حجمها $n + 1$ لتخزين معاملات $p(x)$. افترض أن:

$$p(x) = 1 + 8x - 3x^2 + 5x^4 + 7x^8$$

اذن تكون المصفوفة التي تخزن معاملات $p(x)$ كما هي معطاة في الشكل 3-30.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $p(x)$ | 1 | 8 | -3 | 0 | 5 | 0 | 0 | 0 | 7 |

شكل 3-30: حد متوسط $p(x)$ من الدرجة 8 ومعاملاته.
 بالمثل إذا كانت:

$$q(x) = -5x^2 + 16x^5,$$

اذن تكون المصفوفة التي تخزن معاملات $q(x)$ كما هي معطاة في الشكل 3-31.

| | [0] | [1] | [2] | [3] | [4] | [5] |
|--------|-----|-----|-----|-----|-----|-----|
| $q(x)$ | 0 | 0 | -5 | 0 | 0 | 16 |

شكل 3-31: الحد المتعدد $q(x)$ من الدرجة 5 ومعاملاته.
 بعد هذا نقوم بتعريف العمليات + و- . افترض أن:

$$p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} + a_nx^n,$$

and

$$q(x) = b_0 + b_1x + \dots + b_{m-1}x^{m-1} + b_mx^m.$$

Let $t = \max(n, m)$. Then

$$p(x) + q(x) = c_0 + c_1x + \dots + c_{t-1}x^{t-1} + c_tx^t,$$

where for $i = 0, 1, 2, \dots, t$

$$c_i = \begin{cases} a_i + b_i & \text{if } i \leq \min(n, m) \\ a_i & \text{if } i > m \\ b_i & \text{if } i > n \end{cases}$$

يمكن تعريف الفرق $p(x) - q(x)$ الخاص بـ $p(x)$ و $q(x)$ بالمثل. ينتج أن درجات الحدود المتعددة $p(x) + q(x)$ و $p(x) - q(x)$ أصغر من أو تساوي $\max(n, m)$.
الناتج $p(x) * q(x)$ لكل من $p(x)$ و $q(x)$ يتم تعريفه كما يلي:

$$(x) * q(x) = d_0 + d_1x + \dots + d_{n+m}x^{n+m}.$$

يتم اعطاء المعامل d_k لكل $k = 0, 1, 2, \dots, n + m$ عن طريق الصيغة:

$$d_k = a_0 * b_k + a_1 * b_{k-1} + \dots + a_k * b_0,$$

حيث إذا لم تكن a_i أو b_i مـدة يكون من المفروض أن تكون صفر. على سبيل المثال:

$$d_0 = a_0 b_0$$

$$d_1 = a_0 b_1 + a_1 b_0$$

$$d_{n+m} = a_n b_m$$

لقد تعلمت في الفصل 2 كيفية ائقال معاملات متنوعة. يقوم هذا البرنامج بائقال المعاملات + و- و* لاجراء جمع وطرح وضرب الحدود المتعددة. فضلاً عن هذا يمكننا ائقال معامل استدعاء الدالة () لتقييم حد متعدد عند قيمة معطاة. لتبسيط مدخلات ومخرجات الحدود المتعددة يتم ائقال كل من معاملات << و >>.

بما أن معاملات الحد المتعدد تكون مخزنة في مصفوفة حركية فاننا نقوم باستخدام الفئة `arrayListType` لتخزين والسيطرة على معاملات الحد المتعدد. في الحقيقة نستمد الفئة `polynomialType` لتطبيق عمليات الحدود المتعددة من الفئة `arrayListType` التي تحتاج منا تطبيق العمليات اللازمة فقط للتلاعب في الحدود المتعددة.
الفئة التالية تقوم بتعريف الحدود المتعددة كنوع بيانات مجرد.

```
class polynomialType: public arrayListType<double>

    friend ostream& operator<< (ostream&, const polynomialType&);
        //Overload the stream insertion operator.
    friend istream& operator>> (istream&, polynomialType&);
        //Overload the stream extraction operator.
public:
    polynomialType operator+(const polynomialType&);
        //Overload the operator +
    polynomialType operator-(const polynomialType&);
        //Overload the operator -
    polynomialType operator*(const polynomialType&);
        //Overload the operator *

    double operator() (double x);
        //Overload the operator () to evaluate the
        //polynomial at a given point.
        //Postcondition: The value of the polynomial at x
```

```

// is calculated and returned.

polynomialType(int size = 100);
//constructor

int min(int x, int y);
//Function to return the smaller of x and y.
int max(int x, int y);
//Function to return the larger of x and y.
};

```

في التمرين 19 (في نهاية هذا الفصل) يطلب منك رسم بيان لغة التشكيل الموحدة للفئة `polynomialType`.

إذا كان $p(x)$ حد متوسط من الدرجة 3 يمكننا عمل هدف `p` مثلاً من النوع `polynomialType` وتحديد حجم قائمة المصفوفة عند 4. البيان التالي يعلن الهدف `p`:

`polynomialType p(4);`

يتم تخزين درجة الحد المتعدد في عنصر البيانات `length` الذي يتم توريثه من الفئة `arrayListType`.

بعد هذا نناقش تعريفات الدالات.

المقوم يحدد قيمة `length` عند حجم المصفوفة ويقوم بتهيئة قائمة المصفوفة عند صفر.

```

polynomialType::polynomialType(int size)
: arrayListType<double>(size)
{
    length = size;
    for(int i = 0; i < size; i++)
        list[i] = 0;
}

```

تعريف الدالة لاثقال المعامل () معطى فيما بعد.

```

double polynomialType::operator() (double x)
{
    double value = 0.0;
    for(int i = 0; i < length; i++)
    {
        if(list[i] != 0.0)
            value = value + list[i] * pow(x,i);
    }
    return value;
}

```

افترض أن $p(x)$ حد متعدد من الدرجة n وكان $q(x)$ حد متعدد من الدرجة m . إذا كان $n = m$ اذن يقوم المعامل + بجمع المعاملات المتوافقة لكل من $p(x)$ و $q(x)$. إذا كانت $n > m$ اذن تتم إضافة معاملات m الأولى لـ $p(x)$ مع المعاملات المتوافقة من $q(x)$. يتم بعد هذا نسخ معاملات $p(x)$ المتبقية داخل الحد المتعدد المحتوي على مجموع $p(x)$ و $q(x)$. بالمثل إذا كانت $n < m$ تتم إضافة معاملات n الأولى لـ $q(x)$ مع معاملات $p(x)$ المتوافقة ثم يتم بعد هذا نسخ معاملات $q(x)$ المتبقية داخل الحد المتعدد المحتوي على المجموع. تعريف المعامل - متشابه مع تعريف المعامل +. تكون تعريفات دالات هذين المعاملين كما يلي:

```
polynomialType polynomialType::operator+
    (const polynomialType& right)
{
    int size = max(length, right.length);
    int i;

    polynomialType temp(size); //polynomial to store the sum

    for(i = 0; i < min(length, right.length); i++)
        temp.list[i] = list[i] + right.list[i];

    if(size == length)
        for(i = min(length, right.length); i < length; i++)
            temp.list[i] = list[i];
    else
        for(i = min(length, right.length); i < right.length; i++)
            temp.list[i] = right.list[i];

    return temp;
}

polynomialType polynomialType::operator-
    (const polynomialType& right)
{
    int size = max(length, right.length);
    int i;

    polynomialType temp(size); //polynomial to store the
                                //difference

    for(i = 0; i < min(length, right.length); i++)
        temp.list[i] = list[i] - right.list[i];

    if(size == length)
        for(i = min(length, right.length); i < length; i++)
            temp.list[i] = list[i];
    else
        for(i = min(length, right.length); i < right.length; i++)
            temp.list[i] = -right.list[i];

    return temp;
}
```

تعريف الدالة لاثقال المعامل * لضرب حدين متعددين يتم تركه كتمرين لك. انظر تمرين البرمجة رقم 10 في نهاية هذا الفصل. تعريفات الدالات المتبقية من الفئة polynomialType تكون:

```
int polynomialType::min(int x, int y)
{
    if(x <= y)
        return x;
    else
        return y;
}

int polynomialType::max(int x, int y)
{
    if(x >= y)
        return x;
    else
        return y;
}

ostream& operator<<(ostream& os, const polynomialType& p)
{
    int i;
    int indexFirstNonzeroCoeff = 0;

    for(i = 0; i < p.length; i++)    //determine the index of
                                      //the first nonzero
                                      //coefficient

        if(p.list[i] != 0.0)
        {
            indexFirstNonzeroCoeff = i;
            break;
        }

    if(indexFirstNonzeroCoeff < p.length)
    {
        if(indexFirstNonzeroCoeff == 0)
            os<<p.list[indexFirstNonzeroCoeff]<<" ";
        else
            os<<p.list[indexFirstNonzeroCoeff]<<"x^"
              <<indexFirstNonzeroCoeff<<" ";
    }
}
```

```

    for(i = indexFirstNonzeroCoeff + 1; i < p.length; i++)
    {
        if(p.list[i] != 0.0)
            if(p.list[i] >= 0.0)
                os<<" + "<<p.list[i]
                <<"x^"<<i<<" ";
            else
                os<<" - "<<-p.list[i]
                <<"x^"<<i<<" ";
        }
    }
    else
        os<<"0";

    return os;
}

istream& operator>>(istream& is, polynomialType& p)
{
    cout<<"The degree of this polynomial is: "
        <<p.length - 1<<endl;
    for(int i = 0; i < p.length; i++)
    {
        cout<<"Enter the coefficient of x^"<<i<<": ";
        is>>p.list[i];
    }

    return is;
}

//Test program: Polynomial Operations

#include <iostream>

#include "polynomialType.h"

using namespace std;

int main()
{
    polynomialType p(8); //Line 1
    polynomialType q(4); //Line 2
    polynomialType t; //Line 3

    cin>>p; //Line 4
    cout<<endl<<"Line 5: p(x): "<<p
        <<endl; //Line 5
}

```

```

cout<<"Line 6: p(5): "<<p(5)
    <<endl<<endl; //Line 6

cin>>q; //Line 7
cout<<endl<<"Line 8: q(x): "<<q
    <<endl<<endl; //Line 8

t = p + q; //Line 9

cout<<"Line 10: p(x) + q(x): "
    <<t<<endl; //Line 10

cout<<"Line 11: p(x) - q(x): "
    <<p - q<<endl; //Line 11

return 0;
}

```

تنفيذ الـ

درجة هذا الحد المتعدد: 7

ادخل معامل x^0 : 0

ادخل معامل x^1 : 1

ادخل معامل x^2 : 4

ادخل معامل x^3 : 0

ادخل معامل x^4 : 0

ادخل معامل x^5 : 0

ادخل معامل x^6 : 0

ادخل معامل x^7 : 6

الصف 5: $p(x) : 1x^1 + 4x^2 + 6x^7$

الصف 6: $p(5) : 468855$

درجة هذا الحد المتوسط: 3

ادخل معامل x^0 : 1

ادخل معامل x^1 : 2

ادخل معامل x^2 : 0

ادخل معامل x^3 : 3

الصف 8: $q(x) : 1 + 2x^1 + 3x^3$

الصف 10: $p(x) + q(x) : 1 + 3x^1 + 4x^2 + 3x^3 + 6x^7$

الصف 11: $p(x) - q(x) : -1 - 1x^1 + 4x^2 - 3x^3 + 6x^7$

مراجعة سريعة:

1. تتضمن متغيرات المؤشر عناوين المتغيرات الأخرى مثل قيمها.

2. في C++ لا يرتبط اسم بنوع بيانات المؤشر.

3. يتم اعلان متغير المؤشر باستخدام علامة نجمة (*) بين نوع البيانات والمتغير. على سبيل المثال تقوم البيانات:

int *p;

char *ch;

باعلان أن p و ch متغيرات مؤشر. تشير قيمة P الى مساحة الذاكرة من النوع int وتشير قيمة ch الى مساحة الذاكرة من النوع char. عادةً تسمى p متغير مؤشر من النوع int ويسمى ch متغير مؤشر من النوع char.

4. في C++ يسمى & عنوان المعامل.

5. يقدم عنوان المعامل عنوان معاملة. على سبيل المثال اذا كان p متغير مؤشر من النوع int و num متغير int فان البيان:

p = #

يحدد قيمة p عند عنوان num.

6. عند استخدام * كمعامل أحادي يطلق عليها معامل ازالة الاشارة.

7. موقع الذاكرة الذي يشير اليه قيمة متغير المؤشر يكون في المتناول باستخدام معامل ازالة الاشارة *. على سبيل المثال اذا كان p متغير مؤشر من النوع int فان البيان:

*p = 25;

يحدد قيمة موقع الذاكرة المشار اليه بواسطة قيمة p عند 25.

8. يمكنك استخدام معامل تناول العنصر وهو السهم -> لتناول مكون الهدف المشار اليه بواسطة المؤشر.

9. يتم تهيئة متغيرات المؤشر باستخدام صفر، أو NULL أو عنوان متغير مؤشر من نفس النوع.

10. القيمة الصحيحة الوحيدة التي يمكن تحديدها مباشرةً لمتغير مؤشر هي صفر.

11. العمليات الحسابية الوحيدة التي يمكن اجرائها على متغيرات المؤشر هي الزيادة (++) والنقص (--). واطراف عدد صحيح الى متغير مؤشر، وطرح عدد صحيح من متغير مؤشر، وطرح مؤشر من مؤشر آخر.

12. المؤشر مختلف عن الحلول الحسابية العادية. عندما تتم اضافة عدد صحيح الى مؤشر تكون القيمة المضافة الى قيمة متغير المؤشر هي المرات الصحيحة لحجم الهدف الذي يشير اليه المؤشر. بالمثل عندما يتم طرح عدد صحيح من مؤشر ما تكون القيمة المطروحة من قيمة المؤشر هي المرات الصحيحة لحجم الهدف الذي يشير اليه المؤشر.

13. يمكن مقارنة متغيرات المؤشر باستخدام عمليات متصلة. (انها تعطي معنى لمقارنة مؤشرات من نفس النوع).

14. يمكن تحديد قيمة احدى متغيرات المؤشر عند متغير مؤشر آخر من نفس النوع.
15. المتغير الذي يتم خلقه أثناء تنفيذ البرنامج يسمى متغير حركي.
16. يتم استخدام المعامل new لعمل متغير حركي.
17. يتم استخدام المعامل delete لازالة تخصيص الذاكرة التي يحتلها المتغير الحركي.
18. في C++ يكون كل من new و delete كلمات محفوظة.
19. المعامل new له صيغتان: واحدة لعمل متغير حركي واحد وأخرى لعمل مصفوفة من المتغيرات الحركية.
20. اذا كان p مؤشر من النوع int فان البيان:
 $p = \text{new int};$
يخصص مخزن من النوع int في مكان ما بالذاكرة ويقوم بتخزين عنوان المخزن المخصص في p.
21. المعامل delete له صيغتان: واحدة لازالة تخصيص الذاكرة التي يشغلها متغير حركي واحد وأخرى لازالة تخصيص الذاكرة التي تشغلها مصفوفة من المتغيرات الحركية.
22. اذا كان p متغير من النوع int اذن البيان:
 $\text{Delete } p;$
يزيل تخصيص الذاكرة المشار اليها بواسطة p.
23. اسم المصفوفة عبارة عن مؤشر ثابت وهو يشير دائماً الى نفس موقع الذاكرة وهو موقع مكون الذاكرة الأول.
24. لعمل مصفوفة حركية يتم استخدام صيغة المعامل new التي تصنع مصفوفة من المتغيرات الحركية. على سبيل المثال اذا كان p مؤشر من النوع int فان البيان:
 $p = \text{new int } [10];$
يصنع مصفوفة من 10 مكونات من النوع int ويتم تخزين العنوان الأساسي للمصفوفة في p ونطلق على p مصفوفة حركية.
25. يمكن استخدام تدوين المصفوفة لتناول مكونات المصفوفة الحركية. على سبيل المثال افترض أن p مصفوفة حركية من 10 مكونات. اذن تشير $p[0]$ الى مكون المصفوفة الأول وتشير p [1] الى مكون المصفوفة الثاني وهكذا. بشكل خاص تشير $p[i]$ الى المكون $(i + 1)$ من المصفوفة.
26. المصفوفة التي يتم عملها أثناء تنفيذ البرنامج يطلق عليها مصفوفة حركية.
27. اذا كانت p مصفوفة حركية، اذن البيان:
 $\text{delete } [] p;$
يزيل تخصيص الذاكرة التي يشغلها p – أي مكونات p.

28. في النسخ السطحي يشير مؤشران أو أكثر من نفس النوع الى نفس مساحة الذاكرة أي يشيرون الى نفس البيانات. (انظر القسم "النسخ السطحي مقابل النسخ العميق والمؤشرات").

29. في النسخ العميق يكون لكل مؤشر من نفس النوع نسخته الخاصة من البيانات. (انظر القسم "النسخ السطحي مقابل النسخ العميق والمؤشرات").

30. اذا كانت الفئة لها مدمر فانه يتم تنفيذ المدمر تلقائياً في أى وقت يخرج فيه هدف الفئة من المجال.

31. اذا كانت الفئة لها عناصر بيانات مؤشر فان معاملات الاسناد المدمجة توفر نسخ سطحي للبيانات.

32. يتم تنفيذ مقوم النسخ عندما يتم اعلان الهدف وتتم تهيئته باستخدام قيمة هدف آخر وعندما يتم تمرير الهدف بالقيمة كمعامل.

33. القائمة عبارة عن مجموعة من العناصر من نفس النوع.

34. العمليات التي يتم اجرائها عادةً على القوائم هي: عمل قائمة، أو تحديد ما اذا كانت القائمة فارغة، أو تحديد ما اذا كانت القائمة ممثلة، أو ايجاد حجم القائمة، أو تدمير أو ازالة القائمة، أو تحديد ما اذا كان العنصر هو نفسه المعطى في عناصر القائمة، أو ادخال عنصر في القائمة بموقع محدد، أو ازالة عنصر من القائمة من موقع محدد، أو استبدال عنصر بموقع محدد بعنصر آخر، أو استرجاع عنصر من القائمة من موقع محدد، وبحث القائمة عن عنصر معطى.

تمارين:

1. ضع علامة صح أم خطأ أمام الجمل التالية:

أ. في C++ المؤشر عبارة عن كلمة محفوظة.

ب. في C++ يتم اعلان متغيرات المؤشر باستخدام الكلمة المحفوظة pointer.

ت. البيان delete p; يزيل تخصيص مؤشر المتغير p.

ث. البيان delete p; يزيل تخصيص المتغير الحركي المشار اليه بواسطة p.

ج. باعطاء البيان:

```
int list [10];
```

```
int *p
```

يكون البيان:

```
p = list;
```

صحيح في C++.

ح. باعطاء الاعلان:

```
int *p;
```

يقوم البيان:

```
p = new int [50];
```

بتخصيص مصفوفة من 50 مكون من النوع int ويحتوي p على العنوان الأساسي للمصفوفة.

خ. عنوان المعامل ينتج عنوان وقيمة معاملة.

د. اذا كان p متغير مؤشر فان البيان $p = p * 2$ يكون صحيح في C++.

2. باعطاء البيان:

```
int x;
```

```
int *p;
```

```
int *q;
```

حدد أي الجمل التالية صحيحة وأيها غير صحيح. اذا كانت الجملة غير صحيحة فسر السبب.

أ. $p = q$;

ب. $*p = 56$;

ت. $p = x$;

ث. $*p = *q$;

ج. $q = \&x$;

ح. $*p = q$;

3. ما هي مخرجات الكود التالي من C++؟

```
int x;  
int y;  
int *p = &x;  
int *q = &y;  
*p = 35;  
*q = 98;  
*p = *q;  
cout<<x<<" "<<y<<endl;  
cout<<*p<<" "<<*q<<endl;
```

4. ما هي مخرجات الكود التالي من C++؟

```

int x;
int y;
int *p = &x;
int *q = &y;
x = 35; y = 46;
p = q;
*p = 78;
cout<<x<<" "<<y<<endl;
cout<<*p<<" "<<*q<<endl;

```

5. باعطاء الاعلان التالي:

```
int num = 6;
```

```
int *p = &num;
```

أياً من البيانات التالية يزيد قيمة num؟

أ. p++;

ب. (*p) ++;

ت. num++;

ث. (*num) ++;

6. ما هي مخرجات الكود التالي؟

```

int *p;
int *q;
p = new int;
q = p;
*p = 46;
*q = 39;
cout<<*p<<" "<<*q<<endl;

```

7. ما هي مخرجات الكود التالي؟

```

int *p;
int *q;
p = new int;
*p = 43;
q = p;
*q = 52;
p = new int;
*p = 78;
q = new int;
*q = *p;
cout<<*p<<" "<<*q<<endl;

```

ما الخطأ في الكود التالي؟

```
int *p; //Line 1
int *q; //Line 2

p = new int; //Line 3
*p = 43; //Line 4

q = p; //Line 5
*q = 52; //Line 6

delete q; //Line 7

cout<<*p<<" "<<*q<<endl; //Line 8
```

8. ما هي مخرجات الكود التالي؟

```
int x;
int *p;
int *q;
p = new int[10];
q = p;
*p = 4;

for(int j = 0; j < 10; j++)
{
    x = *p ;
    p++;

    *p = x + j ;
}

for(int k = 0; k < 10; k++)
{
    cout<<*q<<" ";
    q++;
}
cout<<endl;
```

9. ما هي مخرجات الكود التالي؟

```
int *secret;
int j;

secret = new int[10];
secret[0] = 10;
for(j = 1; j < 10; j++)
    secret[j] = secret[j - 1] + 5;
for(j = 0; j < 10; j++)
    cout<<secret[j]<<" ";
cout<<endl;
```

10. وضح الفرق بين النسخ السطحي والنسخ العميق للبيانات.

11. ما الخطأ في الكود التالي؟

```
int *p; //Line 1
int *q; //Line 2

p = new int[5]; //Line 3
*p = 2; //Line 4

for(int i = 1; i < 5; i++) //Line 5
    p[i] = p[i - 1] + i; //Line 6

q = p; //Line 7

delete [] p; //Line 8

for(int j = 0; j < 5; j++) //Line 9
    cout<<q[j]<<" "; //Line 10

cout<<endl; //Line 11
```

12. ما هي مخرجات الكود التالي؟

```
int *p;
int *q;
int i;

p = new int[5];

p[0] = 5;

for(i = 1; i < 5; i++)
    p[i] = p[i - 1] + 2 * i;

cout<<"Array p: ";
for(i = 0; i < 5; i++)
    cout<<p[i]<<" ";
cout<<endl;

q = new int[5];

for(i = 0; i < 5; i++)
    q[i] = p[4 - i];

cout<<"Array q: ";
for(i = 0; i < 5; i++)
    cout<<q[i]<<" ";

cout<<endl;
```

13. ما الهدف من مقوم النسخ؟
14. اذكر موقفين يتم تنفيذ مقوم النسخ فيهما.
15. اذكر ثلاثة أشياء يجب عليك عملها للفئات التي لها عناصر بيانات مؤشر.
16. أ. قم بنقل المعامل + للفئة newString لأداء تركيز المقطع. على سبيل المثال إذا كان المقطع الأول s1 هو "Hello" والمقطع الثاني s2 هو "there" فإن البيان s1 = s3 + s2 يجب أن يحدد "Hello there" للمقطع الثالث s3 ويكون s3 عبارة عن أهداف newString.
- ت. قم بنقل المعامل += للفئة newString لأداء العمليات المقطعية التالية: افترض أن s1 هي "Hello" و s2 هي "there". اذن البيان:


```
s1 += s2;
```

 يجب أن يحدد "Hello there" عند s1 حيث s1 و s2 أهداف من newString.
17. ما هو تأثير البيانات التالية؟
 - أ. `arrayListType<int> intList (100);`
 - ب. `arrayListType<string > stringList (1000);`
 - ت. `arrayListType<double > salesList (-10);`
18. ارسم بيان لغة التشكيل الموحدة للفئة polynomialType وقم كذلك بتوضيح تسلسل التوريث.

تمارين برمجة

1. اكتب برنامج لاختبار معامل لاسناد للفئة cAssignmentOprOverload من المثال 3-5.
2. اكتب برنامج لاختبار مقوم النسخ للفئة pointerDataClass للمثال 3-6.
3. قم بمد تعريف الدالة newString كما يلي:
 - أ. قم بنقل المعاملات + و += لأداء عمليات المقطع.
 - ب. قم بإضافة الدالة length لانتاج طول المقطع.
 - ت. اكتب تعريفات الدالة لتطبيق العمليات المعرفة في الجزء أ.
 - ث. اكتب برنامج اختبار لاختبار عمليات متنوعة على أهداف newString.
4. أ. أعد كتابة تعريف الفئة newString كما هي معرفة وامتدة في تمرين البرمجة رقم 3 حتى يتم ائفال المعاملات المرتبطة كدالات لغير العناصر.
 - ب. اكتب تعريف الفئة newString كما هي مصممة في الجزء أ.
 - ت. اكتب برنامج اختبار لاختبار عمليات متنوعة على الفئة newString.

5. الدالة `removeAt` للفئة `arrayListType` تقوم بإزالة عنصر من القائمة عن طريق نقل عناصر القائمة. بالرغم من هذا إذا كان العنصر الذي تتم إزالته موجود في بداية القائمة والقائمة كانت كبيرة فقد يستهلك الكثير من زمن الحاسب الآلي. بما أن عناصر القائمة غير موجودة في ترتيب محدد يمكنك ببساطة إزالة العنصر عن طريق استبدال العنصر الأخير بالقائمة بالعنصر الذي تتم إزالته وتقليل طول القائمة. أعد كتابة تعريف الدالة `removeAt` باستخدام هذا الأسلوب.

6. الدالة `remove` للفئة `arrayListType` تقوم بإزالة الظهور الأول للعنصر فقط. أضف الدالة `removeAll` إلى الفئة `arrayListType` التي قد تزيل جميع مواقع ظهور العنصر. قم أيضاً بكتابة تعريف الدالة `removeAll` وبرنامج لاختبار هذه الدالة.

7. أضف الدالة `min` إلى الفئة `arrayListType` لإنتاج العنصر الأصغر من القائمة وكذلك قم بكتابة تعريف الدالة `min` وبرنامج لاختبار هذه الدالة.

8. أضف الدالة `max` إلى الفئة `arrayListType` لإنتاج العنصر الأكبر من القائمة وكذلك قم بكتابة تعريف الدالة `max` وبرنامج لاختبار هذه الدالة.

9. يتم ائصال المعاملات + و- كدالات عنصر للفئة `polynomialType`. أعد مثال البرمجة على عمليات الحدود المتعددة حتى يتم ائصال تلك المعاملات كدالات لغير العنصر. قم كذلك بكتابة برنامج اختبار لاختبار تلك المعاملات.

10. اكتب تعريف الدالة لإئصال المعامل * (كدالة عنصر) للفئة `polynomialType` لضرب حدين متعددين، وقم كذلك بكتابة برنامج اختبار لاختبار المعامل *.

11. اجعل الحد المتعدد $p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} + a_nx^n$ رجة n حيث أي عدد حقيقي (أو من a_i) و n عدد صحيح غير سالب. الدالة المستمدة من $p(x)$ التي تتم كتابتها $p'(x)$ وتعرف بأن $p'(x) = a_1 + 2a_2x + \dots + na_nx^{n-1}$.

إذا كان $p(x) = a_1 + 2a_2x + \dots + na_nx^{n-1}$ قم بإئصال المعامل ~ كدالة عنصر للفئة `polynomialType` حتى يقوم المعامل ~ بإنتاج الدالة المشتقة من الحد المتعدد.

12. الفئة `polynomialType` كما هي معطاة في مثال البرمجة عبارة عن عمليات متعددة الحدود تقوم بمعالجة الحدود المتعددة ذات المعاملات الموجودة في شكل أعداد حقيقية. قم بتصميم وتطبيق فئة مماثلة يمكن استخدامها لمعالجة الحدود المتعددة ذات المعاملات الموجودة في شكل أعداد مركبة. يجب أن تقوم فنتك بإئصال المعاملات + و- و* لأداء الجمع والطرح والضرب وكذلك المعامل () لتقييم الحد المتعدد كعدد مركب معطى. قم كذلك بكتابة برنامج لاختبار عمليات متنوعة.

13. باستخدام الفئات قم بتصميم دفتر عناوين مباشر لتتبع الأسماء والعناوين وأرقام الهواتف وتواريخ ميلاد أفراد العائلة و الأصدقاء المقربين وزملاء العمل. يجب أن يكون برنامجك قادراً على معالجة حد أقصى يبلغ 500 مدخل.

أ. قم بتعريف الفئة `addressType` يمكنها تخزين عنوان الشارع والمدينة والدولة والرمز البريدي. استخدم الدالات المناسبة لطبع وتخزين العنوان واستخدم كذلك مقومات لتهيئة عناصر العنوان تلقائياً .

ب. قم بتعريف الفئة `extPersonType` باستخدام الفئة `personType` (كما هي معرفة في المثال 1-6 من الفصل الأول) والفئة `dateType` (كما هي معرفة في تمرين البرمجة 2 من الفصل 2) والفئة `addressType`. أضف عنصر بيانات الى هذه الفئة لتصنيف الشخص كعضو بالعائلة أو صديق أو زميل عمل. أضف أيضاً عنصر بيانات لتخزين رقم الهاتف. أضف (أو ابطل) دالات طبع وتخزين المعلومات المناسبة استخدم المقومات لتهيئة عناصر البيانات تلقائياً .

ت. استمد الفئة `addressBookType` من الفئة `arrayListType` كما هي معرفة في هذا الفصل حتى يمكن للهدف من النوع `addressBookType` تخزين أهداف من النوع `extPersonType`. يجب أن يكون الهدف من النوع `addressBookType` قادراً على معالجة حد أقصى قدره 500 مدخل. أضف أي عمليات لازمة على الفئة `addressBookType` حتى يمكن للبرنامج أداء العمليات التالية:

- (i) تحميل البيانات داخل دفتر العناوين من القرص.
- (ii) البحث عن شخص عن طريق الاسم الأخير.
- (iii) طباعة عنوان ورقم هاتف وتاريخ ميلاد (إذا وجد) شخص محدد.
- (iv) طباعة أسماء الأفراد الذين تكون أعياد ميلادهم في شهر محدد أو بين تاريخين محددين.
- (v) طباعة أسماء جميع الأفراد الذين يتماثل وضعهم مثل عائلة أو صديق أو عمل.
- (vi) طباعة أسماء جميع الأفراد بين الاسمين الأخيرين.

14. (مصفوفات آمنة) في `C++` لا يتم التأكد مما اذا كان مؤشر المصفوفة خارج الحدود أم لا. أثناء تنفيذ البرنامج يمكن أن يتسبب مؤشر المصفوفة الخارج عن الحدود في مشكلات خطيرة. تذكر أيضاً أن مؤشر المصفوفة في `C++` يبدأ عند صفر. قم بتصميم الفئة `safeArray` تقوم بحل مشكلة مؤشر المصفوفة الخارج عن الحدود وتسمح للمستخدم ببدا مؤشر المصفوفة عند أي عدد صحيح موجب أم سالب. يجب أن يكون كل هدف من النوع `safeArray` مصفوفة من

النوع int. أثناء التنفيذ عند تناول مكون المصفوفة وإذا كان المؤشر خارج الحدود يجب أن يقوم البرنامج بالتوقف مع رسالة خطأ مناسبة. على سبيل المثال:

safeArray list (2, 13);

safeArray yourList (-5, 9);

في هذا المثال تكون list مصفوفة من 11 مكونات ونوع المكون int والمكونات هي [2] list و [3] list و [12] list. yourList كذلك عبارة عن مصفوفة من 15 مكون ونوع المكونات int والمكونات هي [-5] yourList و [-4] yourList و و [0] list و و [8] yourList.

15. تمرين البرمجة رقم 14 يعالج مصفوفات int فقط. اعد تصميم الفئة safeArray باستخدام قوالب الفئة حتى يمكن استخدام الفئة في أي تطبيق يتطلب مصفوفات لمعالجة البيانات.

16. قم بتصميم فئة لأداء عمليات مصفوفة متنوعة والمصفوفة عبارة عن مجموعة من الأعداد المرتبة في صفوف وأعمدة. لهذا لكل عنصر في المصفوفة موضع في الصف وموضع في العمود. إذا كانت A مصفوفة من 5 صفوف و 6 أعمدة نقول أن المصفوفة A حجمها 6×5 وأحياناً نعبر عنها بواسطة . بشكل واضح المكان التتمة 6×5 وتخزين مصفوفة هو في مصفوفة ثنائية الأبعاد. يمكن جمع أو طرح مصفوفتين إذا كان حجمهما واحد. افترض أن مصفوفتان من الحجم $m \times n$ $A = [a_{ij}]$ $B = [b_{ij}]$ عنصر A في الصف i والعمود j وهكذا. يتم الوصل a_{ij} إلى مجموع والفرق بين A و B عن طريق:

$$A + B = [a_{ij} + b_{ij}]$$

$$A - B = [a_{ij} - b_{ij}]$$

يتم تعريف ضرب A و B ($A * B$) إذا كان عدد صفوف A هو نفسه عدد صفوف B. إذا

كانت A مصفوفة حجمها $m \times n$ وإذا كانت B مصفوفة حجمها $n \times t$ فاذن

$$A * B = [c_{ik}] \text{ حجمها } m \times t \text{ ويتم إعطاء العنصر } c_{ik} \text{ ريق الصيغة:}$$

$$c_{ik} = a_{i1}b_{1k} + a_{i2}b_{2k} + \dots + a_{in}b_{nk}$$

قم بتصميم وتطبيق الفئة matrixType التي يمكنها تخزين مصفوفة حجمها 100×100 وقم بانثال المعاملات + و - و * لأداء عمليات الجمع والطرح والضرب على التوالي وقم بانثال المعاملات << لاختار مصفوفة وقم أيضاً بكتابة برنامج اختبار لاختبار عمليات متنوعة على المصفوفات.

الفصل

4

مكتبة القالب المعياري

في هذا الفصل سوف:

- تعرف مكتبة القالب المعياري.
- تصبح على علم بالمكونات الرئيسية لمكتبة القالب المعياري.
- *****
- تستكشف كيفية استخدام *****
- *****

قام الفصل 2 بتقديم وشرح القوالب. بمساعدة قوالب الفئة قمنا بعمل (واستخدام) كود عام لمعالجة القوائم. على سبيل المثال قمنا في الفصل رقم 3 باستخدام الفئة `arrayListType` لمعالجة قائمة من الأعداد الصحيحة وقائمة من المقاطع. سوف نقوم في الفصول 5 و 7 و 8 بدراسة أكثر بنيات البيانات أهمية وهي: القوائم المتصلة، والرصات، والصفوف. سوف نقوم في الفصل 5 باستخدام قوالب الفئة بعمل كود عام لمعالجة القوائم المتصلة. بالإضافة الى هذا سوف نقوم باستخدام المبدأ الثاني من البرمجة القائمة على الهدف بعمل كود عام لمعالجة القوائم المرتبة. بعد هذا سوف نقوم في الفصول 7 و 8 باستخدام قوالب الفئة في عمل كود عام لتطبيق الرصات والصفوف. سوف ترى طوال الطريق أن القالب عبارة عن أداة قوية تدعم إعادة استخدام الكود.

C++ القياسية مزودة بمكتبة قالب معياري. من ضمن أشياء أخرى تقدم مكتبة القالب المعياري قوالب فئة لمعالجة القوائم (المتجاورة أو المتصلة) والرصات والصفوف. هذا الفصل يناقش بعض من الخصائص الهامة لمكتبة القالب المعياري ويوضح كيفية استخدام أدوات معينة تقدمها مكتبة القالب المعياري في برنامج ما. الفصل رقم 13 يصف بعض من خصائص مكتبة القالب المعياري التي لم يتم توضيحها في هذا الفصل؟

في الفصول اللاحقة سوف نتعلم كيفية عمل كود خاص بك لتطبيق البيانات والسيطرة عليها وكذلك كيفية استخدام الكود المكتوب بشكل محترف.

مكونات مكتبة القالب المعياري:

الهدف الأساسي لأي برنامج هو احتكار البيانات وتقديم النتائج وتحقيق هذا الهدف يتطلب القدرة على تخزين البيانات داخل ذاكرة الحاسب وتناول جزء محدد من البيانات وكتابة حلول حسابية لاحتكار البيانات.

على سبيل المثال اذا كانت جميع البيانات من نفس النوع وكان لدينا فكرة عن عدد عناصر البيانات اذن يمكننا استخدام مصفوفة لتخزين البيانات. يمكننا اذن استخدام مؤشر لتناول مكون محدد من المصفوفة. باستخدام الحلقة ومؤشر المصفوفة يمكننا السير عبر عناصر المصفوفة. يتم استخدام الحلول الحسابية مثل تلك الخاصة بتهيئة المصفوفة والتخزين والبحث لاحتكار البيانات المخزنة في المصفوفة. على الجانب الآخر اذا لم نكن نريد الاهتمام بحجم البيانات يمكننا استخدام قائمة متصلة كما سوف يتم توضيحها في الفصل 5 لمعالجة البيانات. اذا كانت البيانات التي تحتاج الى معالجة موجودة بأسلوب الداخل آخراً خارج أولاً يمكننا استخدام رصة (الفصل 7). بالمثل لذل كانت البيانات التي تحتاج الى معالجة موجوداً بأسلوب الداخل أولاً يخرج أولاً يمكننا استخدام الصف (الفصل 8). مكتبة القالب المعياري مزودة بتلك الخصائص لاحتكار البيانات بشكل فعال. بشكل رسمي أكثر تحتوي مكتبة القالب المعياري على ثلاث مكونات أساسية:

- حاويات.
- مكررات.
- خوارزميات.

الحاويات والمكررات عبارة عن قوالب للفئة. يتم استخدام المكررات للانتقال عبر عناصر الحاوية ويتم استخدام الخوارزميات للتلاعب في البيانات. هذا الفصل يناقش بعض من الحاويات والمكررات وتتم مناقشة الخوارزميات في الفصل رقم 13.

أنواع الحاويات:

يتم استخدام الحاويات لإدارة أهداف ذات نوع محدد ويتم تصنيف حاويات مكتبة القالب المعياري الى ثلاث فئات:

- حجويات متعاقبة (تسمى أيضاً حاويات متتالية).
- حاويات مترابطة.
- محولات الحاوية.

فضلاً عن هذا يتم تطبيق الحاويات باستخدام قوالب الفئة. الأقسام التالية تصف بعض من الحاويات المتعاقبة ويتم وصف الحاويات المترابطة في الفصل 13 ووصف محولات الحاوية في الفصول 7 و8.

الحاويات المتعاقبة:

كل هدف في الحاوية المتعاقبة له موضع محدد. الحاويات المتعاقبة الثلاث التي تم تعريفها مسبقاً هي:

- vector
- deque
- list

قبل مناقشة أنواع الحاويات المتعاقبة بوجه عام لنقم أولاً بوصف الحاوية المتعاقبة vector باختصار. اننا نفعل هذا لأن الحاويات vector من الناحية المنطقية مثلها مثل المصفوفات ولهذا يمكن معالجتها مثل المصفوفات. بمساعدة الحاويات vector يمكننا وصف العديد من الخصائص المشتركة بين جميع الحاويات. في الحقيقة تستخدم جميع الحاويات الأسماء ذاتها بالنسبة الى العمليات المشتركة وبالطبع هناك عمليات خاصة بالحاويات تتم مناقشتها عند وصف حاوية معينة. يقوم هذا الفصل بمناقشة الحاويات vector و deque. الفصل 5 يناقش الحاويات list.

الحاوية المتعاقبة : vector

تقوم الحاوية vector بتخزين وإدارة أهدافها في مصفوفة حركية. بما أن المصفوفة عبارة عن بنية بيانات تتناول عشوائية اذن يمكن تناول عناصر الحاوية vector بشكل عشوائي. ان ادخال عنصر ما في منتصف مصفوفة أو في بداية مصفوفة يستهلك الوقت خاصةً اذا كانت المصفوفة كبيرة. بالرغم من هذا يكون ادخال العنصر في النهاية سريع للغاية.

اسم الفئة التي تقوم بتطبيق الحاوية vector هي الفئة vector. (تذكر أن الحاويات عبارة عن قوالب فئة). اسم الملف الرئيسي المحتوي على الفئة vector هو الملف vector. لهذا عند استخدام الحاوية vector في برنامج ما يجب أن يتضمن البرنامج البيان التالي:

```
#include <vector>
```

بالإضافة الى هذا لتعريف هدف من النوع vector يجب أن نقوم بتحديد نوع الهدف لأن الفئة vector عبارة عن قالب فئة. على سبيل المثال يقوم البيان:

```
vector<int> intList;
```

بإعلان أن intList حاوية vector وأن نوع المكون هو int. وبالمثل يقوم البيان:

```
vector<string > stringList;
```

بإعلان أن stringList حاوية vector وأن نوع المكون هو string.

اعلان الأهداف vector:

تحتوي الفئة vector على عدة مقومات بما فيها المقوم الافتراضي. لهذا يمكن اعلان الحاوية vector وتهيئتها بعدة طرق. الجدول 1-4 يصف كيفية اعلان وتهيئة حاوية vector ذات نوع محدد.

جدول 1-4: طرق متعددة لاعلان وتهيئة حاوية vector.

| البيان | التأثير |
|--|--|
| <code>vector<elementType> vecList;</code> | يصنع حاوية فارغة vecList دون أي عناصر (يتم استدعاء المقوم الافتراضي). |
| <code>vector<elementType> vecList (otherVecList);</code> | يصنع حاوية vecList ويهيئ vecList مع عناصر الحاوية otherVecList vector ويكون كل من vecList و otherVecList من نفس النوع. |
| <code>vector<elementType> vecList (size);</code> | يصنع حاوية vecList حجمها size ويتم تهيئة vecList باستخدام المقوم الافتراضي. |
| <code>vector<elementType> vecList (n, elem);</code> | يصنع حاوية vecList حجمها n ويتم تهيئتها باستخدام نسخات n من العنصر elem. |
| <code>vector<elementType> vecList (begin, end);</code> | يصنع حاوية vecList ويتم تهيئتها عند العناصر الموجودة في النطاق (بداية، نهاية) أي جميع العناصر في النطاق begin...end-1. |

مثال 1-4:

أ. البيان التالي يعلن أن intList حاوية vector فارغة ونوع العنصر int:

```
vector<int> intList;
```

ب. البيان التالي يعلن أن intList حاوية vector حجمها 10 ونوع العنصر int. تتم تهيئة عناصر intList عند صفر:

```
vector<int> intList
```

ت. البيان التالي يعلن أن intList حاوية vector حجمها 5 ونوع العنصر int. تتم تهيئة الحاوية intList باستخدام عناصر المصفوفة:

```
int intArray[5] = {2, 4, 6, 8, 10};
vector<int> intList(intArray, intArray + 5);
```

تتم تهيئة الحاوية intList باستخدام عناصر المصفوفة intArray. وهي:

```
intList = {2, 4, 6, 8, 10}
```

الآن بعد معرفتنا كيفية اعلان حاوية vector لنقم بمناقشة كيفية الاتلاعب في البيانات المخزنة في حاوية vector. للقيام بهذا يجب أن نعلم العمليات الأساسية التالية:

▪ ادخال عنصر.

▪ حذف عنصر.

▪ الانتقال عبر عناصر حاوية vector.

يمكن تناول العناصر الموجودة في حاوية vector بشكل مباشر باستخدام العمليات المعطاة في جدول 2-4.

جدول 2-4: عمليات تناول عناصر حاوية vector:

| التعبير | التأثير |
|----------------------------------|--|
| <code>vecList.at (index),</code> | يقدم العنصر بالمكان المحدد بواسطة <code>index</code> |
| <code>vecList [index];</code> | يقدم العنصر بالمكان المحدد بواسطة <code>index</code> . |
| <code>vecList.front ();</code> | يقدم العنصر الأول (لا يتحقق مما اذا كانت الحاوية فارغة) |
| <code>vecList.back ();</code> | يقدم العنصر الأخير (لا يتحقق مما اذا كانت الحاوية فارغة) |

من جدول 2-4 ينتج أن العناصر في الحاوية vector يمكن معالجتها كما يمكن في المصفوفة. (تذكر أن مؤشر المصفوفة يبدأ عند الموقع صفر وبالمثال يكون العنصر الاول في الحاويات vector عند الموقع صفر).

مثال 2-4:

انظر الى البيان التالي الذي يعلن أن `intList` حاوية vector حجمها 5 ونوع العنصر `int`:

```
vector<int> intList (5);
```

يمكنك استخدام حلقة ما مثل التالية لتخزين العناصر داخل `intList`:

```
for(int j = 0; j < 5; j++)  
    intList[j] = j;
```

بالمثل يمكنك استخدام الحلقة `for` لاجراج عناصر `intList`.

الفئة `vector` تقدم عمليات متعددة لمعالجة عناصر الحاوية `vector`. افترض أن `vecList` حاوية من النوع `vector`. يمكن تنفيذ ادخال وحذف عنصر داخل `vecList` باستخدام العمليات المعطاة في الجدول 3-4. يتم تنفيذ هذه العمليات كدالات عنصر للفئة `vector` ويتم توضيحها بالخط العريض. كما يوضح الجدول 3-4 كيفية استخدام هذه العمليات.

جدول 3-4: عمليات متعددة على الحاوية vector:

| البيان | التأثير |
|--|---|
| <code>vecList.clear ()</code> | يحذف جميع العناصر من الحاوية |
| <code>vecList.erase (position)</code> | يحذف العنصر الموجود بالموضع المحدد بواسطة <code>position</code> |
| <code>vecList.erase (beg, end)</code> | يحذف جميع العناصر في النطاق بداية من <code>beg</code> وحتى <code>end-1</code> |
| <code>vecList.insert (position, elem)</code> | يتم ادخال نسخة من <code>elem</code> في الموضع المحدد بواسطة <code>position</code> ويتم ارجاع موضع العنصر الجديد. |
| <code>vecList.insert (position, n, elem)</code> | يتم ادخال عدد <code>n</code> من نسخ <code>elem</code> في الموضع المحدد بواسطة <code>position</code> |
| <code>vecList.insert (position, beg, end)</code> | يتم ادخال نسخة من العناصر في النطاق من البداية عند <code>beg</code> وحتى <code>end-1</code> داخل <code>vecList</code> في الموضع المحدد بواسطة <code>position</code> |
| <code>vecList.push_back (elem)</code> | يتم ادخال نسخة من <code>elem</code> داخل <code>vecList</code> في النهاية |
| <code>vecList.pop_back ()</code> | يحذف العنصر الأخير |
| <code>vecList.resize (num)</code> | يغير عدد العناصر الى <code>num</code> . اذا زاد الحجم () يصنع المقوم الافتراضي العناصر الجديدة. |
| <code>vecList.resize (num, elem)</code> | يغير عدد العناصر الى <code>num</code> . اذا زاد الحجم () يصنع المقوم الافتراضي العناصر الجديدة. |

في جدول 3-4 يسمى المعطى `position` في مكتبة القالب المعياري مكرر. المكرر يعمل مثل المؤشر ويتم استخدام المكررات بوجه عام للانتقال عبر عناصر الحاوية. بمعنى آخر وبمساعدة المكرر يمكننا السير عبر عناصر الحاوية ومعالجتها مرة واحدة. القسم التالي يصف كيفية اعلان المكرر داخل حاوية `vector` وكيفية التلاعب في البيانات المخزنة في الحاوية. بما أن المكررات جزء مكمل لمكتبة القالب المعياري فانه تتم مناقشتها بمزيد من التفصيل في القسم "المكررات" الموجود في جزء لاحق من هذا الفصل.

الدالة `push_back` مفيدة حيث يتم استخدامها لاضافة عنصر داخل الحاوية في نهايتها. قام المثال 4-2 باعلان الحاوية `intList` التي حجمها 5. قد تعتقد أنه يمكنك اضافة 5 عناصر فقط داخل الحاوية `intList` وبالرغم من هذا ليس هذا هو الحال. اذا احتجت الى اضافة أكثر من 5 عناصر يمكنك

استخدام الدالة `push_back`. لا يمكنك استخدام معامل تسجيل الدالة كما في المثال 2-4 لاضافة عناصر بعد الموضع رقم 4 الا اذا قمت بزيادة حجم الحاوية. اذا لم تكن تعرف عدد العناصر التي تحتاج الى تخزينها داخل الحاوية `vector` فانك لا تحتاج الى تحديد حجم الحاوية `vector` عند اعلانها. انظر مثال 3-4.

في هذه الحالة يمكنك استخدام الدالة `push_back` كما هو موضح في الأمثلة 3-4 و 4-5 لاضافة عناصر داخل الحاوية `vector`.

مثال 3-4:

البيان التالي يعلن أن `intList` حاوية `vector` حجمها صفر:

```
Vector<int> intList;
```

لاضافة عناصر الى `intList` يمكنك استخدام الدالة `push_back` كما يلي:

```
intList.push_back (34);
```

```
intList.push_back (55);
```

يعد تنفيذ هذه البيانات يكون حجم `intList` هو 2 و `{34, 55}` `intList =`. بالطبع استطعت استخدام الدالة `resize` لزيادة حجم `intList` أولاً ثم استخدمت معامل تسجيل المصفوفة. بالرغم من هذا تكون الدالة `push_back` أكثر ملاءمة لأنها لا تحتاج الى معرفة حجم الحاوية ولكنها ببساطة تضيف العناصر في نهايتها.

اعلان مكرر داخل حاوية `vector`:

بالرغم من أنه يمكننا معالجة حاوية `vector` كأى مصفوفة باستخدام معامل تسجيل المصفوفة الا أنه توجد مواقف سوف نعالج فيها عناصر الحاوية `vector` باستخدام المكرر. (تذكر أن المكرر مماثل للمؤشر). على سبيل المثال افترض أننا نريد ادخال عنصر في موقع محدد بالحاوية `vector`. بما أن العنصر يجب ادخاله في موقع محدد فهذا يتطلب نقل عناصر الحاوية (الا اذا تمت اضافة العنصر في النهاية). بالطبع يجب أن نفكر كذلك في حجم الحاوية. لجعل ادخال العنصر مناسب تقدم الفئة `vector` الدالة `insert` لادخال العناصر في موقع محدد في حاوية `vector`. بالرغم من هذا يجب عند استخدام الدالة `insert` تحديد الموقع الذي يتم ادخال العنصر فيه بواسطة المكرر. بالمثل تحتاج الدالة `erase` التي تحذف العنصر الى استخدام المكرر. هذا القسم يصف كيفية اعلان واستخدام المكررات داخل حاوية `vector`.

تحتوي الفئة vector على معامل typedef يتم اعلانه كعنصر عام. يتم اعلان المكرر للحاوية vector باستخدام المكرر typedef. على سبيل المثال يقوم البيان:

```
vector<int>::iterator intVecIter;
```

باعلان أن intVecIter مكرر داخل الحاوية vector من النوع int.

بما أن المكرر typedef معرف داخل الفئة vector يجب علينا لاستخدام اسم الحاوية (vector)، ونوع عنصر الحاوية، ومعامل حل المجال لاستخدام المكرر typedef.

افترض أن المكرر intVecIter يشير الى عنصر في الحاوية Vector ذات العناصر من النوع int. التعبير:

```
++intVecIter
```

يقدم المكرر intVecIter الى العنصر التالي في الحاوية والتعبير:

```
*intVecIter
```

يعيد العنصر في موقع المكرر الحالي.

لاحظ أن تلك العمليات هي نفسها المؤداة على المؤشرات التي تمت مناقشتها في الفصل 3. تذكر أن عند استخدام * كمعامل أحادي تسمى معامل عدم الاشارة.

نقوم الآن بمناقشة كيفية استخدام المكرر داخل حاوية vector للتلاعب في البيانات المخزنة داخل الحاوية vector. افترض أن لدينا البيانات التالية:

```
vector<int> intList;           //Line 1
vector<int>::iterator intVecIter; //Line 2
```

البيان في الصف 1 يعلن أن intList حاوية vector ونوع العنصر هو int والبيان في الصف 2 يعلن أن intVecIter مكرر داخل حاوية vector عناصرها من النوع int.

الحاويات والدالات begin و end (البداية والنهاية):

كل حاوية لها دالات العنصر begin و end. الدالة begin تنتج موضع العنصر الأول في الحاوية والدالة end تنتج موضع العنصر الأخير في الحاوية. هذه الدالات ليس لها معاملات. بعد تنفيذ البيان التالي:

```
intVecIter = intList.begin ( );
```

يشير المكرر intVecIter الى العنصر الأول في الحاوية هف/هسف>

الحلقة for التالية تستخدم مكرر لايخراج عناصر intList على جهاز الاخراج القياسي:

```
for(intVecIter = intList.begin(); intVecIter != intList.end();
    ++intVecIter)
    cout<<*intVecIter<<" ";
```

مثال 4-4:

انظر الى البيانات التالية:

```
int intArray[7] = {1, 3, 5, 7, 9, 11, 13};    //Line 1
vector<int> vecList(intArray, intArray + 7);  //Line 2
vector<int>::iterator intVecIter;             //Line 3
```

البيان في الصف 2 يعلن ويهيئ الحاوية vecList الآن انظر الى البيانات التالية:

```
intVecIter = vecList.begin();                //Line 4
++intVecIter;                                //Line 5
vecList.insert(intVecIter, 22);               //Line 6
```

البيان في الصف 4 يقوم بتهيئة المكرر intVecIter عند العنصر الأول من vecList والبيان في الصف 5 يقدم intVecIter الى العنصر الثاني من vecList والبيان في الصف 6 يدخل 22 في الموقع المحدد بواسطة intVecIter.

بعد تنفيذ البيان في الصف 6 تكون:

vecList = {1, 22, 3, 5, 7, 9, 11, 13}

لاحظ أن حجم الحاوية يزداد كذلك.

تحتوي الفئة vector أيضاً على دالات عنصر يمكن استخدامها لايجاد عدد العناصر الموجودة حالياً في الحاوية والعدد العناصر الأقصى الذي يمكن ادخاله في الحاوية وهكذا. يقوم الجدول 4-4 بوصف بعض من تلك العمليات. (افترض أن vecCont حاوية vector).

جدول 4-4: دالات لتحديد حجم الحاوية vector:

| التعبير | التأثير |
|---------------------|--|
| vecCont.capacity () | ينتج عدد العناصر الأقصى الذي يمكن ادخاله داخل الحاوية vecCont بدون ***** |
| vecCont.empty () | ينتج true اذا كانت الحاوية vecCont فارغة وبخلاف هذا تنتج false. |
| vecCont.size () | ينتج عدد العناصر الموجودة حالياً في الحاوية vecCont. |
| vecCont.maxsize () | ينتج عدد العناصر الأقصى الذي يمكن ادخاله داخل الحاوية vecCont. |

المثال 4-5 يوضح كيفية استخدام حاوية vector في برنامج وكيفية معالجة العناصر في حاوية

vector.

مثال 5-4:

#include <iostream>

#include <vector>

باستخدام مساحة الاسم std:

```
int main()
{
    vector<int> intList;           //Line 1
    int i;                         //Line 2

    intList.push_back(13);         //Line 3
    intList.push_back(75);         //Line 4
    intList.push_back(28);         //Line 5
    intList.push_back(35);         //Line 6

    cout<<"Line 7: List Elements: "; //Line 7
    for(i = 0; i < 4; i++)         //Line 8
        cout<<intList[i]<<" ";    //Line 9
    cout<<endl;                   //Line 10

    for(i = 0; i < 4; i++)         //Line 11
        intList[i] *= 2;          //Line 12

    cout<<"Line 13: List Elements: "; //Line 13
    for(i = 0; i < 4; i++)         //Line 14
        cout<<intList[i]<<" ";    //Line 15
    cout<<endl;                   //Line 16

    vector<int>::iterator listIt; //Line 17

    cout<<"Line 18: List Elements: "; //Line 18
    for(listIt = intList.begin(); listIt != intList.end(); //Line 19
        ++listIt)
        cout<<*listIt<<" ";      //Line 20
    cout<<endl;                   //Line 21
    listIt = intList.begin();      //Line 22
    ++listIt;                      //Line 23
    ++listIt;                      //Line 24
    intList.insert(listIt,88); //Insert 88 at the
                                //position specified
                                //by listIt. //Line 25

    cout<<"Line 26: List Elements: "; //Line 26
    for(listIt = intList.begin(); listIt != intList.end(); //Line 27
        ++listIt)
        cout<<*listIt<<" ";      //Line 28
    cout<<endl;                   //Line 29

    return 0;
}
```

المخرجات:

الصف 7: عناصر القائمة: 13 75 28 35

الصف 13: عناصر القائمة: 26 150 56 70

الصف 18: عناصر القائمة: 26 150 56 70

الصف 26: عناصر القائمة: 26، 56، 70، 88، 150

البيان في الصف 1 يعلن حاوية vector (أو لـ lajazar Vector) مسماة intList ونوعها int. البيان في الصف 2 يعلن أن i متغير int والبيانات في الصفوف من 3 وحتى 6 تستخدم الدالة push_back لادخال أربعة أعداد – 13، 75، 28، 35 – داخل intList. البيانات في الصفين 8 و9 تستخدم الحلقة for ومعامل تسجيل المصفوفة [] لـ اخراج عناصر intList. في المخرجات انظر الى الخط الذي يحدد الصف رقم 7 الذي يحتوي على مخرجات الصفوف من 7 الى 10 من البرنامج. البيانات في الصفوف 11 و12 تستخدم الحلقة for لمضاعفة قيمة كل عنصر من intList والبيانات في الصفين 14 و15 تخرج عناصر intList. في المخرجات انظر الى الخط الذي يحدد الصف 13 الذي يحتوي على مخرجات الصفوف من 13 وحتى 16 من البرنامج.

البيان في الصف 17 يعلن أن listIt مكرر vector يقوم بمعالجة أي حاوية vector تكون عناصرها من النوع int. باستخدام المكرر listIt تقوم البيانات في الصفوف 19 و20 باخراج عناصر intList. بعد تنفيذ البيان في الصف 22 يشير listIt الى العنصر الأول من intList. البيانات في الصفوف 23 و24 تقوم بتقديم listIt مرتين وبعد تنفيذ هذه البيانات تشير listIt الى العنصر الثالث من intList. البيان في الصف 25 يدخل 88 في intList في الموقع المحدد بواسطة المكرر listIt. بما أن listIt يشير الى المكون الموجود بالموقع 2 (العنصر الثالث من intList) فانه يتم ادخال 88 في الموقع 2 في intList أي تصبح 88 العنصر الثالث من intList. البيانات في الصفوف 27 و28 تخرج intList المعدلة.

دالات عناصر مشتركة بين جميع الحاويات:

القسم التالي ناقش الحاويات vector. ننظر الآن الى العمليات المشتركة بين جميع الحاويات. على سبيل المثال يكون لكل فئة حاوية مقوم افتراضي والعديد من المقومات ذات المعاملات والمدمر ودالة لادخال عنصر داخل الحاوية وهكذا.

تذكر أن الفئة تقوم بتغليف البيانات والعمليات المؤداة على تلك البيانات داخل وحدة واحدة. بما أن كل حاوية عبارة عن فئة فانه يتم تعريف العديد من العمليات بشكل مباشر للحاوية ويتم تقديمها كجزء من تعريف الفئة. تذكر أيضاً أن العمليات التي تسيطر على البيانات يتم تنفيذها بمساعدة الدالات ويطلق عليها دالات عنصر للفئة. الجدول 4-5 يصف بعض من دالات العنصر المشتركة بين جميع الحاويات أي تلك الدالات التي يتم تضمينها كعناصر لـ قالب الفئة التي تطبق الحاوية.

افترض أن ct و ct1 و ct2 حاويات من نفس النوع. الجدول 4-5 يوضح اسم الدالة بالخط العريض ويوضح كيفية استدعاء الدالة.

جدول 4-5: بعض دالات العنصر المشتركة بين جميع الحاويات:

| دالة العنصر | التأثير |
|-----------------|---|
| مقوم افتراضي | يقوم بتهيئة الهدف عند وضع فارغ |
| مقوم ذو معاملات | بالإضافة الى المقوم الافتراضي يكون لكل حاوية مقومات ذات معاملات وسوف نصف تلك المقومات عند مناقشة حاوية محددة. |
| مقوم النسخ | يتم تنفيذه عندما يتم تمرير الهدف كمعامل بالقيمة عند يتم اعلان وتهيئة الهدف باستخدام هدف آخر من نفس النوع وعندما تنتج الدالة قيمتها كعنصر. |
| مدمر | يتم تنفيذه عندما يخرج الهدف من المجال |
| ct.empty () | تنتج true اذا كانت الحاوية ct فارغة وبخلاف هذا تنتج false |
| ct.size () | تنتج عدد العناصر كـ int غير محدد موجود حالياً في الحاوية ct |
| ct.max_size () | تنتج العدد الأقصى من العناصر التي يمكن ادخالها في الحاوية ct. |

جدول 4-5: بعض دالات عنصر مشتركة بين جميع الحاويات (متواصلة)

| دالة العنصر | التأثير |
|----------------------------|--|
| c1.swap (ct2) | يستبدل عناصر الحاويات ct1 و ct2 |
| ct.begin () | ينتج مكرراً للعنصر الأول في الحاوية ct. |
| ct.end () | ينتج مكرراً للعنصر الأخير في الحاوية ct. |
| ct.rbegin () | يعكس البداية وينتج مؤشر للعنصر الأخير في الحاوية ct ويتم استخدام هذه الدالة لمعالجة عناصر ct بالعكس. |
| ct.rend () | يعكس النهاية وينتج مؤشر للعنصر الأول في الحاوية ct |
| ct.insert (position, elem) | يدخل elem في الحاوية ct بالموقع المحدد بواسطة المعطى position ولاحظ أن position هنا عبارة عن مكرراً |
| ct.erase (begin, end) | يحذف كل العناصر الموجودة بين البداية والنهاية-1 من الحاوية ct. |
| ct.clear () | يمسح كل العناصر من الحاوية. بعد استدعاء هذه الدالة تكون الحاوية ct فارغة. |
| دالات المعامل | |
| ct1 = ct2; | ينسخ عناصر ct2 في ct1. بعد هذه العملية تكون العوامل متماثلة في كلا الحاويين. |
| ct1 == ct2 | ينتج true اذا كانت الحاويات ct1 و ct2 متساوية وبخلاف هذا تنتج false. |
| ct1 != ct2 | ينتج true اذا كانت الحاويات ct1 و ct2 غير متساوية وبخلاف هذا تنتج false. |

تلك العمليات مشتركة بين جميع الحاويات ولهذا عند مناقشة حاوية محددة لحفظ المساحة لن يتم ذكر تلك العمليات مرة أخرى.

دالات عنصر مشتركة بين الحاويات المتعاقبة:

الجزء السابق وصف دالات العنصر المشتركة بين جميع الحاويات بالاضافة الى دالات العنصر هذه يقوم الجدول 4-6 بوصف دالات العنصر المشتركة بين جميع الحاويات المتعاقبة – أي الحاويات من النوع vector و deque و list. (افترض أن seqCont حاوية متعاقبة).

جدول 4-6: دالات عنصر مشتركة بين جميع الحاويات المتعاقبة.

| التعبير | التأثير |
|-------------------------------------|--|
| seqCont.insert (position, elem) | نسخة من elem يتم ادخالها في seqCont بالموقع المحدد بواسطة position. ويتم تقديم موقع العنصر الجديد. |
| seqCont.insert (position, n, elem) | عدد n نسخ من elem يتم ادخالها في seqCont بالموقع المحدد بواسطة position. |
| seqCont.insert (position, beg, end) | يتم ادخال نسخة من العناصر بداية من البداية وحتى النهاية-1 داخل seqCont بالموقع المحدد بواسطة position. |
| seqCont.push_back (elem) | يتم ادخال نسخة من elem داخل seqCont في النهاية. |
| seqCont.pop_back () | يحذف العنصر الأخير. |
| seqCont.erase (position) | يحذف العنصر الموجود بالموقع المحدد بواسطة position. |
| seqCont.erase (beg, end) | يحذف جميع العناصر بداية من البداية وحتى النهاية-1. |
| seqCont.clear () | يحذف جميع العناصر من الحاوية. |
| seqCont.resize (num) | يغير عدد العناصر الى num. اذا زاد الحجم () فانه يتم عمل العناصر الجديدة بواسطة مقومها الافتراضي. |
| seqCont.resixe (num, elem) | يغير عدد العناصر الى num. اذا زاد الحجم () فان العناصر الجديدة تكون نسخ من elem. |

خوارزميات النسخ:

قام المثال 4-5 باستخدام الحلقة for لاجراج عناصر الحاوية vector. تقدم مكتبة القالب المعياري طريقة مناسبة لاجراج عناصر الحاوية بمساعدة الدالة copy. يتم توفير الدالة copy كجزء من الحل الحسابي العام ويمكن استخدامها مع اي نوع من الحاويات وكذلك المصفوفات. بما أننا نحتاج كثيراً الى اخراج عناصر الحاوية اذن لنقم بوصف هذه الدالة قبل مناقشة الحاويات. مكتبة القالب المعياري تحتوي على دالات عديدة مثل دالة النسخ copy كجزء من الخوارزميات العامة التي يتم شرحها في الفصل 13.

تقوم الدالة copy بما هو أكثر من اخراج عناصر الحاوية فهي بوجه عام تسمح لنا بنسخ العناصر من مكان الى آخر. على سبيل المثال لاجراج عناصر الحاوية vector أو لنسخ عناصر حاوية vector داخل vector أخرى يمكننا استخدام الدالة copy. نموذج قالب الدالة copy هو:

```
template<class inputIterator, class outputIterator>
outputItr copy(inputIterator first1, inputIterator last,
               outputIterator first2);
```

المعامل first1 يحدد الموقع الذي يبدأ منه نسخ العناصر والمعامل last يحدد الموقع النهائي. المعامل first2 يحدد المكان الذي يتم نسخ العناصر فيه. لهذا تقوم المعاملات first1 و last بتحديد المصدر والمعامل first2 يحدد المقصد. لاحظ أن العناصر الواقعة ضمن النطاق first1.....last-1 يتم نسخها.

يتم تضمين تعريف قالب الدالة copy في الملف الرئيسي algorithm. لهذا لاستخدام الدالة copy يجب أن يتضمن البرنامج البيان التالي:

```
#include <algorithm>
```

تعمل الدالة copy كما يلي. انظر الى البيانات التالية:

```
int intArray[] = {5, 6, 8, 3, 40, 36, 98, 29, 75}; //Line 1
vector<int> vecList(9); //Line 2
```

البيان في الصف 1 يصنع المصفوفة intArray من تسع مكونات:

```
intArray = {5, 6, 8, 3, 40, 36, 98, 29, 75}
intArray[0] = 5, intArray[1] = 6 هنا
```

البيان في الصف 2 يصنع حاوية vector فارغة من تسع مكونات ونوع العنصر هو int. تذكر أن اسم المصفوفة وهو intArray يكون في الحقيقة مؤشر ويحتوي على العنوان الرئيسي للمصفوفة. لهذا تشير intArray الى المكون الأول من المصفوفة ويشير intArray + 1 الى المكون الثاني من المصفوفة وهكذا.

انظر الآن الى البيان التالي:

```
copy(intArray, intArray + 9, vecList.begin()); //Line 3
```

يقوم هذا البيان بنسخ العناصر بداية من الموقع intArray وهو المكون الأول من المصفوفة intArray حتى intArray + 9 - 1 (أي intArray + 8) وهو العنصر الأخير من المصفوفة intArray داخل الحاوية vecList. (لاحظ أن first1 هنا هي intArray، و last هي intArray + 9، و first2 هي (vecList.begin)). بعد تنفيذ البيان في الصف رقم 3:

```
vecList = {5, 6, 8, 3, 40, 36, 98, 29, 75} //Line 4
```

بعد هذا انظر الى البيان:

```
copy(intArray + 1, intArray + 9, intArray); //Line 5
```

هنا first1 هي intArray + 1 أي أن first1 تشير الى موقع العنصر الثاني من المصفوفة intArray و last هي intArray + 9. وكذلك تكون first1 عبارة عن intArray أي أن first2

تشير الى موقع العنصر الأول من المصفوفة intArray. لهذا يتم نسخ عنصر المصفوفة الثاني داخل مكون المصفوفة الأول وعنصر المصفوفة الثالث داخل مكون المصفوفة الثاني وهكذا. بعد تنفيذ البيان في الصف 5:

```
intArray[] = {6, 8, 3, 40, 36, 98, 29, 75, 75} //Line 6
```

لاحظ أن عناصر المصفوفة intArray تم نقلها الى اليسار موضع واحد.

افترض أن vecList كما هي معطاة في الصف 4. انظر الى البيان:

```
copy(vecList.rbegin() + 2, vecList.rend(),  
      vecList.rbegin()); //Line 7
```

تذكر أن الدالة rbegin (البداية العكسية) تنتج مؤشر للعنصر الأخير في الحاوية ويتم استخدامها لمعالجة عناصر الحاوية بالعكس. لهذا تقوم + 2 () vecList.rbegin باننتاج مؤشر الى العنصر من الثالث الى الأخير في الحاوية vecList. بالمثل تقوم الدالة rend (النهاية العكسية) باننتاج مؤشر للعنصر الأول في الحاوية. البيان السابق ينقل عناصر الحاوية vecList موضعين الى اليمين. بعد تنفيذ البيان في الصف 7 تكون الحاوية vecList:

```
vecList = {5, 6, 5, 6, 8, 3, 40, 36, 98}
```

مثال رقم 4-6 يوضح تأثير البيانات السابقة باستخدام برنامج C++. قبل مناقشة المثال 4-6 لنقم بوصف نوع خاص من المكررات يطلق عليها مكررات ostream التي تعمل بشكل جيد مع الدالة copy لنسخ عناصر الحاوية الى جهاز المخرجات.

المكرر ostream والدالة copy:

احدى الطرق لاجراج محتويات الحاوية هي استخدام الحلقة for والدالة begin لتهيئة متغير تحكم الحلقة for ولاستخدام الدالة set في تحديد الحد. يمكن استخدام الدالة copy بشكل آخر لاجراج عناصر الحاوية. في هذه الحالة يقوم مكرر من النوع ostream بتحديد المقصد (تتم مناقشة المكررات ostream بالتفصيل في لاحقاً في هذا الفصل). عندما نصنع مكرر من النوع ostream نقوم كذلك بتحديد نوع العنصر الذي يخرج المكرر.

البيان التالي يوضح كيفية عمل مكرر ostream من النوع int:

```
ostream_iterator<int> screen (cout, " "); // Line A
```

هذا البيان يصنع مكرر ostream مسمى screen مع نوع العنصر int. المكرر screen له معطيان: الهدف cout ومسافة. لهذا تتم تهيئة المكرر screen باستخدام الهدف cout وعندما يقوم هذا المكرر باخراج العناصر تكون منفصلة عن بعضها بواسطة مسافة.

البيان:

`copy (intArray, intArray + 9, screen):`

يخرج عناصر `intArray` على الشاشة.

بالمثل يقوم البيان:

`copy (vecList.begin (), vecList.end (), screen):`

بإخراج عناصر الحاوية `vecList` على الشاشة.

سوف نقوم كثيراً باستخدام الدالة `copy` لإخراج عناصر الحاوية عن طريق استخدام مكرر

`ostream`. حتى نقوم بمناقشة مكررات `ostream` بالتفصيل سوف نستخدم بيانات مماثلة للبيان

الموجود في الصف A لعمل مكرر `ostream`.

بالطبع يمكننا تحديد مكرر `ostream` بشمل مباشر في الدالة `copy`. على سبيل المثال يكون البيان

(الموضح من قبل):

`copy (vecList.begin (), vecList.end (), screen):`

مكافئ للبيان:

`copy (vecList.begin (), vecList.end ();`

`ostream_iterator<int> (cout, " "));`

أخيراً يقوم البيان:

`copy (vecList.begin (), vecList.end ();`

`ostream_iterator<int> (cout, " "));`

بإخراج عناصر `vecList` مع فاصلة ومسافة بينها.

المثال 4-6 يوضح كيفية استخدام الدالة `copy` والمكرر `ostream` في برنامج ما.

مثال 6-4:

```
#include <algorithm>
#include <vector>
#include <iterator>
#include <iostream>

using namespace std;

int main()
{
    int intArray[] = {5, 6, 8, 3, 40, 36, 98, 29, 75};    //Line 1

    vector<int> vecList(9);                               //Line 2

    ostream_iterator<int> screen(cout, " ");             //Line 3

    cout<<"Line 4: intArray: ";                           //Line 4
    copy(intArray, intArray + 9, screen);                //Line 5
    cout<<endl;                                           //Line 6

    copy(intArray, intArray + 9, vecList.begin());        //Line 7

    cout<<"Line 8: vecList: ";                             //Line 8
    copy(vecList.begin(), vecList.end(), screen);         //Line 9
    cout<<endl;                                           //Line 10

    copy(intArray + 1, intArray + 9, intArray);           //Line 11
    cout<<"Line 12: After shifting the elements one "
        <<"position to the left, intArray: "<<endl;       //Line 12
    copy(intArray, intArray + 9, screen);                 //Line 13
    cout<<endl;                                           //Line 14

    copy(vecList.rbegin() + 2, vecList.rend(),           //Line 15
          vecList.rbegin());
    cout<<"Line 16: After shifting the elements down "
        <<"by two positions, vecList:"<<endl;             //Line 16
    copy(vecList.begin(), vecList.end(), screen);         //Line 17
    cout<<endl;                                           //Line 18

    return 0;
}
```

المخرجات:

الصف 4: intArray: 75 29 98 36 40 3 8 6 5

الصف 8: vecList: 75 29 98 36 40 3 8 6 5

الصف 12: بعد نقل العناصر موضع واحد الى اليسار تكون intArray:

75 75 29 98 36 40 3 8 6

الصف 16: بعد نقل العناصر موضعين الى أسفل تكون vecList:

98 36 40 3 8 6 5 6 5

حاوية متعاقبة: deque:

هذا الفصل يصف الحاويات المتعاقبة deque. كلمة deque تعبر عن صف له نهايتان. يتم استخدام الحاويات deque كمصفوفات حركية بطريقة يمكن بها ادخال العناصر عند كلا النهايتين. لهذا يمكن أن تمتد deque في أي من الاتجاهين. يمكن كذلك ادخال العناصر في المنتصف. ان ادخال العناصر في البداية أو في النهاية يكون سريع بينما يكون ادخال العناصر في المنتصف مضيعة للوقت لأنه يكون هناك حاجة الى نقل العناصر في الصف.

اسم الفئة التي تقوم بتعريف الحاويات deque تعرف باسم الفئة deque. تعريف الفئة deque ودالات تطبيق العمليات المتنوعة على الهدف deque يتم تضمينها في الملف الرئيسي deque. لهذا عند استخدام حاوية deque في برنامج ما يجب أن يضم البرنامج البيان التالي:

```
#include <deque>
```

الفئة deque تتضمن العديد من المقومات ولهذا يمكن تهيئة هدف deque بعدة طرق عند اعلانها كما هو موضح في جدول 7-4.

جدول 7-4: طرق متنوعة لاعلان هدف deque

| البيان | التأثير |
|--|---|
| <code>deque<elementType> deq;</code> | يصنع حاوية Deque فارغة دون أية عناصر (يتم استدعاء المقوم الافتراضي deque). |
| <code>Deque<elementType> deq(otherDeq);</code> | يصنع حاوية deque اسمها deq ويهينها لعناصر otherDeq. تكون deq و otherDeq من نفس النوع. |

جدول 7-4: طرق متنوعة لاعلان هدف deque (مستمر):

| البيان | التأثير |
|---|--|
| <code>deque<elementType> deq (size);</code> | يصنع حاوية deque حجمها size وتتم تهيئة deq باستخدام المقوم الافتراضي. |
| <code>deque<elementType> deq (n, elem);</code> | يصنع حاوية deque اسمها deq حجمها n ويتم تهيئة deq باستخدام عدد n نسخ من العنصر elem. |
| <code>deque<elementType> deq (begin, end);</code> | يصنع حاوية deque اسمها deq وتتم تهيئتها عند العناصر في النطاق (البداية، النهاية) أي جميع العناصر في النطاق من البداية...النهاية 1. |

بالاضافة الى العمليات المشتركة بين جميع الحاويات (انظر جدول 6-4) يقوم جدول 8-4 بتوضيح العمليات التي يمكن استخدامها للتحكم في عناصر حاوية deque ويتم توضيح اسم الدالة التي تقوم بتطبيق العمليات بالخط العريض. كما توضح البيانات كيفية استخدام دالة معينة. افترض أن deq عبارة عن حاوية deque.

جدول 4-8: عمليات متنوعة يمكن إجراؤها على هدف deque:

| التعبير | التأثير |
|-----------------------|---|
| deq.assign (n, elem) | يحدد n نسخ من elem. |
| deq.assign (beg, end) | يحدد جميع العناصر في النطاق من البداية ... النهاية-1. |
| deq.push_front (elem) | يدخل elem في بداية deq. |
| deq.pop_front () | يزيل العنصر الأول من deq. |
| deq.at (index) | ينتج العنصر بالموقع المحدد بواسطة index. |
| deq [index] | ينتج العنصر بالموقع المحدد بواسطة index. |
| deq.front () | ينتج العنصر الأول (لا يتحقق مما إذا كانت الحاوية فارغة). |
| deq.back () | ينتج العنصر الأخير (لا يتحقق مما إذا كانت الحاوية فارغة). |

المثال 4-7 يوضح كيفية استخدام حاوية deque في برنامج ما.

مثال 4-7:

```
//Deque Example
#include <iostream>
#include <deque>
#include <algorithm>
#include <iterator>
```

```

using namespace std;

int main()
{
    deque<int> intDeq; //Line 1
    ostream_iterator<int> screen(cout, " "); //Line 2

    intDeq.push_back(13); //Line 3
    intDeq.push_back(75); //Line 4
    intDeq.push_back(28); //Line 5
    intDeq.push_back(35); //Line 6

    cout<<"Line 7: intDeq: "; //Line 7
    copy(intDeq.begin(), intDeq.end(), screen); //Line 8
    cout<<endl; //Line 9
    intDeq.push_front(0); //Line 10
    intDeq.push_back(100); //Line 11

    cout<<"Line 12: After adding two more "
        <<"elements, one at the front "<<endl
        <<"          and one at the back, intDeq: "; //Line 12
    copy(intDeq.begin(), intDeq.end(), screen); //Line 13
    cout<<endl; //Line 14

    intDeq.pop_front(); //Line 15
    intDeq.pop_front(); //Line 16

    cout<<"Line 17: After removing the first "
        <<"two elements, intDeq: "; //Line 17
    copy(intDeq.begin(), intDeq.end(), screen); //Line 18
    cout<<endl; //Line 19

    intDeq.pop_back(); //Line 20
    intDeq.pop_back(); //Line 21

    cout<<"Line 22: After removing the last "
        <<"two elements, intDeq = "; //Line 22
    copy(intDeq.begin(), intDeq.end(), screen); //Line 23
    cout<<endl; //Line 24

    deque<int>::iterator deqIt; //Line 25

    deqIt = intDeq.begin(); //Line 26
    ++deqIt; //deqIt points to the //Line 27
              //second element
    intDeq.insert(deqIt, 444); //insert 444 at the //Line 28
                              //location deqIt

```

```

cout<<"Line 29: After inserting 444, intDeq: "; //Line 29

copy(intDeq.begin(), intDeq.end(), screen); //Line 30
cout<<endl; //Line 31

intDeq.assign(2, 45); //Line 32

cout<<"Line 33: After assigning two "
    <<"copies of 45, intDeq: "; //Line 33
copy(intDeq.begin(), intDeq.end(), screen); //Line 34
cout<<endl; //Line 35

intDeq.push_front(-10); //Line 36
intDeq.push_back(-999); //Line 37

cout<<"Line 38: After inserting two "
    <<"elements, one at the front "<<endl
    <<"          and one at the back, intDeq: "; //Line 38
copy(intDeq.begin(), intDeq.end(), screen); //Line 39
cout<<endl; //Line 40

return 0;
}

```

المخرجات:

الصف 7: intDeq: 35 28 75 13

الصف 12: بعد اضافة عنصرين آخرين واحد في المقدمة وواحد في المؤخرة تكون intDeq: 100 35 28 75 13 0

الصف 17: بعد ازالة أول عنصرين تكون intDeq: 100 35 28 75

الصف 22: بعد ازالة آخر عنصرين تكون intDeq: 28 75

الصف 29: بعد ادخال 444 تكون intDeq: 28 444 75

الصف 33: بعد تحديد نسختين من 45 تصبح intDeq: 45 45

الصف 38: بعد ادخال عنصرين واحد في المقدمة وآخر في المؤخرة تكون intDeq: 999- 45 45 10-

البيان في الصف 1 يعلن أن الحاوية deque المسماة intDeq من النوع int أي أن جميع عناصر intDeq من النوع int. البيان في الصف 2 يعلن أن screen مكرر ostream تتم تهيئته عند جهاز المخرجات القياسي. البيانات في الصفوف من 3 وحتى 6 تستخدم عملية push_back لادخال أربعة أعداد – 13، 75، 28، 25 – داخل intDeq. البيان في الصف 8 يخرج عناصر intDeq. في المخرجات انظر الخط الذي يحدد الصف 7 الذي يتضمن مخرجات البيانات في الصفوف من 7 وحتى 9 من البرنامج.

البيان في الصف 10 يدخل صفر في بداية intDeq والبيان في الصف 11 يدخل 100 في نهاية intDeq. البيان في الصف 13 يخرج intDeq بعد تعديلها.

البيانات في الصفين 15 و 16 تستخدم العملية `pop_front` لازالة أول عنصرين من `intDeq` والبيان في الصف 18 يخرج `intDeq` بعد التعديل. البيانات في الصفوف 20 و 21 تستخدم العملية `pop_back` لازالة آخر عنصرين من `intDeq` والبيان في الصف 23 يخرج `intDeq` بعد تعديلها. البيان في الصف 25 يعلن أن `deqIt` مكرر `deque` يقوم بمعالجة جميع الحاويات `deque` التي تكون بياناتها من النوع `int`. بعد تنفيذ البيان في الصف 26 تشير `deqIt` الى العنصر الأول من `intDeq`. البيان في الصف 27 يقدم `deqIt` الى العنصر التالي من `intDeq`. البيان في الصف 28 يدخل 444 داخل `intDeq` في الموقع المحدد بواسطة `deqIt`. البيان في الصف 30 يقوم باخراج `intDeq`. البيان في الصف 32 يحدد نسختين من 45 عند `IntDeq`. بعد تنفيذ البيان في الصف 32 تتم ازالة العناصر القديمة من `intDeq` وتحتوي `intDeq` على نسختين فقط من 45. مخرجات البيان في الصف 34 توضح هذا النشاط. في المخرجات انظر الى الخط الذي يحدد الصف 33 والذي يحتوي على مخرجات البيانات في الصفوف من 33 وحتى 35 من البرنامج. معنى البيانات المتبقية واضح دون تفسير.

المكررات:

الأمثلة من 4-5 وحتى 4-7 توضح أن المكررات ضرورية لمعالجة عناصر الحاوية بشكل فعال. لنقم بمناقشة المكررات ببعض من التفصيل. تعمل المكررات مثل المؤشرات ويشير المكرر بشكل عام الى عناصر الحاوية (متعاقبة أم مترابطة). لهذا يمكننا بمساعدة المكررات تناول كل عنصر من عناصر الحاوية بنجاح. العمليتان الأكثر انتشاراً بين المكررات هي ++ (معامل الزيادة) و * (معامل عدم الاشارة). افترض أن `cntItr` مكرر داخل حاوية ما. البيان:

```
++cntItr;
```

ينتج قيمة عنصر الحاوية المشار اليه بواسطة `cntItr`.

أنواع المكررات:

يوجد خمسة أنواع من المكررات:

- مكررات الادخال.
- مكررات الاخراج.
- مكررات أمامية.
- مكررات ثنائية الاتجاه.
- مكررات دخول عشوائي.

الخمس أقسام التالية تصف كل من تلك المكررات.

مكررات الادخال:

مكررات الادخال بإمكانية القراءة تنتقل الى الأمام عنصر عنصر وتقوم بانتاج قيم عنصر عنصر. يتم تقديم تلك المكررات لقراءة البيانات من تدفق المدخلات. افترض أن inputIterator عبارة عن مكرر ادخال. الجدول 4-9 يصف بعض من العمليات المؤداة على inputIterator.

جدول 4-9: بعض العمليات المؤداة على مكرر ادخال:

| التعبير | التأثير |
|-----------------------|---|
| ***** | يعطي وصول الى العنصر ***** |
| inputIterator->member | يعطي وصول الى عضو العنصر. |
| ++inputIterator | يتحرك الى الأمام وينتج الموقع الجديد (قبل الزيادة) |
| inputIterator++ | يتحرك الى الأمام وينتج الموقع القديم (بعد الزيادة) |
| inputIt1 == inputIt2 | ينتج true اذا كان المكرران متساويان وبخلاف هذا ينتج false. |
| ***** | ينتج true اذا لم يكن المكرران متساويان وبخلاف هذا ينتج false. |

مكررات الاخراج:

مكررات الاخراج التي بها إمكانية الكتابة تنتقل الى الأمام عنصر عنصر ويتم توفير تلك المكررات لكتابة البيانات على تدفق المخرجات. كما يتم استخدامها كمدخلات. افترض أن outputIterator عبارة عن مكرر اخراج. الجدول 4-10 يصف بعض من العمليات المؤداة على outputIterator.

جدول 4-10: بعض العمليات المؤداة على مكررات الاخراج:

| التعبير | التأثير |
|------------------------------|--|
| *outputIterator value; ***** | يكتب القيمة بالموقع المحدد بواسطة outputIterator. |
| ++outputIterator | يتحرك الى الأمام وينتج الموقع الجديد (قبل الزيادة) |
| outputIterator++ | يتحرك الى الأمام وينتج الموقع القديم (بعد الزيادة) |

لا يمكن استخدام مكررات الاخراج للتكرار عبر نطاق ما مرتين. لهذا اذا قمنا بكتابة نفس الموقع مرتين لن يكون هناك ضمان أن القيمة الجديدة سوف تحل محل القيمة القديمة.

مكررات أمامية:

المكررات الأمامية تجمع وظيفة مكررات الادخال وتقريباً وظيفة مكررات الاخراج. افترض أن forwardIterator عبارة عن مكرر أمام. جدول 4-11 يصف العمليات المؤداة على forwardIterator.

جدول 4-11: عمليات على مكرر أمامي:

| التعبير | التأثير |
|---------------------------|---|
| *forwardIterator | تعطي وصول الى العنصر الذي يشير اليه forwardIterator. |
| forwardIterator -> member | تعطي تناول لعضو العنصر. |
| ++forwardIterator | تتحرك الى الامام وتنتج الموقع الجديد (قبل الزيادة). |
| forwardIterator++ | تتحرك الى الامام وتنتج الموقع القديم (بعد الزيادة). |
| forwardIt1 == forwardIt2 | تنتج true اذا كان المكررين متماثلين وبخلاف هذا تنتج false. |
| forwardIt1 += forwardIt2 | تنتج true اذا لم يكن المكررين متماثلين وبخلاف هذا تنتج false. |
| forwardIt1 = forwardIt2 | اسناد. |

يمكن أن يشير المكرر الأمامي الى نفس العنصر في نفس المجموعة ويمكنه معالجة نفس العنصر أكثر من مرة.

المكررات الثنائية الاتجاه:

المكررات الثنائية الاتجاه عبارة عن مكررات أمامية يمكنها أيضاً الانتقال الى الوراء عبر العناصر. افترض أن biDirectionalIterator عبارة عن مكرر ثنائي الاتجاه. العمليات التي يتم تعريفها للمكررات الأمامية (جدول 4-11) تنطبق كذلك على المكررات الثنائية الاتجاه. الانتقال الى الخلف يتم تعريف عمليات التقليل للمكرر biDirectionalIterator. الجدول 4-12 يوضح عمليات اضافية على المكرر الثنائي الاتجاه. جدول 4-12: عمليات اضافية على المكرر الثنائي الاتجاه:

| التعبير | التأثير |
|-------------------------|---|
| --biDirectionalIterator | يتحرك الى الخلف وينتج الموقع الجديد (قبل الزيادة) |
| biDirectionalIterator-- | يتحرك الى الخلف وينتج الموقع القديم (بعد الزيادة) |

يمكن استخدام المكررات الثنائية الاتجاه مع الحاويات من النوع vector، deque، list، و set، multiset، map، و multimap.

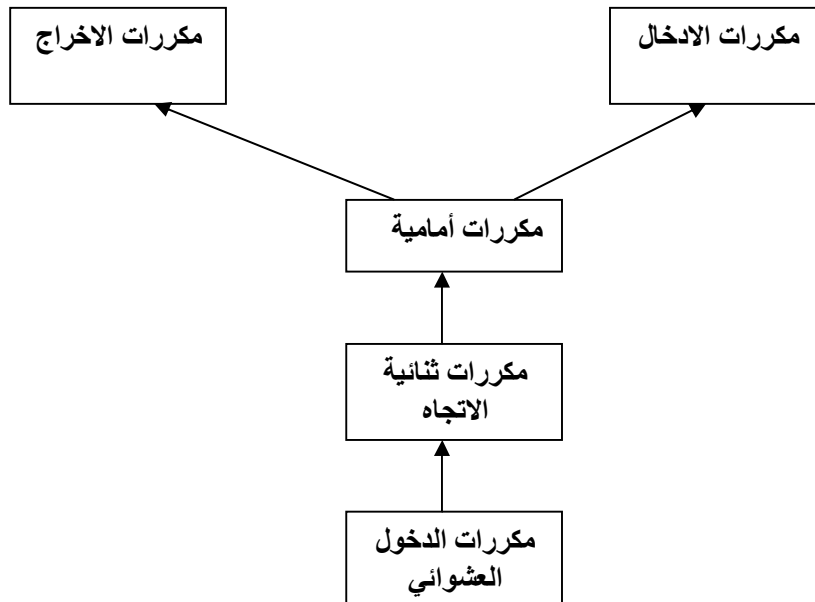
مكررات الدخول العشوائي:

مكررات الدخول العشوائي عبارة عن مكررات ثنائية الاتجاه يمكنها معالجة عناصر الحاوية عشوائياً . يمكن استخدام هذه المكررات مع الحاويات التي تكون من النوع vector و deque و string والمصفوفات . العمليات التي يتم تعريفها للمكررات الثنائية الاتجاه (على سبيل المثال جدول 4-12) تنطبق كذلك على مكررات الدخول العشوائي. (افترض أن rAccessIterator عبارة عن مكرر دخول عشوائي).

جدول 4-13: عمليات اضافية على مكرر الدخول العشوائي:

| التعبير | التأثير |
|--------------------------|---|
| rAccessIterator [n] | تتناول العنصر رقم n. |
| rAccessIterator += n | تحرك rAccessIterator الى الأمام اذا كانت n <= 0 والى الخلف اذا كانت n > 0 |
| rAccessIterator -= n | تحرك rAccessIterator الى الخلف اذا كانت n <= 0 والى الأمام اذا كانت n > 0 |
| rAccessIterator + n | تنتج مكرر العنصر التالي للعنصر n. |
| n + rAccessIterator | تنتج مكرر العنصر التالي للعنصر n. |
| rAccessIterator - n | تنتج مكرر العنصر السابق للعنصر n. |
| rAccessIt1 – rAccessIt2 | تنتج المسافة بين المكررين rAccessIt1 و rAccessIt2. |
| rAccessIt1 < rAccessIt2 | تنتج true اذا كانت rAccessIt1 قبل rAccessIt2 وبخلاف هذا تنتج false. |
| rAccessIt1 <= rAccessIt2 | تنتج true اذا كانت rAccessIt1 قبل rAccessIt2 أو مساوية لها وبخلاف هذا تنتج false. |
| rAccessIt1 > rAccessIt2 | تنتج true اذا كانت rAccessIt1 بعد rAccessIt2 وبخلاف هذا تنتج false. |
| rAccessIt1 >= rAccessIt2 | تنتج true اذا كانت rAccessIt1 قبل rAccessIt2 أو مساوية لها وبخلاف هذا تنتج false. |

شكل 4-1 يوضح تسلسل المكرر.



شكل 4-1: تسلسل المكررات.

الآن تعرف الأنواع المختلفة من المكررات لذلك نقوم الآن بشرح كيفية اعلان مكرر داخل حاوية ما.

`typedef iterator`

كل حاوية (متعاقبة أم مترابطة) تحتوي على مكرر `typedef` لهذا يتم اعلان المكرر في الحاوية باستخدام المكرر `typedef`. على سبيل المثال يقوم البيان:

`Vector<int> : : iterator intVecIter;`

باعلان أن `intVecIter` مكرر داخل حاوية `vector` من النوع `int`. فضلاً عن هذا يمكن استخدام المكرر `intVecIter` على أي `vector<int>` ولكن ليس على أي حاوية أخرى مثل `vector<double>` أو `vector<string>` أو `deque`.

بما أن المكرر عبارة عن `typedef` معرف داخل الحاوية (أي فئة) مثل `vector` يجب علينا استخدام اسم الحاوية المناسب واسم عنصر الحاوية ومعامل حل المجال لاستخدام المعامل `typedef`.

`typedef const_iterator`

إن المكرر يعمل مثل المؤشر ولهذا يمكننا بمساعدة المكرر داخل الحاوية ومعامل عدم الإشارة * تعديل عناصر الحاوية. بالرغم من هذا إذا تم اعلان أن الحاوية `const` يجب علينا منع المكرر من تعديل عناصر الحاوية خاصةً دون قصد. للتعامل مع هذا الموقف يكون لكل حاوية `typedef`, `const_iterator` آخر.

على سبيل المثال البيان:

`vector<int>: : const_iterator intConstVecIt;`

يعلن أن `intConstVecIt` مكرر موجود داخل حاوية `vector` عناصرها من النوع `int`. يتم استخدام المكرر `intConstVecIt` لمعالجة عناصر تلك الحاويات `vector` والتي يتم اعلانها كحاويات `vector` ثابتة من النوع `vector<int>`.

المكرر الذي يكون من النوع `const_iterator` يكون مكرر قراءة فقط.

`typedef reverse_iterator`

تحتوي كل حاوية كذلك على المكرر `typedef reverse_iterator`. يتم استخدام المكرر الذي يكون من هذا النوع للانتقال عبر عناصر الحاوية بالعكس.

`typedef const_reverse_iterator`

المكرر من هذا النوع يكون مكرر قراءة فقط ويتم استخدامه للانتقال عبر عناصر الحاوية بالعكس. يكون ضروري إذا تم اعلان الحاوية بأنها `const` ونحتاج الى الانتقال عبر عناصر الحاوية بالعكس.

بالإضافة الى الأربعة `typedefs` السابقة هناك عدة أنواع من `typedefs` مشتركة بين جميع الحاويات. جدول 4-14 يوضحها.

جدول 4-14: `typedefs` متنوعة مشتركة بين جميع الحاويات:

| التأثير | typedef |
|--|-----------------|
| نوع ناتج طرح مكررين يشيرون الى نفس الحاوية | diferrence_type |
| مؤشر الى نوع العناصر المخزنة في الحاوية. | pointer |
| اشارة الى نوع العناصر المخزنة في الحاوية. | reference |
| اشارة ثابتة الى نوع العناصر المخزنة في الحاوية. الاشارة الثابتة للقراءة فقط. | const reference |
| النوع المستخدم لعد العناصر في الحاوية. يتم استخدام هذا النوع للاشارة خلال الحاويات المتعاقبة فيما عدا الحاويات List. | size_type |
| نوع عناصر الحاوية | value_type |

مكررات التدفق:

هناك مجموعة مفيدة أخرى من المكررات وهي مكررات التدفق: مكررات istream ومكررات ostream. هذا الفصل يوضح نوعي المكررات. يتم استخدام المكررات istream لادخال البيانات في برنامج ما من تدفق المدخلات وتحتوي الفئة istream_iterator على تعريف مكرر تدفق الادخال. التركيب العام لاستخدام مكرر istream هو:

```
istream_iterator <Type> isIdentifier (istream&);
```

حيث تكون Type اما نوع مدمج أو نوع فئة محدد بواسطة المستخدم يتم تعريف مكرر الادخال من أجله. تتم تهيئة المحدد isIdentifier باستخدام المقوم التي تكون معطياته اما هدف للفئة istream مثل cin أو أي نوع ثانوي معرف من istream مثل ifstream.

ostream_iterator

يتم استخدام المكررات ostream لايخراج البيانات من البرنامج الى داخل تدفق المخرجات. تم تعريف تلك المكررات من قبل في هذا الفصل ونقوم باستعراضها هنا من أجل الاكمال.

تحتوي الفئة ostream_iterator على تعريف مكرر تدفق المخرجات والتركيب العام لاستخدام مكرر ostream هو:

```
ostream_iterator <Type> osIdentifier (ostream&);
```

أو:

```
ostream_iterator <Type> osIdentifier (ostream&, char* deLimit);
```

حيث تكون Type اما نوع مدمج أو نوع فئة محدد بواسطة المستخدم يتم تعريف مكرر الاخراج من أجله. تتم تهيئة المحدد osIdentifier باستخدام المقوم التي تكون معطياته اما هدف للفئة ostream مثل cout أو أي نوع ثانوي معرف من ostream مثل ofstream. في الصيغة الثانية المستخدمة لاعلان مكرر ostream يمكننا باستخدام المعطى الثاني (delimiter) من مقوم التهيئة أن نحدد رمز يفصل المخرجات.

تمرين برمجة: تقرير درجات:

مثال البرمجة هذا يوضح مفاهيم التوريث والتكوين.
منتصف الفصل الدراسي بجامعةك المحلية يقترب ومكتب السجل يريد منك اعداد تقارير الدرجات بمجرد تسجيل درجات الطلاب. بعض الطلاب الملتحقين لم يقوموا بعد بسداد مصروفاتهم ومع هذا اذا كان الطالب دفع المصروفات يتم توضيح الدرجات بتقرير الدرجات مع متوسط الدرجات. اذا لم يكن الطالب دفع المصروفات لا يتم طبع الدرجات. بالنسبة الى هؤلاء الطلاب يتضمن تقرير الدرجات رسالة تشير الى الدرجات التي تم منعها بسبب عدم سداد المصروفات. كما يوضح تقرير الدرجات مبلغ الفاتورة.
مكتب السجل ومكتب العمل يريد مساعدتك في كتابة برنامج يمكنه تحليل بيانات الطلاب وطباعة تقارير الدرجات المناسبة. يتم تخزين البيانات في ملف ما بالصيغة التالية:

```
tuitionRate
studentName studentID isTuitionPaid numberOfCourses
courseName courseNumber creditHours grade
courseName courseNumber creditHours grade
.
.
.
studentName studentID isTuitionPaid numberOfCourses
courseName courseNumber creditHours grade
courseName courseNumber creditHours grade
.
.
.
```

معدل المصروفات

| اسم الطالب | هوية الطالب | المصروف المدفوع | عدد البرامج التدريبية |
|--------------|--------------|-----------------|-----------------------|
| اسم البرنامج | رقم البرنامج | ساعات الاعتماد | الدرجة |
| اسم البرنامج | رقم البرنامج | ساعات الاعتماد | الدرجة |

| اسم الطالب | هوية الطالب | المصروف المدفوع | عدد البرامج التدريبية |
|--------------|--------------|-----------------|-----------------------|
| اسم البرنامج | رقم البرنامج | ساعات الاعتماد | الدرجة |
| اسم البرنامج | رقم البرنامج | ساعات الاعتماد | الدرجة |

الصف الأول يشير الى معدل المصروفات لكل ساعة معتمدة. يتم اعطاء بيانات الطلاب بعد هذا.

فيما يلي عينة لملف الادخال:

345
Lisa Miller 890238 Y 4
Mathematics MTH345 4 A
Physics PHY357 3 B
ComputerSci CSC478 3 B
History HIS356 3 A

الصف الأول يشير الى أن معدل المصروفات هو 345 دولار لكل ساعة معتمدة. بعد هذا يتم اعطاء بيانات البرنامج التدريبي للطالبة ليزا ميلر: هوية ليزا ميلر 890238 وقامت بسداد المصروفات وتأخذ 4 برامج تدريبية. رقم البرنامج التدريبي لفصل الرياضيات الذي تأخذه هو MTH345 والبرنامج له 4 ساعات معتمدة ودرجة منتصف الفصل الدراسي هي أ وهكذا.
المخرجات المرجوة لكل طالب تكون صيغتها كما يلي:

Student Name: Lisa Miller
Student ID: 890238
Number of courses enrolled: 4

| Course No | Course Name | Credits | Grade |
|-----------|-------------|---------|-------|
| CSC478 | ComputerSci | 3 | B |
| HIS356 | History | 3 | A |
| MTH345 | Mathematics | 4 | A |
| PHY357 | Physics | 3 | B |

اسم الطالبة: ليزا ميلر.

هوية الطالبة: 890238

عدد البرامج الملتحقة بها: 4

| رقم البرنامج | اسم البرنامج | الاعتمادات | الدرجة |
|--------------|---------------|------------|--------|
| CSC478 | علوم حاسب آلي | 3 | ب |
| HIS356 | تاريخ | 3 | أ |
| MTH345 | رياضيات | 4 | أ |
| PHY357 | فيزياء | 3 | ب |

العدد الكلي للاعتمادات: 13

معدل درجات منتصف الفصل الدراسي: 3.54

هذه المخرجات توضح أنه يجب ترتيب البرامج التدريبي وفقاً لعدد البرامج التدريبية ولحساب معدل الدرجات نفترض أن الدرجة (أ) مكافئة لأربعة نقاط والدرجة (ب) مكافئة لثلاثة نقاط والدرجة (ج) مكافئة لنقطتين والدرجة (د) مكافئة لنقطة واحدة والدرجة (هـ) مكافئة لصفر.

المدخلات: ملف يحتوي على البيانات في الصيغة الموضحة سابقاً . للاشارة السهلة في بقية المناقشة لنقم بافتراض أن اسم ملف المخرجات هو ch4_GradeData.txt وهذا الملف على القرص المرن (أ).

المخرجات: ملف يحتوي على المخرجات في الصيغة الموضحة سابقاً .

تحليل المشكلة وتصميم الخوارزميات:

يجب أن قوم أولاً بتحديد المكونات الرئيسية للبرنامج. الجامعة بها طلاب وكل طالب يأخذ برنامج تدريبي. لهذا المكونان الرئيسيان هما الطالب والبرنامج التدريبي.

لنقم أولاً بوصف المكون: البرنامج التدريبي.

البرنامج التدريبي: السمة الرئيسية لأي برنامج تدريبي هو اسم البرنامج ورقم البرنامج وعدد الساعات المعتمدة. بالرغم من الدرجة التي يتلقاها الطالب ليست سمة فعلية للبرنامج الا أن هذا المكون يتضمن درجة الطالب أيضاً لتبسيط البرنامج.

من العمليات الهامة التي يكون هناك حاجة الى أدائها على هدف من نوع البرنامج التدريبي ما يلي:

1. تحديد معلومات البرنامج التدريبي.

2. طباعة معلومات البرنامج التدريبي.

3. توضيح الساعات المعتمدة.

4. توضيح رقم البرنامج.

5. توضيح الدرجة.

الفئة التالية تقوم بتعريف البرنامج تدريبي كنوع بيانات مجرد (نظر أيضاً شكل 4-2):

```
class courseType
{
public:
    void setCourseInfo(string cName, string cNo,
                      char grade, int credits);
    //Function to set the course information.
    //The course information is set according to the
    //incoming parameters.
    //Postcondition: courseName = cName; courseNo = cNo;
    //                courseGrade = grade;
    //                courseCredits = credits;
```

الفئة courseType
}

عامة:

Void setCourseInfo (string cName, string cNo,
Char grade, int credits);

// دالة لتحديد معلومات البرنامج التدريبي.

// يتم تحديد معلومات البرنامج التدريبي وفقاً للمعاملات الواردة.

// شرط تالي: اسم البرنامج = cName، رقم البرنامج = cNo

درجة البرنامج = grade، اعتمادات البرنامج = credits

```
void print(bool isGrade);
    //Function to print the course information.
    //This function prints the course information on the
    //screen. Furthermore, if the bool parameter isGrade is
    //true, the grade is shown; otherwise, three stars
    //are printed.

void print(ofstream& outp, bool isGrade);
    //Function to print the course information.
    //This function sends the course information to a file.
    //Furthermore, if the bool parameter isGrade is true,
    //the grade is shown; otherwise, three stars are printed.

int getCredits();
    //Function to return the credit hours.
    //Postcondition: The value of the data member
    //                courseCredits is returned.

void getCourseNumber(string& cNo);
    //Function to return the course number.
    //Postcondition: cNo = courseNo

char getGrade();
    //Function to return the grade for the course.
    //Postcondition: The value of the data member courseGrade
    //                is returned.

    //Overload the relational operators.
bool operator==(const courseType&) const;
bool operator!=(const courseType&) const;
bool operator<=(const courseType&) const;
bool operator<(const courseType&) const;
bool operator>=(const courseType&) const;
bool operator>(const courseType&) const;

courseType(string cName = "", string cNo = "",
            char grade = '*', int credits = 0);
    //constructor
    //The object is initialized according to the
    //parameters.
    //Postcondition: courseName = cName; courseNo = cNo;
    //                courseGrade = grade;
    //                courseCredits = credits;
```

```
void print (bool isGrade);  
// دالة لطباعة معلومات البرنامج التدريبي.  
// هذه الدالة تطبع معلومات البرنامج على  
// شاشة. فضلاً عن هذا اذا كان المعامل isGrade حقيقي يتم اظهار الدرجة وبخلاف هذا  
// يتم طبع ثلاثة نجوم.
```

```
void print (ofstream& outp, bool isGrade);  
// دالة لطباعة معلومات البرنامج التدريبي  
// هذه الدالة ترسل معلومات البرنامج الى ملف ما.  
// فضلاً عن هذا اذا كان المعامل isGrade حقيقي يتم اظهار الدرجة وبخلاف هذا  
// يتم طبع ثلاثة نجوم.
```

```
int getCredits ( );  
// دالة لانتاج ساعات الاعتماد.  
// شرط تالي: يتم انتاج قيمة عنصر البيانات courseCredits.
```

```
void getCourseNumber (string& cNo);  
// دالة لانتاج رقم البرنامج التدريبي.  
// شرط تالي: cNo = رقم البرنامج.
```

```
char getGrade ( );  
// دالة لانتاج الدرجة الخاصة بالبرنامج التدريبي.  
// شرط تالي: يتم انتاج قيمة عنصر البيانات courseGrade.
```

```
// انقال المعاملات المترابطة.
```

```
bool operator==(const courseType&) const;  
bool operator!=(const courseType&) const;  
bool operator<=(const courseType&) const;  
bool operator<(const courseType&) const;  
bool operator>=(const courseType&) const;  
bool operator>(const courseType&) const;
```

```
courseType (string cName = " ", string cNo = " ",  
            char grade = '*', int credits = 0);
```

```
// مقوم
```

```
// تتم تهيئة الهدف وفقاً للمعاملات.
```

```
// شرط تالي: اسم البرنامج = cName، رقم البرنامج = cNo،
```

درجة البرنامج = grade، اعتمادات البرنامج = credits

```
private:
    string courseName;    //variable to store the course name
    string courseNo;      //variable to store the course number
    char courseGrade;     //variable to store the grade
    int courseCredits;    //variable to store the number of credits
};
```

خاصة:

string courseName; // متغير لتخزين اسم البرنامج التدريبي.
string courseNo; // متغير لتخزين رقم البرنامج التدريبي.
char courseGrade; // متغير لتخزين الدرجة.
int courseCredits; // متغير لتخزين عدد الاعتمادات.

| courseType |
|--|
| -courseName: string -courseNo: string -courseGrade: char -courseCredits: int |
| +setCourseInfo(string, string, char, int): void +print(bool): void +print(ofstream&, bool): void +getCredits(): int +getCourseNumber(string&): void +getGrade(): char +operator==(const courseType&) const: bool +operator!=(const courseType&) const: bool +operator<=(const courseType&) const: bool +operator<(const courseType&) const: bool +operator>=(const courseType&) const: bool +operator>(const courseType&) const: bool +courseType(string = "", string = "", char = '*', int = 0) |

شكل 2-4: رسم لغة التشكيل الموحدة للفئة courseType.

بعد هذا نناقش تعريفات الدالات لتطبيق عمليات الفئة courseType.

تقوم الدالة setCourseInfo بتحديد قيم عناصر البيانات الخصة وفقاً لقيم المعاملات وتعريفها يكون:

```
void courseType::setCourseInfo(string cName, string cNo,
                                char grade, int credits)
{
    courseName = cName;
    courseNo = cNo;
    courseGrade = grade;
    courseCredits = credits;
}
```

تقوم الدالة print ذات المعامل الواحد بطباعة معلومات البرنامج التدريبي على الشاشة وإذا كان المعامل isGrade حقيقي يتم طبع الدرجة على الشاشة وبخلاف هذا يتم عرض ثلاثة نجوم في مكان الدرجة. كما نقوم كذلك بطبع اسم ورقم البرنامج التدريبي على اليسار بدلاً من اليمين (الشكل الافتراضي). لهذا نحتاج إلى تحديد المسيطر left. هذا المسيطر لا يتم تحديده قبل طباعتنا للدرجة والساعات المعتمدة. الخطوات التالية توضح هذه الدالة:

1. تحديد المسيطر left.
 2. طباعة رقم البرنامج التدريبي.
 3. طباعة اسم البرنامج التدريبي.
 4. إزالة تحديد المسيطر left.
 5. طباعة الساعات المعتمدة.
 6. إذا كانت isGrade حقيقية
إخراج الدرجة
والأخرى ثلاثة نجوم.
- تعريف الدالة print يكون:

```
void courseType::print(bool isGrade)
{
    cout<<left; //Step 1
    cout<<setw(8)<<courseNo<<" "; //Step 2
    cout<<setw(15)<<courseName; //Step 3
    cout.unsetf(ios::left); //Step 4
    cout<<setw(3)<<courseCredits<<" "; //Step 5

    if(isGrade) //Step 6
        cout<<setw(4)<<courseGrade<<endl;
    else
        cout<<setw(4)<<"***"<<endl;
}
```

الدالة print التي لها معاملان ترسل معلومات البرنامج التدريبي إلى ملف ما بدلاً من إرسال المخرجات إلى ملف يتم تمريره كمعامل يكون لهذه الدالة نفس تعريف دالة print السابقة.

تعريف هذه الدالة هو:

```
void courseType::print(ofstream& outp, bool isGrade)
{
    outp<<left; //Step 1
    outp<<setw(8)<<courseNo<<" "; //Step 2
    outp<<setw(15)<<courseName; //Step 3
    outp.unsetf(ios::left); //Step 4
    outp<<setw(3)<<courseCredits<<" "; //Step 5

    if(isGrade) //Step 6
        outp<<setw(4)<<courseGrade<<endl;
    else
        outp<<setw(4)<<"***"<<endl;
}
```

يتم اعلان المقوم بقيم افتراضية. اذا لم يتم تحديد قيم عند اعلان هدف courseType يستخدم المقوم القيم الافتراضية لتهيئة الهدف. باستخدام القيم الافتراضية يتم تهيئة عناصر بيانات الهدف كما يلي:
 رقم البرنامج فارغ، واسم البرنامج فارغ، ودرجة البرنامج *، والساعات المعتمدة عند صفر. بخلاف هذا يتم استخدام القيم المحددة في اعلان الهدف لتهيئة الهدف. تعريفه يكون كما يلي:

```
courseType::courseType(string cName, string cNo,
                        char grade, int credits)
{
    setCourseInfo(cName, cNo, grade, credits);
}
```

تعريفات الدالات المتبقية واضحة.

```
int courseType::getCredits()
{
    return courseCredits;
}

char courseType::getGrade()
{
    return courseGrade;
}

void courseType::getCourseNumber(string& cNo)
{
    cNo = courseNo;
}

bool courseType::operator==(const courseType& right) const
{
    return (courseNo == right.courseNo);
}

bool courseType::operator!=(const courseType& right) const
{
    return (courseNo != right.courseNo);
}

bool courseType::operator<=(const courseType& right) const
{
    return (courseNo <= right.courseNo);
}

bool courseType::operator<(const courseType& right) const
{
    return (courseNo < right.courseNo);
}

bool courseType::operator>=(const courseType& right) const
{
    return (courseNo >= right.courseNo);
}

bool courseType::operator>(const courseType& right) const
{
    return (courseNo > right.courseNo);
}
```

بعد هذا نناقش مكون الطالب.

الطالب: السمات الأساسية للطالب هي اسم الطالب، وهوية الطالب، وعدد البرامج التدريبية الملتحقين بها، وأسماء البرامج التدريبية الملتحقين بها، ودرجة كل برنامج تدريبي. بما أن على كل طالب سداد المصروفات نقوم كذلك بادخال عنصر للإشارة الى ما اذا كان الطالب قام بسداد المصروفات أم لا. كل طالب عبارة عن شخص وكل طالب يأخذ برامج تدريبية. لقد قمنا بالفعل بتصميم الفئة `personType` لمعالجة اسم الشخص الأول واسمه الأخير. كما قمنا كذلك بتصميم فئة لمعالجة معلومات البرنامج التدريبي. لهذا نرى أنه يمكننا نشق الفئة `studentType` لتتبع معلومات الطالب من الفئة `personType` وعنصر واحد من هذه الفئة من النوع `courseType`. يمكننا اضافة المزيد من العناصر حسب الحاجة.

العمليات الأساسية المؤداة على هدف من النوع `studentType` هي:

1. تحديد معلومات الطالب.
2. طباعة معلومات الطالب.
3. حساب عدد الساعات المعتمدة التي تم أخذها.
4. حساب معدل الدرجات.
5. حساب مبلغ السداد.
6. بما أن تقرير الدرجات سوف يطبع البرامج التدريبية بترتيب تصاعدي قم بتصنيف البرامج وفقاً لرقم البرنامج التدريبي.

فئة البيانات التالية تقوم بتعريف `studentType` كنوع بيانات مجرد. نفترض أن الطالب يأخذ ما لا يزيد عن ستة برامج تدريبية في كل فصل دراسي (انظر شكل 4-3 أيضاً).

```

class studentType: public personType
{
public:
    void setInfo(string fname, string lName, int ID,
                bool isTPaid,
                vector<courseType> courses);
    //Function to set a student's information.
    //Postcondition: The data members are set according
    //                to the parameters.

    void print(double tuitionRate);
    //Function to print a student's grade report.

    void print(ofstream& out, double tuitionRate);
    //Function to print a student's grade report.
    //The output is stored in a file specified by the
    //parameter out.

    studentType();
    //default constructor
    //Postcondition: The data members are initialized to
    //                the default values.

    int getHoursEnrolled();
    //Function to return the credit hours in which a student
    //is enrolled.
    //Postcondition: The number of credit hours in which a
    //                student is enrolled is calculated and
    //                returned.

    double getGpa();
    //Function to return the grade point average.
    //Postcondition: The GPA is calculated and returned.

    double billingAmount(double tuitionRate);
    //Function to return the tuition fees.
    //Postcondition: The tuition fees due are calculated
    //                and returned.

private:
    int sId; //variable to store the student ID
    int numberOfCourses; //variable to store the number of
                        //courses
    bool isTuitionPaid; //variable to indicate whether the
                        //tuition is paid

```

الفئة studentType :public personType

عامّة:

```

void setInfo (string fname, string lName, int ID,
              bool isTPaid,
              vector<courseType> courses);
// دالة لتحديد معلومات الطالب.

```

// شرط تالي: يتم تحديد عناصر البيانات وفقاً للعوامل.

```
void print (double tuitionRate);  
// دالة لطباعة تقرير درجات الطالب.
```

```
void print (ofstream& out, double tuitionRate);  
// دالة لطباعة تقرير درجات الطالب.  
// يتم تخزين المخرجات في ملف يحدده المعامل out.
```

```
studentType ( );  
// مقوم افتراضي  
// شرط تالي: يتم تهيئة عناصر البيانات عند القيم الافتراضية.
```

```
int getHoursEnrolled ( );  
// دالة لانتاج الساعات المعتمدة الملتحق بها الطالب.  
// شرط تالي: يتم حساب وانتاج عدد الساعات المعتمدة  
// التي يشترك بها الطالب.
```

```
double getGpa ( );  
// دالة لانتاج متوسط الدرجات.  
// شرط تالي: يتم حساب وانتاج متوسط الدرجات.
```

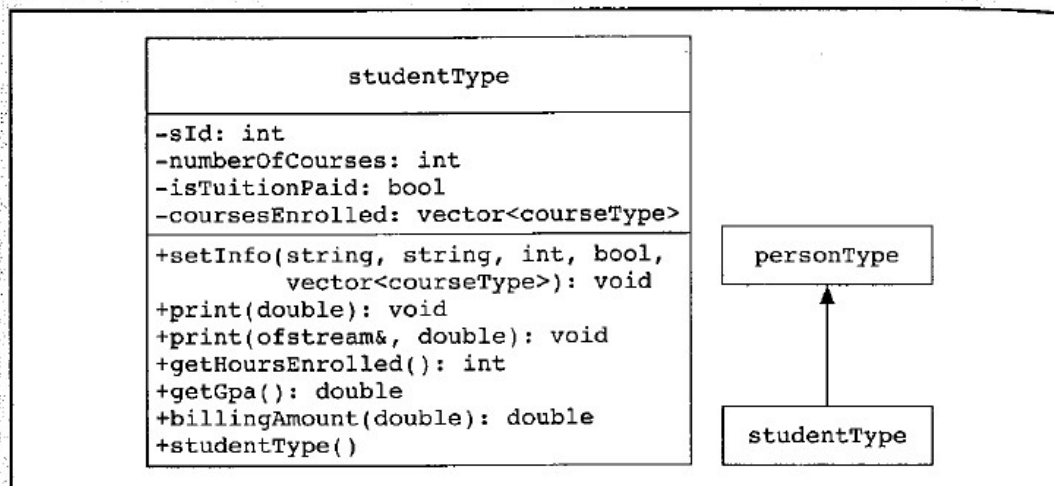
```
double billingAmount (double tuitionRate);  
// دالة لانتاج أتعاب الدروس.  
// شرط تالي: يتم حساب وانتاج رسوم الدروس المستحقة.
```

خاصة:

```
int sId;  
int numberOfCourses;  
bool isTuitionPaid;  
// متغير لتخزين هوية الطالب.  
// متغير لتخزين عدد البرامج التدريبية.  
// متغير للإشارة الى ما اذا تم سداد المصروفات أم لا.
```

```
vector<courseType> coursesEnrolled; //vector to store the  
//courses
```

```
};
```



جدول 3-4: رسم لغة التشكيل الموحدة للفئة studentType وتسلسل التوريث.

بعد هذا نناقش تعريفات دالات تطبيق عمليات الفئة studentType.

تقوم الدالة setInfo أولاً بتهيئة عناصر البيانات الخاصة وفقاً للمعاملات الواردة. الفئة studentType مستمدة من الفئة personType والمتغيرات الخاصة بتخزين الاسم الأول والاسم الأخير عبارة عن عناصر بيانات خاصة لهذه الفئة. لهذا نستدعي دالة العنصر setName للفئة personType ونقوم بتمرير المتغيرات المناسبة لتحديد الاسم الأول والاسم الأخير. لترتيب المصفوفة coursesEnrolled ترتيب تصاعدي نقوم باستخدام الحل الحسابي sort المقدم بواسطة مكتبة القالي المعياري.

من أجل استخدام الحل الحسابي sort لترتيب الحاوية Vector المسماة courses Enrolled نحتاج الى معرفة موقع العنصر الأول والعنصر الأخير في coursesEnrolled. عندما أعلننا coursesEnrolled لم نقم بتحديد حجمها. تقوم الدالة begin الخاصة بالفئة vector بإنتاج موقع العنصر الأول في الحاوية vector والدالة end تحدد موقع العنصر الأخير. لهذا تقوم () coursesEnrolled.begin بتحديد موقع العنصر الأول من الحاوية coursesEnrolled وتقوم () coursesEnrolled.end بتحديد موقع العنصر الأخير. يتم ائصال المعامل <= للفئة courseType ويقارن البرامج التدريبية بواسطة رقم البرنامج ويستخدم الحل الحسابي هذا المعياري لترتيب الحاوية coursesEnrolled. البيان التالي يقوم بترتيب الحاوية coursesEnrolled:

```
sort (coursesEnrolled. begin ( ), coursesEnrolled.end ( ));
```

تعريف الدالة setInfo يكون:

```
void studentType::setInfo(string fName, string lName, int ID,
                          bool isTPaid,
                          vector<courseType> courses)
{
    personType::setName(fName, lName); //set the name

    sId = ID;                          //set the student ID
    isTuitionPaid = isTPaid;           //set isTuitionPaid
    numberOfCourses = courses.size()   //set the number of courses

    coursesEnrolled = courses;         //set the vector coursesEnrolled

    //sort the array coursesEnrolled
    sort(coursesEnrolled.begin(), coursesEnrolled.end());
}
```

يقوم المقوم الافتراضي بتهيئة عناصر البيانات الخاصة عند القيم الافتراضية. لاحظ ان بما ان عنصر البيانات الخاص coursesEnrolled من النوع vector فانه المقوم الافتراضي للفئة vector يتك تنفيذه تلقائياً ويقوم بتهيئة coursesEnrolled.

```
studentType::studentType()
{
    numberOfCourses = 0;
    sId = 0;
    isTuitionPaid = false;
}
```

الدالة print التي لها معامل واحد تخرج تقرير الدرجات على الشاشة ويتم توضيح ما اذا كان الطالب قد سدد مصروفاته أم لا وكذلك الدرجات ومعدل الدرجات. بخلاف هذا تكون المخرجات عبارة عن ثلاثة نجوم تظهر مكان كل درجة ولا يتم عرض معدل الدرجات ويتم اظهار رسالة تشير الى أنه منع نشر الدرجات لعد سداد المصروفات والمبلغ المستحق. هذه الدالة بها الخطوات التالية:

1. اخراج اسم الطالب.
2. اخراج هوية الطالب.
3. اخراج عدد البرامج التدريبية الملتحق بها.
4. اخراج العنوان التالي: رقم البرنامج اسم البرنامج الساعات المعتمدة الدرجة.
5. طباعة معلومات كل برنامج تدريبي.
6. طباعة الساعات المعتمدة الكلية.
7. لاجراج معدل الدرجات والمبلغ المستحق في صيغة عشرية ثابتة مع النقطة العشرية والأصفار قم باستخدام المسيطرين fixed و showpoint وقم كذلك بتحديد الدقة عند منزلتين عشريتين.
8. اذا كان tutitionPaid حقيقي
اخراج معدل الدرجات
بخلاف هذا اخراج المبلغ المستحق ورسالة عن منع نشر الدرجات.

}

هو نفس تعريف الدالة السابقة print ويكون تعريف هذه الدالة هو:

}

هذه الدالة يكون:

```

int studentType::getHoursEnrolled()
{
    int totalCredits = 0;
    int i;

    for(i = 0; i < numberOfCourses; i++)
        totalCredits += coursesEnrolled[i].getCredits();

    return totalCredits;
}

```

إذا لم يكن الطالب سدد المصروفات تقوم الدالة `billingAmount` بحساب وإنتاج المبلغ المستحق بناءً على عدد الساعات المعتمدة التي اشترك بها. وتعريف هذه الدالة يكون:

```

double studentType::billingAmount(double tuitionRate)
{
    return tuitionRate * getHoursEnrolled();
}

```

نناقش الآن الدالة `getGpa` وهذه الدالة تقوم بحساب معدل درجات الطالب. لإيجاد معدل الدرجات نجد النقاط المكافئة لكل درجة ونجمع النقاط ثم نقوم بقسمة المجموع على الساعات المعتمدة الكلية التي يأخذها الطالب. تعريف هذه الدالة يكون كما يلي:

```

double studentType::getGpa()
{
    int i;
    double sum = 0.0;

    for(i = 0; i < numberOfCourses; i++)
    {
        switch(coursesEnrolled[i].getGrade())
        {
            case 'A': sum += coursesEnrolled[i].getCredits() * 4;
                       break;
            case 'B': sum += coursesEnrolled[i].getCredits() * 3;
                       break;

            case 'C': sum += coursesEnrolled[i].getCredits() * 2;
                       break;
            case 'D': sum += coursesEnrolled[i].getCredits() * 1;
                       break;
            case 'F': sum += coursesEnrolled[i].getCredits() * 0;
                       break;
            default: cout<<"Invalid Course Grade"<<endl;
        }
    }

    return sum / getHoursEnrolled();
}

```

البرنامج الرئيسي:

الآن قمنا بتصميم الفئات `studentType` و `courseType` وسوف نستخدم تلك الفئات لإكمال البرنامج.

بما أن الدالة print للفئة studentType تقوم بالحسابات اللازمة لطباعة تقرير الدرجة النهائية فإن البرنامج الأساسي عليه القيام بالقليل للغاية. في الحقيقة كل ما يتم تركه للبرنامج الرئيسي هو اعلان أهداف احتجاز معلومات الطلاب وتحميل البيانات داخل هذه الأهداف ثم طباعة تقارير الدرجات. بما أن المدخلات في ملف وبما أن المخرجات سوف يتم ارسالها الى ملف فاننا نعلن متغيرات تدفق لتتاول ملفات المدخلات والمخرجات. الحل الحسابي للبرنامج يكون جوهرياً :

1. اعلان المتغيرات.
2. فتح ملف المدخلات.
3. اذا لم يكن ملف المدخلات موجود الخروج من البرنامج.
4. فتح ملف المخرجات.
5. الحصول على معدل المصروفات.
6. تحميل بيانات الطلاب.
7. طباعة تقارير الدرجات.

المتغيرات: لتخزين بيانات الطلاب نستخدم الحاوية vector المسماة studentList والتي تكون عناصرها من النوع studentType. كما نحتاج كذلك الى تخزين معدل المصروفات. بما أن البيانات تتم قراءتها من ملف وبما أن المخرجات يتم ارسالها الى ملف فاننا نحتاج الى متغيرين تدفق لتتاول ملفات المدخلات والمخرجات. لهذا نحتاج الى المتغيرات التالية:

```
vector<studentType> studentList;    //vector to store the
                                   //students' data

double tuitionRate;                //variable to store the tuition rate

ifstream infile;                   //input stream variable
ofstream outfile;                  //output stream variable
```

| | |
|---------------------------------|----------------------------------|
| // حاوية لتخزين بيانات الطلاب. | Vector<studentType> studentList; |
| // متغير لتخزين معدل المصروفات. | double tuitionRate; |
| // متغير تدفق الادخال. | ifstream infile; |
| // متغير تدفق الاخراج. | ofstream outfile; |

لتبسيط تعقيد الدالة Main نكتب الدالة getStudentData لتحميل بيانات الطلاب ودالة أخرى printGradeReports لطباعة تقارير الدرجات. القسمان التاليان يصفان تلك الدالات.

الدالة getStudentData: هذه الدالة لها معاملان: معامل لتتاول ملف المدخلات ومعامل لتتاول الحاوية vector المسماة studentList. تعريف هذه الدالة يكون كما يلي:

لكل طالب في الجامعة:

1. احصل على الاسم الأول والاسم الأخير وهوية الطالب و isPaid.
2. اذا كانت isPaid نعم "Y" حدد isTuitionPaid عند true

- وبخلاف هذا حدد isTuitionPaid عند false.
3. احصل على عدد البرامج التدريبية التي يأخذها الطالب.
4. افرغ الحاوية vector المحتوية على معلومات البرنامج التدريبي.
5. لكل برنامج تدريبي:
- أ. احصل على اسم البرنامج والساعات المعتمدة والدرجة.
 - ب. قم بتحميل معلومات البرنامج داخل الهدف courseType.
 - ت. ادخل الهدف المحتوي على معلومات البرنامج داخل الحاوية vector التي تخزن بيانات البرنامج التدريبي.
6. قم بتحميل البيانات داخل الهدف studentType.
7. ادخل الهدف المحتوي على بيانات الطالب داخل قائمة الطالب.

نحتاج الى اعلان عدة متغيرات محلية لقراءة وتخزين البيانات. تعريف الدالة getStudentData هو:

```
void getStudentData(ifstream& inFile,
                    vector<studentType> &studentList)
{
    //local variables
    string fName;           //variable to store the first name
    string lName;           //variable to store the last name
    int ID;                 //variable to store the student ID
    int noOfCourses;        //variable to store the number of courses
    char isPaid;            //variable to store Y/N; that is, is the
                           //tuition paid?
    bool isTuitionPaid;     //variable to store true/false

    string cName;           //variable to store the course name
    string cNo;             //variable to store the course number
    int credits;            //variable to store the course credit hours
```

| | |
|--|---------------------|
| | // متغيرات محلية |
| // متغير لتخزين الاسم الأول | string fName; |
| // متغير لتخزين الاسم الأخير. | string lName; |
| // متغير لتخزين هوية الطالب. | int ID; |
| // متغير لتخزين عدد البرامج التدريبية | int noOfCourses; |
| // متغير لتخزين نعم / لا على سؤال دفع المصروفات. | char isPaid; |
| // متغير لتخزين true / false | bool isTuitionPaid; |
| // متغير لتخزين اسم البرنامج | string cName; |
| // متغير لتخزين رقم البرنامج. | string cNo; |
| // متغير لتخزين الساعات المعتمدة للبرنامج. | int credits; |

```
char grade;                //variable to store the course grade
int i;                    //loop control variable

vector<courseType> courses; //vector of objects to store
                           //the course information
```

// متغير لتخزين درجة البرنامج char grade;
 // متغير سيطرة الحلقة int i;
 // حاوية أهداف لتخزين معلومات البرنامج. vector<courseType> courses;

```
courseType cTemp;
studentType sTemp;

infile>>fName; //Step 1

while(infile)
{
    infile>>lName>>ID>>isPaid; //Step 1

    if(isPaid == 'Y') //Step 2
        isTuitionPaid = true;
    else
        isTuitionPaid = false;

    infile>>noOfCourses; //Step 3

    courses.clear() //Step 4

    for(i = 0; i < noOfCourses; i++) //Step 5
    {
        infile>>cName>>cNo>>credits>>grade; //Step 5.a
        cTemp.setCourseInfo(cName, cNo, //Step 5.b
                           grade, credits); //Step 5.c
        courses.push_back(cTemp);
    }

    sTemp.setInfo(fName, lName, ID, isTuitionPaid,
                 courses); //Step 6
    studentList.push_back(sTemp); //Step 7

    infile>>fName; //Step 1
} //end while
}
```

الدالة **printGradeReports**: تقوم هذه الدالة بطباعة تقارير الدرجات. لكل طالب تقوم باستدعاء الدالة **print** للفئة **studentType** لطباعة تقرير الدرجات.

تعريف الدالة **printGradeReports** يكون:

```
void printGradeReports(ofstream& outfile,
                      vector<studentType> studentList,
                      double tuitionRate)
{
    unsigned int count;

    for(count = 0; count < studentList.size(); count++)
        studentList[count].print(outfile, tuitionRate);
}
```

ترتيب البرنامج:

```

//Header file courseType.h
#ifndef H_courseType
#define H_courseType

#include <fstream>
#include <string>

using namespace std;

//The definition of the class courseType goes here.
.
.
.
#endif

//Implementation file courseTypeImp.cpp
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include "courseType.h"

using namespace std;

//The definitions of the member functions of the class
//courseType go here.
.
.
.

```

```

//Header file personType.h
#ifndef personType_H
#define personType_H

#include <string>
//The definition of the class personType goes here.
.
.
.

#endif

//Implementation file personTypeImp.cpp
#include <iostream>
#include <string>
#include "personType.h"

using namespace std;

//The definitions of the member functions of the class
//personType go here.
.
.
.

//Header file studentType.h
#ifndef H_studentType
#define H_studentType

#include <fstream>
#include <string>
#include <vector>

#include "personType.h"
#include "courseType.h"

using namespace std;

//The definition of the class studentType goes here.
.
.
.

#endif

```

```

//Implementation file studentTypeImp.cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <algorithm>
#include <vector>
#include <iterator>

#include "personType.h"
#include "courseType.h"
#include "studentType.h"

using namespace std;

//The definitions of the member functions of the class
//studentType go here.
.
.
.

//Main Program
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <vector>
#include <iterator>

#include "studentType.h"

using namespace std;

void getStudentData(ifstream& infile,
                   vector<studentType> &studentList);

void printGradeReports(ofstream& outfile,
                      vector<studentType> studentList,
                      double tuitionRate);

int main()
{
    vector<studentType> studentList;

    double tuitionRate;

    ifstream infile;
    ofstream outfile;

```


العدد الكلي للساعات المعتمدة: 17

***** تم احتجاز الدرجات لعدم سداد المصروفات. *****

المبلغ المستحق: 5865.00 دولار.

*_**_**_**_**_**_**_**_**_**_**_**_**_**_**_**_*

هوية الطالب: 746333

| الاعتمادات | الدرجة | اسم البرنامج | رقم البرنامج |
|------------|--------|---------------|--------------|
| 3 | ج | أعمال | BUS128 |
| 4 | ب | كيمياء | CHM348 |
| 3 | ب | علوم حاسب آلي | CSC201 |
| 3 | ب | لغة انجليزية | ENG328 |
| 3 | أ | تاريخ | HIS101 |
| 3 | أ | رياضيات | MTH137 |

3.16 معدل درجات منتصف الفصل الدراسي:

345

5 بيل ويلتون 798324 لا

لغة انجليزية ENG378 3 ب

فلسفة PHL534 3 أ
كيمياء CHM256 4 ج
أحياء BIO234 4 أ
رياضيات MTH346 3 ج

داندي جووت 746333 نعم 6
تاريخ HIS101 3 أ
لغة انجليزية ENG328 3 ب
رياضيات MTH137 3 أ
كيمياء CHM348 4 ب
علوم حاسب آلي CSC201 3 ب
أعمال BUS128 3 ج

مراجعة سريعة

1. مكتبة القالب المعياري تقدم قوالب فئات تعالج القوائم والرسومات والصفوف.
2. المكونات الثلاث الأساسية لمكتبة القالب المعياري هي: الحاويات والمكررات والخوارزميات.
3. حاويات مكتبة القالب المعياري عبارة عن قوالب فئة.
4. يتم استخدام المكررات للانتقال عبر عناصر الحاوية.
5. يتم استخدام الخوارزميات للسيطرة على العناصر في الحاوية.
6. الفئات الرئيسية من الحاويات هي الحاويات المتعاقبة والحاويات المترابطة ومحولات الحاوية.
7. الحاويات المتعاقبة الثلاث المسبقة التعريف هي `vector` و `deque` و `list`.
8. تقوم الحاوية `vector` بتخزين وإدارة أهدافها في مصفوفة حركية.
9. بما أن المصفوفة عبارة عن بنية تناول عشوائي للبيانات فيمكن تناول عناصر الحاوية عشوائياً .
10. اسم الفئة التي تطبق الحاوية `vector` هو `vector`.
11. يتم إدخال عنصر في حاوية `vector` عن طريق استخدام المعاملين `insert` و `push_back`.
12. يتم حذف عنصر في حاوية `vector` عن طريق استخدام العوامل `pop_back` و `erase` و `clear`.
13. يتم إعلان المكرر لحاوية `vector` باستخدام المكرر `typedef` الذي يتم إعلانه كعنصر عام من الفئة `vector`.
14. دالات العنصر المشتركة بين جميع الحاويات هي المقوم الافتراضي، والمقومات ذات العوامل، ومقوم النسخ، والمدمر، والدالات `empty` و `size` و `max_size` و `swap` و `begin` و `end` و `rbegin` و `rend` و `Insert` و `erase` و `clear` ودالات المعامل المرتبطة.
15. دالة العنصر `begin` تنتج مكرر للعنصر الأول في الحاوية.

16. دالة العنصر end تنتج مكرر للعنصر الأخير في الحاوية.
17. بالإضافة الى دالات العنصر المذكورة في رقم 14 تكون دالات العنصر الأخرى المشتركة بين جميع الحاويات المتعاقبة هي insert و push_back و pop_back و erase و clear و resize.
18. يتم استخدام الحل الحسابي copy لنسخ العناصر الموجودة في نطاق محدد الى مكان آخر.
19. باستخدام المكرر ostream يمكن استخدام الدالة copy لايخراج عناصر الحاوية.
20. عندما نصنع مكرر من النوع ostream نقوم كذلك بتحديد نوع العنصر الذي سوف يقوم المكرر باخراجه.
21. يتم تطبيق الحاويات deque كمصفوفات حركية بالطريقة التي يمكن بها ادخال العناصر في كل من نهايتي المصفوفة.
22. الحاوية deque يمكن أن تمتد في أي من الاتجاهين.
23. اسم الفئة المحتوية على تعريف الدالة deque هو deque.
24. بالإضافة الى العمليات المشتركة بين جميع الحاويات هناك عمليات أخرى يمكن استخدامها للسيطرة على عناصر حاوية deque وهي assign و push_front و pop_front و at ومعامل تسجيل المصفوفة [] و front و back.
25. الفئات الخمس من المكررات هي مكررات الادخال، والايخراج، والأمامية، والثنائية الاتجاه، والوصول العشوائي.
26. يتم استخدام مكررات الادخال لادخال البيانات من تدفق المدخلات.
27. يتم استخدام مكررات الاخراج لايخراج البيانات الى تدفق المخرجات.
28. يمكن أن يشير المكرر الأمامي الى نفس العنصر في نفس المجموعة ويعالج نفس العنصر أكثر من مرة.
29. المكررات الثنائية الاتجاه عبارة عن مكررات أمامية يمكن أن تنتقل كذلك الى الخلف عبر العناصر.
30. يمكن استخدام المكررات الثنائية الاتجاه مع الحاويات من النوع list، و set، و multiset، و map، و multimap.
31. مكررات الوصول العشوائي عبارة عن مكررات ثنائية الاتجاه يمكنها معالجة عناصر الحاوية عشوائياً .
32. يمكن استخدام مكررات الوصول العشوائي مع الحاويات من النوع vector، و deque، و string، والمصفوفات.

تمارين

1. ما هي المكونات الثلاث الرئيسية لمكتبة القالب المعياري؟
2. ما الفرق بين حاوية مكتبة القالب المعياري ومكرر مكتبة القالب المعياري؟
3. اكتب بيان يعلن هدف vector يمكنه تخزين 50 عدد عشري.
4. اكتب بيان يقوم باعلان وتخزين عناصر المصفوفة التالية داخل هدف vector:
`char vowels [5] = {'a', 'e', 'i', 'o', 'u'};`
5. اكتب بيان لاعلان أن screen مكرر ostream تتم تهيئته لجهاز الاخراج القياسي الذي يخرج عناصر هدف int للحاوية vector.
 انظر الى البيان التالي:
`vector<int> intVector;`
 افترض أن `intVector = {5, 7, 9, 11, 13}` وافترض أيضاً أن screen عبارة عن مكرر ostream مهيب عند جهاز الاخراج القياسي لاخراج عناصر الهدف vector. ما هو تأثير البيان التالي؟

`copy (vecList.begin (), vecList.end (), screen);`

7. ما هي مخرجات الجزء التالي من البرنامج؟

```
vector<int> vecList(5);
int j;

for(j = 0; j < 5; j++)
    vecList[j] = 2 * j;
for(j = 0; j < 5; j++)
    cout<<vecList[j]<<" ";
cout<<endl;
```

8. ما هي مخرجات الجزء التالي من البرنامج؟ (افترض أن screen مكرر ostream مهيب عند جهاز الاخراج القياسي لاخراج عناصر من النوع int.)

```
int list[5] = {2, 4, 6, 8, 10};
vector<int> vecList(5);

copy(list, list + 5, vecList.begin());

copy(vecList.begin(), vecList.end(), screen);
cout<<endl;
```

9. ماهي مخرجات الجزء التالي من البرنامج؟ (افترض أن screen مكرر ostream مهبط عند جهاز الاخراج القياسي لاجراج عناصر من النوع int).

```
vector<int> vecList;
vector<int>::iterator vecIt;

vecList.push_back(3);
vecList.push_back(5);
vecList.push_back(7);
vecIt = vecList.begin();
++vecIt;
vecList.erase(vecIt);
vecList.push_back(9);

copy(vecList.begin(), vecList.end(), screen);
cout<<endl;
```

10. ماهي مخرجات الجزء التالي من البرنامج؟ (افترض أن screen مكرر ostream مهبط عند جهاز الاخراج القياسي لاجراج عناصر من النوع int).

```
int list[5] = {2, 4, 6, 8, 10};
vector<int> vecList(7);

copy(list, list + 5, vecList.begin());

vecList.push_back(12);

copy(vecList.begin(), vecList.end(), screen);
cout<<endl;
```

11. ماهي مخرجات الجزء التالي من البرنامج؟ (افترض أن screen مكرر ostream مهبط عند جهاز الاخراج القياسي لاجراج عناصر من النوع double).

```
vector<double> sales(3);

sales[0] = 50.00;
sales[1] = 75.00;
sales[2] = 100.00;

sales.resize(5);

sales[3] = 200.00;
sales[4] = 95.00;

copy(sales.begin(), sales.end(), screen);
cout<<endl;
```

12. ماهي مخرجات الجزء التالي من البرنامج؟ (افترض أن screen مكرر ostream مهبط عند جهاز الاخراج القياسي لاجراج عناصر من النوع int).

```
vector<int> intVector;
vector<int>::iterator vecIt;
```

```
intVector.push_back(15);
intVector.push_back(2);
intVector.push_back(10);
intVector.push_back(7);
vecIt = intVector.begin();
vecIt++;
intVector.erase(vecIt);
intVector.pop_back();
```

```
copy(intVector.begin(), intVector.end(), screen);
```

13. افترض أن vecList عبارة عن حاوية vector و

```
vecList = {12, 16, 8, 23, 40, 6, 18, 9, 75}
```

قم بتوضيح vecList بعد تنفيذ البيان التالي:

```
copy (vecList.begin ( ) + 2, vecList.end ( ), vecList.begin ( ));
```

14. افترض أن vecList عبارة عن حاوية vector و

```
vecList = {12, 16, 8, 23, 40, 6, 18, 9, 75}
```

قم بتوضيح vecList بعد تنفيذ البيان التالي:

```
copy (vecList.rbegin ( ) + 3, vecList.rend ( ), vecList.rbegin ( ));
```

15. ما هي مخرجات الجزء التالي من البرنامج؟

```
deque<int> intDeq;
ostream_iterator<int> screen(cout, " ");
deque<int>::iterator deqIt;
```

```
intDeq.push_back(5);
intDeq.push_front(23);
intDeq.push_front(45);
intDeq.push_back(35);
intDeq.push_front(0);
intDeq.push_back(50);
intDeq.push_front(34);
```

```
deqIt = intDeq.begin();
intDeq.insert(deqIt, 76);
intDeq.pop_back();
```

```
++deqIt;
++deqIt;
```

```
intDeq.erase(deqIt);
intDeq.push_front(2 * intDeq.back());
intDeq.push_back(3 * intDeq.front());
```

```
copy(intDeq.begin(), intDeq.end(), screen);
cout<<endl;
```

1. اكتب برنامج يسمح للمستخدم بادخال الأسماء الأخيرة لخمسة مرشحين في انتخابات محلية والأصوات التي يتلقاها كل مرشح. يجب أن يقوم البرنامج باخراج اسم كل مرشح والأصوات التي حصل عليها ونسبة الأصوات الكلية التي حصل عليها المرشح. يجب أن يقوم برنامجك باخراج الفائز في الانتخابات أيضاً . عينة المخرجات تكون كما يلي:

| المرشح | الأصوات التي حصل عليها | % من الأصوات الكلية |
|---------|------------------------|---------------------|
| جونسون | 5000 | 25.91 |
| ميلر | 4000 | 20.72 |
| دافي | 6000 | 31.09 |
| روبنسون | 2500 | 12.95 |
| أنتوني | 1800 | 9.33 |
| المجموع | 19300 | |

الفائز في الانتخابات هو: دافي

2. اكتب برنامج يسمح للمستخدم بادخال أسماء الطلاب يليها درجات امتحانهم ويخرج المعلومات التالية (افترض أن العدد الأقصى من الطلاب في الفصل هو 50):

أ. متوسط الفصل.

ب. أسماء جميع الطلاب التي تقل درجات امتحانهم عن متوسط الفصل مع رسالة مناسبة.

ت. أعلى درجة اختبار وأسماء جميع الطلاب الحاصلين على أعلى درجة.

3. اكتب برنامج يستخدم هدف vector لتخزين مجموعة من الأعداد الحقيقية. البرنامج يخرج أصغر الأعداد وأكبرها ومتوسطها. عند اعلان الهدف vector لا تحدد الحجم واستخدم الدالة push_back لادخال العناصر في الهدف vector.

4. اكتب تعريف قالب الدالة reverseVector لعكس عناصر الهدف vector ونموذجه هو:

```
template<class elemType>
void reverseVector(vector<elemType> &list);
//Reverses the elements of the vector list.
//Example: Suppose list = {4, 8, 2, 5}.
//      After a call to this function, list =
//      {5, 2, 8, 4}.
```

// يعكس عناصر القائمة vector.

// مثال: افترض أن القائمة = {4, 8, 2, 5}.

// بعد استدعاء هذه الدالة تكون القائمة = (5، 2، 8-4).
 قم أيضاً بكتابة برنامج لاختبار الدالة reverseVector. عند اعلان الهدف object لا تحدد الحجم
 واستخدم الدالة push_back لادخال العناصر في الهدف vector.
 5. اكتب تعريف قالب الدالة seqSearch لتطبيق البحث المتتالي على هدف vector ونموذجه
 كما يلي:

```
template<class elemType>
int seqSearch(const vector<elemType> &list, const
              elemType& item);
//If the item is found in the list, returns the
//position of the item in the list; otherwise,
//returns -1.
```

// اذا تم العثور على العنصر في القائمة تنتج مكان العنصر
 // في القائمة وبخلاف هذا تنتج -1.
 قم أيضاً بكتابة برنامج لاختبار الدالة seqSearch. واستخدم الدالة push_back لادخال العناصر
 في الهدف vector.
 6. أعد عمل تمرين الترجمة رقم 6 من الفصل الأول حتى يتم الحفاظ على قائمة الكتب في هدف
 vector.
 7. أعد عمل تمرين البرمجة رقم 13 من الفصل الثالث حتى يتم تخزين دفتر العناوين في هدف
 vector.
 8. (سوق البورصة) اكتب برنامج لمساعدة شركة محلية تتاجر في البورصة بجعل أنظمتها آلية.
 تقوم الشركة بالاستثمار في سوق البورصة فقط. في نهاية كل يوم تجاري سوف تود الشركة
 انتاج وتحديد أسهمها حتى يمكن للمستثمرين رؤية كيفية أداء حاملها في هذا اليوم. لنفترض
 أن الشركة تستثمر على سبيل المثال في 10 بورصات مختلفة. اذن المخرجات المرجوة هي
 انتاج قائمة مخزنة برمز البورصة.
 يتم توفير بيانات الاخراج في ملف ما بالصيغة التالية:

Symbol openingPrice closingPrice todayHigh todayLow prevClose
 volume

على سبيل المثال تكون بيانات العينة هي:

```
MSMT 112.50 115.75 116.50 111.75 113.50 6723823
CBA 67.50 75.50 78.75 67.50 65.75 378233
.
.
.
```

هدفی بورسۃ عن طریق رمزهما.

أ. 2. قم بإثقال معامل الادخال << للاخراج السهل.

أ. 3. بما أن البيانات مخزنة في ملف قم بإثقال معامل استخلاص التدفق >> للادخال السهل.
على سبيل المثال افترض أن infile عبارة عن هدف ifstream وتم فتح ملف الادخال باستخدام
الهدف infile. افترض كذلك أن myStock عبارة عن هدف بورصة. اذن البيان:

infile>>myStock;

يقرأ البيانات من ملف الادخال ويقوم بتخزينها في الهدف myStock. (لاحظ أن هذا البيان يقرأ
ويخزن البيانات في المكونات المترابطة من myStock).

ب. الآن بعد تصميمك وتطبيقك للفئة stockType لتطبيق هدف بورصة في برنامج ما حان
الوقت لعمل قائمة بأهداف البورصة. لنقم بتسمية الفئة التي تطبق قائمة أهداف البورصة
stockListType. لتخزين قائمة البورصات تحتاج الى اعلان vector. نوع مكونات هذا
vector هو stockType.

ت. تعريف الفئة stockType يكون:

```
class stockListType
{
public:
    void addToStockList(stockType s);
        //Function to add the stocks to the list.
    ...

private:
    vector<stockType> list; //vector to store the list
        //of stocks
};
```

ث. اكتب برنامج يستخدم هاتين الفئتين لجعل تحليل الشركة لبيانات البورصة آلياً .

الفصل

5

القوائم المتصلة

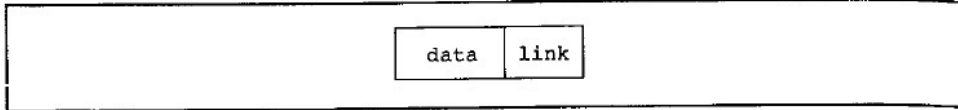
في هذا الفصل سوف تتعلم:

لقد رأيت بالفعل كيفية ترتيب البيانات ومعالجتها بالتتابع باستخدام مصفوفة تسمى **قائمة متعاقب**. لقد قمت بأداء عدة عمليات على القوائم المتعاقبة مثل الترتيب والادخال والحذف والبحث. كما وجدت أن البيانات إذا لم تكن مرتبة فلن البحث عن عنصر في القائمة قد يضيع الكثير من الوقت خاصة مع القوائم الكبيرة. بمجرد أن يتم ترتيب البيانات يمكنك استخدام حل بحثي آخر يسمى البحث الثنائي (تمت مناقشته في الفصل 9) وتحسين الحل البحثي. بالرغم من هذا إذا كانت القائمة مرتبة يصبح الإدخال والحذف مضيقاً للوقت خاصة مع القوائم الأكبر لأن هذه العمليات تتطلب تحسين البيانات حيث أن القائمة الناتجة يجب أن يتم ترتيبها أيضاً. فضلاً عن هذا بما أنه يجب تثبيت حجم المصفوفة أثناء التنفيذ فإنه يمكن إضافة عناصر جديدة إذا كانت هناك مساحة. لهذا تكون هناك قيود عند تنظيمك للبيانات في مصفوفة.

هذا الفصل يساعدك على التغلب على بعض من تلك المشكلات. الفصل الثالث قام بتوضيح كيفية تخصيص الذاكرة (متغيرات) بشكل حركي وإزالة تخصيصها باستخدام المؤشرات. يستخدم هذا الفصل المؤشرات لتنظيم ومعالجة البيانات في القوائم المسماة **قوائم متصلة**. تذكر أن عند تخزين البيانات في مصفوفة تكون الذاكرة الخاصة بمكونات المصفوفة contiguous – أي يتم تخصيص القوالب واحد تلو الآخر. على الجهة الأخرى كما سوف ترى لا تكون كونات القائمة المتصلة التي تم تخصيصها حركياً **** بالضرورة.

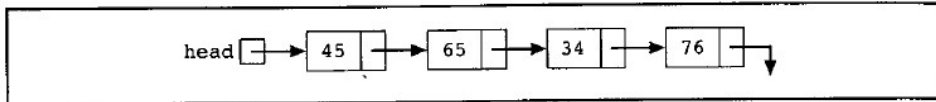
القوائم المتصلة:

القوائم المتصلة عبارة عن مجموعة من المكونات تسمى عقد. كل عقدة (فيما عدا العقدة الأخيرة) تحتوي على عنوان العقدة التالية. لهذا يكون لكل عقدة في القائمة المتصلة مكونان: واحد لتخزين المعلومات (أي البيانات) وآخر لتخزين العنوان المسمى وصلة للعقدة التالية في القائمة. يتم تخزين عنوان العقدة الأول في القائمة في موقع منفصل يسمى الرئيسي أو الأول. الشكل 5-1 تمثيل تصويري للعقدة.



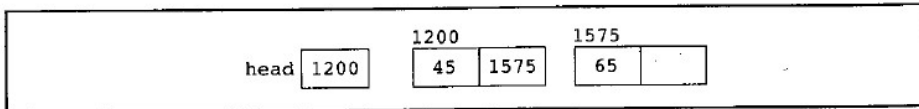
شكل 5-1: تركيب العقدة

القوائم المتصلة: قائمة عناصر تسمى عقد يتم فيها تحديد ترتيب العقد بواسطة العنوان المسمى وصلة والمخزن في كل عقدة. القائمة في شكل 5-2 تعتبر مثال على القائمة المتصلة:



شكل 5-2: قائمة متصلة

السهم السفلي في العقدة الأخيرة يشير الى أن مجال هذه الوصلة هو NULL. ويشير السهم في كل عقدة الى أن عنوان العقدة التي يشير اليها مخزن في هذه العقدة. من أجل استيعاب أفضل لهذه الملحوظة افترض أن العقدة الأولى موجودة في موقع الذاكرة 1200 والعقدة الثانية موجودة في موقع الذاكرة 1575. بهذا يكون لدينا شكل 5-3.



شكل 5-3: قائمة متصلة وقيم الوصلات.

قيمة الجزء الرئيسي هي 1200 وجزء بيانات العقدة الأولى 45 ومكون الوصلة للعقدة الأولى يحتوي على 1575 عنوان العقدة الثانية. اننا نقوم باستخدام الأسهم في أي مكان نرسم به شكل القائمة المتصلة. في الشكلين 5-2 و 5-3 قمنا لأغراض توضيحية بافتراض أن البيانات المخزنة في العقدة من النوع int. ومع هذا يمكن أن تكون البيانات المخزنة في العقدة من أي نوع بما فيها الفئة أو struct. في الحقيق بعد وصف الخصائص الأساسية للقوائم المتصلة في الأقسام القليلة التالية وكيفية بناء قائمة متصلة نقوم باستخدام قوالب الفئة لعمل كود عام لمعالجة القوائم المتصلة التي يمكن استخدامها بعد ذلك

في مجموعة متنوعة من التطبيقات. فضلاً عن هذا ومن أجل البساطة وسهولة الفهم والوضوح تقوم الأشكال 3-5 و 4-5 و 5-5 و 6-5 باستخدام الأعداد الصحيحة كقيم لعناوين الذاكرة. بالرغم من هذا تكون عناوين الذاكرة ثنائية في ذاكرة الحاسب الآلي.

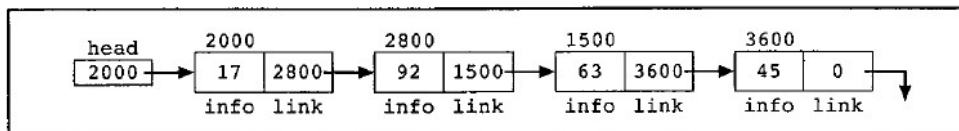
بما أن كل عقدة من القائمة المتصلة لها مكونان فإننا نحتاج إلى إعلان كل عقدة كقوة أو struct. يعتمد نوع بيانات كل عقدة على التطبيق المحدد – أي نوع البيانات الذي تتم معالجته ومع هذا يكون مكون وصلة كل عقدة عبارة عن مؤشر ونوع بيانات متغير المؤشر هذا يكون عبارة عن نوع العقدة ذاتها. بالنسبة إلى القائمة المتصلة السابقة يكون تعريف العقدة كما يلي (افترض أن نوع البيانات هو int):

```
struct nodeType
{
    int info;
    nodeType *link;
};
```

البيان التالي يعلن أن head مؤشر من النوع nodeType.
nodeType *head;

القوائم المتصلة: بعض الخصائص:

من أجل مساعدتك على الحصول على استيعاب أفضل لمفهوم القائمة المتصلة والعقدة يتم فيما يلي وصف بعض الخصائص الهامة للقوائم المتصلة.
انظر إلى القائمة المتصلة في شكل 4-5.



شكل 4-5: قائمة متصلة بها أربعة عقد.

هذه القائمة المتصلة بها أربعة عقد وعنوان العقدة الأولى مخزن في المؤشر head. كل عقدة لها مكونان وهما Info (المعلومات) لتخزين المعلومات، و link (الوصلة) لتخزين عنوان العقدة التالية. من أجل البساطة نفترض أن info من النوع int.

افترض أن العقدة الأولى موجود بالموقع 2000 والعقدة الثانية بالموقع 2800 والثالثة موجودة بالموقع 1500 والرابعة موجودة بالموقع 3600. لهذا تكون قيمة head 2000 وقيمة المكون link للعقدة الأولى تكون 2800 وقيمة المكون link للعقدة الثانية تكون 1500 وهكذا. كذلك القيمة صفر الموجودة

في المكون link من العقدة الأخيرة تعني أن هذه القيمة NULL التي نشير إليها عن طريق رسم سهم إلى أسفل. العدد الموجود أعلى كل عقدة هو عنوان العقدة.

الجدول 1-5 يوضح قيم head وبعض العقد الخاصة بالقائمة المتصلة في شكل 4-5.

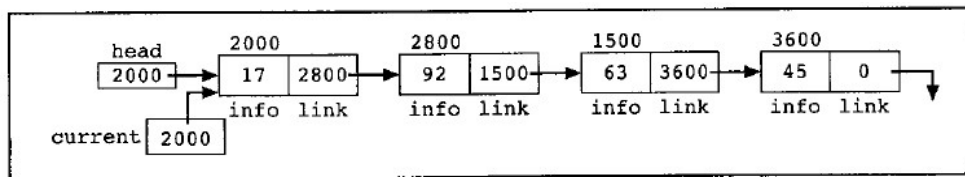
جدول 1-5: قيم head وبعض من عقد القائمة المتصلة في شكل 4-5.

| القيمة | | |
|------------------|------|---|
| head | 2000 | **** |
| head->info | 17 | لأن head = 2000 ومعلومات العقدة بالموقع 2000 = 17 |
| head->link | 2800 | **** |
| head->link->info | 92 | لأن head->link = 2800 ومعلومات العقدة بالموقع 2800 هي 92. |

افترض أن current مؤشر من نفس نوع المؤشر head. اذن البيان:

current = head;

يقوم بنسخ قيمة head داخل current. انظر شكل 5-5.



شكل 5-5: قائمة متصلة بعد تنفيذ البيان current = head.

جدول 2-5 يوضح قيم current وبعض من عقد القائمة المتصلة في شكل 5-5.

جدول 2-5: قيم current وبعض من عقد القائمة المتصلة في شكل 5-5.

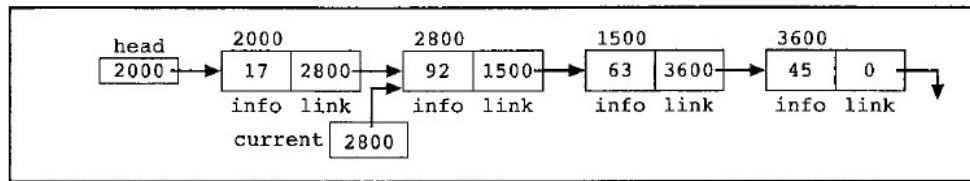
| القيمة | |
|---------------------|------|
| current | 1000 |
| current->info | 17 |
| current->link | 2800 |
| current->link->info | 92 |

الآن انظر إلى البيان:

current = current->link;

يقوم هذا البيان بنسخ قيمة current->link وهي 2800 داخل current. لهذا بعد تنفيذ هذا البيان تشير current إلى العقدة الثانية في القائمة. (عند العمل مع القوائم المرتبطة نستخدم تلك الأنواع من

البيانات لتقديم مؤشر الى العقدة التالية في القائمة كما يحدث عندما نجتاز قائمة متصلة). انظر شكل 5-6.



شكل 5-6: قوائم متصلة بعد تنفيذ البيان `current = current->link`; الجدول 5-3 يوضح قيم `current` جدول 5-2 وبعض من عقد القائمة المتصلة بعد تقديم `current` الى العقدة التالية.

جدول 5-3: قيم `current` وبعض من عقد القائمة المتصلة في شكل 5-6.

| القيمة | |
|--------|--|
| 1000 | <code>current</code> |
| 17 | <code>current->info</code> |
| 2800 | <code>current->link</code> |
| 92 | <code>current->link->info</code> |

أخيراً يوضح جدول 5-4 قيم بعض المؤشرات والعقد الأخرى للقائمة المتصلة في شكل 5-6.

جدول 5-4: قيم بعض مؤشرات وعقد متنوعة للقائمة المتصلة في شكل 5-6.

| القيمة | |
|-------------|--|
| 1500 | <code>head->link->link</code> |
| 63 | <code>head->link->link->info</code> |
| 3600 | <code>head->link->link->link</code> |
| 45 | <code>head->link->link->link->info</code> |
| 3600 | <code>current->link->link</code> |
| 45 | <code>current->link->link->info</code> |
| 0 (أي NULL) | <code>current->link->link->link</code> |
| غير موجود | <code>current->link->link->link->info</code> |

من الآن عند العمل مع القوائم المتصلة نستخدم فقط الأسهم.

اجتياز القائمة المتصلة:

العمليات الأساسية للقائمة المتصلة هي:

- بحث القائمة لتحديد ما اذا كان عنصر محدد موجود بها.

- ادخال عنصر في القائمة.
- حذف عنصر من القائمة.

هذه العمليات تتطلب اجتياز القائمة أي أننا باعطائنا مؤشر للعقدة الأولى من القائمة يجب علينا الانتقال عبر عقد القائمة.

افترض أن المؤشر head يشير الى العقدة الأولى في القائمة ووصلة العقدة الأخيرة هي NULL. لا يمكننا استخدام المؤشر head لاجتياز القائمة لأننا اذا استخدمنا head لاجتياز القائمة فقد نفقد عقد القائمة. تحدث هذه القائمة لأن الوصلات تذهب في اتجاه واحد فقط. يحتوي المؤشر head على عنوان العقدة الأولى وتحتوي العقدة الأولى على عنوان العقدة الثانية وتحتوي العقدة الثانية على عنوان العقدة الثالثة وهكذا. اذا نقلنا head الى العقدة الثانية يتم فقد العقدة الأولى (الا اذا قمنا بحفظ مؤشر لهذه العقدة). اذا ظللنا نقوم بتقديم head الى العقدة التالية سوف نفقد جميع عقد القائمة (الا اذا قمنا بحفظ مؤشرات لكل عقدة قبل تقديم head وهذا أمر غير عملي لأنه قد يحتاج الى وقت اضافي ومساحة ذاكرة اضافية من الحاسب الآلي للحفاظ على القائمة).

لهذا نريد دائماً أن يشير head الى العقدة الأولى وينتج من هذا أننا يجب أن نجتاز القائمة باستخدام مؤشر آخر من نفس النوع. افترض أن current عبارة عن مؤشر من نفس نوع head. الكود التالي يجتاز القائمة:

```
current = head;
while(current != NULL)
{
    //Process current
    current = current->link;
}
```

على سبيل المثال افترض أن head يشير الى قائمة متصلة من الأعداد. الكود التالي يخرج البيانات المخزنة في كل عقدة:

```
current = head;
while(current != NULL)
{
    cout<<current->info<<" ";
    current = current->link;
}
```

ادخال وحذف العنصر:

هذا القسم يناقش كيفية ادخال عنصر في قائمة متصلة أو حذفه منها. انظر الى تعريف العقدة التالي. (للبساطة نفترض أن نوع المعلومات هو int. القسم "القائمة المتصلة كنوع بيانات مجرد" الموجود لاحقاً في هذا الفصل والذي ناقش القوائم المتصلة كنوع بيانات مجرد باستخدام القوالب يستخدم التعريف العام للعقدة.)

```

struct nodeType
{
    int info;
    nodeType *link;
};

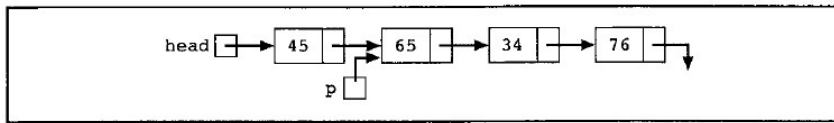
```

نقوم باستخدام اعلان المتغير التالي:

```
nodeType *head, *p, *q, *newNode;
```

الادخال:

انظر الى القائمة المتصلة الموضحة في شكل 7-5.



شكل 7-5: قائمة متصلة قبل ادخال العنصر.

افترض أن p تشير الى العقدة ذات المعلومات 65 وعقدة جديدة ذات معلومات 50 سوف يتم عملها وادخالها بعد p. البيانات التالية تقوم بعمل وتخزين 50 في info العقدة الجديدة:

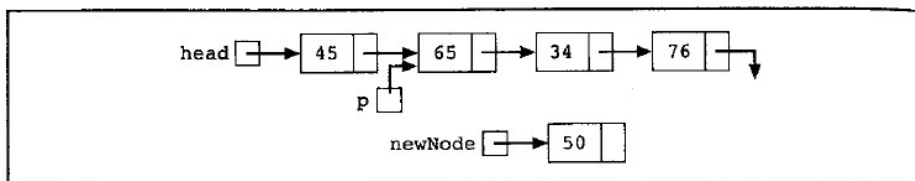
```

newNode = new nodeType; //create newNode
assert(newNode != NULL); //if unable to allocate memory
                        //space, terminate the program
newNode->info = 50;      //store 50 in the new node

```

| | |
|-------------------------------------|--------------------------|
| // تصنع عقدة جديدة. | newNode = new nodeType; |
| // اذا ام تكن قادرة على تخصيص مساحة | assert (newNode !=NULL); |
| // ذاكرة توقف البرنامج | |
| // يقوم بتخزين 50 في العقدة الجديدة | newNode->infor = 50; |

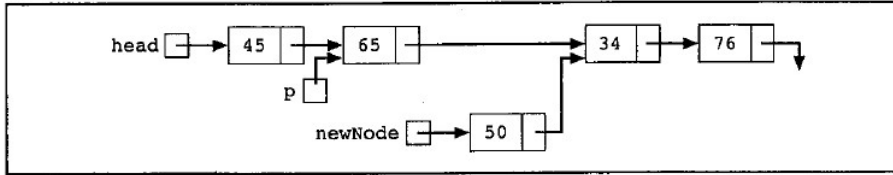
البيان الأول (newNode = new nodeType;) يصنع عقدة في مكان ما بالذاكرة ويقوم بتخزين عنوان العقدة المصنوعة حديثاً في newNode. البيان التالي يقوم بايقاف البرنامج اذا لم يكن النظام قادراً على تخصيص مساحة الذاكرة. البيان الثالث (newNode->infor = 50;) يقوم بتخزين 50 في الحقل info من العقدة الجديدة. انظر شكل 5-8 .



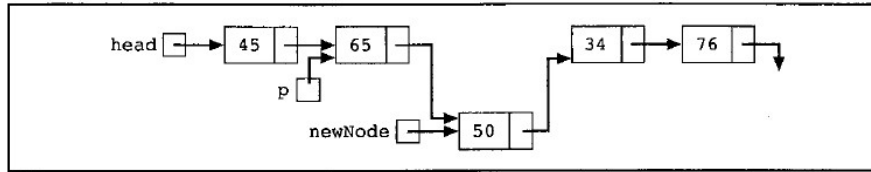
شكل 5-8: عمل newNode عقدة جديدة وتخزين 50 داخلها.
البيانات التالية تقوم بادخال العقدة في القائمة المتصلة عند المكان اللازم:

```
newNode->link = p->link;  
p->link = newNode;
```

بعد تنفيذ البيان الأول (newNode->link = p->link;) القائمة الناتجة تكون كما هي موضحة في شكل 5-9.



شكل 5-9: القائمة المتصلة بعد تنفيذ البيان newNode->link = p->link;
بعد تنفيذ البيان الثاني (p->link = newNode;) القائمة الناتجة تكون كما هي موضحة في شكل 5-10.

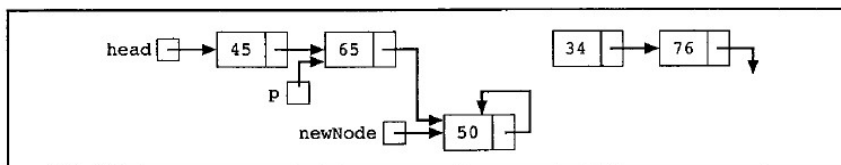


شكل 5-10: القائمة المتصلة بعد تنفيذ البيان p->link = newNode;
لاحظ أن تتابع البيانات لادخال العقدة هام للغاية لأننا من أجل ادخال newNode في القائمة قمنا باستخدام مؤشر واحد فقط هو p لضبط وصلات عقدة القائمة المتصلة. اذا عكسنا تتابع البيانات لا نحصل على النتيجة المرجوة.

على سبيل المثال افترض أننا نقوم بتنفيذ البيانات بالترتيب التالي:

```
p->link = newNode;  
newNode->link = p->link;
```

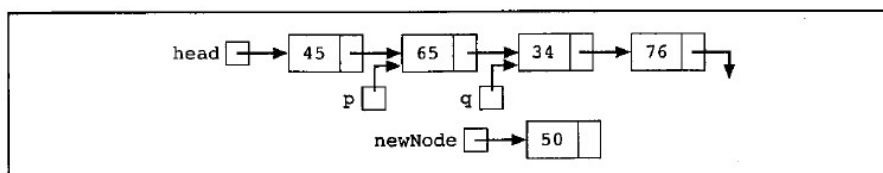
شكل 5-11 يوضح القائمة الناتجة بعد تنفيذ تلك البيانات.



شكل 5-11: القائمة المتصلة بعد تنفيذ البيان `p->link = newNode;` يليه تنفيذ البيان `newNode-`

`>link = p->link;`

يتضح من شكل 5-11 أن `newNode` تشير الى نفسها ويتم فقد بقية القائمة. باستخدام مؤشرين يمكننا تبسيط كود الادخال نوعاً ما. افترض أن `q` تشير الى العقدة ذات `info 34`. انظر شكل 5-12.



شكل 5-12: القائمة المتصلة ذات المؤشرات `p` و `q`.

البيان التالي يدخل `newNode` بين `p` و `q`:

```

newNode->link = q;
p->link = newNode;

```

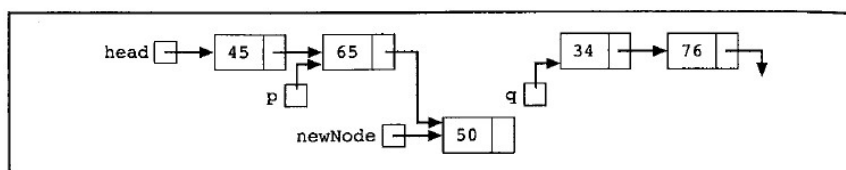
الترتيب الذي يتم به تنفيذ تلك البيانات لا يهم. لتوضيح هذا افترض أننا نقوم بتنفيذ البيانات بالترتيب التالي:

```

p->link = newNode;
newNode->link = q;

```

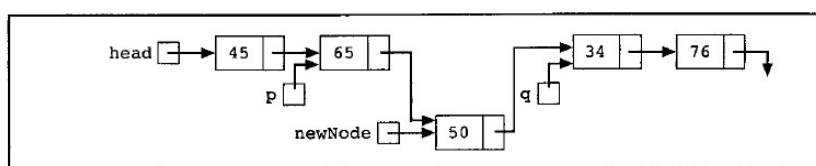
بعد تنفيذ البيان `p->link = newNode;` القائمة الناتجة تكون كما هي موضحة في شكل 5-13



شكل 5-13: القائمة المتصلة بعد تنفيذ البيان `p->link = newNode;`

بما أننا لدينا مؤشر `q` يشير الى القائمة المتبقية فان بقية القائمة لا يتم فقدها.

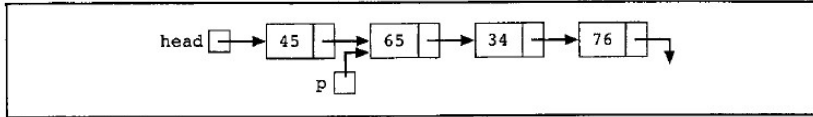
بعد تنفيذ البيان `newNode->link = q;` تكون القائمة كما هي موضحة في شكل 5-14.



شكل 5-14: القائمة المتصلة بعد تنفيذ البيان `newNode->link = q`.

الحذف:

انظر الى القائمة المتصلة الموضحة في شكل 5-15.

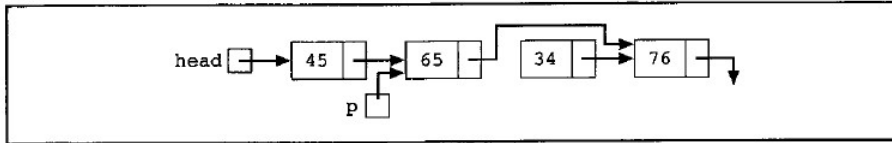


شكل 5-15: العقدة التي يتم حذفها ذات info 34.

افترض أن العقدة ذات info 34 سوف يتم حذفها من القائمة. البيان التالي يحذف العقدة من القائمة:

`p->link = p->link->link;`

شكل 5-16 يوضح القائمة الناتجة بعد تنفيذ البيان السابق.

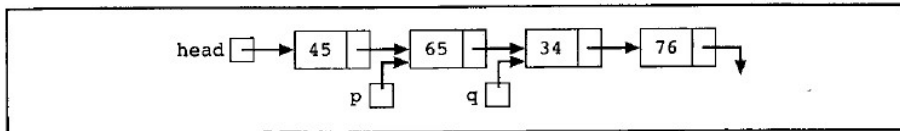


شكل 5-16: القائمة المتصلة بعد تنفيذ البيان `p->link = p->link->link;`

يتضح من شكل 5-16 أنه تم حذف العقدة ذات info 34 من القائمة. بالرغم من هذا لا تزال الذاكرة مشغولة بهذه العقدة أي أن هذه العقدة عالقة. لازالة تخصيص الذاكرة نحتاج مؤشر الى هذه العقدة. البيانات التالية تحذف العقدة من القائمة وتزيل تخصيص الذاكرة التي تشغلها هذه العقدة:

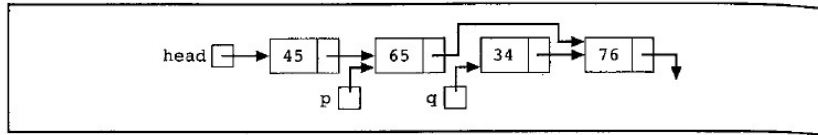
```
q = p->link;  
p->link = q->link;  
delete q;
```

بعد تنفيذ البيان `q = p->link` تكون القائمة كما هي موضحة في شكل 5-17.

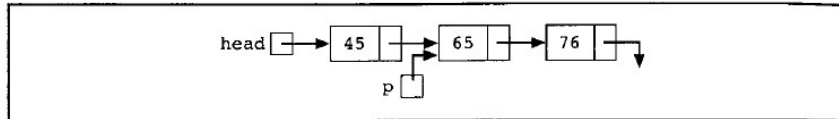


شكل 5-17: القائمة المتصلة بعد تنفيذ البيان `q = p->link`

بعد تنفيذ البيان `p->link = q->link;` تكون القائمة الناتجة كما هي موضحة في شكل 5-18.



شكل 5-18: القائمة المتصلة بعد تنفيذ البيان $p \rightarrow \text{link} = q \rightarrow \text{link};$ بعد تنفيذ البيان delete q; تكون القائمة كما هي موضحة في شكل 5-19.



شكل 5-19: القائمة المتصلة بعد تنفيذ البيان delete q; ينتج من المناقشة السابقة أن من أجل حذف عقدة من قائمة متصلة تحتاج الى مؤشرين – مؤشر لاجتياز القائمة وآخر لازالة تخصيص مساحة ذاكرة العقدة التي يتم حذفها. البيان delete q; يحدد فقط الذاكرة التي تشغلها العقدة والتي تشير اليها q كذاكرة غير مخصصة حتى يمكن استخدامها لاحقاً. المؤشر q قد لا يزال يحتوي على عنوان مساحة الذاكرة هذه. لتجنب أية آثار جانبية قد تحب تحديد q عند NULL بعد البيان delete q.

بناء قائمة متصلة:

بعد علمك الآن بكيفية ادخال عقدة في قائمة متصلة سوف نتعلم بعد هذا كيفية بناء قائمة متصلة. أولاً ننظر الى القائمة المتصلة بوجه عام. اذا كانت البيانات التي نقرأها غير مرتبة فان القائمة المرتبطة لا تحتاج أن يتم ترتيبها. مثل هذه القائمة يمكن بناءها بطريقتين: بالأسلوب الأمامي والأسلوب الخلفي. في الأسلوب الأمامي يتم دائماً ادخال عقدة جديدة في نهاية القائمة المتصلة وفي الأسلوب الخلفي يتم دائماً ادخال عقدة جديدة في بداية القائمة المتصلة. سوف ننظر الى كلتا الحالتين.

بناء قائمة متصلة الى الأمام:

افترض أن العقد في الصيغة المعتادة من info-link و info من النوع int. لنفترض أننا نعالج البيانات التالية:

2 15 8 24 34

نحتاج الى ثلاث مؤشرات لبناء القائمة: مؤشر للإشارة الى العقدة الأولى في القائمة التي لا يمكن نقلها، ومؤشر للإشارة الى العقدة الأخيرة في القائمة، ومؤشر لعمل العقدة الجديدة.

انظر الى اعلان المتغير التالي:

```
nodeType *first, *last, *newNode;
int num;
```

افترض أن first يشير الى العقدة الأولى في القائمة. مبدئياً القائمة فارغة لذلك كل من first و last يكونان NULL. لهذا يجب أن يكون لدينا البيانات:

```
first = NULL;
last = NULL;
```

لتهيئة first و last عند NULL.

بعد هذا ننظر الى البيانات التالية:

```
1  cin>>num;                //read and store a number
                                //in num
2  newNode = new nodeType;    //allocate memory of the
                                //type nodeType and store
                                //the address of the
                                //allocated memory in
                                //newNode
3  assert(newNode != NULL)    //if unable to allocate
                                //memory space,
                                //terminate the program
4  newNode->info = num;        //copy the value of num
                                //into the info field
                                //of newNode
5  newNode->link = NULL;      //initialize the link
                                //field of newNode to
                                //NULL
6  if (first == NULL)         //if first is NULL, the
                                //list is empty; make
                                //first and last point to
                                //newNode
{
    6a      first = newNode;
    6b      last = newNode;
}
7  else                        //the list is not empty
{
```

1 // قراءة وتخزين العدد في num

2 // تخصيص ذاكرة من النوع nodeType وتخزين

// عنوان الذاكرة المخصصة

// في newNode

3 // اذا لم يكن غير قادراً على تخصيص مساحة ذاكرة

// يوقف البرنامج

4 // نسخ قيمة num داخل حقل

info من newNode

5 // تهيئة حقل الوصلة من newNode عند NULL

6 // اذا كانت first = NULL تكون القائمة فارغة

// وتجعل first و last تشير الى newNode

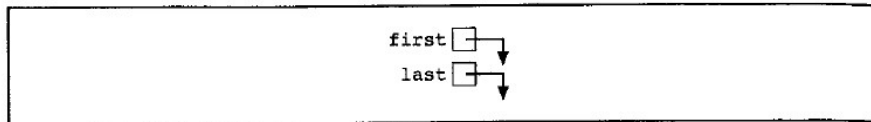
7 // القائمة ليست فارغة.

```
7a      last->link = newNode; //insert newNode at the
                                     //end of the list
7b      last = newNode;        //set last so that it
                                     //points to the actual
                                     //last node in the list
}
```

أ7 // ادخال newNode في نهاية القائمة

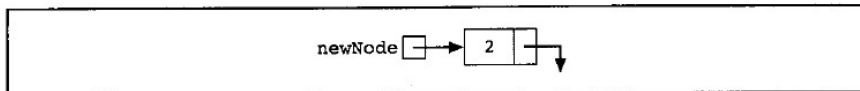
ب7 // وضع last حتى تشير الى العقدة الأخيرة في القائمة.

لنقم الآن بتنفيذ هذه البيانات. مبدئياً كل من first و last يكون NULL. لهذا تكون لدينا القائمة كما هي موضحة في شكل 5-20.



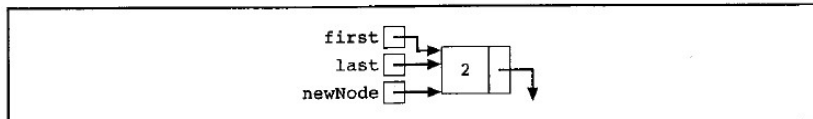
شكل 5-20: قائمة متصلة فارغة.

يعد تنفيذ البيان 1 num تكون 2. البيان 2 يصنع عقدة ويقوم بتخزين عنوان هذه العقدة في newNode. البيان 3 يوقف البرنامج اذا لم يستطع النظام تخصيص مساحة الذاكرة. البيان 4 يقوم بتخزين 2 في حقل info من newNode ويقوم البيان 5 بتخزين NULL في حقل link من newNode. انظر شكل 5-21.



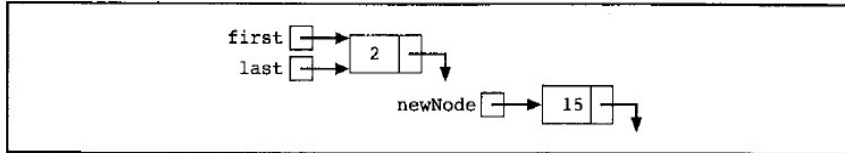
شكل 5-21: newNode ذات 2 info.

بما أن first = NULL نقوم بتنفيذ البيانات 6أ و 6ب. شكل 5-22 يوضح القائمة الناتجة.

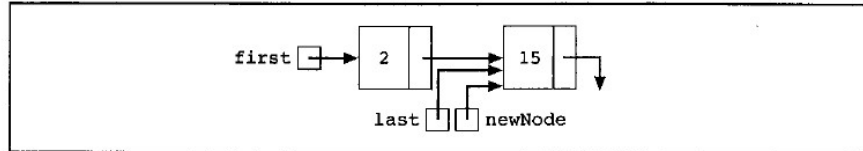


شكل 5-22: قائمة متصلة بعد ادخال newNode فيها.

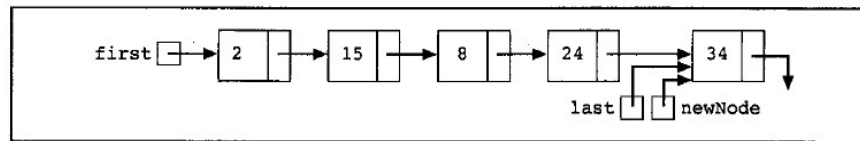
نقوم الآن بتكرار البيانات من 1 وحتى 7. بعد تنفيذ البيان 1 num تساوي 15. البيان 2 يصنع عقدة و يقوم بتخزين عنوان العقدة في newNode. البيان 4 يقوم بتخزين 15 في حقل info من newNode و يقوم البيان 5 بتخزين NULL في الحقل link من newNode. انظر شكل 5-23



شكل 5-23: القائمة المتصلة و newNode ذات info 15.
بما أن first ليست NULL نقوم بتنفيذ البيانات 7 و 7. شكل 5-24 يوضح القائمة الناتجة.



شكل 5-24: القائمة المتصلة بعد ادخال newNode في النهاية.
نقوم الآن بتكرار البيانات من 1 وحتى 7 ثلاث مرات أخرى. شكل 5-25 يوضح القائمة الناتجة.



شكل 5-25: القائمة المتصلة بعد ادخال 8، و 24، و 34.
يمكننا وضع البيانات السابقة في حلقة وتنفيذ الحلقة حتى تتم ملاقة الشروط المحددة لبناء القائمة المتصلة. في الحقيقة يمكننا كتابة دالة C++ لبناء قائمة متصلة.

افترض أننا نقرأ قائمة من الاعداد الصحيحة تنتهي بالرقم 999. تقوم الدالة التالية buildListForward ببناء قائمة متصلة (بأسلوب أمامي) و انتاج مؤشر للقائمة المبنية.

```

nodeType* buildListForward()
{
    nodeType *first, *newNode, *last;
    int num;

    cout<<"Enter a list of integers ending with -999.\n";
    cin>>num;
    first = NULL;

    while(num != -999)
    {
        newNode = new nodeType;
        assert(newNode != NULL);

        newNode->info = num;
        newNode->link = NULL;

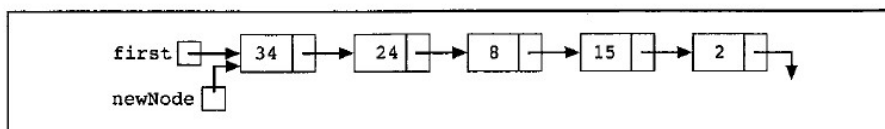
        if(first == NULL)
        {
            first = newNode;
            last = newNode;
        }
        else
        {
            last->link = newNode;
            last = newNode;
        }
        cin>>num;
    } //end while

    return first;
} //end buildListForward

```

بناء قائمة متصلة الى الخلف:

ننظر الآن الى حالة بناء قائمة متصلة الى الخلف. بالنسبة الى البيانات المعطاة من قبل – 2، 15، 8، 24، 34 – تكون القائمة المتصلة كما هي موضحة في شكل 5-26.



شكل 5-26: القائمة المتصلة بعد بنائها الى الخلف.

بما أن العقدة الجديدة يتم دائماً ادخالها في بداية القائمة فاننا لا نحتاج الى معرفة نهاية القائمة ولهذا لا يكون هناك حاجة الى المؤشر last. بعد ادخال العقدة الجديدة في البداية تصبح العقدة الجديدة هي أول عقدة في القائمة. لهذا نحتاج الى تحديث قيمة المؤشر first لكي يشير بشكل صحيح الى العقدة الأولى في القائمة. بعد هذا نرى أننا نحتاج الى مؤشرين فقط لبناء القائمة المتصلة: مؤشر الى القائمة ومؤشر لعمل العقدة الجديدة. بما أن القائمة تكون فارغة في البداية يجب تهيئة المؤشر first عند NULL.

في الكود الوهمي يكون الحل الحسابي هو:

1. تهيئة first عند NULL.

2. لكل عنصر في القائمة:

أ- عمل العقدة الجديدة newNode.

ب- تخزين العنصر في newNode.

ت- ادخال newNode قبل first.

ث- تحديث قيمة المؤشر first.

دالة C++ التالية تبني قائمة متصلة الى الخلف وتنتج مؤشر للقائمة المبنية:

```
nodeType* buildListBackward()
{
    nodeType *first, *newNode;
    int num;

    cout<<"Enter a list of integers ending with -999.\n";
    cin>>num;
    first = NULL;

    while(num != -999)
    {
        newNode = new nodeType; //create a node
        assert(newNode != NULL); //if unable to allocate
                                   //memory space,
                                   //terminate the program
        newNode->info = num; //store the data in
                              //newNode
        newNode->link = first; //put newNode at the
                               //beginning of the list
        first = newNode; //update the head pointer
                          //of the list
        cin>>num; //read the next number
    }

    return first;
} //end buildListBackward
```

القائمة المتصلة كنوع بيانات مجرد:

الأقسام السابقة علمتك الخصائص الأساسية للقوائم المتصلة وكيفية بناء القوائم المنفصلة والتحكم فيها. بما أن القائمة المتصلة عبارة عن بنية بيانات شديدة الأهمية فإن هذا القسم بدلاً من مناقشة قوائم محددة مثل قائمة الأعداد الصحيحة أو قائمة مقاطع يقوم بمناقشة القوائم المتصلة كنوع بيانات مجرد. باستخدام القوالب يقدم تعريف عام للقوائم المتصلة نستخدمه بعد ذلك في القسم التالي ولاحقاً في هذا الكتاب. يقوم كذلك تمرين البرمجة الموجود في نهاية هذا الفصل باستخدام التعريف العام للقوائم المتصلة.

العمليات الأساسية على القوائم المتصلة هي:

1. تهيئة القائمة.

2. التحقق مما إذا كانت القائمة فارغة.

3. اخراج القائمة.
 4. ايجاد طول القائمة.
 5. تدمير القائمة.
 6. استعادة المعلومات الموجودة في العقدة الأولى.
 7. استعادة المعلومات الموجودة في العقدة الأخيرة.
 8. البحث عن عنصر محدد في القائمة.
 9. ادخال عنصر في القائمة.
 10. حذف عنصر من القائمة.
 11. عمل نسخة من القائمة المتصلة.
- أولاً نناقش هذه العمليات على قوائم كيفية – أي القوائم التي يمكن ترتيبها أو عدم ترتيبها. القسم التالي يناقش القوائم المتصلة المرتبة.
- إذا كانت القائمة كيفية يمكننا ادخال عنصر جديد اما في النهاية أو في البداية. فضلاً عن هذا يمكن لهذه القائمة أن يتم بنائها مبدئياً اما بأسلوب أمامي أو خلفي. الدالة `buildListForward` تحتاج أن يتم ادخال العنصر الجديد في النهاية والدالة `buildListBachward` تحتاج أن يتم ادخال العنصر الجديد في البداية. للملاءمة بين كل من العاملين نقوم بكتابة دالتين: `insertFirst` لادخال العنصر الجديد في بداية القائمة، و `insertLast` لادخال العنصر الجديد في نهاية القائمة. وحتى نجعل الحل الحسابي أكثر كفاءة نوعاً ما نبقي على مؤشرين في القائمة وهما: `first` الذي يشير الى العقدة الأولى في القائمة و `last` الذي يشير الى العقدة الأخيرة في القائمة. بالاضافة الى هذا نقوم باثقال معامل ادخال التدفق لادخال عناصر القائمة.
- الفئة التالية تقوم بتعريف القائمة المتصلة كنوع بيانات مجرد:

```

//Definition of the node

template<class Type>
struct nodeType
{
    Type info;
    nodeType<Type> *link;
};

template<class Type>
class linkedListType
{
    friend ostream& operator<<(ostream&, const linkedListType<Type>&);
    //Overload the stream insertion operator.

public:
    const linkedListType<Type>& operator=
        (const linkedListType<Type>&);
    //Overload the assignment operator.
    void initializeList();
    //Function to initialize the list to an empty state.
    //Postcondition: first = NULL; last = NULL;
    //                count = 0
    bool isEmptyList();
    //Function to determine whether the list is empty.
    //Postcondition: Returns true if the list is empty;
    //                otherwise, returns false.

    int length();
    //Function to return the number of nodes in the
    //list.
    //Postcondition: The value of count is returned.
    void destroyList();
    //Function to delete all the nodes from the list.
    //Postcondition: first = NULL; last = NULL;
    //                count = 0
    Type front();
    //Function to return the first element of the list.
    //Precondition: The list must exist and must not
    //    be empty.
    //Postcondition: If the list is empty, then the
    //    program terminates; otherwise, the first
    //    element of the list is returned.
    Type back();
    //Function to return the last element of the
    //list.
    //Precondition: The list must exist and must not
    //    be empty.

```

```

    //Postcondition: If the list is empty, then the
    //  program terminates; otherwise, the last
    //  element of the list is returned.

bool search(const Type& searchItem);
    //Function to determine whether searchItem is in
    //the list.
    //Postcondition: Returns true if searchItem is found
    //  in the list; otherwise, returns false.

void insertFirst(const Type& newItem);
    //Function to insert newItem in the list.
    //Postcondition: first points to the new list,
    //  newItem is inserted at the beginning
    //  of the list, last points to the last node, and
    //  count is incremented by 1.

void insertLast(const Type& newItem);
    //Function to insert newItem at the end of the list.
    //Postcondition: first points to the new list,
    //  newItem is inserted at the end of the list,
    //  last points to the last node in the list, and
    //  count is incremented by 1.

void deleteNode(const Type& deleteItem);
    //Function to delete deleteItem from the list.
    //Postcondition: If found, the node containing
    //  deleteItem is deleted from the list, first points
    //  to the first node, last points to the last
    //  node of the updated list, and count is decremented
    //  by 1.

linkedListType();
    //default constructor
    //Initializes the list to an empty state.
    //Postcondition: first = NULL; last = NULL;
    //  count = 0

linkedListType(const linkedListType<Type>& otherList);
    //copy constructor

~linkedListType();
    //destructor
    //Deletes all the nodes from the list.
    //Postcondition: The list object is destroyed.

protected:
    int count;          //variable to store the number of
                        //elements in the list

```

```

nodeType<Type> *first; //pointer to the first node of
                        //the list
nodeType<Type> *last;  //pointer to the last node of
                        //the list
private:
void copyList(const linkedListType<Type>& otherList);
//Function to make a copy of otherList.
//Postcondition: A copy of otherList is created
//               and assigned to this list.

};

```

لاحظ أن عناصر بيانات الفئة `linkedListType` عناصر محمية وليست خاصة لأننا سوف نشق منها فئات أخرى. تتم الإشارة إلى هذه الفئة في قسم "القوائم المتصلة المرتبة" الموجود في جزء لاحق من هذا الفصل وفي الفصلين 9 و10.

لاحظ أيضاً أن الدالة `copyList` يتم إعلانها كدالة خاصة وهذا لأننا نستخدم هذه الدالة فقط في تطبيق نقوم النسخ والدالة لا تقال معامل الاسناد. لعمل نسخة من قائمة أخرى يمكنك استخدام معامل الاسناد بعد ائقاله.

تعريف الفئة `linkedListType` يتضمن دالة عنصر لا تقال معامل الاسناد. بالنسبة إلى الفئات التي تتضمن عناصر بيانات مؤشر يجب ان يتم ائقال معامل الاسناد بوضوح (انظر الفصلين 2 و3). لنفس السبب يتضمن تعريف الفئة مقوم نسخ أيضاً. بعد هذا نناقش تطبيق دالات العنصر. تكون القائمة فارغة اذا كانت `first` `NULL` ولهذا يكون تعريف الدالة `isEmptyList` لتطبيق هذه العملية كما يلي:

```

template<class Type>
bool linkedListType<Type>::isEmptyList()
{
    return(first == NULL);
}

```

لمقوم الافتراضي:

يقوم المقوم الافتراضي بتبئة القائمة على حالة فارغة. تذكر أنه عند اعلان هدف من النوع `linkedListType` وعدم تمرير أي قيمة يتم تنفيذ المقوم الافتراضي تلقائياً ويكون تعريفه كما يلي:

```

template<class Type>
linkedListType<Type>::linkedListType()
{
    first = NULL;
    last = NULL;
    count = 0;
}

```

من تعريف الدالة isEmptyList والمقوم الافتراضي ينتج أن كل من هذه الدالات من النوع $O(1)$.
تدمير القائمة:

تقوم الدالة destroyList بإزالة تخصيص الذاكرة التي تشغلها كل عقدة. اننا نقوم باجتياز القائمة بدءاً من العقدة الأولى ونزيل تخصيص الذاكرة عن طريق استدعاء المعامل delete. اننا نحتاج الى مؤشر مؤقت لإزالة تخصيص الذاكرة. بمجرد تدمير القائمة بأكملها يجب أن نقوم بتحديد المؤشرات first و last عند NULL والمؤشر count عند صفر. تعريفه يكون:

```
template<class Type>
void linkedListType<Type>::destroyList()
{
    nodeType<Type> *temp;    //pointer to deallocate
                             //the memory occupied by
                             //the node
    while(first != NULL)    //while there are nodes in
                             //the list
    {
        temp = first;        //set temp to the current
                             //node
        first = first->link;  //advance first to the
                             //next node
        delete temp;         //deallocate the memory
                             //occupied by temp
    }

    last = NULL;    //initialize last to NULL; first has
                   //already been set to NULL by the
                   //while loop
    count = 0;
}
```

| | |
|--|-----------------------|
| //مؤشر لإزالة تخصيص الذاكرة | nodeType<Type> *temp; |
| // التي تشغلها العقدة. | |
| // عندما لا يكون هناك عقد في القائمة | while (first != NULL) |
| // تحديد temp عند العقدة التالية | temp = first; |
| // تقدم first الى العقدة التالية | first = first->link; |
| // يزيل تخصيص الذاكرة التي يشغلها temp | delete temp; |
| // يهيئ last عند NULL وتم تحديد first بالفعل | last = NULL; |
| // عند NULL عن طريق الحلقة while. | |

إذا كان هناك n عنصر بالقائمة فإن الحلقة while يتم تنفيذها n مرة. ينتج من هذا أن الدالة destroyList من النوع $O(n)$.

تهيئة القائمة:

الدالة initializeList تهيئ القائمة في وضع فارغ. لاحظ أن المقوم الافتراضي أو مقوم النسخ قام بالفعل بتهيئة القائمة عندما تم اعلان هدف القائمة. في الحقيقة تقوم هذه العملية باعادة تهيئة القائمة في وضع فارغ لذلك يجب أن يحذف العقد (إذا وجدت) من القائمة. يمكن اتمام هذه المهمة عن طريق

استخدام عملية destroyList التي تقوم كذلك باعادة تحديد المؤشرين first و last عند NULL والمؤشر count عند صفر. تعريفه يكون:

```
template<class Type>
void linkedListType<Type>::initializeList()
{
    destroyList(); //if the list has any nodes,
                  //delete them
}
```

destroyList (); // اذا كانت هناك عقد في القائمة
// احذفها.

الدالة initializeList تستخدم الدالة destroyList التي هي من النوع $O(n)$ ولهذا تكون الدالة initilaizeList من النوع $O(n)$.

اثقال معامل ادخال التدفق:

لاخراج البيانات المخزنة في عقد قائمة متصلة نقوم بإثقال معامل ادخال التدفق. الآن لاجراج البيانات الموجودة في كل عقدة يجب أن نجر القائمة بدءاً من العقدة الأولى. بما أن المؤشر first يشير دائماً الى العقدة الأولى في القائمة فاننا نحتاج الى مؤشر آخر لاجتياز القائمة. (اذا استخدمنا first لاجتياز القائمة سوف يتم فقد القائمة كلها).

تعريف دالة اثقال معامل ادخال التدفق هو:

```
template<class Type>
ostream& operator<<(ostream& osObject,
                  const linkedListType<Type>& list)
{
    nodeType<Type> *current; //pointer to traverse the list

    current = list.first;    //set current so that it points
                           //to the first node
    while(current != NULL)   //while more data to output
    {
        osObject<<current->info<<" ";
        current = current->link;
    }

    return osObject;
}
```

كما في حالة الدالة destroyList تكون دالة اثقال معامل ادخال التدفق من النوع $O(n)$.

طول القائمة:

يتم تخزين طول القائمة المتصلة (أي عدد العقد في القائمة) في المتغير count. لهذا تقوم هذه الدالة بإنتاج قيمة هذا المتغير. تعريفه هو:

```
template<class Type>
int linkedListType<Type>::length()
{
    return count;
} //end length
```

استعادة بيانات العقدة الأولى:

تقوم الدالة front بانتاج المعلومات الموجودة في العقدة الأولى وتعريفها يكون:

```
template<class Type>
Type linkedListType<Type>::front()
{
    assert(last != NULL);
    return first->info; //return the info of the
                        //first node
} //end front
```

لاحظ أنه اذا كانت القائمة فارغة يقوم البيان assert بايقاف البرنامج. لهذا قبل استدعاء هذه الدالة تأكد من رؤية مما اذا كانت القائمة غير فارغة.

استعادة بيانات العقدة الأخيرة:

تقوم الدالة back بانتاج المعلومات الموجودة في العقدة الأخيرة وتعريفها واضح وهو:

```
template<class Type>
Type linkedListType<Type>::back()
{
    assert(last != NULL);
    return last->info; //return the info of the
                      //last node
} //end back
```

لاحظ أنه اذا كانت القائمة فارغة يقوم البيان assert بايقاف البرنامج. لهذا قبل استدعاء هذه الدالة تأكد من رؤية مما اذا كانت القائمة غير فارغة.

ينتج من تعريفات الدالات length و front و back أن كل من هذه الدالات من النوع $O(1)$.

بحث القائمة:

تقوم دالة العنصر search ببحث القائمة من أجل عنصر محدد. اذا تم العثور على العنصر تنتج true. بما أن القائمة المتصلة ليست بنية بيانات وصول عشوائي اذن يجب أن نقوم ببحث القائمة بشكل متتالي بدءاً من العقدة الأولى.

الخطوات التالية تصف هذه الدالة:

1. قارن عنصر البحث مع العقدة الحالية في القائمة. اذا كانت معلومات العقدة الحالية هي نفسها معلومات عنصر البحث أوقف البحث وبخلاف هذا اجعل العقدة التالية هي العقدة الحالية.
2. أعد الخطوة الأولى حتى يتم العثور على العنصر أو لا تعد هناك بيانات في القائمة لمقارنتها مع عنصر البحث.

تعريف الدالة search هو:

```
template<class Type>
bool linkedListType<Type>::search(const Type& searchItem)
{
    nodeType<Type> *current; //pointer to traverse the list
    bool found;

    current = first; //set current to point to the
                    //first node in the list
    found = false;   //set found to false

    while(current != NULL && !found)    //search the list
        if(current->info == searchItem) //the item is found
            found = true;
        else
            current = current->link; //make current point
                                    //to the next node

    return found;
} //end search
```

يتوقف عدد المرات التي يتم بها تنفيذ الحلقة while في الدالة search على مكان وجود عنصر البحث في القائمة. افترض أن القائمة بها عدد n من العناصر. اذا لم يكن عنصر البحث موجود في القائمة سوف يتم تنفيذ الحلقة while عدد n من المرات. من الجهة الأخرى اذا كان عنصر البحث هو العنصر الأول سوف يتم تنفيذ الحلقة while مرة واحدة وبالمثل اذا كان عنصر البحث عو ال عنصر i في القائمة فانه يتم تنفيذ الحلقة while عدد i مرات. يمكننا من تلك الملاحظات توضيح أن دالة البحث من النوع $O(n)$. نقوم بتحليل حل البحث المتتالي بوضوح في الفصل 9.

ادخال عقدة أولى:

تقوم الدالة insertFirst بادخال العنصر الجديد في بداية القائمة – أي قبل العقدة المشار اليها بواسطة first. هناك حاجة الى الخطوات التالية لاستخدام هذه الدالة:

1. اصنع عقدة جديدة.
2. اذا لم تستطع عمل عقدة جديدة قم بايقاف البرنامج.
3. قم بتخزين العنصر الجديد في العقدة الجديدة.
4. ادخل العقدة قبل first.
5. قم بزيادة count بمقدار 1.

تعريف الدالة insertFirst هو:

```
template<class Type>
void linkedListType<Type>::insertFirst(const Type& newItem)
{
    nodeType<Type> *newNode; //pointer to create the new
                             //node

    newNode = new nodeType<Type>; //create the new node

    assert(newNode != NULL); //if unable to allocate
                             //memory, terminate
                             //the program

    newNode->info = newItem; //store newItem in
                             //the node

    newNode->link = first;   //insert newNode before
                             //first
    first = newNode;        //make first point to the
                             //actual first node
    count++;                //increment count

    if(last == NULL) //if the list was empty, newNode is
                     //also the last node in the list
        last = newNode;
}
```

ادخال عقدة أخيرة:

تعريف دالة العنصر insertLast مماثل لتعريف دالة العنصر insertFirst. هنا نقوم بادخال عقدة جديدة بعد last. في البداية تكون الدالة insertLast كما يلي:

```
template<class Type>
void linkedListType<Type>::insertLast(const Type& newItem)
{
    nodeType<Type> *newNode; //pointer to create the new node

    newNode = new nodeType<Type>; //create the new node

    assert(newNode != NULL); //If unable to allocate memory,
                             //terminate the program

    newNode->info = newItem; //store the new item in the node
    newNode->link = NULL;    //set the link field of newNode
                             //to NULL

    if(first == NULL) //if the list is empty, newNode is
                     //both the first and last node
    {
        first = newNode;
        last = newNode;
        count++; //increment count
    }
    else //the list is not empty, insert newNode after last
    {
        last->link = newNode; //insert newNode after last
        last = newNode; //make last point to the actual last node
        count++; //increment count
    }
} //end insertLast
```

من تعريفات الدالات insertFirst و insertLast ينتج أن كل من هذه الدالات من النوع $O(1)$.
حذف عقدة:

بعد هذا نناقش تطبيق الدالة deleteNode التي تقوم بحذف عقدة من القائمة ذات معلومات محددة.
نحتاج الى النظر الى عدة حالات:

الحالة 1: القائمة فارغة.

الحالة 2: العقدة الأولى هي العقدة ذات المعلومات المحددة. في هذه الحالة نحتاج الى تعديل المؤشر first.

الحالة 3: العقدة ذات المعلومات المحددة في مكان ما بالقائمة. اذا كانت القائمة الذي يتم حذفها هي العقدة الأخيرة يجب أن نقوم بتعديل المؤشر last.

الحالة 4: القائمة لا تحتوي على العقدة ذات المعلومات المحددة.

اذا كانت القائمة فارغة يمكننا ببساطة طباعة رسالة تشير الى أن القائمة فارغة. اذا لم تكن القائمة فارغة نبحث عن العقدة ذات المعلومات المعطاة واذا تم العثور على هذه العقدة نحذف هذه العقدة. بعد حذف العقدة يتم تقليل count بمقدار 1. الحل الحسابي يكون:

اذا كانت القائمة فارغة

المخرجات (لا يمكن الحذف من قائمة فارغة)،

بطريقة أخرى

}

اذا كانت العقدة الأولى هي العقدة ذات المعلومات المعطاة

اضبط first و last (اذا لزم الأمر)

و count وأزل تخصيص الذاكرة،

بطريقة أخرى

}

ابحث في القائمة عن العقدة ذات المعلومات المعطاة

اذا تم العثور على هذه العقدة احذفها واضبط قيم

Last (اذا لزم الأمر) و count.

{

{

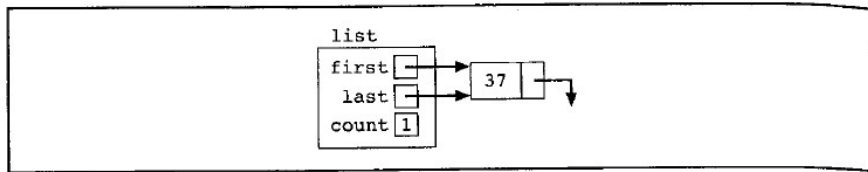
لنقم بعد ذلك بتوضيح هذه الحالات:

الحالة 1: القائمة فارغة.

اذا كانت القائمة فارغة اخراج رسالة خطأ كما هو موضح في الكود اللاحق.

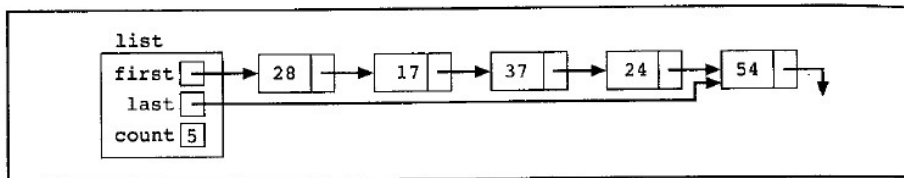
الحالة 2: اذا كانت القائمة غير فارغة تكون العقدة التي يتم حذفها هي العقدة الأولى.

هذه الحالة لها شكلان: قائمة بها عقدة واحدة فقط وقائمة بها أكثر من عقدة. انظر الى القائمة ذات العقدة الواحدة الموضحة في شكل 5-27.



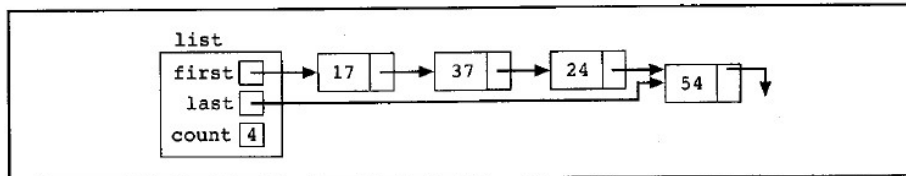
شكل 5-27: قائمة ذات عقدة واحدة.

افترض أننا نريد حذف 37. بعد الحذف تصبح القائمة فارغة. لهذا بعد الحذف يتم تحديد first و last عند القيمة NULL والمؤشر count يتم تحديده عند صفر. انظر الآن الى القائمة المحتوية على أكثر من عقدة كما هو موضح في شكل 5-28.



شكل 5-28: قائمة بها أكثر من عقدة واحدة.

افترض أن العقدة التي يتم حذفها هي 28. بعد حذف هذه العقدة تصبح العقدة الثانية العقدة الأولى. لهذا بعد حذف هذه العقدة تتغير قيمة المؤشر first أي أن first بعد الحذف تحتوي على عنوان العقدة ذات المعلومات 17 ويتم تقليل count بمقدار 1. شكل 5-29 يوضح القائمة بعد حذف 28.

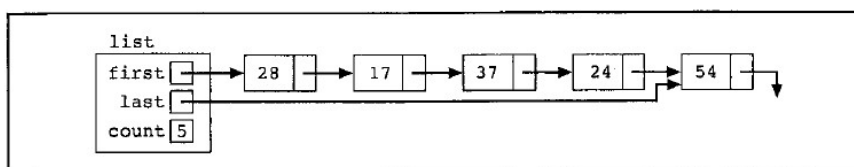


شكل 5-29: القائمة بعد حذف العقدة ذات المعلومات 28.

الحالة 3: العقدة التي يتم حذفها ليست العقدة الأخيرة ولكنها موجودة في مكان ما بهذه القائمة. هذه الحالة لها حالتين فرعيتين: (أ) العقدة التي يام حذفها ليست العقدة الأخيرة و(ب) العقدة التي يتم حذفها هي العقدة الأخيرة. انقم بتوضيح كلتا الحالتين.

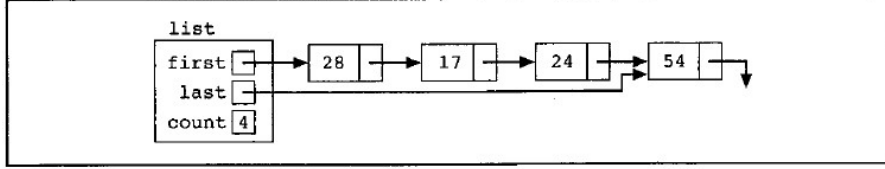
الحالة 3أ: العقدة التي يتم حذفها ليست العقدة الأخيرة.

انظر الى القائمة الموضحة في شكل 5-30.



شكل 5 - 30: القائمة قبل حذف 37

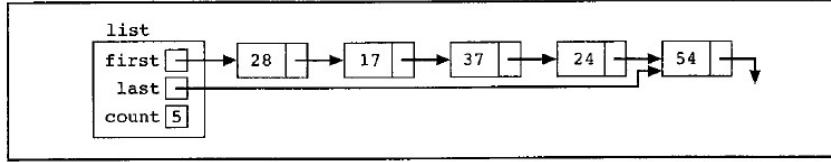
افترض أن العقدة التي يتم حذفها هي 37. بعد حذف هذه العقدة تكون القائمة الناتجة هي الموضحة في شكل 5-31. (لاحظ أن حذف 37 لا يحتاج منا تغيير قيم first و last. حقل الوصلة للعقدة السابقة - أي 17 - يتغير. بعد الحذف تحتوي العقدة ذات المعلومات 17 على عنوان العقدة ذات 24).



شكل 5-31: القائمة بعد حذف 37

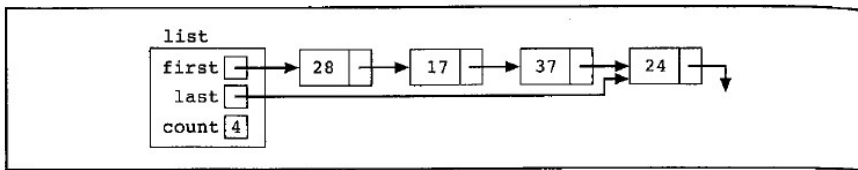
الحالة 3ب: العقدة التي يتم حذفها هي العقدة الأخيرة.

انظر الى القائمة الموضحة في شكل 5-32. افترض أن العقدة التي يتم حذفها هي 54.



شكل 5-32: القائمة قبل حذف 54

بعد حذف 54 تصبح القائمة ذات المعلومات 24 العقدة الأخيرة. لهذا يتطلب منا حذف 54 تغيير قيمة المؤشر last. بعد حذف 54 تحتوي last على عنوان العقدة ذات المعلومات 24. كما يتم تقليل count بمقدار 1. شكل 5-33 يوضح القائمة الناتجة.



شكل 5-33: القائمة بعد حذف 54

الحالة 4: العقدة التي يتم حذفها غير موجودة في القائمة. في هذه الحالة لا تحتاج القائمة الى أي تعديل. نقوم ببساطة باخراج رسالة خطأ تشير الى أن العنصر الذي يراد حذفه غير موجود بالقائمة. من الحالات 2 و 3 و 4 ينتج أن حذف عقدة ما يتطلب منا اجتياز القائمة. بما أن القائمة المتصلة ليست بنية بيانات وصول عشوائي اذن يجب علينا بحث القائمة بشكل متتالي. (اننا نتعامل مع الحالة 1 بشكل منفصل لأنها لا تتطلب منا اجتياز القائمة).

اننا نبحث القائمة بالتتابع بدءاً من العقدة الثانية. اذا كانت العقدة التي يتم حذفها موجودة في منتصف القائمة فاننا نحتاج الى تعديل حقل الوصلة للعقدة الموجود أمام العقدة التي يتم حذفها. لهذا نحتاج الى

مؤشر الى العقدة السابقة. عندما نقوم ببحث القائمة عن المعلومات المعطاة نستخدم مؤشرين: مؤشر للتحقق من معلومات العقدة الحالية ومؤشر لتتبع العقدة الموجودة قبل العقدة الحالية. اذا كانت العقدة التي يتم حذفها هي العقدة الأخيرة يجب أن نقوم بتعديل المؤشر last.

تعريف الدالة deleteNode هو:

```
template<class Type>
void linkedListType<Type>::deleteNode(const Type& deleteItem)
{
    nodeType<Type> *current; //pointer to traverse the list
    nodeType<Type> *trailCurrent; //pointer just before current
    bool found;

    if(first == NULL) //Case 1; list is empty.
        cerr<<"Cannot delete from an empty list.\n";
    else
    {
        if(first->info == deleteItem) //Case 2
        {
            current = first;
            first = first->link;

            if(first == NULL) //list has only one node
                last = NULL;
            delete current;
        }
        else //search the list for the node with the given info
        {
            found = false;
            trailCurrent = first; //set trailCurrent to point to
                                //the first node
            current = first->link; //set current to point to the
                                //second node

            while(current != NULL && !found)
            {
                if(current->info != deleteItem)
                {
                    trailCurrent = current;
                    current = current->link;
                }
                else
                    found = true;
            } // end while

            if(found) //Case 3; if found, delete the node
            {
                trailCurrent->link = current->link;
                count--;

                if(last == current) //node to be deleted was
                                    //the last node
                    last = trailCurrent; //update the value of last

                delete current; //delete the node from the list
            }
            else
                cout<<"Item to be deleted is not in the list."<<endl;
        } //end else
    } //end deleteNode
}
```



```

        //copy the remaining list
while(current != NULL)
{
    newNode = new nodeType<Type>; //create a node

    assert(newNode!= NULL);

    newNode->info = current->info; //copy the info
    newNode->link = NULL;          //set the link of
                                   //newNode to NULL
    last->link = newNode;          //attach newNode after last
    last = newNode;               //make last point to
                                   //the actual last node
    current = current->link;       //make current point to
                                   //the next node
} //end while
} //end else
} //end copyList

```

تحتوي الدالة copyList على حلقة loop. عدد المرات التي يتم بها تنفيذ الحلقة while يعتمد على عدد العناصر الموجودة في القائمة. اذا احتوت القائمة على عدد n من العناصر يتم تنفيذ الحلقة while عدد n مرات. يمكن من هذا توضيح أن الدالة copyList من النوع $O(n)$.

المدمر:

الغرض من المدمر هو ازالة تخصيص الذاكرة التي تحتلها عقد القائمة عند خروج هدف الفئة من المجال. بما أن الذاكرة مخصصة حركياً فان اعادة ضبط المؤشرين first و last لا يزيل تخصيص الذاكرة التي تشغلها العقد في القائمة. يجب أن نجتاز القائمة بدءاً من العقدة الأولى وحذف كل عقدة في القائمة. لتدمير القائمة يمكنك استدعاء الدالة destroyList ولهذا يكون تعريف المدمر هو:

```

template<class Type>
linkedListType<Type>::~~linkedListType() //destructor
{
    destroyList();
} //end destructor

```

مقوم النسخ:

بما أن الفئة linkedListType تحتوي على عناصر بيانات المؤشر فان تعريف هذه الفئة يحتوي على مقوم النسخ. تذكر أن المعامل الرسمي اذا كان معامل قيمة فان مقوم النسخ يقدم المعامل الرسمي بنسخته الخاصة من البيانات. يتم تنفيذ مقوم النسخ كذلك عندما يتم اعلان الهدف وتهيئته باستخدام هدف آخر. (لمزيد من المعلومات انظر الفصل 3).

يصنع مقوم النسخ نسخة متطابقة من القائمة المتصلة. يمكن تحقيق هذا من خلال استدعاء الدالة copyList. الآن نتحقق الدالة copyList مما اذا كانت القائمة الأصلية فارغة عن طريق التحقق من قيمة first. لهذا يجب أن نقوم بتهيئة المؤشر first عند NULL قبل استدعاء الدالة copyList.

تعريف مقوم النسخ يكون:

```
template<class Type>
linkedListType<Type>::linkedListType
(const linkedListType<Type>& otherList)
{
    first = NULL;
    copyList(otherList);
} //end copy constructor
```

اثقال معامل الاسناد:

تعريف دالة اثقال معامل الاسناد للفئة linkedListType شديد الشبه مع تعريف مقوم النسخ. اننا نعطي تعريفه هنا من أجل الاكمال.

```
template<class Type>
const linkedListType<Type>& linkedListType<Type>::operator=
(const linkedListType<Type>& otherList)
{
    if(this != &otherList) //avoid self-copy
        copyList(otherList);

    return *this;
}
```

المدمر يستخدم الدالة destroyList التي تكون من النوع $O(n)$. مقوم النسخ ودالة اثقال معامل الاسناد يستخدمان الدالة copyList التي هي من النوع $O(n)$. لهذا تكون كل من هذه الدالات من النوع $O(n)$.

الجدول 5-5 يقوم بتلخيص التركيب الزمني لعمليات الفئة linkedListType.
جدول 5-5: التركيب الزمني لعمليات الفئة linkedListType.

| الدالة | التركيب الزمني |
|--------------------------|----------------|
| isEmptyList | $O(1)$ |
| المقوم الافتراضي | $O(1)$ |
| destroyList | $O(n)$ |
| initializaList | $O(n)$ |
| اثقال معامل ادخال التدفق | $O(n)$ |
| length | $O(1)$ |
| front | $O(1)$ |
| back | $O(1)$ |
| search | $O(n)$ |
| insertFirst | $O(1)$ |
| insertLast | $O(1)$ |
| deleteNode | $O(n)$ |
| copyList | $O(n)$ |
| المدمر | $O(n)$ |
| مقوم النسخ | $O(n)$ |
| اثقال معامل الاسناد | $O(n)$ |

عندما تصنع الملف الرئيسي للفئة `linkedListType` يجب أن تقوم بتضمين البيانات التالية قبل تعريف الفئة:

```
#include <iostream>
#include <cassert>
using namespace std;
```

قوائم متصلة مرتبة:

الآن بعد حصولك على فكرة عن كيفية استخدام متغيرات المؤشر في C++ لتخصيص وإزالة تخصيص الذاكرة حركياً وكيفية بناء ومعالجة القوائم المتصلة يقوم هذا الفصل بمناقشة كيفية بناء قوائم متصلة مرتبة ووصف العمليات المتنوعة على تلك القوائم. العمليات التالية يتم إجراؤها عادةً على القائمة المرتبة:

1. تهيئة القائمة.
2. التحقق مما إذا كانت القائمة فارغة.
3. اخراج القائمة.
4. اخراج القائمة بترتيب عكسي.
5. تدمير القائمة.
6. بحث القائمة من أجل عنصر محدد.
7. ادخال عنصر في القائمة.
8. حذف عنصر من القائمة.
9. ايجاد طول القائمة.
10. عمل نسخة من القائمة.

العديد من العمليات المؤداة على قوائم متصلة مرتبة تكون مشابهة للعمليات المؤداة على القوائم العامة التي تمت مناقشتها في القسم الأخير. بما أن القائمة مرتبة نحتاج فقط إلى تعديا خوارزميات تطبيق عمليات البحث والادخال والحذف. لهذا نشق الفئة لتعريف القائمة المتصلة كنوع بيانات مجرد من الفئة `linkedListType`.

لاخراج البيانات المخزنة في كل عقدة بترتيب عكسي تحتاج القائمة إلى أن يتم اجتيازها بالترتيب العكسي بدءاً من العقدة الأخيرة. بما أن الوصلات تتواجد في اتجاه واحد فقط فإنه لا يمكننا اجتياز القائمة إلى الخلف باستخدام الوصلات. نقوم في الفصل 6 بتوضيح كيفية استخدام التكرار لاجتياز قائمة متصلة إلى الخلف.

الفئة التالية تقوم بتعريف قائمة متصلة مرتبة كنوع بيانات مجرد.

```

template<class Type>
class orderedLinkedListType: public linkedListType<Type>
{
public:
    bool search(const Type& searchItem);
        //Function to determine whether searchItem is in the list.
        //Postcondition: Returns true if searchItem is found in
        //                the list; otherwise, it returns false.

    void insertNode(const Type& newItem);
        //Function to insert newItem in the list.
        //Postcondition: first points to the new list and newItem is
        //                inserted at the proper place in the list, and
        //                count is incremented by 1.

    void deleteNode(const Type& deleteItem);
        //Function to delete deleteItem from the list.
        //Postcondition: If found, the node containing deleteItem
        //                is deleted from the list; first points
        //                to the first node of the new list, and
        //                count is decremented by 1.
        //                If deleteItem is not in the list, an
        //                appropriate message is printed.
};

```

الفئة `orderedLinkedListType` مشتقة من الفئة `linkedListType`. تقدم الفئة `linkedListType` الدالة `insertFirst` لإدخال عنصر في بداية القائمة والدالة `insertLast` لإدخال عنصر في نهاية القائمة. بالرغم من هذا نقوم الآن بمناقشة القوائم المتصلة المرتبة التي يتم فيها ترتيب العناصر وفقاً لبعض المقاييس. إذا كان هناك عنصر يتم إدخاله في قائمة مرتبة يجب إدخاله في المكان الصحيح. بالرغم من أنه يمكنك استخدام الدالة `insertFirst` و `insertLast` لإدخال عنصر في البداية أو النهاية إلا أنه لا يوجد ضمان أن القائمة الناتجة سوف تكون مرتبة. لهذا لن نستخدم تلك الدالات مع القوائم المتصلة المرتبة. فضلاً عن هذا وبما أن العناصر سوف يتم إدخالها في المكان الصحيح كما يتم التوضيح في القسم القادم "إدخال عقدة" فإن المؤشر `last` لا يلعب دوراً في بناء قوائم مرتبة. لهذا نقوم بإهمال المؤشر `last` الذي يتم ضبطه عند `NULL` ونستخدم المؤشر `first` فقط. كما أن الدالة `back` من الفئة `linkedListType` تستخدم المؤشر `Last` لإنتاج العنصر الأخير من القائمة. بما أن المؤشر `last` لا يستخدم بالنسبة إلى القائمة المتصلة المرتبة (إلا إذا قمت بإلغاء تعريف هذه الدالة للفئة `orderedLinkedListType`، انظر تمرين البرمجة 8).

بحث القائمة:

أولاً نناقش عملية البحث. الحل الحسابي لتطبيق عملية البحث مماثل للبحث بالنسبة إلى القوائم العامة التي تمت مناقشتها في القسم "القائمة المتصلة كنوع بيانات مجرد" الموجود سابقاً في هذا الفصل. هنا بما أن القائمة مرتبة يمكننا تحسين الحل الحسابي للبحث نوعاً ما. كما حدث من قبل نبدأ البحث عند

العقدة الأولى في القائمة ونوقف البحث بمجرد إيجاد عقدة في القائمة لها معلومات أكبر من أو مساوية لعنصر البحث أو بمجرد إجرائنا للبحث في القائمة بأكملها.

هذا الحل الحسابي يتكون من الخطوات التالية:

1. قارن عنصر البحث مع العقدة الحالية في القائمة. إذا كانت معلومات العقدة التالية أكبر من أو مساوية لعنصر البحث قم بإيقاف البحث وبخلاف هذا اجعل العقدة التالية العقدة الحالية.
 2. كرر الخطوة الأولى حتى يتم العثور على عنصر في القائمة أكبر من أو مساوي لعنصر البحث أو حتى لا يعد هناك بيانات أخرى في القائمة لمقارنتها مع عنصر البحث.
- لاحظ أن الحلقة لا تتحقق بوضوح مما إذا كان عنصر البحث مساوي لعنصر ما في القائمة أم لا. لهذا بعد تنفيذ الحلقة يجب أن نتحقق مما إذا كان عنصر البحث مساوي للعنصر في القائمة.

```
template<class Type>
bool orderedLinkedListType<Type>::search(const Type& searchItem)
{
    bool found;
    nodeType<Type> *current; //pointer to traverse the list

    found = false;    //initialize found to false
    current = first;  //start the search at the first node

    while(current != NULL && !found)
        if(current->info >= searchItem)
            found = true;
        else
            current = current->link;

    if(found)
        found = (current->info == searchItem); //test for equality

    return found;
} //end search
```

ادخال عقدة:

لادخال عنصر في قائمة متصلة مرتبة نجد أولاً المكان الذي من المفترض ذهاب العنصر الجديد اليه ثم نقوم بادخاله في القائمة. لايجاد المكان المناسب للعنصر الجديد في القائمة نقوم كما قمنا من قبل ببحث القائمة. هنا نقوم باستخدام مؤشرين: `current` و `trailCurrent` لبحث القائمة. المؤشر `current` يشير الى العقدة التي تتم مقارنة معلوماتها مع العنصر الذي يتم ادخاله والمؤشر `trailCurrent` يشير الى العقدة الواقعة قبل `current` مباشرة. بما أن القائمة مرتبة يكون الحل الحسابي للبحث هو نفسه كما سبق. تنشأ الحالات التالية:

الحالة 1: القائمة فارغة مبدئياً. العقدة المحتوية على العنصر الجديد هي العقدة الوحيدة ولهذا تكون العقدة الأولى في القائمة.

الحالة 2: القائمة غير فارغة والعنصر الجديد أصغر من أصغر عنصر في القائمة. يذهب العنصر الجديد في بداية القائمة. في هذه الحالة نحتاج الى ضبط مؤشر رأس القائمة – أي المؤشر first. وكذلك يتم تقليل count بمقدار 1.

الحالة 3: القائمة غير فارغة والعنصر الذي يتم ادخاله أكبر من العنصر الأول في القائمة. يتم ادخال العنصر في مكان ما بالقائمة.

الحالة 3أ: العنصر الجديد أكبر من جميع عناصر القائمة. في هذه الحالة يتم ادخال العنصر الجديد في نهاية القائمة. لهذا تكون قيمة current هي NULL ويتم ادخال العنصر الجديد بعد trailCurrent. وكذلك يتم تقليل count بمقدار 1.

الحالة 3ب: يتم ادخال العنصر الجديد في مكان ما في منتصف القائمة. في هذه الحالة يتم ادخال العنصر الجديد بين trailCurrent وcurrent. ويتم كذلك تقليل count بمقدار 1.

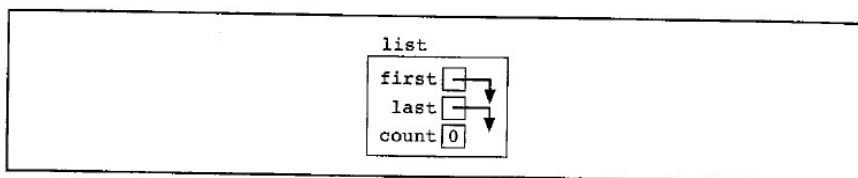
البيانات التالية يمكنها تحقيق كلتا الحالتين 3أ و3ب (افترض أن newNode يشير الى العقدة الجديدة):

```
trailCurrent->link = newNode;
newNode->link = current;
```

لنقم بعد هذا بتوضيح هذه الحالات.

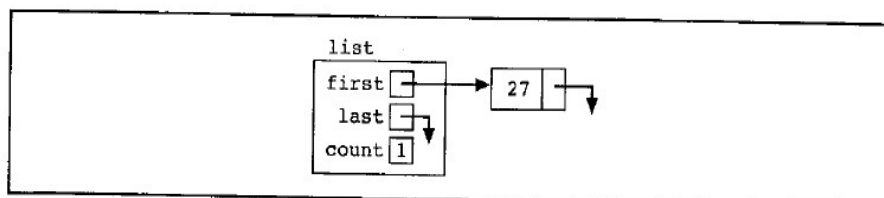
الحالة 1: القائمة فارغة.

انظر الى القائمة الموضحة في شكل 5-34.



شكل 5-34: قائمة فارغة.

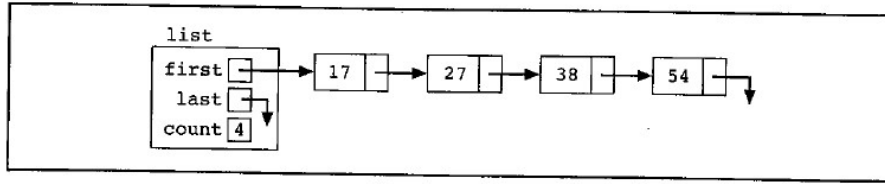
افترض أننا نريد ادخال 27 في القائمة. لانجاز هذه المهمة نصنع عقدة ونقوم بنسخ 27 داخل العقدة ونقوم بتحديد وصلة العقدة عند NULL ونجعل first يشير الى العقدة. الشكل 5-35 يوضح القائمة الناتجة.



شكل 5-35: القائمة بعد ادخال 27.

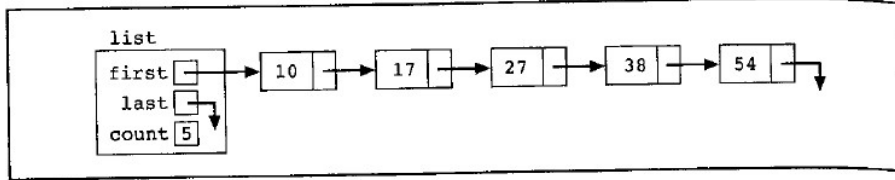
لاحظ أن بعد ادخال 27 تتغير قيم first وcount.

الحالة 2: القائمة غير فارغة والعنصر الذي يتم ادخاله أصغر من أصغر عنصر في القائمة. انظر الى القائمة الموضحة في شكل 5-36.



شكل 5-36: قائمة غير فارغة قبل ادخال 10.

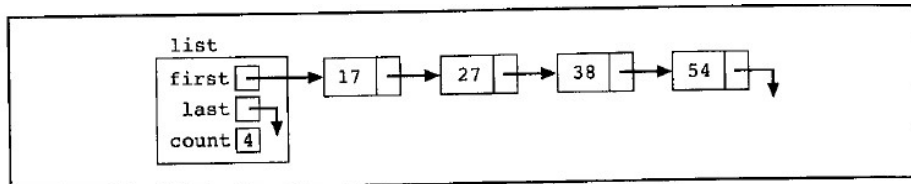
افترض أنه يتم ادخال 10. بعد ادخال 10 في القائمة تصبح العقدة ذات المعلومات 10 العقدة الأولى من القائمة. هذا يتطلب منا تغيير قيمة first. كما يتم زيادة count بمقدار 1. الشكل 5-37 يوضح القائمة الناتجة.



شكل 5-37: القائمة بعد ادخال 10.

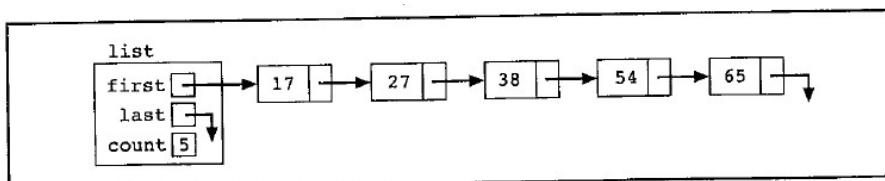
الحالة 3: القائمة غير فارغة والعنصر الذي يتم ادخاله أكبر من العنصر الأول في القائمة. كما وضح من قبل هذه الحالة لها شكلين.

الحالة 3أ: العنصر الذي يتم ادخاله أكبر من أكبر عنصر في القائمة أي أنه يذهب الى نهاية القائمة. انظر القائمة الموضحة في شكل 5-38.



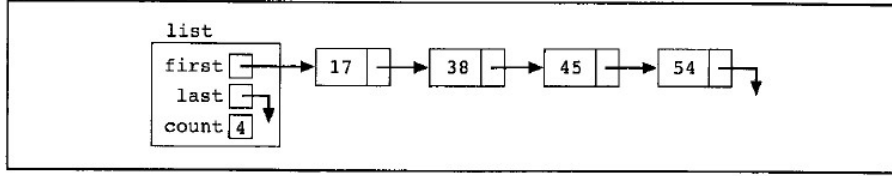
شكل 5-38: القائمة قبل ادخال 65.

افترض أننا نريد ادخال 65 في القائمة. بعد ادخال 65 تكون القائمة الناتجة كما هو موضح في شكل 5-39.



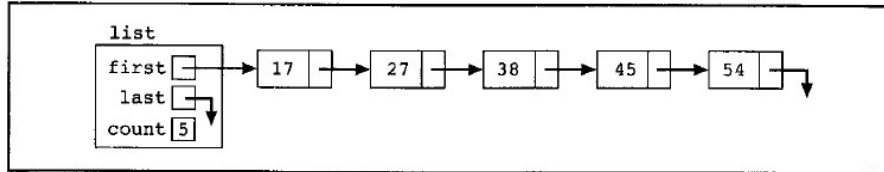
شكل 5-39: القائمة بعد ادخال 65.

الحالة 3ب: العنصر الذي يتم ادخاله يذهب الى مكان ما في منتصف القائمة. انظر الى القائمة الموضحة في شكل 5-40.



شكل 5-40: القائمة قبل ادخال 27.

افترض أننا نريد ادخال 27 في هذه القائمة. بشكل واضح تذهب 27 بين 17 و 38 وهذا يتطلب أن يتم تغيير وصلة العقدة ذات المعلومات 17. بعد ادخال 27 تكون القائمة الناتجة كما هو موضح في شكل 5-41.



شكل 5-41: القائمة بعد ادخال 27.

بالنسبة الى الحالة 3 يجب أن نقوم أولاً باجتياز القائمة للعثور على المكان الذي يتم ادخال العنصر الجديد فيه. كما وضح من قبل يجب أن نجتاز القائمة بمؤشرين – مثل current و trailCurrent. المؤشر current يعبر القائمة ويقارن معلومات العقدة في القائمة مع العنصر الذي يتم ادخاله. المؤشر trailCurrent يشير الى العقدة الواقعة مباشرة قبل current. على سبيل المثال في الحالة 3ب عندما يتوقف البحث تشير trailCurrent الى العقدة 17 وتشير current الى العقدة 38 ويتم ادخال العنصر بعد trailCurrent. في الحالة 3أ بعد بحث القائمة للعثور على مكان من أجل 65 يشير trailCurrent الى العقدة 54 ويشير current الى NULL.

مبدئياً تكون الدالة insertNode كما يلي:

```
template<class Type>
void orderedLinkedListType<Type>::insertNode(const Type& newitem)
{
    nodeType<Type> *current;           //pointer to traverse the list
    nodeType<Type> *trailCurrent;       //pointer just before current
    nodeType<Type> *newNode;           //pointer to create a node

    bool found;
```

```

newNode = new nodeType<Type>; //create the node
assert(newNode != NULL);

newNode->info = newitem;    //store newitem in the node
newNode->link = NULL;       //set the link field of the node
                             //to NULL

if(first == NULL) //Case 1
{
    first = newNode;
    count++;
}
else
{
    current = first;
    found = false;

    while(current != NULL && !found) //search the list
        if(current->info >= newitem)
            found = true;
        else
        {
            trailCurrent = current;
            current = current->link;
        }

    if(current == first) //Case 2
    {
        newNode->link = first;
        first = newNode;
        count++;
    }
    else //Case 3
    {
        trailCurrent->link = newNode;
        newNode->link = current;
        count++;
    }
} //end else
} //end insertNode

```

الدالة insertNode لا تتحقق مما اذا كان العنصر الذي يتم ادخاله موجود بالفعل في القائمة أي أنها لا تتحقق من المكررات. تمرين البرمجة رقم 7 في نهاية هذا الفصل يطلب منك مراجعة تعريف الدالة insertNode لذلك قبل ادخال العنصر تحقق مما اذا كان موجود بالفعل في القائمة أم لا. اذا كان العنصر الذي يتم ادخاله موجود بالفعل في القائمة تقوم الدالة باخراج رسالة خطأ مناسبة أي أن المكررات غير مسموح بها.

حذف عقدة:

لحذف عنصر محدد من قائمة متصلة مرتبة نقوم أولاً ببحث القائمة لرؤية ما اذا كان العنصر الذي يتم حذفه موجود في القائمة أم لا. الدالة لتطبيق هذه العملية مماثلة لعملية الحذف المؤداة على القوائم المتصلة العامة بما أن القائمة هنا مرتبة يمكننا تحسين الحل الحسابي نوعاً ما للقوائم المتصلة المرتبة. كما في حالة insertNode نبحث القائمة بمعاملين هما current و trialCurrent. كما هو الحال في العملية insertNode تنشأ عدة حالات:

الحالة 1: القائمة فارغة مبدئياً. يكون لدينا خطأ ولا يمكننا الحذف من قائمة فارغة.

الحالة 2: العنصر الذي يتم حذفه موجود في العقدة الأولى من القائمة. يجب أن نقوم بتعديل المؤشر الرأسي للقائمة – أي first.

الحالة 3: العنصر الذي يتم حذفه موجود في مكان ما في القائمة. في هذه الحالة يشير current الى العقدة المحتوية على العنصر الذي يتم حذفه ويشير trailCurrent الى العقدة الواقعة قبل العقدة المشار اليها بواسطة current مباشرةً.

الحالة 4: القائمة غير فارغة ولكن العنصر الذي يتم حذفه غير موجود في القائمة.

بعد حذف العقدة يتم تقليل count بمقدار 1. تعريف الدالة deleteNode هو:

```
template<class Type>
void orderedLinkedListType<Type>::deleteNode
    (const Type& deleteItem)
{
    nodeType<Type> *current;      //pointer to traverse the list
    nodeType<Type> *trailCurrent; //pointer just before current
    bool found;

    if(first == NULL) //Case 1
        cerr<<"Cannot delete from an empty list."<<endl;
    else
    {
        current = first;
        found = false;

        while(current != NULL && !found) //search the list
            if(current->info == deleteItem)
                found = true;
            else
            {
                trailCurrent = current;
                current = current->link;
            }

        if(current == NULL) //Case 4
            cout<<"The item to be deleted is not in the list."
                <<endl;
```

```

else
    if(current->info == deleteItem) //item to be deleted
                                    //is in the list
    {
        if(first == current)        //Case 2
        {
            first = first->link;

            delete current;
        }
        else                          //Case 3
        {
            trailCurrent->link = current->link;
            delete current;
        }
        count--;
    }
    else                             //Case 4
        cout<<"The item to be deleted is not in the list."
            <<endl;
}
} //end deleteNode

```

الملف الرئيسي للقائمة المتصلة المرتبة:

من أجل الاكمال نوضح الآن كيفية عمل الملف الرئيسي الذي يقوم بتعريف الفئة `orderedLinkedListType` والعمليات المؤداة على هذه القوائم. (نفترض أن تعريف الفئة `linkedListItemType` وتعريفات الدالات لتطبيق العمليات موجودة في الملف الرئيسي `linkedlist.h`.)

// قائمة متصلة مرتبة مشتقة من قائمة متصلة عامة

// الملف الرئيسي: `orderedLinkedList.h`

```

#ifndef H_orderedLinkedListType
#define H_orderedLinkedListType

#include <iostream>
#include <cassert>
#include "linkedlist.h"

using namespace std;

template<class Type>
class orderedLinkedListType: public linkedListItemType<Type>
{
public:
    bool search(const Type& searchItem);
    void insertNode(const Type& newItem);
    void deleteNode(const Type& deleteItem);
};

```


تنفيذ العينة: في تنفيذ هذه العينة يتم تظليل مدخلات المستخدم.

الصف 3: ادخل الأعداد الصحيحة المنتهية ب-999

23 65 34 72 12 82 36 55 29 -999

الصف 9: القائمة 1: 12 23 29 34 36 55 65 72 82

الصف 11: القائمة 2: 12 23 29 34 36 55 65 72 82

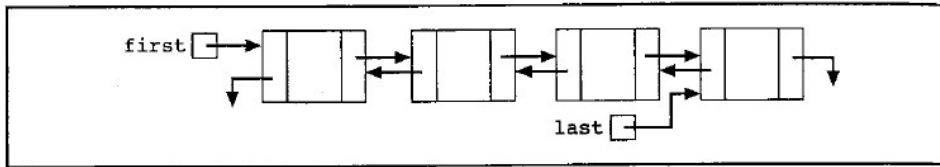
الصف 12: ادخل الرقم الذي يتم حذفه: 36

الصف 16: بعد حذف العقدة تكون القائمة 2:

12 23 29 34 55 65 72 82

القوائم المتصلة من الطرفين:

القائمة المتصلة من الطرفين عبارة عن قائمة متصلة تحتوي فيها كل عقدة على مؤشر next ومؤشر back. بمعنى آخر تحتوي كل عقدة (فيما عدا العقدة الأخيرة) على عنوان العقدة التالية وتحتوي كل عقدة (فيما عدا العقدة الأولى) على عنوان العقدة السابقة. انظر شكل 5-42.



شكل 5-42: قائمة متصلة من الطرفين

يمكن اجتياز القائمة المتصلة من الطرفين في أي من الاتجاهين. أي يمكننا اجتياز القائمة بدءاً من العقدة الأولى أو اذا تم اعطاء مؤشر الى العقدة الأخيرة يمكننا اجتياز القائمة بدءاً من العقدة الأخيرة. كما قيل من قبل فان العمليات المعتادة المؤداة على القائمة المتصلة من الطرفين هي:

1. تهيئة القائمة.
2. تدمير القائمة.
3. تحديد ما اذا كانت القائمة فارغة أم لا.
4. بحث القائمة عن عنصر محدد.
5. استعادة العنصر الأول من القائمة.
6. استعادة العنصر الأخير من القائمة.
7. ادخال عنصر في القائمة.
8. حذف عنصر من القائمة.
9. ايجاد طول القائمة.
10. اخراج القائمة.
11. عمل نسخة من القائمة.

الفئة التالية تقوم بتعريف قائمة متصلة من الطرفين كنوع بيانات مجرد:

```
//Definition of the node
template<class Type>
struct nodeType
{
    Type info;
    nodeType<Type> *next;
    nodeType<Type> *back;
};

template <class Type>
class doublyLinkedList
{
    friend ostream& operator<<(ostream&,
                               const doublyLinkedList<Type>&);
    //Overload the stream insertion operator.
public:
    const doublyLinkedList<Type>& operator=
        (const doublyLinkedList<Type> &);
    //Overload the assignment operator.

    void initializeList();
    //Function to initialize the list to an empty state.
    //Postcondition: first = NULL; last = NULL; count = 0

    bool isEmptyList();
    //Function to determine whether the list is empty.
    //Postcondition: Returns true if the list is empty;
    //                otherwise, returns false.

    void destroy();
    //Function to delete all the nodes from the list.
    //Postcondition: first = NULL; last = NULL; count = 0

    void reversePrint();
    //Function to output the info contained in each node
    //in reverse order.

    int length();
    //Function to return the number of nodes in the list.
    //Postcondition: The value of count is returned.

    Type front();
    //Function to return the first element of the list.
    //Precondition: The list must exist and must not be empty.
```

```

    //Postcondition: If the list is empty, the program
    //                terminates; otherwise, the first
    //                element of the list is returned.

Type back();
    //Function to return the last element of the list.
    //Precondition: The list must exist and must not be empty.
    //Postcondition: If the list is empty, the program
    //                terminates; otherwise, the last
    //                element of the list is returned.

bool search(const Type& searchItem);
    //Function to determine whether searchItem is in the list.
    //Postcondition: Returns true if searchItem is found
    //                in the list; otherwise, returns false.

void insertNode(const Type& insertItem);
    //Function to insert newItem in the list.
    //Precondition: If the list is nonempty, it must be in
    //                order.
    //Postcondition: newItem is inserted at the proper place
    //                in the list; first points to the first
    //                node, last points to the last node of the
    //                new list and count is incremented by 1.

void deleteNode(const Type& deleteItem);
    //Function to delete deleteItem from the list.
    //Postcondition: If found, the node containing
    //                deleteItem is deleted from the list, first points
    //                to the first node of the new list, last points to
    //                the last node of the new list, and count is
    //                decremented by 1; otherwise, an appropriate
    //                message is printed.

doublyLinkedList();
    //default constructor
    //Initializes the list to an empty state.
    //Postcondition: first = NULL; last = NULL; count = 0

doublyLinkedList(const doublyLinkedList<Type>& otherList);
    //copy constructor
~doublyLinkedList();
    //destructor
    //Postcondition: The list object is destroyed.

protected:
    int count;
    nodeType<Type> *first; //pointer to the first node
    nodeType<Type> *last;  //pointer to the last node

private:
    void copyList(const doublyLinkedList<Type>& otherList);
        //Function to make a copy of otherList.
        //Postcondition: A copy of otherList is created and
        //                assigned to this list.
};

```

دالات تطبيق عمليات القائمة المتصلة من الطرفين مماثلة لعمليات تمت مناقشتها في القسمين السابقين. هنا بما أن كل عقدة لها مؤشران back و next فان بعض العمليات تتطلب تعديل المؤشرين في كل عقدة. نقوم هنا باعطاء تعريف كل دالة مع أربع استثناءات. يتم ترك تعريفات الدالات copyList ومقوم النسخ، والمدمر، واثقال معامل الاسناد كتمارين لك. (انظر تمرين البرمجة 9 في نهاية هذا الفصل).

المقوم الافتراضي:

يقوم المقوم الافتراضي بتهيئة القائمة المتصلة من الطرفين في وضع فارغ ويحدد first و Last عند NULL ويحدد count عند صفر وتعريفه يكون:

```
template<class Type>
doublyLinkedList<Type>::doublyLinkedList()
{
    first= NULL;
    last = NULL;
    count = 0;
}
```

:isEmptyList

هذه العملية تنتج true اذا كانت القائمة فارغة وبخلاف هذا تنتج false. تكون القائمة فارغة اذا كان المؤشر first عند NULL وتعريفه يكون:

```
template<class Type>
bool doublyLinkedList<Type>::isEmptyList()
{
    return(first == NULL);
}
```

تدمير القائمة:

هذه العملية تحذف جميع العقد الموجودة في القائمة تاركة القائمة في حالة فارغة. اننا نجتاز القائمة بدءاً من العقدة الأولى ثم نحذف كل عقدة. تعريفها يكون:

```
template<class Type>
void doublyLinkedList<Type>::destroy()
{
    nodeType<Type> *temp; //pointer to delete the node

    while(first != NULL)
    {
        temp = first;
        first = first->next;
        delete temp;
    }
    last = NULL;

    count = 0;
}
```

تهيئة القائمة:

هذه العملية تعيد تهيئة القائمة المتصلة من الطرفين في حالة فارغة. يمكن انجاز هذه المهمة باستخدام العملية destroy. تعريف الدالة initializeList هو:

```
template<class Type>
void doublyLinkedList<Type>::initializeList()
{
    destroy();
}
```

طول القائمة:

طول القائمة هو عدد العقد في القائمة. وتعريفه هو:

```
template<class Type>
int doublyLinkedList<Type>::length()
{
    return count;
}
```

انقال معامل ادخال التدفق:

لاخراج البيانات المخزنة في عقد القائمة المتصلة من الطرفين يتم انقال معامل ادخال التدفق. لاجراج البيانات الموجودة في كل عقدة يجب أن نجتاز القائمة بدءاً من العقدة الأولى. بما أن المؤشر first يشير دائماً الى العقدة الأولى في القائمة نقوم باستخدام مؤشر آخر لاجتياز القائمة.

تعريف الدالة لانقال معامل ادخال التدفق هو:

```
template<class Type>
ostream& operator<<(ostream& osObject,
                    const doublyLinkedList<Type>& list)
{
    nodeType<Type> *current; //pointer to traverse the list

    current = list.first;    //set current to point to
                             //the first node

    while(current != NULL)
    {
        cout<<current->info<<" "; //output the info
        current = current->next;
    } //end while

    return osObject;
}
```

الطباعة العكسية للقائمة:

هذه الدالة تخرج المعلومات الموجودة في كل عقدة بترتيب عكسي. اننا نجتاز القائمة بترتيب عكسي بدءاً من العقدة الأخيرة وتعريفها هو:

```
template<class Type>
void doublyLinkedList<Type>::reversePrint()
{
    nodeType<Type> *current; //pointer to traverse
                           //the list

    current = last; //set current to point to the
                   //last node

    while(current != NULL)
    {
        cout<<current->info<<" ";
        current = current->back;
    } //end while
} //end reversePrint
```

بحث القائمة:

الدالة search تنتج true اذا كان عنصر البحث موجود في القائمة وبخلاف هذا تنتج false. هذا الحل الحسابي مماثل تماماً لحل البحث الحسابي للقائمة المتصلة المرتبة وتعريفها هو:

```
template<class Type>
bool doublyLinkedList<Type>::search(const Type& searchItem)
{
    bool found;
    nodeType<Type> *current; //pointer to traverse the list

    found = false;
    current = first;

    while(current != NULL && !found)
        if(current->info >= searchItem)
            found = true;
        else
            current = current->next;

    if(found)
        found = (current->info == searchItem); //test for equality

    return found;
} //end search
```

العنصر الأول والأخير:

الدالة front تنتج العنصر الأول من القائمة والدالة back تنتج العنصر الأخير. اذا كانت القائمة فارغة تقوم كلتا الدالتين بإيقاف البرنامج. تعريفاتها تكون كما يلي:

```

template<class Type>
Type doublyLinkedList<Type>::front()
{
    assert(first != NULL);

    return first->info;
}

template<class Type>
Type doublyLinkedList<Type>::back()
{
    assert(last != NULL);

    return last->info;
}

```

ادخال عقدة:

بما أننا نقوم بادخال عنصر في قائمة متصلة من الطرفين فان ادخال عقدة في القائمة يتطلب تعديل مؤشرين في عقدات محددة. كما قيل من قبل نعثر على المكان الذي يكون من المفترض ادخال العنصر الجديد فيه ثم نصنع العقدة ونقوم بتخزين العنصر الجديد وتعديل حقول وصلة العقدة الجديدة وعقدات أخرى محددة في القائمة. هناك أربع حالات:

الحالة 1: ادخال في قائمة فارغة.

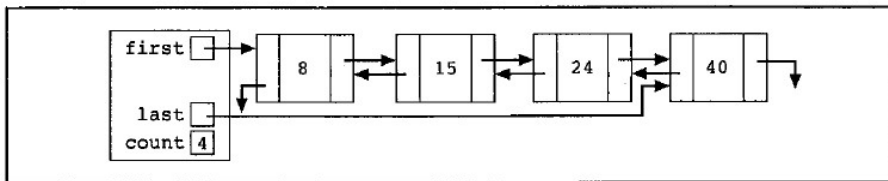
الحالة 2: الادخال في بداية قائمة غير فارغة.

الحالة 3: الادخال في نهاية قائمة غير فارغة.

الحالة 4: الادخال في مكان ما في قائمة غير فارغة.

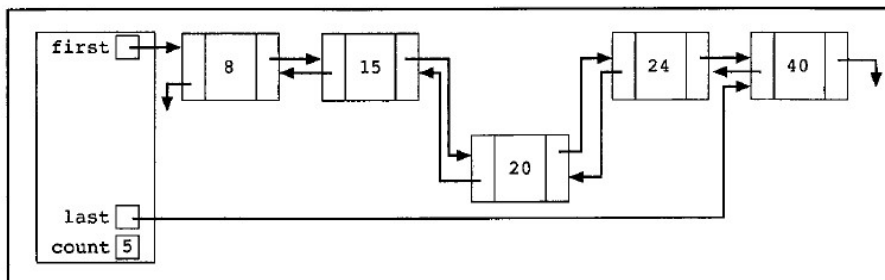
الحالتان 1 و 2 تقتضي منا تغيير قيمة المؤشر first والحالات 3 و 4 متشابهة. بعد ادخال العنصر يتم زيادة count بمقدار 1 وبعد هذا نوضح الحالة 4.

انظر الى القائمة المتصلة من الطرفين الموضحة في شكل 5-43.



شكل 5-43: قائمة متصلة من الطرفين قبل ادخال 20.

افترض أنه يتم ادخال 20 في القائمة. بعد ادخال 20 يوضح الشكل 5-44 القائمة الناتجة.



شكل 44-5: قائمة متصلة من الطرفين بعد ادخال 20.

شكل 44-5 يوضح أن هناك حاجة الى تعديل المؤشر next من العقدة 15 وكل من المؤشران next و back للعقدة 20.

تعريف الدالة insertNode هو:

```
template<class Type>
void doublyLinkedList<Type>::insertNode(const Type& insertItem)
{
    nodeType<Type> *current;           //pointer to traverse the list
    nodeType<Type> *trailCurrent;       //pointer just before current
    nodeType<Type> *newNode;           //pointer to create a node
    bool found;

    newNode = new nodeType<Type>; //create the node
    assert(newNode != NULL);

    newNode->info = insertItem; //store the new item in the node
    newNode->next = NULL;
    newNode->back = NULL;

    if(first == NULL) //if the list is empty, newNode is
                      //the only node
    {
        first = newNode;
        last = newNode;
        count++;
    }
    else
    {
        found = false;
        current = first;

        while(current != NULL && !found) //search the list
        {
            if(current->info >= insertItem)
                found = true;
            else
            {
                trailCurrent = current;
                current = current->next;
            }
        }

        if(current == first) //insert new node before first
        {
            first->back = newNode;
            newNode->next = first;
            first = newNode;
            count++;
        }
        else
        {
            //insert newNode between trailCurrent and current
            if(current != NULL)
            {
                trailCurrent->next = newNode;
                newNode->back = trailCurrent;
                newNode->next = current;
                current->back = newNode;
            }
        }
    }
}
```

```

else
{
    trailCurrent->next = newNode;
    newNode->back = trailCurrent;
    last = newNode;
}
count++;
} //end else
} //end else
} //end insertNode

```

حذف عقدة:

هذه العملية تحذف عنصر محدد (إذا وجد) من قائمة متصلة من الطرفين. كما سبق نقوم أولاً ببحث القائمة لرؤية ما إذا كان العنصر الذي يتم حذفه موجود في القائمة أم لا ويكون الحل الحسابي هو نفس الحل السابق. هذه العملية (إذا كان العنصر المحذوف في القائمة) مثلها مثل عملية insertNode تتطلب تعديل مؤشرين في عقد معينة. عملية الحذف لها العديد من الحالات:

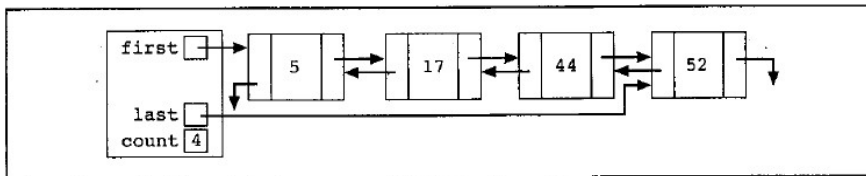
الحالة 1: القائمة فارغة.

الحالة 2: العنصر الذي يتم حذفه موجود في العقدة الأولى من القائمة وهذا يقتضي تعديل قيمة المؤشر first.

الحالة 3: العنصر الذي يتم حذفه موجود في مكان ما في القائمة.

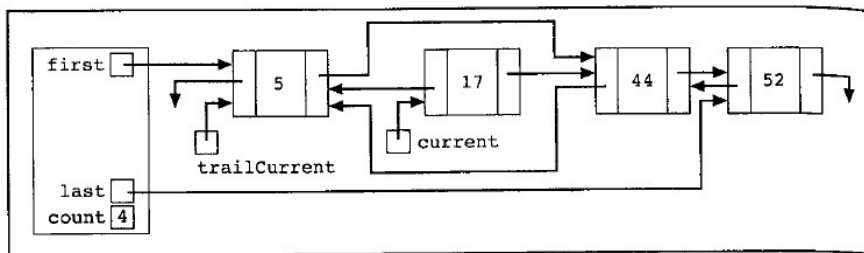
الحالة 4: العنصر الذي يتم حذفه غير موجود في القائمة.

بعد حذف العقدة يتم تقليل count بمقدار 1. لنقم بتوضيح الحالة 3. انظر الى القائمة الموضحة في شكل 5-45.

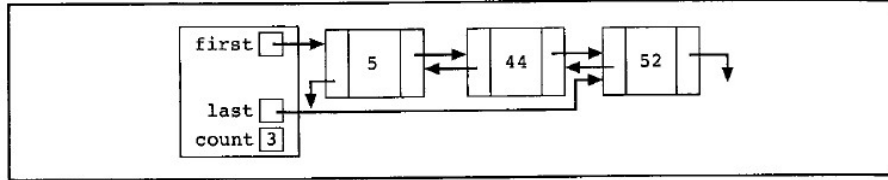


شكل 5-45: قائمة متصلة من الطرفين قبل حذف 17.

افترض أن العنصر الذي يتم حذفه هو 17. نقوم ببحث القائمة بمؤشرين وإيجاد العقدة ذات المعلومات 17 ثم نقوم بتعديل حقول الوصلة للعقد المتأثرة. انظر شكل 5-46.



شكل 5-46: القائمة بعد تعديل وصلات العقد قبل وبعد العقدة ذات المعلومات 17.
بعد هذا نقوم بتقليل count بمقدار 1 ونحذف العقدة المشار اليها بواسطة current. انظر شكل 5-47.



شكل 5-47: القائمة بعد حذف العقدة ذات المعلومات 17.

تعريف الدالة deleteNode هو:

```

template<class Type>
void doublyLinkedList<Type>::deleteNode(const Type& deleteItem)
{
    nodeType<Type> *current;          //pointer to traverse the list
    nodeType<Type> *trailCurrent;      //pointer just before current

    bool found;

    if(first == NULL)
        cout<<"Cannot delete from an empty list"<<endl;
    else
        if(first->info == deleteItem) //node to be deleted is the
            //first node
        {
            current = first;
            first = first->next;
        }
  
```

```

        if(first != NULL)
            first->back = NULL;
        else
            last = NULL;

        count--;
        delete current;
    }
    else
    {
        found = false;
        current = first;

        while(current != NULL && !found) //search the list
            if(current->info == deleteItem)
                found = true;
            else
                current = current->next;

        if(current == NULL)
            cout<<"The item to be deleted is not in the list."
                <<endl;
        else
            if(current->info == deleteItem) //check for equality
            {
                trailCurrent = current->back;
                trailCurrent->next = current->next;

                if(current->next != NULL)
                    current->next->back = trailCurrent;

                if(current == last)
                    last = trailCurrent;

                count--;
                delete current;
            }
            else
                cout<<"The item to be deleted is not in the list."
                    <<endl;
    }
} //end deleteNode
} //end deleteNode

```

تحليلات دالات الفئة doublyLinkedList مماثلة لتحليلات الفئة linkedListType ولهذا يتم تركها لكي تكون تمارين لك.

الحاوية المتعاقبة: list

الفصل الرابع ذكر ثلاثة أنواع من الحاويات المتعاقبة – vector، deque، و list ويتم وصف الحاويات المتعاقبة vector و deque في الفصل الرابع وهذا الفصل يصف الحاوية المتعاقبة من مكتبة القالب المعياري. يتم استخدام حاويات القائمة كقوائم متصلة من الطرفين. لهذا يشير كل عنصر في القائمة الى ما يسبقه مباشرةً والى ما يليه مباشرةً (فيما عدا العنصر الأول والأخير). تذكر أن القائمة المتصلة ليست بنية بيانات وصول عشوائي مثل المصفوفة. لهذا من أجل الوصول الى العنصر الخامس مثلاً يجب أن نقوم أولاً باجتياز الأربعة عناصر الأولى.

اسم الفئة المحتوية على تعريف الفئة list هو list. تعريف الفئة list وتعريفات الدالات لتطبيق عمليات متنوعة على قائمة موجودة في الملف الرئيسي list. لهذا لاستخدام list في برنامج يجب أن يتضمن البرنامج البيان التالي:

#include<list>

الفئة list مثلها مثل فئات الحاوية الأخرى تحتوي على عدة مقومات ولهذا يمكن تهيئة هدف list بعدة طرق عند اعلانه كما هو موضح في جدول 5-6. جدول 5-6: طرق متنوعة لاعلان هدف list.

| البيان | التأثير |
|--------|---------|
| ***** | ***** |
| ***** | ***** |
| | |
| | |
| | |

الجدول 4-5 (في الفصل 4) يصف العمليات المشتركة بين جميع الحاويات والجدول 4-6 يصف العمليات المشتركة بين جميع الحاويات المتعاقبة. بالإضافة الى تلك العمليات المشتركة يقوم الجدول 5-7 بوصف العمليات الخاصة بالحوية list. اسم الدالة التي تطبق العملية موضح بالخط العريض. افترض أن listCont عبارة عن حاوية من النوع list. جدول 5-7: عمليات خاصة بالحوية list.

| التعبير | التأثير |
|---|---|
| listCont.assign (n, elem) | يحدد n نسخ من elem. |
| listCont.assign (beg, end) | يحدد جميع العناصر في النطاق من البداية ... النهاية -1. |
| listCont.push_front (elem) | يدخل عنصر في بداية listCont. |
| listCont.pop_front () | يزيل آخر عنصر من listCont. |
| listCont.front () | يُنتج العنصر الأول ولا يتحقق مما إذا كانت الحاوية فارغة. |
| listCont.back () | يُنتج العنصر الأخير ولا يتحقق مما إذا كانت الحاوية فارغة. |
| listCont.remove (elem) | يحذف جميع العناصر المتساوية مع elem. |
| listCont.remove_if (oper) | يحذف جميع العناصر التي يكون oper (elem) فيها true. |
| listCont.unique () | ***** |
| listCont.unique (oper) | ***** |
| listCont.splice (pos, listCont2) | يتم نقل جميع عناصر listCont2 الى listCont1 قبل الموقع المحدد بواسطة المكرر pos بعد أن تكون هذه العملية listCont2 فارغة. |
| listCont.splice (pos, listCont2, pos2) | يتم نقل جميع العناصر بدءاً من pos2 من listCont2 الى listCont1 قبل الموقع المحدد بواسطة المكرر pos |
| listCont. (pos, listCont2, beg, end) | يتم نقل جميع العناصر في نطاق البداية... النهاية-1 من listCont2 الى listCont1 قبل الموقع المحدد بواسطة المكرر pos |
| listCont. sort () | يتم ترتيب عناصر listCont ومعيار الترتيب هو < |

جدول 5-7: العمليات الخاصة بالحوية list (مستمر)

| التعبير | التأثير |
|---|---|
| listCont.sort (oper) | يتم تخزين عناصر listCont ويتم تحديد معيار الترتيب بواسطة oper. |
| listCont.merge (listCont2) | افتراض أن عناصر listCont1 و listCont2 مرتبة. هذه العملية تنقل جميع عناصر listCont2 داخل listCont1 وبعد هذه العملية تكون العناصر في listCont1 مرتبة وتكون listCont2 فارغة. |
| listCont.merge (listCont2, oper) | افتراض أن عناصر listCont1 و listCont2 مرتبة وفقاً لمعيار الترتيب oper. هذه العملية تنقل جميع عناصر listCont2 داخل listCont1 وبعد هذه العملية تكون العناصر في listCont1 مرتبة وفقاً لمعيار الترتيب oper. |
| listCont.reverse () | يتم عكس عناصر listCont. |

جدول 1-5 يوضح كيفية استخدام عدة عمليات على حاوية قائمة.

مثال 1-5:

```
//List Container Example
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    list<int> intList1, intList2, intList3, intList4;           //Line 1

    ostream_iterator<int> screen(cout, " ");                  //Line 2

    intList1.push_back(23);                                     //Line 3
    intList1.push_back(58);                                     //Line 4
    intList1.push_back(58);                                     //Line 5
    intList1.push_back(58);                                     //Line 6
    intList1.push_back(36);                                     //Line 7
    intList1.push_back(15);                                     //Line 8
    intList1.push_back(93);                                     //Line 9
    intList1.push_back(98);                                     //Line 10
    intList1.push_back(58);                                     //Line 11
```

```

cout<<"Line 12: intList1: ";           //Line 12
copy(intList1.begin(), intList1.end(), screen); //Line 13
cout<<endl;                           //Line 14

intList2 = intList1;                   //Line 15

cout<<"Line 16: intList2: ";           //Line 16
copy(intList2.begin(), intList2.end(), screen); //Line 17
cout<<endl;                           //Line 18

intList1.unique();                     //Line 19

cout<<"Line 20: After removing the consecutive "
    <<"duplicates,"<<endl
    <<"    intList1: ";               //Line 20
copy(intList1.begin(), intList1.end(), screen); //Line 21
cout<<endl;                           //Line 22

intList2.sort();                       //Line 23

cout<<"Line 24: After sorting, intList2: "; //Line 24
copy(intList2.begin(), intList2.end(), screen); //Line 25
cout<<endl;                           //Line 26

intList3.push_back(13);                //Line 27
intList3.push_back(23);                //Line 28
intList3.push_back(25);                //Line 29
intList3.push_back(136);               //Line 30
intList3.push_back(198);               //Line 31

cout<<"Line 32: intList3: ";           //Line 32
copy(intList3.begin(), intList3.end(), screen); //Line 33
cout<<endl;                           //Line 34

intList4.push_back(-2);                //Line 35
intList4.push_back(-7);                //Line 36
intList4.push_back(-8);                //Line 37

cout<<"Line 38: intList4: ";           //Line 38
copy(intList4.begin(), intList4.end(), screen); //Line 39
cout<<endl;                           //Line 40

intList3.splice(intList3.begin(), intList4); //Line 41

cout<<"Line 42: After moving the elements of "
    <<"intList4 into intList3,"<<endl
    <<"    intList3: ";               //Line 42
copy(intList3.begin(), intList3.end(), screen); //Line 43
cout<<endl;                           //Line 44

```

```

intList3.sort(); //Line 45

cout<<"Line 46: After sorting, intList3: "; //Line 46
copy(intList3.begin(), intList3.end(), screen); //Line 47
cout<<endl; //Line 48

intList2.merge(intList3); //Line 49

cout<<"Line 50: After merging intList2 and intList3, "
    <<"intList2: " <<endl <<" "; //Line 50
copy(intList2.begin(), intList2.end(), screen); //Line 51
cout<<endl; //Line 52

intList2.unique(); //Line 53

cout<<"Line 54: After removing the consecutive "
    <<"duplicates, intList2: " <<endl
    <<" "; //Line 54
copy(intList2.begin(), intList2.end(), screen); //Line 55
cout<<endl; //Line 56

return 0;
}

```

المخرجات:

الصف 12: intList1: 58 98 93 15 36 58 58 58 23
 الصف 16: intList2: 58 98 93 15 36 58 58 58 23
 الصف 20: بعد ازالة المكررات المتتالية
 intList1: 58 98 93 15 36 58 23
 الصف 24: بعد الترتيب intList2: 98 93 58 58 58 36 23 15
 الصف 32: intList3: 198 136 25 23 13
 الصف 38: intList4: 8- 7- 2-
 الصف 42: بعد نقل عناصر intList4 داخل intList3
 intList3: 198 136 25 23 13 8- 7- 2-
 الصف 46: بعد الترتيب intList3: 198 136 25 23 13 2- 7- 8-
 الصف 50: بعد دمج intList2 و intList3 تكون
 : 198 136 98 93 58 58 58 36 25 23 23 15 13 2- 7- 8-
 الصف 54: بعد ازالة المكررات المتتالية تكون intList2
 : 198 136 98 93 58 36 25 23 15 13 2- 7- 8-

في غالبية الحالات تكون مخرجات البرنامج السابق واضحة. البيانات في الصفوف من 3 والى 11 تدخل 23، 58، 58، 58، 36، 15، 93، 98، 58 (بهذا الترتيب) داخل intList1. البيان في الصف 15 يقوم بنسخ عناصر intList1 داخل intList2. بعد تنفيذ هذا البيان تكون intList1 و intList2 متطابقة. البيان في الصف 19 يحذف أي حدوث متتالي للعناصر ذاتها. على سبيل المثال يظهر العدد 58 بشكل متتالي ثلاث مرات. العملية unique تحذف حالتها ظهور للعدد 58. لاحظ أن هذه العملية ليس لها أثر على العدد 58 الذي يظهر في نهاية intList1.

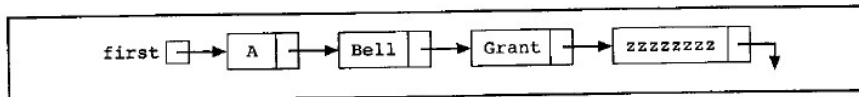
البيان في الصف 23 يقوم بترتيب intList2. البيانات في الصفوف من 27 وحتى 31 تدخل 13، 23، 25، 136، 198 داخل intList3. بالمثل تقوم البيانات في الصفوف من 35 وحتى 37 بادخال -2، -7، 8 داخل intList4. البيان في الصف 41 يستخدم العملية splice لنقل عناصر intList4 الى بداية intList3. بعد العملية splice تكون intList4 فارغة. البيان في الصف 45 يقوم بترتيب intList3 والبيان في الصف 49 يقوم بدمج intList2 و intList3 داخل intList2. بعد عملية الدمج تكون intList فارغة. معاني البيانات المتبقية متشابهة.

قوائم متصلة ذات عقد header وTrailer:

عند ادخال وحذف عناصر من قائمة متصلة (خاصة قائمة مرتبة) رأينا أن هناك حالات خاصة مثل ادخال أو (حذف) اما في بداية (العقدة الأولى) القائمة أو في قائمة فارغة. هذه الحالات في حاجة الى معاملتها بشكل منفصل. كنتيجة لهذا لم تكن الحلول الحسابية للادخال والحذف بسيطة وواضحة كما نريد. من أحد طرق تبسيط تلك الحلول الحسابية عمد ادخال عنصر أبدأً قبل العنصر الأخير أو بعد العنصر الأخير وعدم حذف العقدة الأولى أبدأً. بعد هذا نناقش كيفية القيام بذلك.

افترض أن عقد القائمة مرتبة أي أنها مرتبة على أساس معين. افترض أيضاً أنه من الممكن بالنسبة لنا أن نحدد ما اذا كانت الأساسات الأصغر والأكبر موجودة في مجموعة البيانات المعطاة. يمكن عمل عقدة تسمى header في بداية القائمة المحتوية على قيمة أصغر من أصغر قيمة في مجموعة البيانات. بالمثل يمكننا عمل عقدة تسمى trailer في نهاية القائمة المحتوية على قيمة أكبر من أكبر قيمة في مجموعة البيانات. هاتان العقدتان header و trailer تعمل من أجل تبسيط حلول الادخال والحذف وليست جزءاً من القائمة الفعلية. القائمة الفعلية تكون بين هاتين العقدتين.

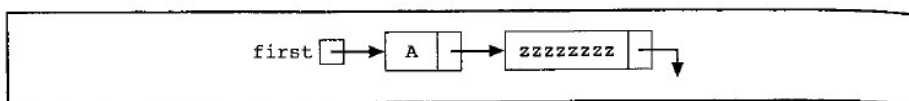
افترض على سبيل المثال أن البيانات مرتبة حسب الاسم الأخير. افترض كذلك أن الاسم الأخير عبارة عن مقطع مكون على الأكثر من ثمانية رموز. أصغر اسم أخير يكون أكبر من المقطع "A" وأكبر اسم أخير يكون أصغر من المقطع "zzzzzzzz". يمكننا عمل العقدة header بالقيمة "A" والعقدة trailer بالقيمة "zzzzzzzz". القائمة الموجودة في شكل 5-48 توضح هذه الفكرة.



شكل 5-48: قائمة متصلة غير فارغة بها عقد header و trailer.

القائمة المتصلة الفارغة ذات العقد header و trailer بها عقدتان فقط وهما header و trailer.

شكل 5-49 يوضح قائمة متصلة فارغة بها عقدتان header و trailer.



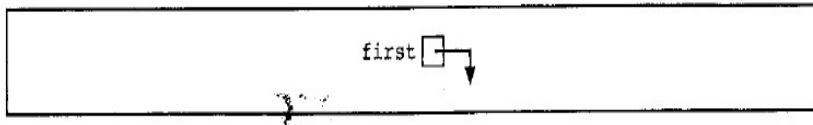
شكل 5-49: قائمة متصلة فارغة بها عقد Header و Trailer.
كما سبق تكون العمليات المعتادة المؤداة على القوائم ذات العقد header و trailer هي:

1. تهيئة القائمة (في وضع فارغ).
2. تحديد ما اذا كانت القائمة فارغة.
3. تدمير القائمة.
4. اخراج القائمة.
5. ايجاد طول القائمة.
6. بحث القائمة عن عنصر محدد.
7. استعادة العنصر الأول والأخير من القائمة.
8. ادخال عنصر في القائمة.
9. حذف عنصر من القائمة.
10. نسخ القائمة.

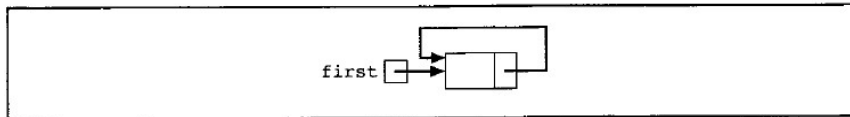
نترك لك كتمرين تصميم فئة لتطبيق قائمة متصلة ذات عقد header و trailer. (انظر تمرين البرمجة 11 في نهاية هذا الفصل).

قوائم متصلة دائرية:

القائمة المتصلة التي تشير فيها العقدة الأخيرة الى العقدة الأولى تسمى قائمة متصلة دائرية. الأشكال 50-5 وحتى 52-5 توضح عدة قوائم متصلة دائرية.

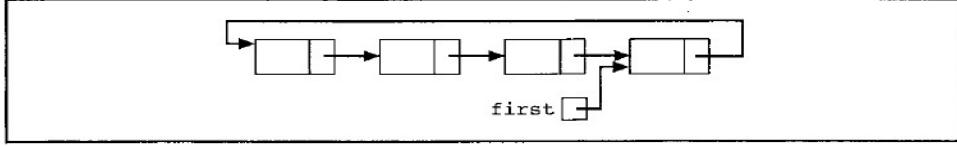


شكل 50-5: قائمة متصلة دائرية فارغة.



شكل 51-5: قائمة متصلة دائرية ذات عقدة واحدة.

في القائمة المتصلة الدائرية التي بها أكثر من عقدة واحدة من التقليدي جعل المؤشر first يشير الى العقدة الأخيرة من القائمة. اذن باستخدام المؤشر first يمكنك تناول كل من العقد الأولى والأخيرة من القائمة. على سبيل المثال يشير first الى العقدة الأخيرة ويشير first->link الى العقدة الأولى. الشكل 52-5 يوضح قائمة متصلة دائرية ذات أكثر من عقدة.



شكل 5-52: قائمة متصلة دائرية ذات أكثر من عقدة.

كما سبق تكون العمليات المعتادة المؤداة على قائمة متصلة دائرية هي:

11. تهيئة القائمة (في وضع فارغ).

12. تحديد ما اذا كانت القائمة فارغة.

13. تدمير القائمة.

14. اخراج القائمة.

15. ايجاد طول القائمة.

16. بحث القائمة عن عنصر محدد.

17. استعادة العنصر الأول والأخير من القائمة.

18. ادخال عنصر في القائمة.

19. حذف عنصر من القائمة.

20. نسخ القائمة.

نترك لك كتمرين تصميم فئة لتطبيق قائمة متصلة دائرية ومرتبطة. (انظر تمرين البرمجة 12 في نهاية هذا الفصل).

تمرين برمجة: تخزين الفيديو:

بالنسبة الى العائلة والأفراد يكون المكان المفضل للذهاب اليه في نهاية الأسبوع وفي العطلات هو متجر الفيديو لتأجير فيلم. هناك متجر فيديو جديد على وشك أن يفتح في حيك وبالرغم من هذا ليس لديه برنامج لتتبع شرائطه وزبائنه. مديرو المتجر يريدون من أي شخص كتابة برنامج من أجل نظامهم حتى يمكن لمتجر الفيديو أن يعمل. يجب أن يكون البرنامج قادراً على أداء العمليات التالية:

1. ايجار فيديو أي التحقق من خروجه.

2. اعادة الفيديو أو التأكد من رجوعه.

3. عمل قائمة بالفيديوهات التي يمتلكها المتجر.

4. عرض تفاصيل فيديو محدد.

5. طبع قائمة بجميع الفيديوهات الموجودة في المتجر.

6. التحقق مما اذا كان هناك فيديو محدد في المتجر.

7. الحفاظ على قاعدة بيانات الزبائن.

8. طباعة قائمة بجميع الفيديوهات التي يقوم يستأجرها كل زبون.

لنقم بكتابة برنامج لمتجر الفيديو. هذا المثال يوضح أسلوب التصميم القائم على الهدف وبشكل خاص التوريث والانتقال.

متطلبات البرمجة تخبرنا أن متجر الفيديو به مكونان أساسيان: الشرائط والزبائن ونقوم بوصف هذين المكونين بالتفصيل ونحتاج كذلك الى الحفاظ على ثلاثة قوائم:

- قائمة بجميع الشرائط الموجودة في المتجر.

- قائمة بجميع زبائن المتجر.

- قائمة بجميع الفيديوهات المؤجرة في الوقت الحالي.

سوف نقوم بعمل البرنامج في جزئين. في الجزء الأول نقوم بتصميم وتطبيق واختبار مكون الشرائط. وفي الجزء الثاني نقوم بتصميم وتطبيق مكون الزبائن الذي تتم اضافته الى مكون الشرائط الذي يتم عمله في الجزء الأول. بعد اتمام الجزئين 1 و2 يمكننا اجراء جميع العمليات المذكورة من قبل.

الجزء 1: مكون الشرائط:

هدف الشرائط: هذه هي المرحلة الأولى التي نبدأ منها مناقشة مكونات الهدف. الأشياء الشائعة المرتبطة بالشريط هي:

- اسم الفيلم.

- اسم النجوم.

- اسم المنتج.

- اسم المخرج.

- اسم شركة الانتاج.

- عدد النسخ في المتجر.

من هذه القائمة نرى أن بعض من العمليات المؤداة على هدف الشرائط هي:

1. تحديد معلومات الشريط: العنوان، والنجوم، وشركة الانتاج، وغيرها.

2. عرض تفاصيل شريط محدد.

3. التحقق من عدد النسخ في المتجر.

4. اخراج (ايجار) الشريط أي اذا كان عدد النسخ أكبر من صفر قم بتقليل عدد النسخ بمقدار واحد.

5. ادخال (استرجاع) الشريط. لاسترجاع الشريط يجب أن تحقق أولاً مما اذا كان المتجر يمتلك

هذا الشريط أم لا واذا كان يمتلكها قم بزيادة عدد النسخ بمقدار واحد.

6. التحقق مما اذا كان متوفر شريط معين-أي أن عدد النسخ الموجودة حالياً في المتجر أكبر من صفر.

لحذف شريط من قائمة الشرائط يجب أن يتم بحث قائمة الفيديو من أجل الشريط الذي يتم حذفه. لهذا نحتاج الى التحقق من اسم الشريط لاكتشاف الشريط الذي سوف يتم حذفه من القائمة. يكون الشريطان متماثلين اذا كان لهما نفس العنوان.

الفئة التالية تقوم بتعريف هدف الشريط كنوع بيانات مجرد:

```
#include <iostream>
#include <string>

using namespace std;

class videoType
{
    friend ostream& operator<<(ostream&, const videoType&);

public:
    void setVideoInfo(string title, string star1,
                     string star2, string producer,
                     string director, string productionCo,
                     int setInStock);
    //Function to set the details of a video.
    //Private data members are set according to the parameters.
    //Postcondition: videoTitle = title; movieStar1 = star1;
    // movieStar2 = star2; movieProducer = producer;
    // movieDirector = director;
    // movieProductionCo = productionCo;
    // copiesInStock = setInStock

    int getNoOfCopiesInStock() const;
    //Function to check the number of copies in stock.
    //Postcondition: The value of the data member copiesInStock is
    // returned.

    void checkOut();
    //Function to rent a video.
    //Postcondition: The number of copies in stock is decremented
    // by one.

    void checkIn();
    //Function to check in a video.
    //Postcondition: The number of copies in stock is incremented
    // by one.

    void printTitle() const;
    //Function to print the title of a movie.

    void printInfo() const;
    //Function to print the details of a video.
    //Postcondition: The title of the movie, stars, director, and
    // so on are displayed on the screen.
```

```

bool checkTitle(string title);
    //Function to check whether the title is the same as the title
    //of the video.
    //Postcondition: Returns true if the title is the same as
    //                the title of the video; otherwise, returns
    //                false.
void updateInStock(int num);
    //Function to increment the number of copies in stock by
    //adding the value of the parameter num.
    //Postcondition: copiesInStock = copiesInStock + num
void setCopiesInStock(int num);
    //Function to set the number of copies in stock.
    //Postcondition: copiesInStock = num
string getTitle();
    //Function to return the title of the video.
    //Postcondition: The title of the video is returned.
videoType(string title = "", string star1 = "",
           string star2 = "", string producer = "",
           string director = "", string productionCo = "",
           int setInStock = 0);
    //constructor
    //Private data members are set according to the incoming
    //parameters. If no values are specified, the default
    //values are assigned.
    //Postcondition: videoTitle = title; movieStar1 = star1;
    // movieStar2 = star2; movieProducer = producer;
    // movieDirector = director;
    // movieProductionCo = productionCo;
    // copiesInStock = setInStock

    //Overload the relational operators.
bool operator==(const videoType&) const;
bool operator!=(const videoType&) const;

private:
    string videoTitle;        //variable to store the name of the
                               //movie
    string movieStar1;        //variable to store the name of the
                               //star
    string movieStar2;        //variable to store the name of the
                               //star
    string movieProducer;     //variable to store the name of the
                               //producer
    string movieDirector;     //variable to store the name of the
                               //director

    string movieProductionCo; //variable to store the name of the
                               //production company
    int copiesInStock;        //variable to store the number of
                               //copies in stock
};

```

نترك رسم لغة التشكيل الموحدة للفئة videoType كتمرين لك.
للاخراج السهل نقوم باثقال معامل ادخال تدفق المخرجات << للفئة videoType.
بعد هذا نقوم بكتابة تعريفات كل دالة من الفئة videoType.

```
void videoType::setVideoInfo(string title, string star1,
                             string star2, string producer,
                             string director,
                             string productionCo, int setInStock)
{
    videoTitle = title;
    movieStar1 = star1;
    movieStar2 = star2;
    movieProducer = producer;
    movieDirector = director;
    movieProductionCo = productionCo;
    copiesInStock = setInStock;
}

void videoType::checkOut()
{
    if(getNoOfCopiesInStock() > 0)
        copiesInStock--;
    else
        cout<<"Currently out of stock"<<endl;
}

void videoType::checkIn()
{
    copiesInStock++;
}

int videoType::getNoOfCopiesInStock() const
{
    return copiesInStock;
}
```

```

void videoType::printTitle() const
{
    cout<<"Video Title: "<<videoTitle<<endl;
}

void videoType::printInfo() const
{
    cout<<"Video Title: "<<videoTitle<<endl;
    cout<<"Stars: "<<movieStar1<<" and "<<movieStar2<<endl;
    cout<<"Producer: "<<movieProducer<<endl;
    cout<<"Director: "<<movieDirector<<endl;
    cout<<"Production Company: "<<movieProductionCo<<endl;
    cout<<"Copies in stock: "<<copiesInStock<<endl;
}

bool videoType::checkTitle(string title)
{
    return(videoTitle == title);
}

void videoType::updateInStock(int num)
{
    copiesInStock += num;
}

void videoType::setCopiesInStock(int num)
{
    copiesInStock = num;
}

string videoType::getTitle()
{
    return videoTitle;
}

videoType::videoType(string title, string star1,
                    string star2, string producer,
                    string director,
                    string productionCo, int setInStock)
{
    setVideoInfo(title, star1, star2, producer, director,
                productionCo, setInStock);
}

bool videoType::operator==(const videoType& other) const
{
    return (videoTitle == other.videoTitle);
}

```

```

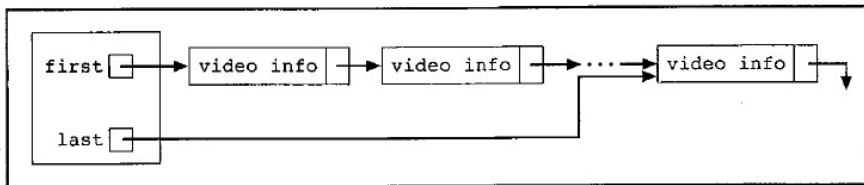
bool videoType::operator!=(const videoType& other) const
{
    return (videoTitle != other.videoTitle);
}

ostream& operator<<(ostream& os, const videoType& video)
{
    os<<endl;
    os<<"Video Title: "<<video.videoTitle<<endl;
    os<<"Stars: "<<video.movieStar1<<" and "
        <<video.movieStar2<<endl;
    os<<"Producer: "<<video.movieProducer<<endl;
    os<<"Director: "<<video.movieDirector<<endl;
    os<<"Production Company: "<<video.movieProductionCo<<endl;
    os<<"Copies in stock: "<<video.copiesInStock<<endl;
    os<<"_____ "<<endl;
    return os;
}

```

قائمة الفيديو:

هذا البرنامج يقتضي منا الاحتفاظ بقائمة لجميع الشرائط الموجودة في المتجر ويجب أن نكون قادرين على اضافة شريط جديد في قائمتنا. اننا لن نعرف عدد الشرائط في المتجر واطافة أو حذف شريط من المتجر سوف يغير عدد الشرائط في المتجر. لهذا نستخدم قائمة متصلة لعمل قائمة بالشرائط. انظر شكل 5-53 (هذا الشكل لا يوضح المتغير count).



شكل 5-53: قائمة الشرائط videoList.

مبكراً في هذا الفصل قمنا بتعريف الفئة `linkedListType` لعمل قائمة متصلة الأهداف. كما قمنا بتعريف العمليات الأساسية مثل ادخال وحذف شريط موجود في القائمة. بالرغم من هذا هناك بعض العمليات الخاصة للغاية بقائمة الشرائط مثل اخراج شريط، وادخال شريط، وتحديد عدد نسخ الشريط، وهكذا. هذه العمليات غير متاحة في الفئة `linkedListType`. لهذا نشق الفئة `videoListType` من الفئة `linkedListType` ونضيف هذه العمليات.

تعريف الفئة videoListType هو:

```
#include <iostream>
#include <string>
#include "linkedList.h"
#include "videoType.h"

using namespace std;

class videoListType: public linkedListType<videoType>
{
public:
    bool videoSearch(string vTitle);
        //Function to search the list to see whether a
        //particular title, specified by the parameter title,
        //is in the store.
        //Postcondition: Returns true if the title is found;
        //                otherwise, returns false.

    bool isVideoAvailable(string vTitle);
        //Function to return true if at least one copy of a
        //particular video is in the store.

    void videoCheckOut(string vTitle);
        //Function to check out a video, that is, rent a video.
        //Postcondition: copiesInStock is decremented by one.

    void videoCheckIn(string vTitle);
        //Function to check in a video returned by a customer.
        //Postcondition: copiesInStock is incremented by one.

    bool videoCheckTitle(string vTitle);
        //Function to determine whether a particular video is in
        //the store.
        //Postcondition: Returns true if the video title is the
        //                same as vTitle; otherwise, returns false.

    void videoUpdateInStock(string vTitle, int num);
        //Function to update the number of copies of a video
        //by adding the value of the parameter num. The
        //parameter vTitle specifies the name of the video for
        //which the number of copies is to be updated.
        //Postcondition: copiesInStock = copiesInStock + num

    void videoSetCopiesInStock(string vTitle, int num);
        //Function to reset the number of copies of a video.
        //The parameter vTitle specifies the name of the video
        //for which the number of copies is to be reset; the
```

```
bool videoSearch (string vTitle);  
// دالة لبحث القائمة لرؤية ما اذا كان عنوان محدد  
// بواسطة العامل title موجود في المتجر أم لا.  
// شرط تالي: ينتج true اذا تم العثور على العنوان  
// وبخلاف هذا تنتج false.
```

```
bool isVideoAvailable (string vTitle);  
// دالة لانتاج true اذا كانت هناك نسخة واحدة على الأقل  
// من شريط معين موجودة في المتجر.
```

```
void videoCheckOut (string vTitle);  
// دالة لاجراج الشريط أي تأجير.  
// شرط تالي: يتم تقليل عدد النسخ في المتجر بمقدار 1.
```

```
void videoCheckIn (string vTitle);  
// دالة لادخال الشريط الذي يعيده الزبون.  
// شرط تالي: يتم زيادة عدد النسخ في المتجر بمقدار 1.
```

```
bool videoCheckTitle (string vTitle);  
// دالة لتحديد ما اذا كان هناك شريط معين في المتجر.  
// شرط تالي: تنتج true اذا كان عنوان الشريط هو نفسه vTitle  
// وبخلاف هذا تنتج false.
```

```
void videoUpdateInStock (string vTitle, int num);  
// دالة لتحديث عدد نسخ الشريط عن طريق اضافة قيمة المعامل num.  
// المعامل vTitle يحدد اسم الشريط الذي يتم تحديث عدد نسخه.  
// شرط تالي: النسخ في المخزن = النسخ في المخزن + العدد
```

```
void videoSetCopiesInStock (string vTitle, int num);  
// دالة لاعادة تحديد عدد نسخ الشريط.  
// المعامل vTitle يحدد اسم الشريط الذي يتم اعادة تحديد عدد نسخه  
// والمعامل num يحدد عدد النسخ.  
// شرط تالي: النسخ في المخزن = العدد
```

```

//parameter num specifies the number of copies.
//Postcondition: copiesInStock = num

void videoPrintTitle();
//Function to print the titles of all the videos in the store.

private:
void searchVideoList(string vTitle, bool& found,
    nodeType<videoType>* &current);
//Function to search the video list for a particular
//video, specified by the parameter vTitle.
//Postcondition: If the video is found, the parameter
//                found is set to true; otherwise,
//                it is set to false. The parameter current
//                points to the node containing the video.
};

```

void videoPrintTitle ();
 // دالة لطباعة عناوين جميع الشرائط في المتجر.
 خاصة:

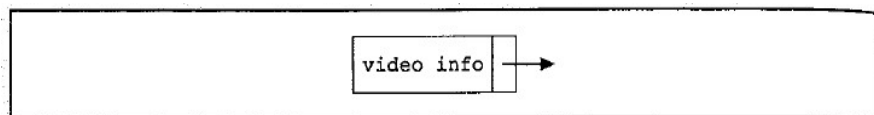
void searchVideoList (string vTitle, bool& found,
 nodeType<videoType>* ¤t);
 // دالة لبحث قائمة الشريط عن شريط محدد بواسطة المعامل vTitle.
 // شرط تالي: اذا تم العثور على الشريط يتم ضبط المعامل found عند true
 // بخلاف هذا ينتج false. المعامل current يشير الى العقدة المحتوية على الشريط.

لاحظ أن الفئة videoListType مستمدة من الفئة linkedListType عبر توريث عام. فضلاً عن هذا linkedListType عبارة عن قالب فئة وقد قمنا بتمرير الفئة videoType كمعامل لهذه الفئة. أي أن الفئة videoListType ليست قالباً. بما أننا الآن نتعامل مع نوع بيانات خاص للغاية فلا يعد هناك حاجة الى أن تكون الفئة videoListType قالباً. لهذا يكون نوع كل عقدة في القائمة المتصلة videoType. من خلال دالات عنصر الفئة videoType يمكن الآن تناول عناصر معينة – مثل videoTitle وcopiesInStock لهدف من النوع videoType.

تعريفات الدالات لتطبيق عمليات الفئة videoListType معطاة فيما بعد.

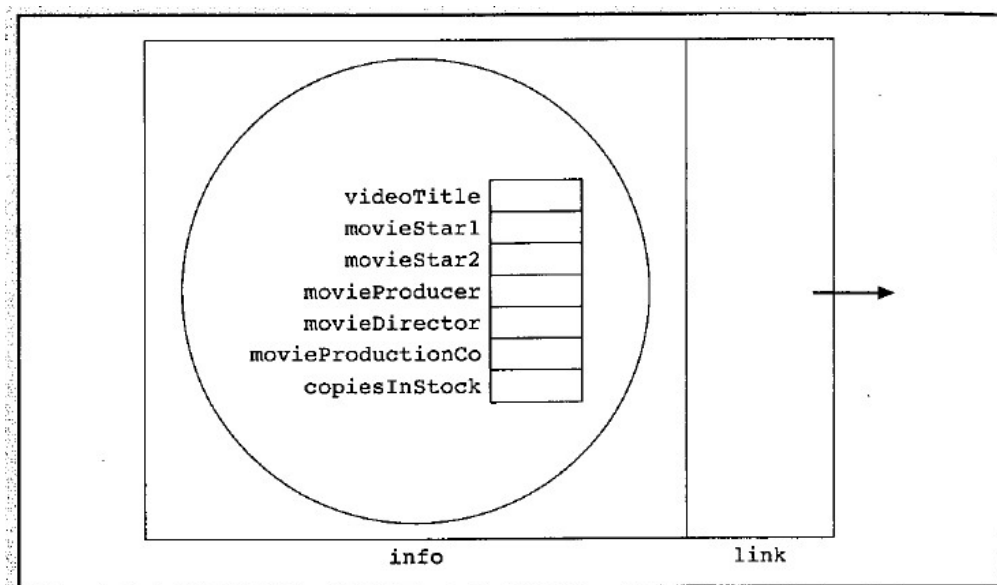
العمليات الأساسية المؤداة على قائمة الشريط هي ادخال الشريط واخراج الشريط. هاتان العمليتان تحتاج الى أن يتم بحث القائمة واخراج وادخال موقع الشريط للعثور عليه في قائمة الشرائط. هناك عمليات أخرى مثل التحقق مما اذا كان شريط محدد موجود في المتجر وتحديث عدد نسخ الشريط وغيرها تحتاج أن يتم بحث قائمة الشرائط. لتبسيط عملية البحث نكتب دالة تقوم ببحث قائمة الشرائط من أجل شريط محدد. اذا تم العثور على الشريط فانه يحدد المعامل found عند true وينتج مؤشر الى الشريط حتى يمكن أداء عمليات الاخراج ولاادخال والعمليات الأخرى على الشريط. لاحظ أن

الدالة searchVideoList عنصر بيانات خاص من الفئة videoListType لأنه يتم استخدامه فقط للتحكم الداخلي. أولاً نقوم بوصف اجراء البحث.
انظر الى عقد قائمة الشرائط الموضحة في شكل 5-54.



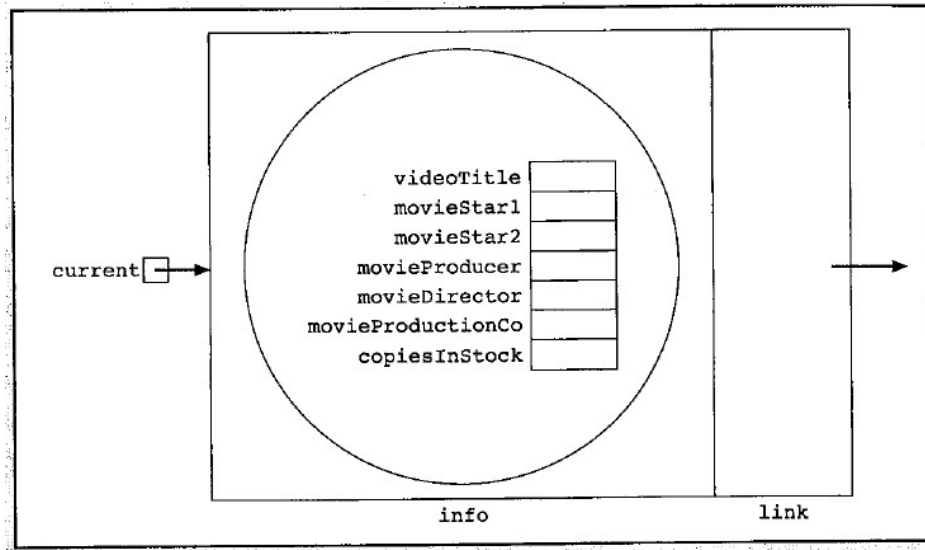
شكل 5-54: عقدة لقائمة شرائط.

المكون info يكون من النوع videoType ويحتوي على المعلومات الضرورية عن الشريط وبشكل محدد يحتوي على سبعة عناصر وهي: عنوان الشريط، ونجم الفيلم الأول، ونجم الفيلم الثاني، ومنتج الفيلم، ومخرج الفيلم، وشركة انتاج الفيلم، والنسخ في المخزن. (انظر تعريف الفئة videoType). لهذا تكون عقدة قائمة الشرائط بالشكل الموضح في شكل 5-55.



شكل 5-55: عقد قائمة شرائط توضح المكونات info.

جميع عناصر البيانات هذه عناصر خاصة ولا يمكن تناولها بشكل مباشر. دالات عنصر الفئة videoType تساعدنا على التحقق من و/أو تحديد قيمة مكون معين.
افترض أن المؤشر – مثل current – يشير الى عقدة في قائمة الشرائط. انظر شكل 5-56.



شكل 5-56: المؤشر current وعقدة قائمة الشرائط.

الآن:

current->info

يشير الى الجزء info من العقدة. افترض أننا نريد أن نعرف ما اذا كان عنوان الشريط المخزن في هذه العقدة هو نفس الاسم المحدد بواسطة المتغير title. التعبير:

current->info.checkTitle (title)

يكون صحيح اذا كان عنوان الشريط المخزن في هذه العقدة هم نفس العنوان المحدد بواسطة المعامل title ويكون بخلاف هذا خطأ. (لاحظ أن دالة العنصر checkTitle عبارة عن دالة منتجة للقيمة. انظر اعلانها في الفئة videoType).

كمثال آخر افترض أننا نريد تحديد عنصر البيانات copiesInStock (النسخ في المخزن) لهذه العقدة عند 10. بما أن copiesInStock عنصر بيانات خاص لا يمكن تناوله بشكل مباشر. لهذا يكون البيان:

current->info.copiesInStock = 10; //illegal

غير صحيح وينتج خطأ في زمن التجميع. علينا استخدام دالة العنصر setCopiesInStock كما يلي:

current->info.setCopiesInStock (10);

بما أننا علمنا الآن كيفية تناول عنصر بيانات شريط مخزن في عقدة ما لنقم بوصف الحل الحسابي لبحث قائمة الشرائط:

إذا كانت قائمة الشرائط خالية

خطأ

غير ذلك

إذا كان عنوان الشريط الحالي هو نفس العنوان المرغوب فيه أوقف البحث

غير ذلك

تحقق من العقدة التالية.

تعريف الدالة التالية يؤدي البحث المطلوب:

```
void videoListType::searchVideoList(string vTitle, bool& found,
                                     nodeType<videoType>* &current)
{
    found = false;    //set found to false

    if(first == NULL) //list is empty
        cerr<<"Cannot search an empty list. "<<endl;
    else
    {
        current = first; //set current point to first
                        //node in the list.
        found = false;    // set found to false

        while(!found && current != NULL) //search the list
            if(current->info.checkTitle(vTitle)) //the item is
                                                    //found
                found = true;
            else
                current = current->link; //make current point
                                        //to the next node
    } //end else
}
```

هذا التعريف ناجح والمعامل found محدد عند true والمعامل current يشير الى العقدة المحتوية على معلومات الشريط. إذا كان البحث غير ناجح يتم تحديد found عند false و current تكون NULL.

تعريفات الدالات الأخرى للفئة videoListType هي:

```
bool videoListType::isVideoAvailable(string vTitle)
{
    bool found;
    nodeType<videoType> *location;
```

```

    searchVideoList(vTitle, found, location);

    if(found)
        found = (location->info.getNoOfCopiesInStock() > 0);
    else
        found = false;

    return found;
}

void videoListType::videoCheckIn(string vTitle)
{
    bool found = false;
    nodeType<videoType> *location;

    searchVideoList(vTitle, found, location); //search the list

    if(found)
        location->info.checkIn();
    else
        cout<<"The store does not carry this video."<<endl;
}

void videoListType::videoCheckOut(string vTitle)
{
    bool found = false;
    nodeType<videoType> *location;

    searchVideoList(vTitle, found, location); //search the list

    if(found)
        location->info.checkOut();
    else
        cout<<"The store does not carry this video."<<endl;
}

bool videoListType::videoCheckTitle(string vTitle)
{
    bool found = false;
    nodeType<videoType> *location;

    searchVideoList(vTitle, found, location); //search the list

    return found;
}

```

```

void videoListType::videoUpdateInStock(string vTitle, int num)
{
    bool found = false;
    nodeType<videoType> *location;

    searchVideoList(vTitle, found, location); //search the list

    if(found)
        location->info.updateInStock(num);
    else
        cout<<"The store does not carry this video."<<endl;
}

void videoListType::videoSetCopiesInStock(string vTitle, int num)
{
    bool found = false;
    nodeType<videoType> *location;

    searchVideoList(vTitle, found, location);

    if(found)
        location->info.setCopiesInStock(num);
    else
        cout<<"The store does not carry this video."<<endl;
}

bool videoListType::videoSearch(string vTitle)
{
    bool found = false;
    nodeType<videoType> *location;

    searchVideoList(vTitle, found, location);

    return found;
}

void videoListType::videoPrintTitle()
{
    nodeType<videoType>* current;

    current = first;
    while(current != NULL)
    {
        current->info.printTitle();
        current = current->link;
    }
}

```

الجزء 2: مكون الزبائن:

هدف الزبائن: الخصائص الأساسية للزبون هي:

- الاسم الأول للزبون.
- الاسم الأخير للزبون.
- رقم حساب الزبون.
- قائمة الشروط المؤجرة.

افترض أننا نريد فقط معرفة عدد الشروط المؤجرة بواسطة زبون ما بدلاً من بحث قائمة الشروط المؤجرة (الذي قد يأخذ وقت طويل إذا كانت القائمة بها العديد من المدخلات) نقوم بإضافة عنصر رابع لهدف الزبون وهو عدد الايجارات.

كل زبون عبارة عن شخص وقد قمنا بالفعل بتصميم الفئة `personType` في المثال 1-5 في الفصل 1 وقمنا بوصف العمليات الضرورية المؤداة على اسم الشخص. لهذا يمكننا اشتقاق الفئة `customerType` من الفئة `personType` وإضافة العناصر الإضافية التي نحتاجها. بالرغم من هذا نقوم أولاً بإعادة تعريف الفئة `personType` للحصول على مميزات الخصائص الجديدة للتصميم القائم على الهدف الذي تعلمته مثل ائصال المعامل ثم اشتقاق الفئة `customerType`.
العمليات الأساسية المؤداة على هدف من النوع `personType` هي:

1. تحديد الاسم.
2. طباعة الاسم.
3. اظهار الاسم الأول.
4. اظهار الاسم الأخير.

بالمثل تكون العمليات الأساسية المؤداة على هدف من النوع `customerType` هي:

1. تحديد الاسم، ورقم الحساب، وعدد الايجارات.
2. طباعة الاسم، ورقم الحساب، وعدد الايجارات.
3. تأجير شريط أي اضافته الى قائمة الشروط المؤجرة.
4. اعادة شريط أي حذفه من قائمة الشروط المؤجرة.
5. اظهار رقم الحساب.

يتم ترك تفاصيل تطبيق مكون الزبون كتمرين لك. (انظر تمرين البرمجة رقم 14 في نهاية هذا الفصل).

البرنامج الأساسي:

نقوم الآن بكتابة البرنامج الأساسي لاختبار هدف الشروط. نفترض أن البيانات اللازمة للشروط يتم تخزينها في ملف ما ونفتح الملف ونصنع قائمة للشروط التي يمتلكها متجر الفيديو. البيانات الموجودة في ملف المدخلات تكون في الصيغة التالية:

عنوان الشريط (اسم الفيلم)

نجم الفيلم الأول

نجم الفيلم الثاني

منتج الفيلم

مخرج الفيلم

شركة انتاج الفيلم

عدد النسخ

.

.

.

نكتب دالة وهي `createVideoList` لقراءة البيانات من ملف المدخلات وعمل قائمة بالشرائط. كما نقوم بكتابة دالة وهي `displayMenu` لظهور الاختيارات المختلفة – مثل ادخال أو اخراج شريط – التي يمكن للمستخدم عملها. الحل الحسابي للدالة `main` هو:

1. افتح ملف المدخلات.
 2. اذا لم يكن ملف المدخلات موجود اخرج من البرنامج.
 3. اصنع قائمة شرائط (`createVideoList`).
 4. اظهر القائمة (`displayMenu`).
 5. لم يتم عمل `while` قم بأداء العمليات المتنوعة.
- فتح ملف المدخلات واضح. لنقم بوصف الخطوات 2 و3 ويتم عملهما بكتابة دالتين منفصلتين وهما:
- `createVideoList` و `displayMenu`.

createVideoList: تقوم هذه الدالة بقراءة البيانات من ملف المدخلات وتصنع قائمة متصلة للشرائط. بما أن البيانات تتم قراءتها من ملف وتم فتح ملف المدخلات في الدالة `main` فاننا نقوم بتمرير مؤشر ملف المدخلات لهذه الدالة كما نقوم بتمرير مؤشر قائمة الشرائط التي تم اعلانها في الدالة `main` لهذه الدالة. كلا هذين المعاملين معاملان اشارة. بعد هذا نقرأ البيانات الخاصة بكل شريط ثم ندخل الشريط في القائمة. الحل الحسابي العام هو:

أ. اقرأ البيانات وقم بتخزينها في هدف الشرائط.

ب. ادخل الشريط في القائمة.

ت. أعد الخطوتين الأولى والثانية لكل شريط في الملف.

displayMenu: تقوم هذه الدالة بابلاغ المستخدم عما يفعل وتحتوي على بيانات المخرجات التالية.

اختر واحدة مما يلي:

1: للتحقق من وجود عنوان محدد في المخزن.

- 2: لاخراج شريط.
 - 3: لادخال شريط.
 - 4: للتحقق من وجود شريط محدد في المخزن.
 - 5: لطباعة عناوين جميع الشرائط.
 - 6: لطباعة قائمة بجميع الشرائط.
 - 9: للخروج
- في الكود اللاحق تكون الخطوة رقم 4:

```
get choice
while (choice != 9)
{
    switch(choice)
    {
```

- الحالة 1: أ. احصل على اسم الفيلم.
 ب. ابحث قائمة الشرائط
 ج. اذا تم العثور عليه انتج "نجاح"
 بخلاف هذا انتج "فشل"
- الحالة 2: أ. احصل على اسم الفيلم.
 ب. ابحث قائمة الشرائط
 ج. اذا تم العثور عليه قم باخراج الشريط
 بخلاف هذا انتج "فشل"
- الحالة 3: أ. احصل على اسم الفيلم.
 ب. ابحث قائمة الشرائط
 ج. اذا تم العثور عليه قم بادخال الشريط
 بخلاف هذا انتج "فشل"
- الحالة 4: أ. احصل على اسم الفيلم.
 ب. ابحث قائمة الشرائط
 ج. اذا تم العثور عليه
 اذا كان عدد النسخ < صفر
 انتج "نجاح"
 بخلاف هذا انتج لخارج المخزن حالياً "
 بخلاف هذا انتج "فشل"
- الحالة 5: اطبع عناوين الشرائط
- الحالة 6: اطبع جميع الشرائط الموجودة في المتجر.
 الافتراضي: اختيار سيئ

```
    }//end switch
    displayMenu();
    get choice;
} //end while
```



```

        cout<<"Enjoy your movie: "<<title<<endl;
    }
    else
        cout<<"The video is currently "
            <<"out of stock."<<endl;
    }
    else
        cout<<"The video is not in the store."
            <<endl;
    break;
case 3: cout<<"Enter the title: ";
        getline(cin,title);
        cout<<endl;
        if(videoList.videoSearch(title))
        {
            videoList.videoCheckIn(title);
            cout<<"Thanks for returning "<<title
                <<endl;
        }
        else
            cout<<"This video is not from our store."
                <<endl;

        break;
case 4: cout<<"Enter the title: ";
        getline(cin,title);
        cout<<endl;
        if(videoList.videoSearch(title))
        {
            if(videoList.isVideoAvailable(title))
                cout<<"The video is currently in stock."
                    <<endl;
            else
                cout<<"The video is out of stock."
                    <<endl;
        }
        else
            cout<<"The video is not in the store."
                <<endl;

        break;
case 5: videoList.videoPrintTitle();
        break;
case 6: cout<<videoList<<endl;
        break;
default: cout<<"Bad Selection"<<endl;
} //end switch

```

```

        displayMenu();                //display the menu
        cout<<"Enter your choice: ";
        cin>>choice;                  //get the next request
        cin.get(ch);
        cout<<endl;
    }//end while

    return 0;
}

void createVideoList(ifstream& infile, videoListType& videoList)
{
    string Title;
    string Star1;
    string Star2;
    string Producer;
    string Director;
    string ProductionCo;
    char ch;
    int InStock;

    videoType newVideo;

    getline(infile, Title);
    while(infile)
    {
        getline(infile, Star1);
        getline(infile, Star2);
        getline(infile, Producer);
        getline(infile, Director);
        getline(infile, ProductionCo);
        infile>>InStock;
        infile.get(ch);
        newVideo.setVideoInfo(Title, Star1, Star2, Producer,
                               Director, ProductionCo, InStock);
        videoList.insertFirst(newVideo);

        getline(infile, Title);
    }//end for
}

void displayMenu()
{
    cout<<"Select one of the following "<<endl;
    cout<<"1: To check whether a particular video is in the store"
        <<endl;
    cout<<"2: To check out a video"<<endl;
    cout<<"3: To check in a video"<<endl;
    cout<<"4: To check whether a particular title is in stock"
        <<endl;

    cout<<"5: To print the titles of all the videos"<<endl;
    cout<<"6: To print a list of all the videos"<<endl;
    cout<<"9: To exit"<<endl;
}

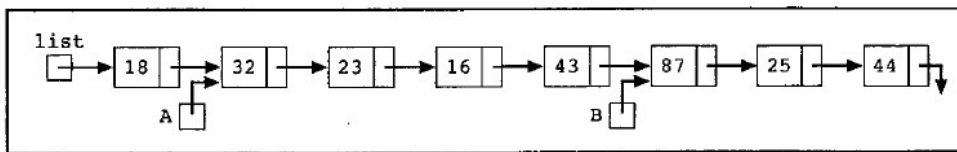
```

1. القائمة المتصلة عبارة عن قائمة من العناصر تسمى عقد يتم فيها تحديد ترتيب العقد عن طريق العنوان المسمى الوصلة والذي يكون مخزن في كل عقدة.
2. المؤشر الى قائمة متصلة – أي المؤشر الى العقدة الأولى في القائمة – يتم تخزينه في موقع منفصل يسمى الأساس أو الأول.
3. القائمة المتصلة عبارة عن بنية بيانات حركية.
4. طول القائمة المتصلة هو عدد العقد الموجودة في القائمة.
5. اذ حال عنصر وحذف عنصر من قائمة متصلة لا يتطلب نقل البيانات ويتم فقط تعديل المؤشرات.
6. يتم اجتياز القائمة المتصلة (المفردة) في اتجاه واحد فقط.
7. البحث في القائمة المتصلة يكون متتابع.
8. المؤشر الأول (first) أو (head) للقائمة المتصلة يكون ثابت دائماً ومشيراً الى العقدة الأولى في القائمة.
9. لاجتياز قائمة متصلة يجب أن يقوم البرنامج باستخدام مؤشر مختلف عن المؤشر الرئيسي (head) للقائمة مهيئ عند العقدة الأولى في القائمة.
10. في القائمة المتصلة من الطرفين يكون لكل عقدة وصلتان: وصلة تشير الى العقدة التالية ووصلة تشير الى العقدة السابقة.
11. يمكن اجتياز القائمة المتصلة من الطرفين في اي اتجاه.
12. في القائمة المتصلة من الطرفين يتطلب ادخال وحذف العنصر تعدسل المؤشرين الموجودين في العقدة.
13. حاويات القوائم يتم استخدامها كقوائم متصلة من الطرفين ولهذا يشير كل عنصر في القائمة الى العنصر الذي يسبقه مباشرة والعنصر الذي يليه مباشرة (فيما عدا العنصران الأول والأخير).
14. اسم الفئة المحتوية على تعريف الفئة list هو list.
15. بالاضافة الى العمليات المشتركة بين الحاويات المتعاقبة (انظر الفصل 4) هناك عمليات أخرى يمكن استخدامها للتحكم في العناصر الموجودة في حاوية القائمة وهي assign، push_front، pop_front، front، back، remove، remove_if، unique، splice، sort، merge، وreverse.
16. القائمة المتصلة ذات العقدتين header و trailer تقوم بتبسيط عمليات الادخال والحذف.

17. في القائمة المتصلة ذات العقدتين header و trailer لا تكون هاتان العقدتان جزء من القائمة الحقيقية. عناصر القائمة الحقيقية تكون بين العقدتان header و trailer.
18. القائمة المتصلة ذات العقدتين header و trailer تكون فارغة اذا كانت العقد الوحيدة الموجودة في القائمة هي header و trailer.
19. القائمة المتصلة الدائرية هي القائمة التي تشير فيها العقدة الأخيرة الى العقدة الأولى اذا لم تكن القائمة فارغة.

تمارين

- ضع علامة صح أم خطأ أمام العبارات التالية:
 - أ- في القائمة المتصلة يتم تحديد ترتيب العناصر بواسطة الترتيب الذي تم عمل العقد به لتخزين العناصر.
 - ب- في القائمة المتصلة تكون الذاكرة المخصصة للعقد متتابعة.
 - ت- يمكن اجتياز قائمة متصلة مفردة في أي اتجاه.
 - ث- في القائمة المتصلة يتم دائماً ادخال العقد اما في البداية أو في النهاية لأن القائمة المتصلة ليست بنية بيانات وصول عشوائي.
 - ج- ادخال (وحذف) عنصر في قائمة متصلة بها عقدتان header و trailer يكون أبسط منه في قائمة متصلة عادية لأن القائمة الأولى ليس لها حالات خاصة.
 - ح- يجب ألا يتم استخدام المؤشر head للقائمة المتصلة لاجتياز القائمة.
- انظر الى القائمة المتصلة الموضحة في شكل 5-57 وافترض أن العقد موجودة في الصيغة المعتادة info-link. استخدم هذه القائمة للإجابة عن التمارين من 2 وحتى 7. اذا لزم الأمر قم باعلان متغيرات اضافية. (افترض أن list و p و s و A و B مؤشرات من النوع nodeType).



شكل 5-57: قائمة متصلة من أجل التمارين من 2 وحتى 7.

2. ما هي مخرجات بيانات C++ التالية:

أ. `cout<<list->info;`

ب. `cout<<A->info;`

ج. `cout<<B->link->info;`

د. `cout<<list->link->link->info`

3. ما هي قيمة التعبيرات التالية؟

أ- `list->info >= 18`

ب- `list->link == A`

ت- `A->link->info == 16`

ث- `B->link == NULL`

ج- `list->info == 18`

4. ضع علامة صح أم خطأ أمام البيانات التالية وإذا كان البيان غير صحيح فسر السبب:

أ- `A = B;`

ب- `list->link = A->link;`

ت- `list->link->info = 45;`

ث- `*list = B;`

ج- `*A = *B;`

ح- `B = A->link->info;`

خ- `A->info = B->info;`

د- `list = B->link->link;`

ذ- `B = B->link->link->link;`

5. اكتب بيانات C++ للقيام بما يلي:

أ- جعل A يشير إلى العقدة المحتوية على 23.info.

ب- جعل list تشير إلى العقدة المحتوية على 16.info.

ت- جعل B يشير إلى العقدة الأخيرة في القائمة.

ث- جعل list تشير إلى قائمة فارغة.

ج- تحديد قيمة العقدة المحتوية على من 25 إلى 35.

ح- عمل وادخال العقدة ذات 10.info بعد العقدة المشار إليها بواسطة A.

خ- حذف العقدة ذات 23.info وكذلك إزالة تخصيص الذاكرة التي تشغلها هذه العقدة.

6. ما هي مخرجات كود C++ التالي؟

```
p = List;
while(p != NULL)
    cout<<p->info<<" ";
    p = p->link;
cout<<endl;
```

7. اذا كان الكود التالي من C++ صحيح قم بتوضيح المخرجات واذا كان غير صحيح فسر السبب.
أ.

```
s = A;
p = B;
s->info = B;
p = p->link;
cout<<s->info<<" "<<p->info<<endl;
```

ب.

```
p = A;
p = p->link;
s = p;
p->link = NULL;
s = s->link;
cout<<p->info<<" "<<s->info<<endl;
```

8. وضح ما يتم انتاجه من كود C++ التالي وافترض أن العقدة موجودة بالصيغة المعتادة
info-link حيث info من النوع int. (list ومُشاران من النوع nodeType).

```
list = new nodeType;
list->info = 10;
ptr = new nodeType;
ptr->info = 13;
ptr->link = NULL;
list->link = ptr;
ptr = new nodeType;
ptr->info = 18;
ptr->link = list->link;
list->link = ptr;
cout<<list->info<<" "<<ptr->info<<" ";
ptr = ptr->link;
cout<<ptr->info<<endl;
```

9. وضح ما يتم انتاجه من كود C++ التالي وافترض أن العقدة موجودة بالصيغة المعتادة
info-link حيث info من النوع int. (list ومُشاران من النوع nodeType).

```
list = new nodeType;
list->info = 20;
ptr = new nodeType;
ptr->info = 28;
ptr->link = NULL;
list->link = ptr;
ptr = new nodeType;
ptr->info = 30;
ptr->link = list;
list = ptr;
ptr = new nodeType;
ptr->info = 42;
```

```

ptr->link = list->link;
list->link = ptr;
ptr = list;
while(ptr != NULL)
{
    cout<<ptr->info<<endl;
    ptr = ptr->link;
}

```

10. انظر الى بيانات C++ التالية (الفئة linkedListType كما هي معرفة في هذا الفصل).

```

list.insertFirst(15);
list.insertLast(28);
list.insertFirst(30);
list.insertFirst(2);
list.insertLast(45);
list.insertFirst(38);
list.insertLast(25);
list.deleteNode(30);
list.insertFirst(18);
list.deleteNode(28);
list.deleteNode(12);
cout<<list<<endl;

```

ما هي مخرجات هذا البرنامج؟

11. افترض أن المدخلات هي:

999- 75 48 19 78 36 45 32 4 30 18

ما هي مخرجات كود C++ التالي؟ (الفئة linkedListType كما هي معرفة في هذا الفصل).

```

linkedListType<int> list;
linkedListType<int> copyList;
int num;

cin>>num;
while(num != -999)
{
    if(num % 5 == 0 || num % 5 == 3)
        list.insertFirst(num);
    else
        list.insertLast(num);
    cin>>num;
}

cout<<"List : "<<list<<endl;

copyList = list;

copyList.deleteNode(78);
copyList.deleteNode(35);

cout<<"Copy List = "<<copyList<<endl;

```

12. افترض أن `intList` خاوية قائمة وأن:

`intList = {3, 23, 23, 43, 56, 11, 11, 23, 25}`

وضح `intList` بعد تنفيذ البيان التالي:

`intList.unique ();`

13. افترض أن `intList1` و `intList2` حاويات قائمة وأن:

`intList1 = {3, 58, 78, 85, 6, 15, 93, 98, 25}`

`intList2 = {5, 24, 16, 11, 60, 9}`

وضح `intList1` بعد تنفيذ البيان التالي:

`intList1.splice (intList1.begin (), intList2);`

14. ما هي مخرجات البرنامج التالي؟

```
list<int> intList;
ostream_iterator<int> screen(cout, " ");
list<int>::iterator listIt;

intList.push_back(5);
intList.push_front(23);
intList.push_front(45);
intList.pop_back();
intList.push_back(35);

intList.push_front(0);
intList.push_back(50);
intList.push_front(34);

copy(intList.begin(), intList.end(), screen);
cout<<endl;

listIt = intList.begin();
intList.insert(listIt,76);

++listIt;
++listIt;
intList.insert(listIt,38);

intList.pop_back();

++listIt;
++listIt;
```

```

intList.erase(listIt);
intList.push_front(2 * intList.back());
intList.push_back(3 * intList.front());

copy(intList.begin(), intList.end(), screen);
cout<<endl;

```

15. ارسم شكل لغة التشكيل الموحدة للفئة videoType لمثال البرمجة في هذا الفصل.

16. ترسم شكل لغة التشكيل الموحدة للفئة videoListType لمثال البرمجة في هذا الفصل.

تمارين برمجة

1. (زيارة دفتر العناوين مرة أخرى) تمرين البرمجة رقم 13 يمكنه معالجة حد أقصى قدره 500 مدخل فقط. باستخدام القوائم المتصلة أعد عمل البرنامج لمعالجة كمية مدخلات حسب الحاجة وأضف العمليات التالية الى برنامجك:

أ- أضف أو احذف مدخل جديد الى دفتر العناوين.

ب- عندما يتوقف البرنامج اكتب البيانات الموجودة في دفتر العناوين على قرص.

2. قم بمد الفئة linkedListType عن طريق اضافة العمليات التالية:

أ- ايجاد وحذف العقدة ذات المعلومات الأصغر في القائمة. (احذف الظهور الأول فقط واجتاز القائمة مرة واحدة فقط).

ب- ايجاد وحذف جميع حالات ظهور معلومات محددة من القائمة. (اجتاز القائمة مرة واحدة فقط).

3. قم بمد الفئة linkedListType عن طريق اضافة العمليات التالية:

أ- كتابة دالة تنتج معلومات العنصر رقم k من القائمة المتصلة. اذا لم يتواجد هذا العنصر أيقاف البرنامج.

ب- كتابة دالة تحذف العنصر رقم k في القائمة المتصلة واذا لم يتواجد هذا العنصر اخراج رسالة خطأ.

4. أضف الدالة splitMid لتقسيم قائمة متصلة الى قائمتين فرعيتين ذات حجم متساوي تقريباً (كما

هو موضح هنا. على سبيل المثال افترض أن القائمة المعطاة هي 13 72 89 65 34. ان تكون

القائمتان الفرعيتان هما 13 72 89 و 34 65. بالمثل اذا كانت القائمة الأصلية هي 12 67 34

65 فان القائمتين الفرعيتين هما 12 67 و 34 65.

أ- أضف العملية splitMid الى الفئة linkedListType كما يلي:

```

void splitMid(linkedListType<Type> &sublist);
//This operation splits the given list into two
//sublists of (almost) equal size.
//Precondition: The list must exist.
//Postcondition: first points to the first node,
// and last points to the last node of the first
// sublist. sublist.first points to the first
// node, and sublist.last points to the last node
// of the second sublist.

```

// هذه العملية تقسم القائمة المعطاة الى قائمتين فرعيتين ذات حجم متساوي (تقريباً)
 // شرط مسبق: يجب أن تتواجد القائمة.
 // شرط تالي: تشير first الى العقدة الأولى وتشير last الى العقدة الأولى من
 // القائمة الفرعية الأولى. Sublist.first يشير الى العقدة الأولى ويشير sublist.last الى
 // العقدة الأخيرة من القائمة الفرعية الثانية.

انظر الى البيانات التالية:

```
linkedListType<int> myList;  
linkedListType<int> subList;
```

افترض أن myList تشير الى قائمة بها العناصر 12 89 27 65 34 (بهذا الترتيب). فان البيان:
 myList.splitMid (subList);
 يقسم myList الى قائمتين فرعيتين وتشير myList الى القائمة ذات العناصر 27 65 34 وتشير
 subList الى القائمة الفرعية ذات العناصر 12 89.

ب- اكتب تعريف قالب الدالة لتطبيق العملية splitMid.

5. أضف الدالة splitAt لتقسيم قائمة متصلة معطى معلومات العقدة الموجودة بها.

افترض أن oldList تشير الى القائمة ذات العناصر:

10 18 34 6 28 92 56 48

والقائمة التي يتم تقسيمه عند العقدة ذات 6 info. تكون القائمتين الفرعيتين هما

10 18 34 و 6 28 92 56 48

أ- أضف الدالة التالية الى الفئة linkedListType:

```
void splitAt(linkedListType<Type> &secondList,  
             const Type& item);  
//Splits the list at the node with the info item  
//into two sublists.  
//Precondition: The list must exist.  
//Postcondition: first and last point to the  
// first and last nodes of the first sublist.  
// secondList.first and secondList.last  
// point to the first and last nodes of the  
// second sublist.
```

// تقسم القائمة عند العقدة ذات العنصر info الى قائمتين فرعيتين.

// شرط مسبق: يجب أن تتواجد القائمة.

// شرط تالي: تشير first و last الى العقدتين الأولى والأخيرة والقائمة الفرعية الأولى

// يشير كل من secondList.first و secondList.last الى العقدتين
// الأولى والأخيرة من القائمة الفرعية الثانية.
انظر الى البيانات التالية:

```
linkedListType<int> myList;  
linkedListType<int> otherList;
```

افترض أن myList تشير الى القائمة ذات العناصر 34 65 18 39 27 89 12 (بهذا الترتيب).
البيان:

```
myList.splitAt (otherList, 18);
```

يقسم myList الى قائمتين فرعيتين حيث تشير myList الى القائمة ذات العناصر 34 65 وتشير
otherList الى القائمة الفرعية ذات العناصر 18 39 27 89 12.

ب- اكتب تعريف قالب الدالة لتطبيق العملية splitAt.

6. أ- أضف العملية التالية الى الفئة orderedLinkedListType:

```
void mergeLists(orderedLinkedListType<Type> &list1,  
orderedLinkedListType<Type> &list2);
```

// هذه العملية تصنع قائمة جديدة عن طريق دمج

// عناصر list1 و list2.

// شرط مسبق: كلتا القائمتين list1 و list2 مرتبتين.

// شرط تالي: يشير first الى القائمة المدمجة وكل من

// list1 و list2 فارغتين.

مثال: انظر الى البيانات التالية:

```
orderedLinkedListType<int> newList;  
orderedLinkedListType<int> list1;  
orderedLinkedListType<int> list2;
```

افترض أن list1 تشير الى القائمة ذات العناصر 2 6 7 وتشير list2 الى القائمة ذات العناصر 3 5
8. فان البيان:

```
newList.mergeLists (list1, list2)
```

يصنع قائمة متصلة جديدة تحتوي على العناصر بالترتيب التالي 2 3 5 6 7 8 ويشير الهدف
newList الى هذه القائمة. بعج تنفيذ البيان السابق تكون list1 و list2 فارغتين.

ب- اكتب تعريف قالب الدالة mergeLists لتطبيق العملية mergeLists.

7. الدالة insertNode للفئة orderedLinkedListType لا تتحقق مما اذا كان العنصر الذي يتم
ادخاله موجود بالفعل في القائمة أم لا أي أنها لا تتحقق من المكررات. أعد كتابة تعريف

insertNode جتى تتحقق قبل ادخال العنصر مما اذا كان موجود بالفعل في القائمة. اذا كان العنصر الذي يتم ادخاله موجود بالفعل في القائمة تقوم الدالة باخراج رسالة خطأ مناسبة. قم كذلك بكتابة برنامج لاختبار دالتك.

8. اكتب تعريف قالب الدالة back لانتاج العنصر الأخير من قائمة متصلة مرتبة. فضلاً عن هذا أضف هذه العملية على الفئة orderedLinkedListType.

9. اكتب تعريفات الدالة copyList ومقوم النسخ والمدمر ودالة ائقال معامل الاسناد للفئة doublyLinkedList.

10. اكتب برنامج لاختبار عمليات متنوعة على الفئة doublyLinkedList.

11. (قائمة متصلة ذات عقد header و trailer). قام هذا الفصل بتعريف وتحديد عمليات متنوعة على القائمة المتصلة ذات العقد header و trailer.

أ- اكتب تعريف الفئة التي تقوم بتعريف قائمة متصلة ذات عقد header و trailer كنوع بيانات مجرد.

ب- اكتب تعريف دالات العنصر للفئة المعرفة في الجزء أ. (يمكنك افتراض أن عناصر القائمة المتصلة ذات العقد header و trailer مرتبة ترتيب تصاعدي).

ت- اكتب برنامج لاختبار العمليات المتنوعة على الفئة المعرفة في الجزء أ.

12. (قوائم متصلة دائرية) هذا الفصل قام بتعريف وتحديد عمليات متنوعة على قائمة متصلة دائرية.

أ- اكتب تعريف الفئة التي تقوم بتعريف قائمة متصلة دائرية مرتبة كنوع بيانات مجرد.

ب- اكتب تعريفات دالات عنصر الفئة المعرفة في الجزء أ. (يمكنك افتراض أن عناصر القائمة المتصلة الدائرية مرتبة ترتيب تصاعدي).

ت- اكتب برنامج لاختبار العمليات المتنوعة على الفئة المعرفة في الجزء أ.

13. اكتب تعريف قالب الدالة seqSearch لتطبيق بحث متتابع على هدف القائمة. نمودجه هو:

```
template<class elemType>
list<elemType>::iterator seqSearch
    (list<elemType> &searchList,
     const elemType& item);
//If the item is found in the list, returns a
//pointer to the item in the list; otherwise,
//returns list.end().
```

قم كذلك بكتابة برنامج لاختبار الدالة seqSearch.

14. (مثال البرمجة، متجر الفيديو)

أ- اكمل تصميم وتطبيق الفئة customerType التي تم تعريفها في مثال البرمجة الخاص بمتجر الفيديو.

ب- قم بتصميم وتطبيق الفئة `customerListType` لعمل والحفاظ على قائمة بعملاء متجر الفيديو.

15. (مثال البرمجة، متجر الفيديو) اكمل تصميم وتطبيق برنامج متجر الفيديو.

16. أعد عمل برنامج متجر الفيديو حتى يتم الاحتفاظ بقائمة الشرائط، وقائمة الزبائن، وقائمة الشرائط المؤجرة بواسطة زبون في حاوية قائمة.

الفصل
6
الاستدعاء الذاتي

في هذا الفصل سوف:

***** ■
***** ■

حتى الآن قمنا باستخدام الأسلوب الأكثر شيوعاً المسمى المكرر بالنسبة الى حلول المشكلة. بالرغم من هذا وفي مشكلات معينة يكون استخدام أسلوب المكررات للحصول على حل معقد للغاية. يقوم هذا الفصل بتقديم أسلوب آخر لحل المشكلات وهو الاستدعاء الذاتي ويوفر العديد من الأمثلة لتوضيح كيفية عمل الاستدعاء الذاتي.

تعريفات تكرارية:

عملية حل المشكلة عن طريق تقليلها الى صور أصغر تسمى التكرار. التكرار عبارة عن طريقة شديدة القوة لحل مشكلات محددة قد يكون حلها بدون ذلك معقد للغاية. لنقم بالنظر الى مشكلة مألوفة بالنسبة الى كل فرد تقريباً .

في الرياضيات يتم تعريف معامل العدد الصحيح كما يلي:

$$0! = 1 \quad (6-1)$$

$$n! = n \times (n - 1)! \text{ if } n > 0 \quad (6-2)$$

في هذا التعريف يتم تعريف $0!$ بأنها 1 ، وإذا كانت n عدد صحيح أكبر من صفر فاننا أولاً نجد $(n-1)!$ ثم نضربها في n . لايجاد $(n-1)!$ نقوم بتطبيق التعريف مرة أخرى. اذا كانت $(n-1) > 0$ فاننا نستخدم المعادلة رقم 6-2 وبخلاف هذا نستخدم المعادلة 6-1. لهذا وبالنسبة الى عدد صحيح n أكبر من صفر يتم الحصول على $n!$ عن طريق ايجاد $(n-1)!$ أولاً (أي يتم تقليل $n!$ الى صورة أصغر منها) ثم نقوم بضرب $(n-1)!$ في n .

لنقم بتطبيق هذا التعريف لايجاد $3!$. هنا $n = 3$ وبما أن $n > 0$ فاننا نستخدم المعادلة 6-2 لكي نحصل على:

$$3! = 3 \times 2!$$

بعد هذا نوجد $2!$ وهنا $n = 2$ وبما أن $n > 0$ نستخدم المعادلة 6-2 للحصول على:

$$2! = 2 \times 1!$$

الآن لايجاد $1!$ نستخدم مرة أخرى المعادلة 6-2 لأن $n = 1 > 0$. لهذا:

$$1! = 1 \times 0!$$

أخيراً نستخدم المعادلة 1-6 لايجاد 0! وهي 1. استبدال 0! في 1! يعطي 1! = 1. وهذا يعطي 2! = 2 × 1! = 2 × 1 = 2 وهذا بدوره يعطي 3! = 3 × 2! = 3 × 2 = 6. الجل في المعادلة 1-6 مباشر – أي أن الجانب الأيمن من المعادلة لا يحتوي على عوامل. الحل في المعادلة 2-6 معطى على أساس صورة أصغر منه. تعريف العامل كما هو معطى في المعادلتين 1-6 و 2-6 يسمى **تعريف تكراري** وتسمى المعادلة 1-6 **المعادلة الأساسية** (الحالة التي يتم فيها اعطاء الحل مباشرة) والمعادلة 2-6 تسمى **الحالة العامة أو حالة تكرارية**.

تعريف تكراري: تعريف يتم فيه تعريف شئ ما على أساس صورة أصغر منه. من المثال السابق يكون من الواضح أن:

- كل تعريف تكراري يجب أن يكون له حالة أساسية واحدة (أو أكثر).
- الحالة العامة يجب أن يتم اختزالها الى حالة أساسية.
- الحالة الأساسية توقف التكرار.

مفهوم التكرار في علوم الحاسب الآلي يعمل بالمثل. هنا نتحدث عن الحلول الحسابية التكرارية والدالات التكرارية. الحل الحسابي الذي يجد الحل لمشكلة محددة عن طريق اختزال المشكلة الى صور أصغر منها يسمى **حل حسابي تكراري**. الحل الحسابي التكراري يجب أن يكون له حالة أساسية واحدة أو أكثر ويجب أن يتم اختزال الحل العام الى حالة أساسية.

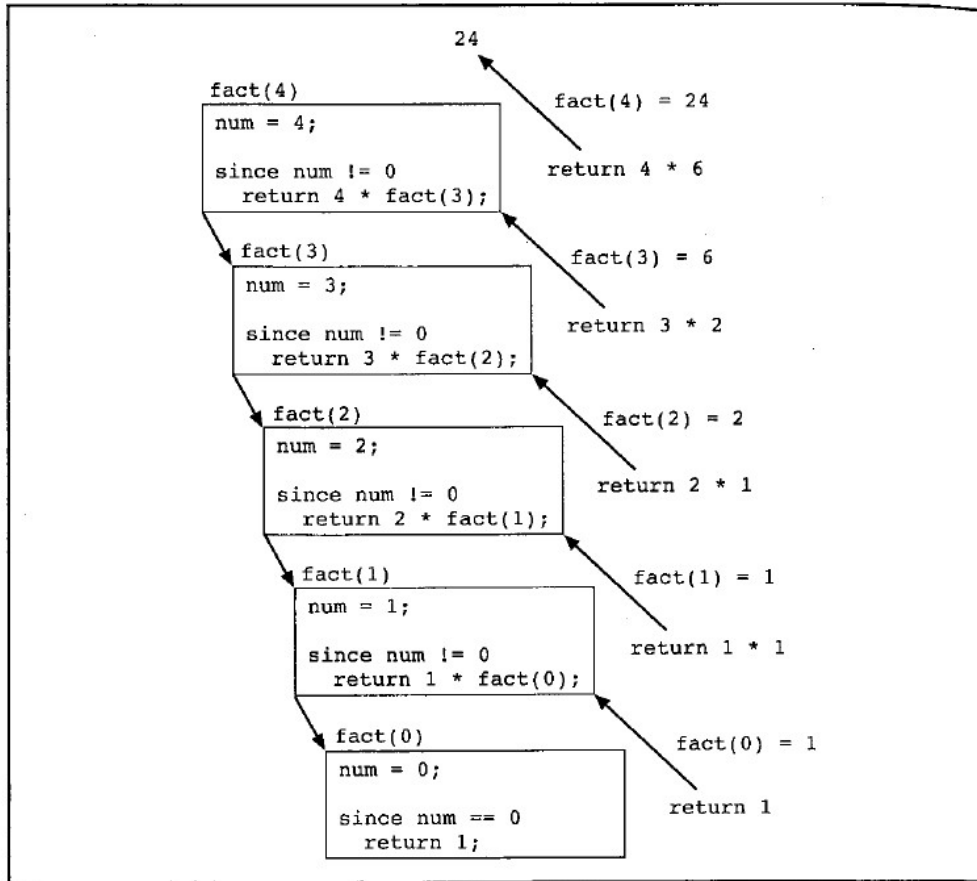
الدالة التي تستدعي نفسها تسمى **دالة تكرارية**. هذا يعني أن هيكل الدالة التكرارية يحتوي على بيان يتسبب في تنفيذ نفس الدالة قبل اكمال الاستدعاء الحالي. يتم تطبيق الحلول الحسابية التكرارية باستخدام الدالات التكرارية.

بعد هذا لنقم بكتابة الدالة التكرارية التي تطبق دالة المعامل.

```
int fact(int num)
{
    if(num == 0)
        return 1;
    else
        return num * fact(num - 1);
}
```

الشكل 1-6 يتتبع تنفيذ البيان التالي:

```
cout<<fact(4)<<endl;
```



شكل 6-1: تنفيذ التعبير fact (4)

مخرجات البيان cout السابق هي:

24

في شكل 6-1 تمثل الأسهم المتجهة الى الأسفل الاستدعاءات المتتالية للدالة fact والأسهم المتجهة الى أعلى تمثل القيم التي يتم ارجاعها الى المستدعي أي دالة الاستدعاء. أثناء تتبع تنفيذ الدالة التكرارية fact لنقم بملاحظة ما يلي:

- منطقياً يمكنك التفكير في الدالة التكرارية بأنها تمتلك نسخ غير محدودة من نفسها.
- كل استدعاء الى دالة تكرارية – أي كل استدعاء تكراري – له كوده الخاص به ومجموعته الخاصة من المعاملات والمتغيرات المحلية.
- بعد اكمال استدعاء تكراري محدد يعود التحكم الى البيئة الداعية وهي الاستدعاء السابق. الاستدعاء (التكراري) التالي يجب أن يتم تنفيذه كاملاً قبل عودة التحكم الى الاستدعاء السابق. التنفيذ في الاستدعاء السابق يبدأ من النقطة التي تلي الاستدعاء التكراري مباشرة.

التكرار المباشر وغير مباشر:

تسمى الدالة تكرارية بشكل مباشر اذا استدعت نفسها. الدالة التي تستدعي دالة أخرى وتتسبب في النهاية في استدعاء الدالة الأساسية يقال أنها تكرارية بشكل غير مباشر. على سبيل المثال اذا قامت الدالة أ باستدعاء الدالة ب والدالة ب تستدعي الدالة أ اذن تكون الدالة أ تكرارية بشكل غير مباشر. التكرار الغير مباشر قد يكون على عمق عدة مراحل. على سبيل المثال افترض أن الدالة أ تستدعي الدالة ب والدالة ب تستدعي الدالة ج والدالة ج تستدعي الدالة د والدالة د تستدعي الدالة أ فاذن تكون الدالة أ تكرارية بشكل غير مباشر.

التكرار الغير مباشر يتطلب التحليل الحريص نفسه مثل التكرار المباشر. يجب أن تكون الحالات الأساسية محددة ويجب توفير حلول مناسبة لها. بالرغم من هذا قد يكون التتبع عن طريق التكرار الغير مباشر عملية شاقة. لهذا يجب أن تمارس المزيد من الحرص عند تصميم دالات تكرارية غير مباشرة. من أجل البساطة تتضمن المشكلات الموجودة في هذا الكتاب التكرار المباشر فقط. الدالة التكرارية التي يكون فيها البيان الأخير الذي يتم تنفيذه هو الاستدعاء التكراري تسمى دالة تكرارية ذيلية. الدالة fact مثال على الدالة التكرارية الذيلية.

التكرار اللانهائي:

شكل 6-1 يوضح أن تتابع الاستدعاءات التكرارية وصل في النهاية الى استدعاء لم يقم بعمل المزيد من الاستدعاءات التكرارية. هذا يعني أن تتابع الاستدعاءات التكرارية وصل في النهاية الى حالة أساسية. على الجانب الآخر اذا كان كل استدعاء تكراري يتسبب في استدعاء تكراري آخر اذن يقال أن الدالة التكرارية بها تكرار لا نهائي. نظرياً يتم تنفيذ التكرار النهائي الى الأبد. كل استدعاء الى دالة تكرارية يقتضي من النظام تخصيص ذاكرة للمتغيرات المحلية والمعاملات الرسمية. كما يقوم النظام بحفظ المعلومات حتى يمكن تحويل التحكم مرة أخرى الى المستدعي الصحيح بعد انتهاء الاستدعاء. لهذا وبما أن ذاكرة الحاسب الآلي محدودة اذا قمت بتنفيذ دالة تكرارية لا نهائية على الحاسب الآلي فان الدالة سوف يتم تنفيذها حتى تنفذ ذاكرة النظام وتتسبب في ايقاف غير عادي للبرنامج. يجب أن يتم تصميم وتحليل الدالات (الحلول الحسابية) التكرارية بحرص ويجب أن تتأكد من أن كل استدعاء تكراري يتم اختزاله في النهاية الى حالة أساسية. هذا الفصل يقدم عدة أمثلة توضح كيفية تصميم وتطبيق الحلول الحسابية التكرارية.

لتصميم دالة تكرارية يجب عليك القيام بما يلي:

أ- قم باستيعاب مقتضيات المشكلة.

ب- قم بتحديد الشروط القاصرة. على سبيل المثال يتم تحديد الشرط القاصر للقائمة عن طريق

عدد العناصر الموجودة في القائمة.

ت- قم بتحديد الحالات الأساسية وتقديم حل مباشر لكل حالة أساسية.

ث- قم بتحديد الحالات العامة وتقديم حل لكل حالة عامة على أساس صورة أصغر منها.

حل المشكلة باستخدام التكرار:

الأمثلة من 1-6 وحتى 5-6 توضح كيفية عمل الحلول الحسابية التكرارية وكيفية تطبيقها في C++ باستخدام الدالات التكرارية.

مثال 1-6: العنصر الأكبر في المصفوفة:

يقوم هذا المثال باستخدام حل حسابي تكراري لايجاد العنصر الأكبر في المصفوفة. انظر الى القائمة المعطاة في شكل 2-6.

| | [0] | [1] | [2] | [3] | [4] | [5] |
|------|-----|-----|-----|-----|-----|-----|
| list | 5 | 8 | 2 | 10 | 9 | 4 |

شكل 2-6: قائمة بها ستة عناصر

العنصر الأكبر في القائمة المعطاة في شكل 2-6 هو 10.

افترض أن list هو اسم المصفوفة المحتوية على عناصر القائمة وافترض كذلك أن list [b]...list [a] تمثل عناصر المصفوفة list [a]، وlist [a+1]، و....، وlist [b]. على سبيل المثال list [0]...list [5] تمثل عناصر المصفوفة list [0]، وlist [1]، وlist [2]، وlist [3]، وlist [4]، وlist [5]. بالمثل list [1]...list [5] تمثل عناصر المصفوفة list [1]، وlist [2]، وlist [3]، وlist [4]، وlist [5]. لكتابة حل حسابي تكراري لايجاد أكبر عنصر في list لنقم بالتفكير على أساس التكرار.

إذا كانت list طولها 1 فإنها تحتوي على عنصر واحد فقط وهو أكبر عنصر. افترض أن طول list أكبر من 1. لايجاد أكبر عنصر في list [a]...list [b] نقوم أولاً بايجاد العنصر الأكبر في list [a]...list [b] ثم نقارن هذا العنصر الأكبر مع list [a]. هذا يعني أنه يتم اعطاء أكبر عنصر في list [a]...list [b] عن طريق:

$\text{maximum (list [a], largest (list [a + 1]...list [b]))}$

لنقم بتطبيق هذه الصيغة لايجاد أكبر عنصر في القائمة الموضحة في شكل 2-6. هذه القائمة بها ستة عناصر معطاة بواسطة list [0]...list [5]. العنصر الأكبر في list هو:

$\text{maximum (list [0], largest (list [1]...list [5]))}$

هذا يعني أن أكبر عنصر في list هو الحد الأقصى من list [0] والعنصر الأكبر في list [5]...list [1]. لايجاد أكبر عنصر في list [1]...list [5] نقوم باستخدام نفس الصيغة مرة أخرى لأن طول هذه القائمة أكبر من 1. أكبر عنصر في list [1]...list [5] يكون اذن:

$\text{maximum (list [1], largest (list [2]...list [5]))}$ وهكذا.

اننا نرى أنه في كل وقت نستخدم فيه الصيغة السابقة لإيجاد أكبر عنصر في قائمة فرعية يتم اختزال طول القائمة الفرعية في الاستدعاء التالي بمقدار واحد. أخيراً يكون طول القائمة الفرعية 1 وفي هذه الحالة تحتوي القائمة الفرعية على عنصر واحد فقط وهو أكبر عنصر في القائمة الفرعية. انطلاقاً من هذه النقطة نقوم بالتتبع من خلال الاستدعاءات التكرارية. تتم ترجمة هذه المناقشة الى الحل الحسابي التكراري التالي الذي يتم تمثيله في الكود الوهمي:

الحالة الأساسية: حجم القائمة هو 1

العنصر الوحيد في القائمة هو أكبر عنصر.

الحالة العامة: حجم القائمة أكبر من 1

إيجاد أكبر عنصر في $list[a] \dots list[b]$

أ- اوجد أكبر عنصر في $list[a+1] \dots list[b]$ واطلق عليها max.

ب- قارن العناصر $list[a]$ و max:

إذا كانت $(list[a] \geq max)$

فان أكبر عنصر في $list[a] \dots list[b]$ يكون هو $list[a]$.

وبخلاف هذا

فان أكبر عنصر في $list[a] \dots list[b]$ يكون هو max.

تتم ترجمة هذا الحل الحسابي الى دالة C++ التالية لإيجاد أكبر عنصر في المصفوفة:

```
int largest(const int list[], int lowerIndex, int upperIndex)
{
    int max;

    if(lowerIndex == upperIndex)    //the size of the sublist is 1
        return list[lowerIndex];
    else
    {
        max = largest(list, lowerIndex + 1, upperIndex);
        if(list[lowerIndex] >= max)
            return list[lowerIndex];
        else
            return max;
    }
}
```

انظر -

| | | | | |
|------|-----|-----|-----|-----|
| | [0] | [1] | [2] | [3] |
| list | 5 | 10 | 12 | 8 |

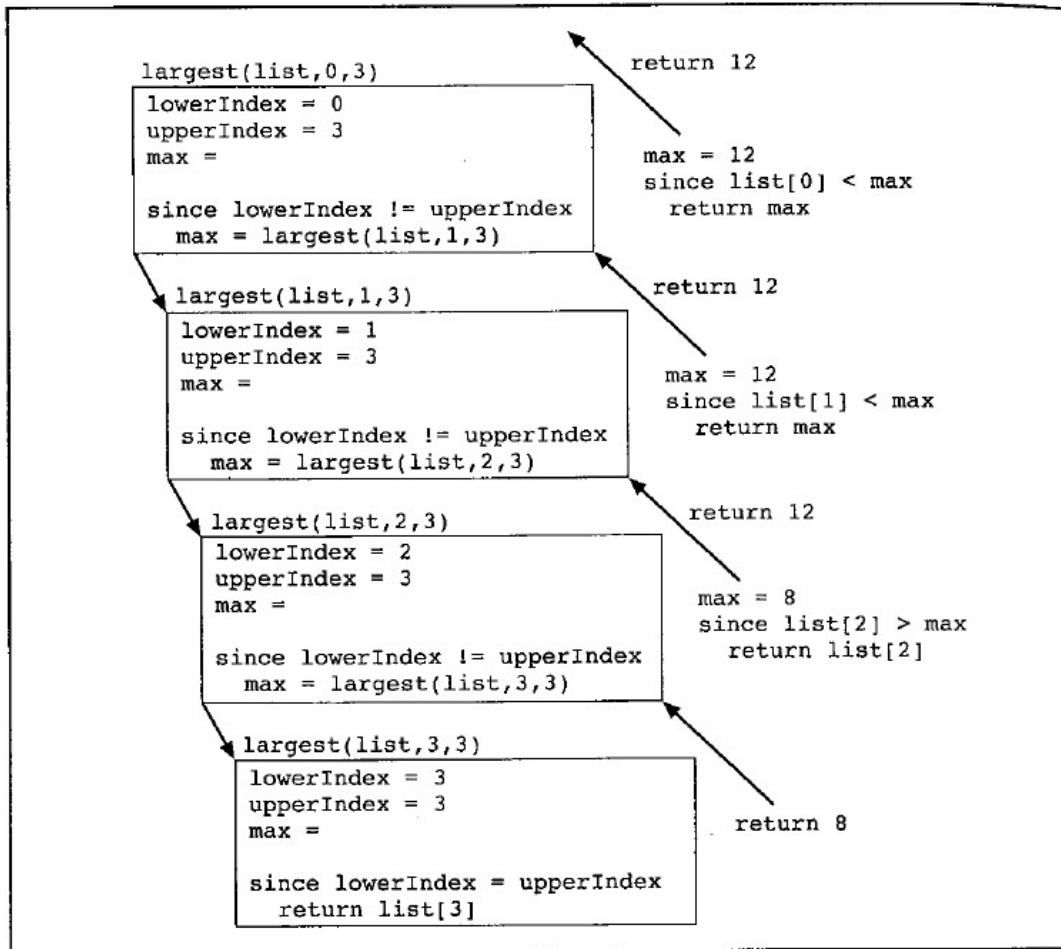
شكل 6-3: قائمة ذات أربعة عناصر

لنقم بتتبع تنفيذ البيان التالي:

`cout<<largest (list, 0, 3);`

هنا `upperindex = 3` والقائمة بها أربعة عناصر. الشكل 4-6 يتتبع تنفيذ

`largest (list, 0, 3)`.



شكل 4-6: تنفيذ التعبير `largest (list, 0, 3)`

القيمة الناتجة من التعبير `largest (list, 0, 3)` هي 12 وهي أكبر عنصر في `list`.

برنامج C++ التالي يستخدم الدالة largest لتحديد أكبر عنصر في القائمة:

```
//Largest Element in an Array

#include <iostream>

using namespace std;

int largest(const int list[], int lowerIndex, int upperIndex);

int main()
{
    int intArray[10] = {23, 43, 35, 38, 67, 12, 76, 10, 34, 8};

    cout<<"The largest element in intArray: "
         <<largest(intArray,0,9);
    cout<<endl;
    return 0;
}

int largest(const int list[], int lowerIndex, int upperIndex)
{
    int max;

    if(lowerIndex == upperIndex)    //the size of the sublist is 1
        return list[lowerIndex];
    else
    {
        max = largest(list, lowerIndex + 1, upperIndex);
        if(list[lowerIndex] >= max)
            return list[lowerIndex];
        else
            return max;
    }
}
```

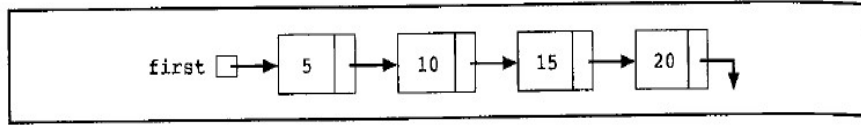
تنفيذ العينة:

العنصر الأكبر في intArray هو: 76

مثال 2-6: طباعة قائمة متصلة بترتيب عكسي:

تكون عقد القائمة المتصلة المرتبة (كما هي موضحة في الفصل 5) مرتبة ترتيباً تصاعدياً. بالرغم من هذا فهناك تطبيقات محددة تقتضي طباعة البيانات بترتيب تنازلي وهذا يعني أنه يجب علينا طباعة القائمة بالعكس. نقوم الآن بمناقشة الدالة reversePrint. باعطاء مؤشر للقائمة تقوم هذه الدالة بطباعة عناصر القائمة بترتيب عكسي.

انظر الى القائمة المتصلة الموضحة في شكل 5-6.



شكل 5-6: قائمة متصلة.

بالنسبة الى الشكل 5-6 يجب أن تكون المخرجات بالصيغة التالية:

5 10 15 20

بما أن الوصلات في اتجاه واحد فقط لا يمكننا اجتياز القائمة الى الخلف بدءاً من العقدة الأخيرة. لنرى كيفية استخدامنا للتكرار بفاعلية لطباعة القائمة بترتيب عكسي.

لنقم بالتفكير على اساس التكرار. لا يمكننا طباعة معلومات كل عقدة حتى نطبع بقية القائمة (أي ذيل العقدة الأولى). بالمثل لا يمكننا طباعة معلومات العقدة الثانية حتى نطبع ذيل العقدة الثانية وهكذا. في كل وقت ننظر فيه الى ذيل عقدة ما نقوم بتقليل حجم القائمة بمقدار 1. أخيراً يتم تقليل حجم القائمة الى صفر وفي هذا الحالة يتوقف التكرار.

الحالة الأساسية: القائمة فارغة

لا تصرف

الحالة العامة: القائمة غير فارغة

أ. طباعة الذيل.

ب. طباعة العنصر

لنقم بكتابة هذا الحل الحسابي في الكود الافتراضي. (افترض أن current مؤشر الى قائمة متصلة).

```
if(current != NULL)
{
    reversePrint(current->link); //print the tail
    cout<<current->info<<" "; //print the node
}
```

هنا لا نرى الحالة الأساسية فهي مختبئة. يتم طبع القائمة فقط اذا كان المؤشر current للقائمة ليس NULL. كذلك يكون الاستدعاء التكرار على ذيل القائمة بداخل البيان if. بما أن ذيل القائمة سوف يصبح فارغاً في النهاية فان البيان if في الاستدعاء التالي يفشل والتكرار يتوقف. لاحظ أيضاً أن البيان لطبع معلومات العقدة يظهر بعد الاستدعاء التكراري وبهذا عند عودة التحويل مرة أخرى الى دالة الاستدعاء يجب أن نقوم بتنفيذ هذا البيان. تذكر أن الدالة تتواجد فقط بعد تنفيذ البيان الأخير. (بالبيان الأخير لا نقصد البيان المادي الأخير ولكن نقصد البيان المنطقي الأخير). لنقم بكتابة قالب دالة لتطبيق الحل الحسابي السابق ثم نطبقه على قائمة ما.

```

template<class Type>
void orderedLinkedListType<Type>::reversePrint
(nodeType<Type> *current) const
{
    if(current != NULL)
    {
        reversePrint(current->link);    //print the tail
        cout<<current->info<<" ";      //print the node
    }
}

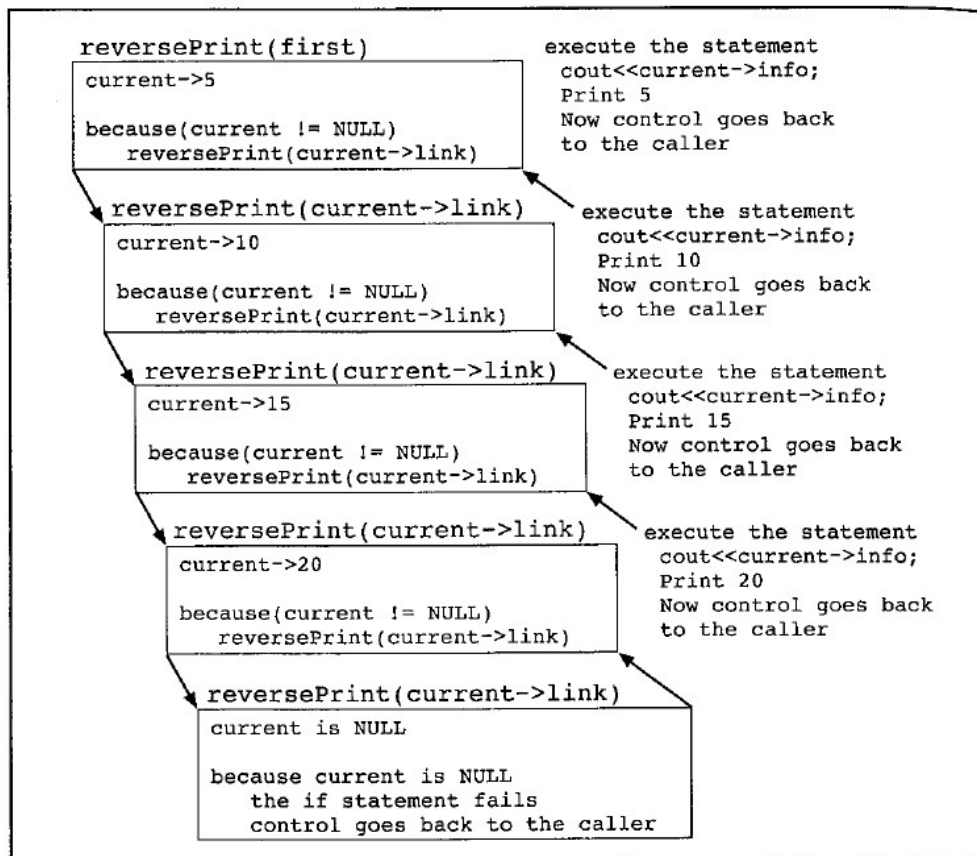
```

انظر الى البيان:

reversePrint(first);

حيث first مؤشر الى النوع <Type>.nodeType

لنقم بتتبع تنفيذ هذا البيان وهو استدعاء دالة للقائمة الموضحة في شكل 6-5. بما أن المعامل الرسمي عبارة عن معامل قيمة فان قيمة المعامل الحقيقي يتم تمريرها الى المعامل الرسمي. انظر شكل 6-6.



شكل 6-6: تنفيذ البيان reversePrint (first);

الدالة printListReverse:

بما أننا قمنا بكتابة الدالة reversePrint يمكننا كتابة تعريف الدالة printListReverse التي يمكن استخدامها لطباعة قائمة متصلة مرتبة موجودة في هدف من النوع orderedLinkedListType. تعريفها يكون:

```
template<class Type>
void orderedLinkedListType<Type>::printListReverse() const
{
    reversePrint(first);
    cout<<endl;
}
```

يمكننا ادخال الدالة printListReverse كعنصر عام في تعريف الفئة والدالة reversePrint كعنصر خاص. اننا نقوم بادخال الدالة reversePrint كعنصر خاص لأنه يتم استخدامه فقط لتطبيق الدالة printListReverse.

مثال 3-6: عدد fibonacci:

انظر الى التتابع التالي من الأعداد:

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

باعطاء العددين الأوليان من التتابع (لنقل a_1 و a_2) فانه يتم الوصول الى العدد رقم n بالنسبة الى $n \geq 3$ من هذا التتابع عن طريق:

$$a_n = a_{n-1} + a_{n-2}$$

بهذا:

$$\begin{aligned} a_3 &= a_2 + a_1 \\ &= 1 + 1 \\ &= 2, \end{aligned}$$

$$\begin{aligned} a_4 &= a_3 + a_2 \\ &= 2 + 1 \\ &= 3, \end{aligned}$$

وهكذا.

هذا التتابع يطلق عليه تتابع fibonacci. في التتابع السابق و $a_1 = 1$ $a_2 = 1$ بالرغم من هذا فانك تستطيع باستخدام هذه العملية وبالحصول على أول عددين أن تقوم بتحديد العدد رقم n وهو a_n حيث $n \geq 3$. العدد الذي يتم تحديده بهذه الطريقة يسمى عدد fibonacci رقم n . افترض أن

و . اذ $a_1 = 3$ $a_2 = 6$

$$\begin{aligned} a_3 &= a_2 + a_1 = 6 + 3 = 9 \\ a_4 &= a_3 + a_2 = 9 + 6 = 15 \end{aligned}$$

نقوم في هذا المثال بكتابة دالة تكرارية هي rFibNum لتحديد رقم fibonacci المرغوب فيه. الدالة rFibNum تأخذ ثلاثة أعداد كعوامل تمثل أول عددين من تتابع Fibonacci وعدد n وعدد n المرجو. الدالة rFibNum تنتج عدد Fibonacci رقم n في التتابع. عدد Fibonacci الثالث في تتابع Fibonacci هو مجموع أول عددين Fibonacci. عدد Fibonacci الرابع في تتابع Fibonacci هو مجموع أعداد Fibonacci الثاني والثالث. لهذا عند حساب رابع عدد Fibonacci نقوم بجمع العدد Fibonacci الثاني والثالث (وهو مجموع أول عددين Fibonacci). الحل الحسابي التكراري التالي يحسب عدد Fibonacci رقم n حيث تعبر a عن أول عدد Fibonacci و b عدد Fibonacci الثاني و n هو عدد Fibonacci رقم n:

$$rFibNum(a,b,n) = \begin{cases} a & \text{if } n = 1 \\ b & \text{if } n = 2 \\ rFibNum(a,b,n-1) + rFibNum(a,b,n-2) & \text{if } n > 2 \end{cases} \quad (6-3)$$

افترض اننا نريد تحديد:

$$rFibNum(2, 5, 4)$$

هنا $a = 2$ و $b = 5$ و $n = 4$. هذا يعني أننا نريد تحديد عدد Fibonacci الرابع من التتابع الذي يكون العدد الأول به هو 2 والعدد الثاني به هو 5. بما أن n هي $4 > 2$ فان:

$$1. \quad rFibNum(2, 5, 4) = rFibNum(2, 5, 3) + rFibNum(2, 5, 2)$$

بعد هذا نقوم بتحديد $rFibNum(2, 5, 3)$ و $rFibNum(2, 5, 2)$. لنقم أولاً بتحديد $rFibNum(2, 5, 3)$. هنا $a = 2$ و $b = 5$ و $n = 3$. بما أن n تساوي 3 اذن:

$$1. \quad rFibNum(2, 5, 3) = rFibNum(2, 5, 2) + rFibNum(2, 5, 1)$$

هذا البيان يقتضي منا تحديد $rFibNum(2, 5, 2)$ و $rFibNum(2, 5, 1)$. في $rFibNum(2, 5, 2)$ تكون $a = 2$ و $b = 5$ و $n = 2$. لهذا من التعريف المعطى في المعادلة 6-3 ينتج أن:

$$1.1. \quad rFibNum(2, 5, 2) = 5$$

لايجاد $rFibNum(2, 5, 1)$ لاحظ أن $a = 2$ و $b = 5$ و $n = 1$. لهذا من التعريف المعطى في المعادلة 6-3،

$$1.2. \quad rFibNum(2, 5, 1) = 2$$

نقوم باستبدال قيم $rFibNum(2, 5, 2)$ و $rFibNum(2, 5, 1)$ داخل (1.1) للحصول على:

$$rFibNum(2, 5, 3) = 5 + 2 = 7$$

بعد هذا نقوم بتحديد $rFibNum(2, 5, 2)$. كما في (1.1) تكون $rFibNum(2, 5, 2) = 5$.

يمكننا استبدال قيم $rFibNum(2, 5, 3)$ و $rFibNum(2, 5, 2)$ داخل (1) للحصول على:

$$rFibNum(2, 5, 4) = 7 + 5 = 12$$

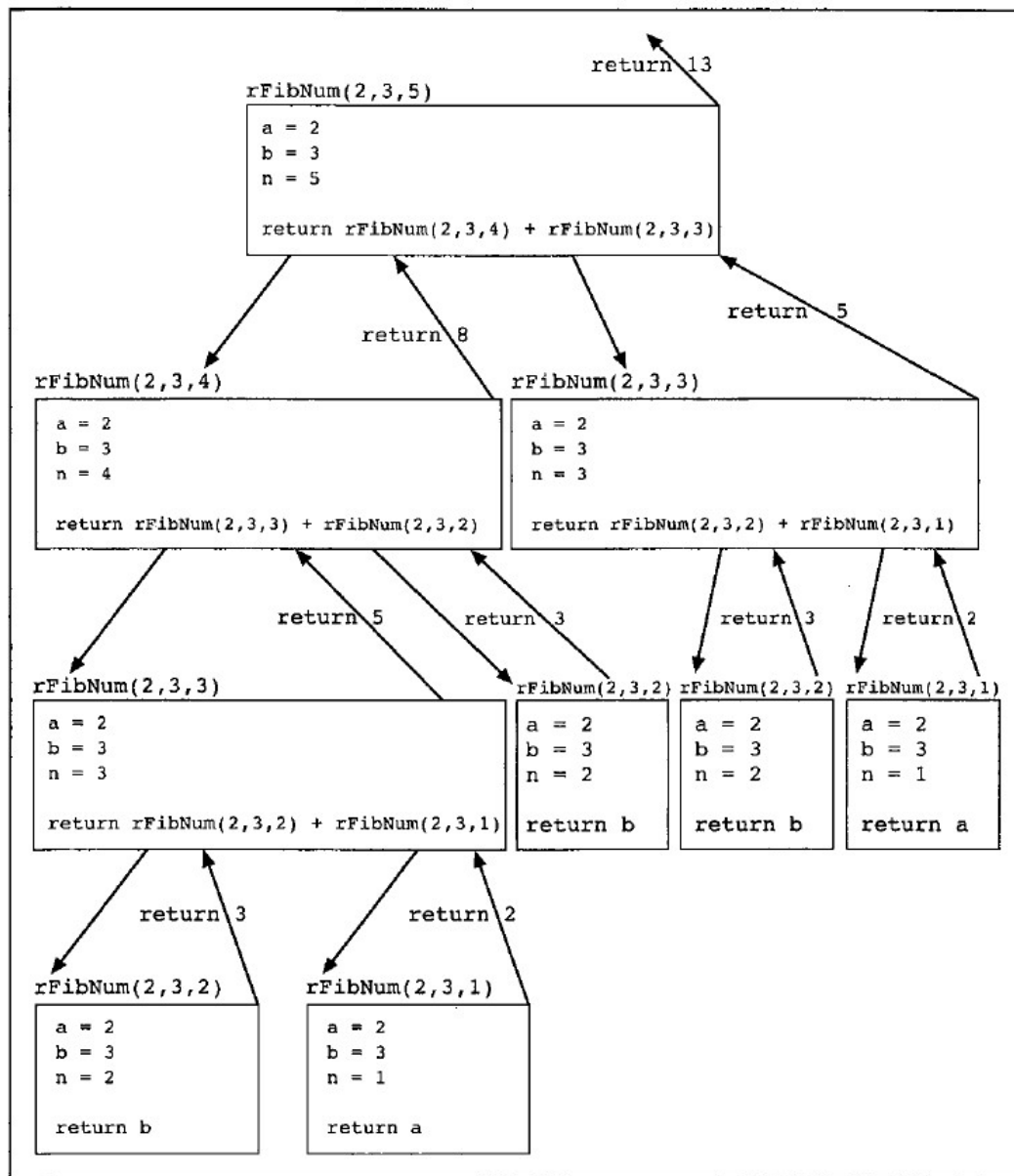
تقوم الدالة التكرارية التالية بتطبيق هذا الحل الحسابي:

```

int rFibNum(int a, int b, int n)
{
    if(n == 1)
        return a;
    else if(n == 2)
        return b;
    else
        return rFibNum(a, b, n - 1) + rFibNum(a, b, n - 2);
}

```

لنقم بتنفيذ البيان التالي: `cout<<rFibNum(2, 3, 5)<<endl`; في هذا البيان العدد الأول هو 2 والعدد الثاني هو 3 ونحن نريد تحديد العدد الخامس من هذا التسلسل. الشكل 6-7 يتتبع تنفيذ التعبير `rFibNum(2, 3, 5)` والقيمة الناتجة هي 13 وهي عدد Fibonacci الخامس من التسلسل الذي يكون فيه العدد الأول هو 2 والعدد الثاني هو 3.



شكل 6-7: تنفيذ التعبير `rFibNum(2, 3, 5)`

برنامج C++ التالي يستخدم الدالة rFibNum:

```
//Chapter 6: Fibonacci Number

#include <iostream>

using namespace std;

int rFibNum(int a, int b, int n);

int main()
{
    int firstFibNum;
    int secondFibNum;
    int nth;

    cout<<"Enter the first Fibonacci number: ";
    cin>>firstFibNum;
    cout<<endl;

    cout<<"Enter the second Fibonacci number: ";
    cin>>secondFibNum;
    cout<<endl;

    cout<<"Enter the position of the desired Fibonacci number: ";
    cin>>nth;
    cout<<endl;

    cout<<"The Fibonacci number at position "<<nth<<" is: "
        << rFibNum(firstFibNum, secondFibNum, nth)<<endl;

    return 0;
}

int rFibNum(int a, int b, int n)
{
    if(n == 1)
        return a;
    else if(n == 2)
        return b;
    else
        return rFibNum(a, b, n - 1) + rFibNum(a, b, n - 2);
}
```

تنفيذ العينة: في تشغيل هذه العينة يتم تظليل مدخلات المستخدم.

تنفيذ العينة 1:

ادخل عدد Fibonacci الأول: 2

ادخل عدد Fibonacci الثاني: 5

العدد Fibonacci في الموقع 6 هو: 31

تنفيذ العينة 2:

ادخل عدد Fibonacci الأول: 3

ادخل عدد Fibonacci الثاني: 4

ادخل موقع عدد Fibonacci المرجو: 6

العدد Fibonacci في الموقع 6 هو: 29

تنفيذ العينة 3:

ادخل عدد Fibonacci الأول: 12

ادخل عدد Fibonacci الثاني: 18

ادخل موقع عدد Fibonacci المرجو: 15

العدد Fibonacci في الموقع 15 هو: 9582

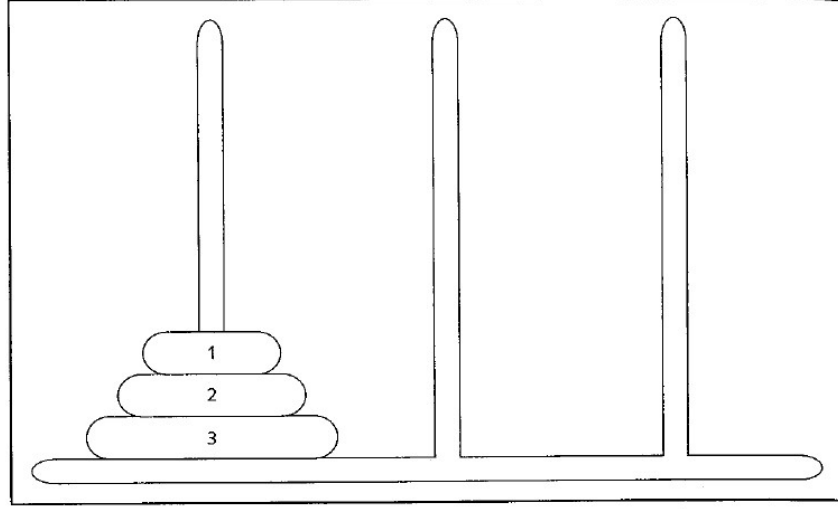
مثال 4-6: برج هانوي:

في القرن التاسع عشر أصبحت اللعبة المسماة برج هانوي منتشرة في أوروبا. هذه اللعبة تمثل عمل جاري في معبد براهما. في خلق الكون تم اعطاء رجال الدين في معبد براهما ثلاثة ابر من الألماظ مع وجود ابرة محتوية على 64 قرص ذهبي. كل قرص ذهبي أصغر قليلاً من القرص الواقع أسفله. مهمة رجل الدين هي نقل جميع ال 64 قرص من الابر الأولى الى الابر الثالثة. قواعد نقل الأقراص تكون كما يلي:

- قرص واحد فقط يمكن تحريكه في المرة الواحدة.
- يجب أن يتم وضع القرص المزال على احدى الابر.
- لا يمكن وضع قرص أكبر فوق قرص أصغر منه.

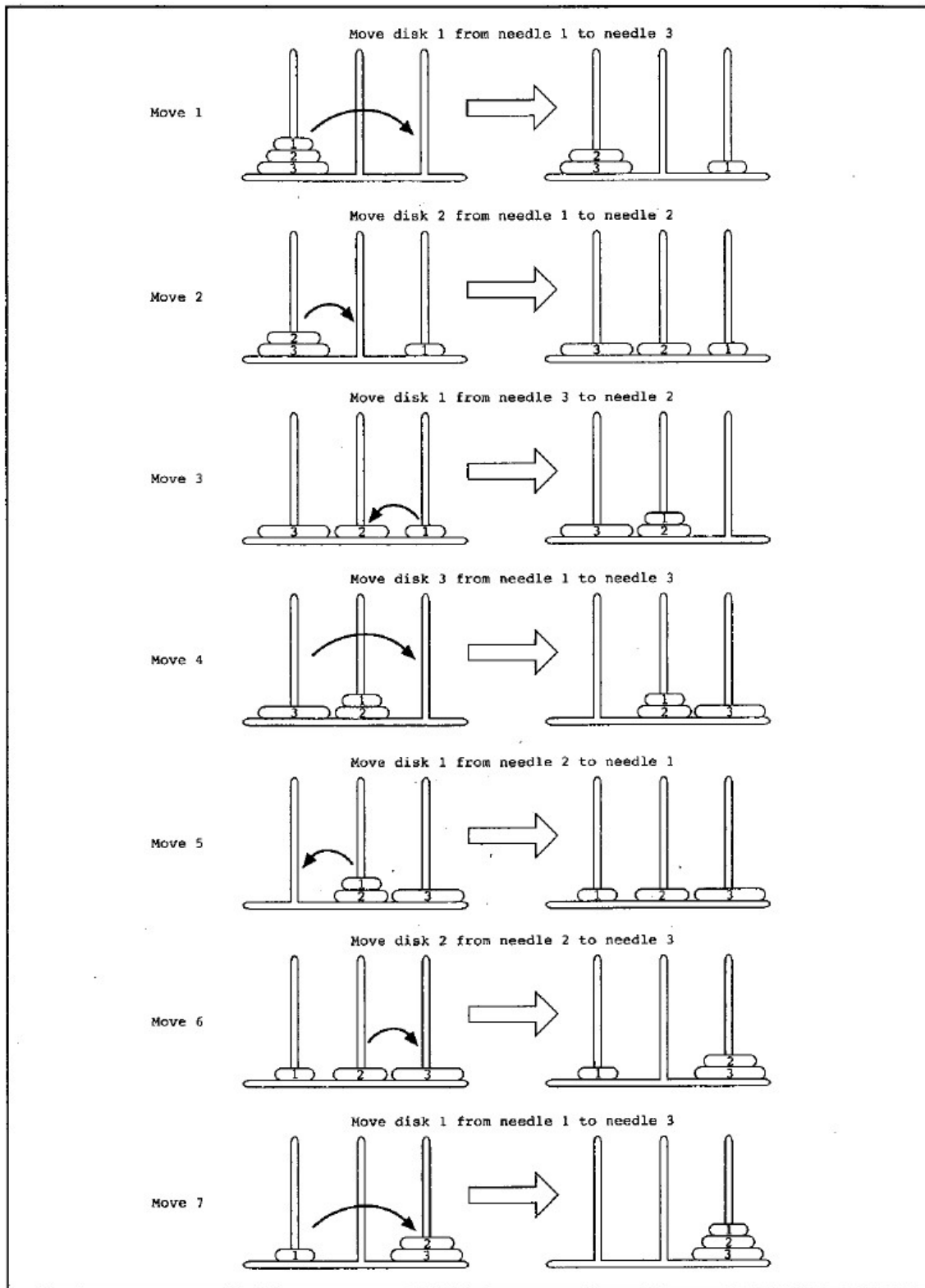
تم اخبار رجال الدين أنه بمجرد نقلهم للأقراص من الابر الأولى الى الابر الثالثة سوف يصل الكون الى النهاية.

هدفنا هو كتابة برنامج يطبع تتابع الحركات اللازمة لتحويل الأقراص من الابر الأولى الى الابر الثالثة. الشكل 6-8 يوضح مشكلة برج هانوي مع ثلاثة أقراص.



شكل 6-8: مشكلة برج هانوي مع ثلاثة أقراص

كما سبق ففكر على أساس التكرار. لنقم أولاً بالتفكير في الحالة التي تحتوي فيها الابرّة الأولى على قرص واحد فقط. في هذه الحالة يمكن نقل القرص مباشرةً من الابرّة 1 إلى الابرّة 3. لذلك لنقم بالنظر إلى الحالة التي تحتوي فيها الابرّة الأولى على قرصين فقط. في هذه الحالة نقوم أولاً بنقل القرص الأول من الابرّة 1 إلى الابرّة 2 ثم ننقل القرص الثاني من الابرّة 1 إلى الابرّة 3. في النهاية ننقل القرص الأول من الابرّة 2 إلى الابرّة 3 وبعد هذا نفكر في الحالة التي يحتوي فيها أول ابرة على ثلاثة أقراص ثم نقوم بتعميم هذا على حالة الـ 64 قرص (في الحقيقة لعدد arbitrary من الأقراص). افترض أن الابرّة 1 تحتوي على ثلاثة أقراص. لنقل القرص رقم 3 إلى الابرّة 3 يجب أن يتم أولاً نقل القرصين العلويين إلى الابرّة 2. يمكن عند هذا نقل القرص رقم 3 من الابرّة 1 إلى الابرّة 3. لنقل القرصين العلويين من الابرّة 2 إلى الابرّة 3 نقوم باستخدام نفس الطريقة كما سبق. هذه المرة نستخدم الابرّة 1 كابرّة وسطي. الشكل 6-9 يوضح حلاً لمشكلة برج هانوي مع ثلاثة أقراص.



شكل 9-6: حل لمشكلة برج هانوي مع ثلاثة أقراص.

لنقم الآن بتعميم هذه المشكلة على حالة 64 قرص. للبدء تحتوي الابرّة الأولى على جميع الـ 64 قرص. لا يمكن نقل القرص رقم 64 من الابرّة 1 الى الابرّة 3 الا اذا كانت الـ 63 قرصاً العلويين على الابرّة الثانية. لذلك نقوم أولاً بنقل الـ 63 قرص العلويين من الابرّة 1 الى الابرّة 2 ثم ننقل القرص رقم 64 من الابرّة 1 الى الابرّة 3. الآن تكون الـ 63 قرص العلويين على الابرّة 2. لننقل القرص رقم 63 من ابرّة 2 الى ابرّة 3 نقوم أولاً بنقل الـ 26 قرص العلويين من الابرّة 2 الى الابرّة 1 ثم ننقل القرص رقم 63 من الابرّة 2 الى الابرّة 3. لننقل الـ 62 القرص الباقيين نستخدم اجراء مماثل. تتم ترجمة هذه المناقشة الى الحل الحسابي التكراري التاليين معطى في الكود الوهمي. افترض أن الابرّة 1 تحتوي على عدد n من الأقراص حيث $1 \leq n$.

1. انقل الأقراص العلوية $n-1$ من الابرّة 1 الى الابرّة 2 باستخدام الابرّة 3 كابرّة انتقالية.
 2. انقل القرص رقم n من الابرّة رقم 1 الى الابرّة رقم 3.
 3. انقل الأقراص العلوية $n-1$ من الابرّة 2 الى الابرّة 3 باستخدام الابرّة 1 كابرّة انتقالية.
- تتم ترجمة هذا الحل الحسابي التكراري الى دالة C++ التالية:

```
{
    if(count > 0)
    {
        moveDisks(count - 1, needle1, needle2, needle3);
        cout<<"Move disk "<<count<<" from "<<needle1
            <<" to "<<needle3<<". "<<endl;
        moveDisks(count - 1, needle2, needle3, needle1);
    }
}
```

لنقم بعد هذا بتحديد الوقت الذي يتم استهلاكه لنقل جميع الأقراص من ابرّة 1 الى ابرّة 3. اذا كانت الابرّة 1 تحتوي على ثلاثة أقراص فان عدد الحركات اللازمة لنقل جميع الأقراص الثلاثة من الابرّة 1 الى الابرّة 3 هي $2^3 - 1 = 7$ تتوت الابرّة 1 على 64 قرص فان عدد الحركات اللازمة لنقل جميع الأقراص الأربعة والستين من الابرّة 1 الى الابرّة 3 هو $2^{64} - 1$. أن:

$$2^{10} = 1024 \approx 1000 = 10^3,$$

يكون لدينا:

$$2^{64} = 2^4 * 2^{60} \approx 2^4 * 10^{18} = 1.6 * 10^{19}.$$

هنا الرمز \approx يعني تقريبا مكافئ. عدد الثواني في العام هو تقريباً $3.2 * 10^7$. ن أن رجال الدين يحركون قرص واحد كل ثانية ولا يأخذون راحة. الآن:

$$1.6 * 10^{19} = 5 * 3.2 * 10^{18} = 5 * (3.2 * 10^7) * 10^{11} = (3.2 * 10^7) * (5 * 10^{11}).$$

الوقت اللازم لتحريم جميع الأقراص الأربعة والستين من الالبرة 1 الى الالبرة 3 هو
عاماً من $10^{11} * 5$ عالمنا يبلغ تقريباً 15 بليون $(1.5 * 10^{10} =)$ مأً . كذلك:

$$5 * 10^{11} = 50 * 10^{10} \approx 33 * (1.5 * 10^{10}).$$

هذه الحسابات توضح أن عالمنا سوف يستمر حوالي 33 مرة مما استمر بالفعل.
افترض أن هناك حاسب آلي يمكنه توليد 1 بليون (= 109) حركة في الثانية. اذن عدد الحركات التي
يمكن للحاسب توليدها في عام واحد هو:

$$(3.2 * 10^7) * 10^9 = 3.2 * 10^{16}.$$

لذلك فان الزمن اللازم لتوليد 264 حركة هو:

$$2^{64} \approx 1.6 * 10^{19} = 1.6 * 10^{16} * 10^3 = (3.2 * 10^{16}) * 500.$$

بهذا قد يحتاج الحاسب الآلي الى حوالي 500 عام لتوليد 264 حركة بمعدل 1 بليون حركة في الثانية.

مثال 5-6: تحويل العدد من عشري الى أحادي:

هذا المثال يناقش ويصمم برنامج يستخدم التكرار لتحويل عدد صحيح غير سالب بالصيغة العشرية –
أي القاعدة 10- الى عدد أحادي مكافئ له – أي القاعدة 2. أولاً نقوم بتعريف بعض المصطلحات.
لتكن x عدد صحيح غير سالب. نطلق على باقي x بعد القسمة على 2 ؟؟؟؟؟؟؟؟؟؟؟
بهذا يكون ؟؟؟؟؟؟؟؟؟؟؟ للعدد 33 هو 10 لأن 33 % 2 تكون 1 و ؟؟؟؟؟؟؟؟؟؟؟ للعدد 28 يكون صفر لأن 28
% 2 يكون صفر.

نقوم أولاً بتوضيح الحل الحسابي لتحويل عدد صحيح في القاعدة 10 الى العدد المكافئ في صيغة
ثنائية بمساعدة مثال.

افترض أننا نريد ايجاد التمثيل الثنائي للعدد 35. أولاً نقوم بقسمة 35 على 2. ناتج القسمة يكون 17
والباقي – أي ؟؟؟؟؟؟؟ - يكون 1. بعد هذا نقوم بقسمة 17 على 2 ويكون ناتج القسمة 8 والباقي – أي
؟؟؟؟؟؟؟؟ - يكون 1. بعد هذا نقوم بقسمة 8 على 2 وناتج القسمة يكون 4 والباقي – أي ؟؟؟؟؟؟؟ - يكون
صفر. نستمر في هذه العملية حتى يصبح الناتج صفراً .

؟؟؟؟؟؟ للعدد 35 لا يمكن طبعه حتى نقوم بطباعة ؟؟؟؟؟؟ للعدد 17 و ؟؟؟؟؟ للعدد 17 لا يمكن طبعه حتى
نقوم بطباعة ؟؟؟؟؟؟ للعدد 8 وهكذا. بهذا يكون التمثيل الثنائي للعدد 35 هو التمثيل الثنائي للعدد 17
(أي ناتج قسمة 35 بعد القسمة على 2) يليه ؟؟؟؟؟؟ للعدد 35.

بهذا ومن أجل تحويل عدد صحيح غير سالب num في القاعدة 10 الى عدد ثنائي مكافئ نقوم أولاً
بتحويل ناتج num/2 الى عدد ثنائي مكافئ ثم نقوم بتقريب ؟؟؟؟ num الى التمثيل الثنائي للعدد
num/2.

تتم ترجمة هذه المناقشة الى الحل الحسابي التكراري التالي حيث تعبر $\text{binary}(\text{num})$ عن التمثيل الثنائي للعدد num :

1. $\text{binary}(\text{num}) = \text{العدد}$ اذا كان العدد = صفر.
2. $\text{binary}(\text{num}) = \text{العدد الثنائي}(\text{num}/2)$ يليه قسمة $\text{num}/2$ اذا كان $\text{num} < \text{صفر}$.

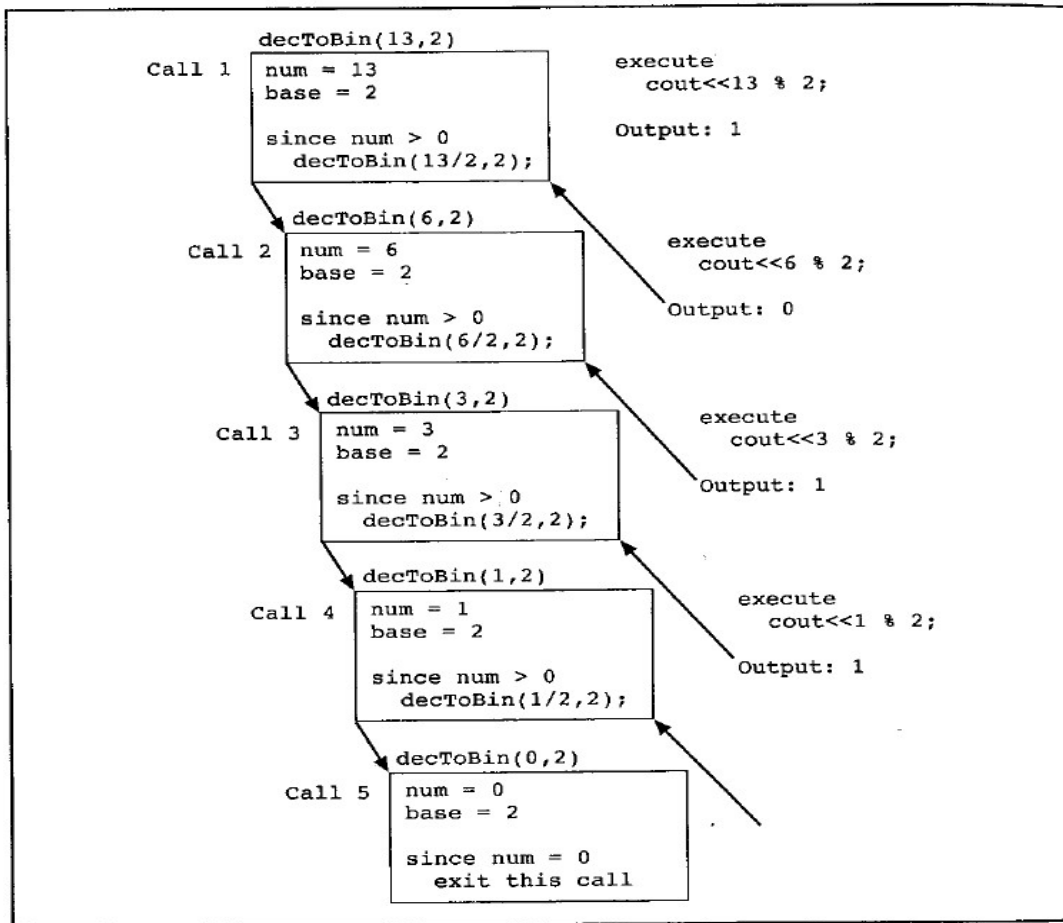
الدالة التكرارية التالية تطبق هذا الحل الحسابي:

```
void decToBin(int num, int base)
{
    if(num > 0)
    {
        decToBin(num/base, base);
        cout<<num % base;
    }
}
```

شكل 10-6 يتتبع تنفيذ البيان التالي:

$\text{decToBin}(13, 2);$

حيث num تساوي 13 والقاعدة هي 2.



شكل 10-6: تنفيذ البيان $\text{decToBin}(13, 2);$

بما أن البيان if في الاستدعاء 5 يفشل فإن هذا الاستدعاء لا يطبع أي شيء. الناتج الأول يتم تقديمه بالاستدعاء رقم 4 والثاني يتم إنتاجه بالاستدعاء رقم 3 الذي يطبع 1 والناتج الثالث يتم إنتاجه بالاستدعاء رقم 2 الذي يطبع صفر والناتج الرابع يتم إنتاجه من الاستدعاء 1 الذي يطبع 1. بهذا تكون مخرجات هذا البيان هي:

1101

برنامج C++ التالي يستخدم الدالة decToBin:

```
//Chapter 6: Program - Decimal to Binary

#include <iostream>

using namespace std;

void decToBin(int num, int base);

int main()
{
    int decimalNum;
    int base;

    base = 2;

    cout<<"Enter the number in decimal: ";
    cin>>decimalNum;
    cout<<endl;
    cout<<"Decimal "<<decimalNum<<" = ";
    decToBin(decimalNum, base);
    cout<<" binary"<<endl;

    return 0;
}

void decToBin(int num, int base)
{
    if(num > 0)
    {
        decToBin(num/base, base);
        cout<<num % base;
    }
}
```

تنفيذ العينة: في هذه العينة يتم تظليل مدخلات المستخدم.

ادخل الرقم عشرياً : 57

الرقم العشري 57 = الرقم الثنائي 111001

تكرار أم مكرر؟

قامت البرامج في الفصول السابقة باستخدام حلقة لعمل مجموعة من البيانات لأداء حسابات معينة أي أن البرامج في الفصول السابقة استخدمت بنية تحكم تكرارية لإعادة مجموعة من البيانات. بشكل رسمي أكثر تستخدم **بنيات التحكم التكرارية** مثل while أو for أو do...while لإعادة مجموعة من البيانات. في بداية هذا الفصل قمنا بتصميم دالة تكرارية لحساب معامل عدد صحيح غير سالب. من دالة المعامل ينتج أنه يتم تكرار مجموعة من البيانات عن طريق وجود الدالة تستدعي ذاتها. بالإضافة الى هذا فان هناك بنية تحكم اختيارية تتحكم في الاستدعاءات المتكررة في التكرار.

في هذا الفصل قمنا كذلك باستخدام التكرار لتحديد أكبر عنصر في القائمة ولتحديد عدد Fibonacci. باستخدام بنية تحكم تكرارية يمكنك كذلك كتابة حل حسابي لإيجاد أكبر عنصر في المصفوفة وبالمثل يمكن تصميم حل حسابي يستخدم بنية تحكم تكرارية لإيجاد معامل عدد صحيح غير سالب. السبب الوحيد لاعطاء حل تكراري لهذه المشكلات هو توضيح كيفية عمل التكرار.

اننا بهذا نرى أن هناك عادةً طريقتين لحل مشكلة معينة – المكررات والتكرار. السؤال الواضح هو ما هي الطريقة الأفضل – المكررات أم التكرار؟ لا توجد اجابة بسيطة لهذا السؤال فبالإضافة الى طبيعة المشكلة هناك عامل رئيسي آخر في تحديد طريقة الحل الأفضل وهي الكفاءة.

عندما يتم استدعاء دالة ما يتم تخصيص مساحة من الذاكرة من أجل معاملاتها الرسمية ومتغيراتها المحلية وعندما تتوقف الدالة يتم ازالة تخصيص مساحة الذاكرة. ان هذا الفصل أثناء تتبع تنفيذ دالات تكرارية يوضح لنا أن كل استدعاء (تكراري) له مجموعة خاصة به من المعاملات والمتغيرات المحلية (الآلية). هذا يعني أن كل استدعاء (تكراري) يقتضي من النظام تخصيص مساحة ذاكرة من أجل معاملاتها الرسمية ومتغيراتها المحلية ثم ازالة تخصيص مساحة الذاكرة عندما تخرج الدالة. بهذا يرتبط ؟؟؟؟ بتنفيذ دالة (تكرارية) على أساس مساحة الذاكرة وزمن الحاسب الآلي. لهذا يتم تنفيذ الدالة التكرارية أكثر ببطء من نظيرتها المكررة. على الجهة الأخرى سوف يكون من الواضح الحاسبات الآلية الأبطأ خاصةً تلك ذات مساحة الذاكرة المحدودة والتنفيذ (البطيء) للدالة التكرارية.

بالرغم من هذا فان حاسبات اليوم الآلية سريعة وتمتلك ذاكرة غير مكلفة. لهذا لا يكون تنفيذ الدالة التكرارية ملحوظاً. ان الاحتفاظ بقوة حاسبات الوقت الحالي مع الاختيار بين البديلين – التكرار أو المكررات – تعتمد على طبيعة المشكلة. بالطبع تكون الكفاءة شديدة الأهمية بالنسبة الى المشكلات مثل أنظمة التحكم في المهمات ولهذا سوف يملي عامل الكفاءة طريقة الحل.

كقاعدة عامة اذا كنت تعتقد أن الحل المكرر أكثر وضوحاً وسهولة في فهمه أكثر من الحل التكراري قم باستخدام الحل المكرر فسوف يكون أكثر كفاءة. من الجهة الأخرى تتواجد مشكلات التي يكون الحل التكراري لها أكثر وضوحاً أو سهولة لعمله مثل مشكلة برج هانوي. (في الحقيقة من الصعب اقامة حل مكرر لمشكلة برج هانوي). ان تذكر قوة التكرار اذا كان تعريف المشكلة تكراري بالوراثة فانه يجب عليك النظر في حل تكراري.

التكرار والتراجع: لغز عدد n من الملكات:

هذا القسم يصف أسلوب لتصميم الحل الحسابي وحل المشكلة يسمى التراجع. لنقم بالنظر الى لغز الثماني ملكات التالي: ضع ثماني ملكات على لوحة شطرنج (8 × 8 لوح مربع) حتى لا يمكن لأي ملكتين أن تهاجم بعضها البعض. حتى لا تكون أي ملكتين في موقف هجوم لا يمكن أن يكونوا في نفس الصف أو نفس العمود أو نفس الخط المائل الشكل 6-11 يعطي حل واحد ممكن للغز الثماني ملكات.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | Q | | | | |
| | | | | | Q | | |
| | | | | | | | Q |
| | Q | | | | | | |
| | | | | | | Q | |
| Q | | | | | | | |
| | | Q | | | | | |
| | | | | Q | | | |

شكل 6-11: حل للغز الثماني ملكات.

في عام 1850 تم التفكير في لغز الثماني ملكات بواسطة سي اف جاوس الذي لم يكن قادراً على الحصول على حل كامل. تم وضع مصطلح التراجع لأول مرة بواسطة دي اتش ليمر في عام 1950. في عام 1960 أعطى آر جيه والكر حل حسابي التراجع كما تم تقديم وصف عام للتتبع الخلفي مع تنوع من التطبيقات بواسطة اس جولومب وال بوميرت.

التراجع:

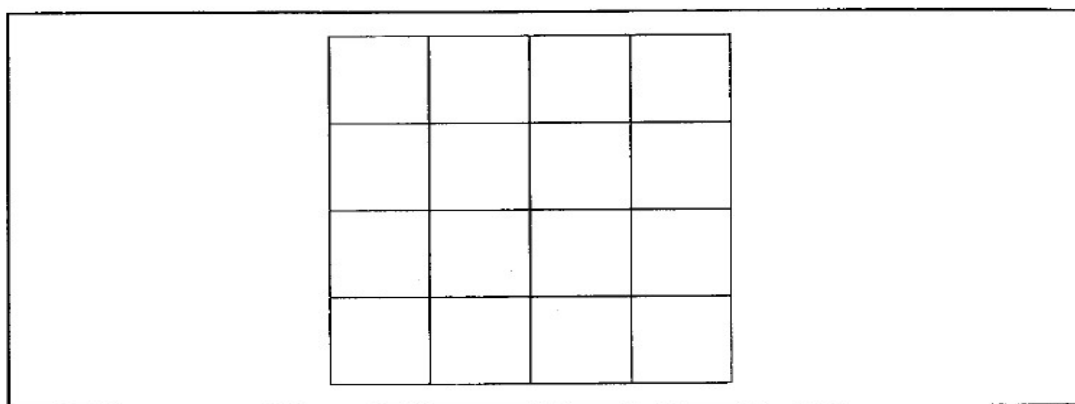
يحاول الحل الحسابي للتراجع أن يجد حلاً للشكلة عن طريق اقامة حلول جزئية ثم التأكد من أن أي حل جزئي لا يتعدى على متطلبات المشكلة. يحاول الحل الحسابي التوسع في الحل الجزئي نحو الكمال. بالرغم من هذا اذا تم تحديد أن الحل الجزئي لن يؤدي الى حل أي أن الحل الجزئي سوف ينتهي في نهية مغلقة فان الحل الحسابي يرجع عن طريق ازالة الجزء المضاف حديثاً ومحاولة احتمالات أخرى.

لغز الثماني ملكات:

في التراجع يمكن تمثيل حل لغز الملكات n في صورة (x_1, x_2, \dots, x_n) ، عدد صحيح مثل x_i يجب أن $1 \leq x_i \leq n$ في صف مختلف. في هذا tuple تقوم x_i بتحديد رقم العمود الذي يتم فيه وضع الملكة رقم i في الصف رقم i. لهذا يكون الحل بالنسبة الى لغز الثماني ملكات هو ثماني tuple $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ حيث x_i هو العمود الذي يتم فيه وضع الملكة رقم i في الصف رقم i.

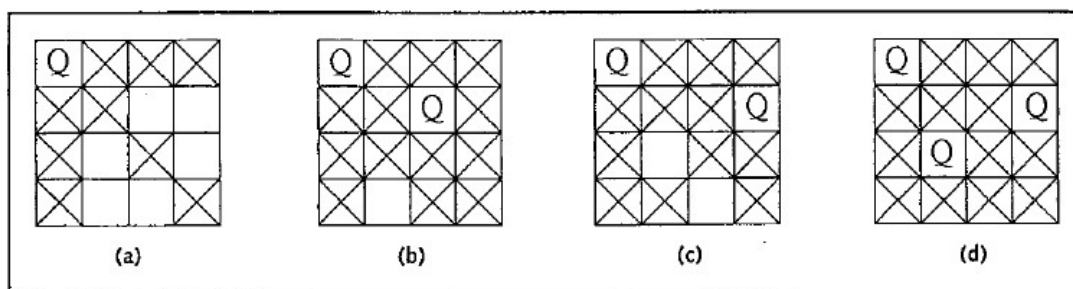
هذا يعني أن أول ملكة يتم وضعها في الصف الأول والعمود الرابع والملكة الثانية يتم وضعها في الصف الثاني والعمود السادس وهكذا. يتضح أن كل x_i عبارة عن عدد صحيح مثل $1 \leq x_i \leq 8$. لنقم مرة أخرى بالتفكير في الثماني tuple $(x_1-x_2-x_3-x_4-x_5-x_6-x_7-x_8)$ حيث x_i عدد صحيح مثل 1 $8 \geq x_i \geq 1$. بما أن كل x_i لها ثمانية اختيارات إذن هناك 8^8 من تلك tuple و 8^8 حلول ممكنة. بالرغم من هذا وبما أنه لا يمكن وضع ملكتين في نفس الصف فإنه لا يوجد عنصران من الثماني tuple $(x_1-x_2-x_3-x_4-x_5-x_6-x_7-x_8)$ متماثلان. ينتج من هذا أن عدد الحلول الممكنة للثماني tuple هو 8!.

الحل الذي نقوم بعمله يمكن في الحقيقة تطبيقه على أي عدد من الملكات. لهذا ولتوضيح أسلوب التراجع ننظر الى لغز الأربع ملكات أي أنه يتم اعطاؤك لوحة مربعة 4×4 (انظر شكل 6-12) و عليك وضع أربع ملكات على اللوحة حتى لا تهاجم أي ملكتين بعضهما البعض.



شكل 6-12: لوحة مربعة للغز الأربع ملكات.

نبدأ بوضع الملكة الأولى في الصف الأول والعمود الأول كما هو موضح في شكل 6-13 (أ). علامة خطأ في صندوق ما تعني أنه لا يمكن وضع ملكة في هذا الصندوق).

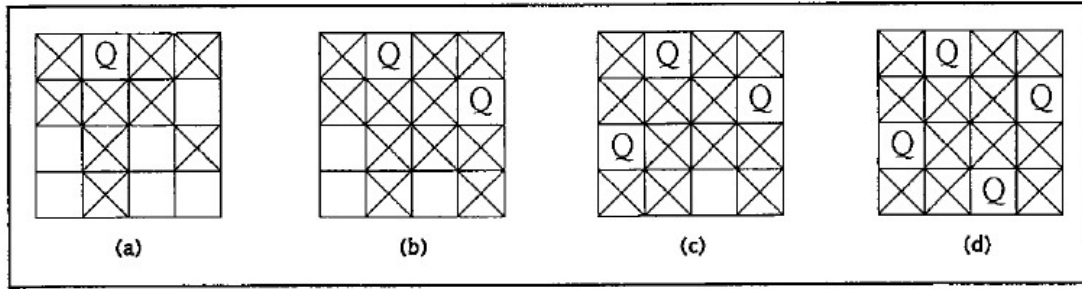


شكل 6-13: إيجاد حل للغز الأربع ملكات.

بعد وضع الملكة الأولى نحاول وضع الملكة الثانية في الصف الثاني. بوضوح تكون الملكة الأولى في الصف الثاني حيث يمكن وضع الملكة الثانية في العمود الثالث. لهذا نقوم بوضع الملكة الثانية في هذا العمود وانظر شكل 6-13 (ب).

بعد هذا نحاول وضع الملكة الثالثة في الصف الثالث ونجد أنه لا يمكن وضع الملكة الثالثة في الصف الثالث ولهذا نصل الى نهاية مغلقة. عند هذه النقطة نتتبع تركيب اللوحة السابقة ونضع الملكة الثانية في العمود الرابع. انظر شكل 6-13 (ج). بعد هذا نحاول وضع الملكة الثالثة في الصف الثالث. هذه المرة نضع الملكة الثالثة في العمود الثاني من الصف الثالث بنجاح. انظر شكل 6-13 (د). بعد وضع الملكة الثالثة في الصف الثالث وعندما نحاول وضع الملكة الرابعة نكتشف أن الملكة الرابعة لا يمكن وضعها في الصف الرابع.

نقوم بالتراجع الى الصف الثالث ونحاول وضع الملكة في أي عمود آخر. بما أنه لا يوجد أي عمود آخر متوفر للملكة رقم ثلاثة فاننا نقوم بالتراجع للصف 2 ونحاول وضع الملكة الثانية في أي عمود آخر وهذا لا يمكن فعله. لذلك نقوم بالتراجع للصف الأول ونضع الملكة الأولى في العمود التالي. بعد وضع الملكة الأولى في العمود الثاني نضع الملكات المتبقية في الصفوف المتتالية. هذه المرة نحصل على الحل كما هو موضح في شكل 6-14.



شكل 6-14: ايجاد حل للغز الأربع ملكات.

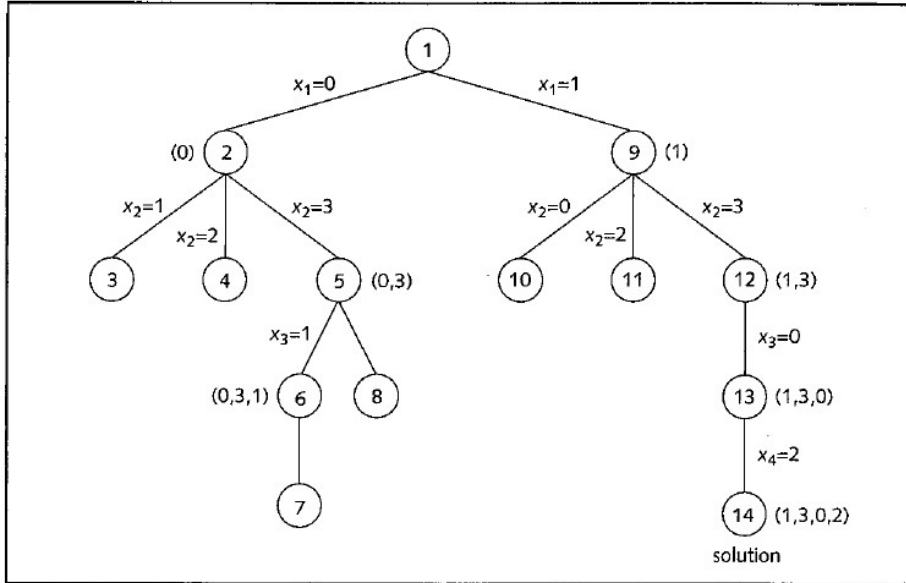
التراجع ولغز الأربع ملكات:

افترض أن صفوف اللوحة المربعة للغز الأربع ملكات مرقمة من صفر حتى 3 والعواميد مرقمة من صفر وحتى 3. (تذكر أنه في C++ يبدأ مؤشر المصفوفة عند صفر).

بالنسبة الى لغز الأربع ملكات نبدأ بوضع الملكة الأولى في الصف الأول والعمود الأول وبهذا نقوم بتوليد tuple (0). بعد هذا نضع الملكة الثانية في العمود الثالث من الصف الثاني وبهذا نقوم بتوليد tuple (0, 2). عندما نحاول وضع ثالث ملكة في الصف الثالث حددنا أن الملكة الثالثة لا يمكن وضعها في الصف الثالث. لهذا نعود الى الحل الجزئي (0, 2) ونحذف 2 من tuple ثم نقوم بتوليد tuple (0, 3) أي يتم وضع ثاني ملكة في العمود الرابع من الصف الثاني. بعد هذا نحاول بالحل الجزئي (0, 3) وضع ثالث ملكة في الصف الثالث وتوليد tuple (0, 3, 1). بعد هذا وبالحل الجزئي (0, 3, 1) عندما نحاول وضع رابع ملكة في الصف الرابع يتحدد أنه لا يمكن فعل هذا ولذلك ينتهي الحل الجزئي (0, 3, 1) في نهاية مسدودة.

من الحل الجزئي (0, 3, 1) يتم دعم الحل الحسابي بالتراجع لوضع أول ملكة ويزيل جميع عناصر tuple. يقوم الحل الحسابي بعد هذا بوضع أول ملكة في العمود الثاني من الصف الأول وبهذا يولد الحل الجزئي (1). في هذه الحالة يكون تتابع الحل الجزئي الناتج هو (1) و (1, 3) و (1, 3, 0) و (1, 3, 0, 1).

3، 0، 2) وهذا يمثل حل للغز الأربع ملكات. الحلول الناتجة من حل التراجع يمكن تمثيلها بشجرة كما هو موضح في شكل 6-15.



شكل 6-15: شجرة الأربع ملكات.

لغز الثماني ملكات:

لنقم الآن بالتفكير في لغز الثماني ملكات. انه مثل لغز الأربع ملكات لا يمكن أن يكون هناك ملكتان في نفس الصف أو نفس العمود أو نفس الخط المائل. ان تحديد ما اذا كان هناك ملكتان في نفس الصف أو العمود سهل لأننا يمكننا التحقق نت أماكن صفوفها وعواميدها. لنقم بوصف كيفية تحديد ما اذا كان هناك ملكتان في نفس الخط المائل أم لا.

انظر الى اللوحة المربعة 8×8 الموضحة في شكل 6-16. الصفوف مرقمة من صفر الى 7 والعواميد مرقمة من صفر الى 7. (تذكر أنه في C++ يبدأ مؤشر المصفوفة عند صفر).

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 |
| 1,0 | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 | 1,7 |
| 2,0 | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | 2,6 | 2,7 |
| 3,0 | 3,1 | 3,2 | 3,3 | 3,4 | 3,5 | 3,6 | 3,7 |
| 4,0 | 4,1 | 4,2 | 4,3 | 4,4 | 4,5 | 4,6 | 4,7 |
| 5,0 | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 | 5,6 | 5,7 |
| 6,0 | 6,1 | 6,2 | 6,3 | 6,4 | 6,5 | 6,6 | 6,7 |
| 7,0 | 7,1 | 7,2 | 7,3 | 7,4 | 7,5 | 7,6 | 7,7 |

شكل 6-16: لوحة مربعة 8×8

انظر الى الخط المائل من أعلى اليسار الى أدنى اليمين كما مشار اليه بواسطة السهم. أماكن المربعات على هذا الخط المائل هي (0، 4) و(1، 5) و(2، 6) و(3، 7). لاحظ أنه بالنسبة الى تلك المدخلات يكون مكان الصف – مكان العمود هو 4- على سبيل المثال $4-0 = 5-1 = 6-2 = 7-3$ يمكن توضيح أن مكان الصف – مكان العمود لكل مربع على الخط المائل من أعلى اليسار الى أدنى اليمين يكون متماثل.

انظر الآن الى الخط المائل من أعلى اليمين الى أدنى اليسار كما مشار اليه بالسهم. أماكن المربعات على هذا الخط المائل هي (0، 6) و(1، 5) و(2، 4) و(3، 3) و(4، 2) و(5، 1) و(6، 0). هنا يكون مكان الصف + مكان العمود = 6. يمكن توضيح أن مكان الصف + مكان العمود لكل مربع على الخط المائل من أعلى اليمين الى أدنى اليسار يكون متماثل.

يمكننا استخدام هذه النتائج لتحديد ما اذا كان هناك ملكتين على نفس الخط المائل أم لا. افترض أن هناك ملكة في الموضع (أ، ب) أي (الصف أ والعمود ب) وهناك ملكة أخرى في الموضع (س، ص) أي (الصف س والعمود ص). هاتان الملكتان تكونان على نفس الخط المائل اذا كان $أ + ب = س + ص$ أو $أ - ب = س - ص$. المعادلة الأولى تشير الى أن $ب - ص = س - أ$ نفس الخط المائل اذا كانت $ب - ص = |أ - س|$ حيث $|ب - ص|$ هي القيمة المطلقة للمقدار $ب - ص$ وهكذا.

بما أن حل لغز الثماني ملكات يتم تمثيله بثنائي tuple فاننا نستخدم المصفوفة queensInRow حجمها 8 حيث يقوم queensInRow[k] بتحديد موضع العمود الذي تكون به الملكة رقم k في الصف k. على سبيل المثال queensInRow[0] = 3 تعني أن الملكة الأولى يتم وضعها في العمود 3 (وهو العمود الرابع) من الصف صفر (وهو الصف الأول).

افترض أننا وضعنا الملكات k-1 الأولى في الصفوف k-1 الأولى. بعد هذا نحاول وضع الملكة رقم k في عمود من الصف رقم k. اننا نكتب الدالة canPlaceQueen(k, i) والتي تنتج كلمة صحيح اذا كان يمكن وضع الملكة رقم k في العمود رقم i من الصف k وبخلاف هذا تنتج كلمة خطأ.

الملكات k-1 الأولى موجودة في الصفوف k-1 الأولى ونحن نحاول وضع الملكة k في الصف k. لهذا يجب أولاً أن يكون الصف k فارغاً ومن هذا ينتج أن الملكة k يمكن وضعها في العمود I من الصف k شريطة أن لا يكون هناك ملكة أخرى في العمود i وألا يكون هناك ملكات على الخطوط المائلة التي يقع عليها المربع (k, i). الحل الحسابي العام للدالة canPlaceQueen(k, i) هو:

```
for(int j = 0; j < k; j++)
    if((queensInRow[j] == i) //there is already a queen in column i
        || (abs(queensInRow[j] - i) == abs(j-k))) //there is already
            //a queen in one of the diagonals
            //on which square (k,i) lies
        return false
return true
```

الحلقة for تتحقق مما اذا كان هناك ملكة بالفعل اما في العمود I أو في واحد من الخطوط المائلة التي يقع عليها المربع (k, i). اذا وجدت أي ملكة على أي موضع تقوم الحلقة for بانتاج القيمة false وبخلاف هذا تنتج القيمة true.

الفئة التالية تقوم بتعريف لغز الثماني ملكات كنوع بيانات مجرد:

```
class nQueensPuzzle
{
public:
    nQueensPuzzle(int queens = 8);
    //constructor
    //Postcondition: noOfSolutions = 0; noOfQueens = queens;
    // queensInRow is a pointer to the array
    // that stores the n-tuple.
    // If the user does not specify any value for the
    // parameter queens, then the default value, which is 8,
    // is assigned to it.
    // bool canPlaceQueen(int k, int i);

    // اذا لم يحدد المستخدم أي قيمة للمعامل الماكة فانه يتم تحديد القيمة الافتراضية التي تبلغ 8 له.

    //Function to determine whether a queen can be placed
    //in row k and column i.
    //Postcondition: Returns true if a queen can be placed in
    // row k and column i; otherwise, returns false.

    void queensConfiguration(int k);
    //Function to determine all the solutions to the n-queens
    //puzzle using backtracking.
    //The function is called with the value 0.
    //Postcondition: All n-tuples representing solutions to the
    // n-queens puzzle are generated and printed.

    void printConfiguration();
    //Function to output an n-tuple containing a solution
    //to the n-queens puzzle.

    int solutionsCount();
    //Function to return the total number of solutions.
    //Postcondition: The value of noOfSolutions is returned.

private:
    int noOfSolutions;
    int noOfQueens;
    int *queensInRow;
};
```

// شرط تالي: تنتج true اذا كان يمكن وضع الملكة في الصف k والعمود i
 وبخلاف هذا تنتج false.

```
void queensConfiguration (int, k);
// دالة لتحديد جميع الحلول للغز الثماني ملكات باستخدام التراجع.
// يتم استدعاء الدالة بالقيمة صفر.
// شرط تالي: جميع n-tuples التي تمثل حلول للغز الثماني ملكات يتم انتاجها وطبعها.
```

```
void printConfiguration ( );
// دالة لايخراج n-tuple المحتوي على حل للغز الثماني ملكات.
```

```
int solutionCount ( );
// دالة لانتاج عدد الحلول الكلي.
// شرط تالي: يتم انتاج قيمة عدد الحلول.
```

يتم اعطاء تعريفات دلالات العنصر للفئة nQueensPuzzle بعد هذا:

```
nQueensPuzzle::nQueensPuzzle(int queens)
{
    noOfQueens = queens;
    queensInRow = new int[noOfQueens];
    noOfSolutions = 0;
}

bool nQueensPuzzle::canPlaceQueen(int k, int i)
{
    for(int j = 0; j < k; j++)
        if((queensInRow[j] == i)
            || (abs(queensInRow[j] - i) == abs(j-k)))
            return false;
    return true;
}
```

باستخدام التكرار تقوم الدالة queensConfiguration بتطبيق أسلوب لبتراجع لتحديد جميع حلول لغز الثماني ملكات. المعامل k يحدد الملكة التي يتم وضعها في الصف k وتعريفه يكون:

```

void nQueensPuzzle::queensConfiguration(int k)
{
    for(int i = 0; i < noOfQueens; i++)
    {
        if(canPlaceQueen(k, i))
        {
            queensInRow[k] = i; //place the kth queen in column i
            if(k == noOfQueens - 1) //all the queens are placed
                printConfiguration(); //print the n-tuple
            else
                queensConfiguration(k + 1) //determine the place for
                                           //the (k+1)th queen
        }
    }
}

void nQueensPuzzle::printConfiguration()
{
    noOfSolutions++;
    cout<<"(";
    for(int i = 0; i < noOfQueens - 1; i++)
        cout<<queensInRow[i]<<" ";

    cout<<queensInRow[noOfQueens - 1]<<")"<<endl;
}

int nQueensPuzzle::solutionsCount()
{
    return noOfSolutions;
}

```

مراجعة سريعة

1. عملية حل المشكلة عن طريق اختزالها الى صور أصغر تسمى التكرار.
2. التعريف التكراري يقوم بتعريف المشكلة على أساس صور أصغر منها.
3. كل تعريف تكراري له حالة أساسية واحدة أو أكثر.
4. الحل الحسابي التكراري يحل المشكلة عن طريق اختزالها الى صور أصغر منها.
5. كل حل حسابي تكراري له حالة أساسية واحدة أو أكثر.
6. يتم الحصول على حل المشكلة في الحالة الأساسية بشكل مباشر.
7. يطلق على الدالة دالة تكرارية اذا استدعت نفسها.
8. يتم تطبيق الحلول الحسابية التكرارية باستخدام الدالات التكرارية.
9. يجب أن يكون لكل دالة تكرارية حالة أساسية واحدة أو أكثر.
10. الحل العام يقوم بتقسيم المشكلة الى صور أصغر.
11. يجب أن يتم اختزال الحالة العامة الى حالة أساسية في النهاية،
12. الحالة الأساسية توقف التكرار.
13. أثناء تتبع دالة تكرارية:

- منطقياً يمكنك التفكير في أن الدالة التكرارية لها نسخ غير محدودة منها.
- كل استدعاء لدالة تكرارية – أي كل استدعاء تكراري – له كود خاص به ومجموعة خاصة به من العوامل والمتغيرات المحلية.
- بعد اكمال استدعاء تكراري معين يعود التحكم الى البيئة الداعية وهو الاستدعاء السابق. الاستدعاء (التكراري) الحالي يجب أن يتم تنفيذه كاملاً قبل عودة التحكم الى الاستدعاء السابق. التنفيذ في الاستدعاء السابق يبدأ من النقطة التالية مباشرةً للاستدعاء التكراري.

14. يطلق على الدالة تكرارية بشكل مباشر اذا استدعت نفسها.
15. الدالة التي تستدعي دالة أخرى وتتسبب في النهاية في استدعاء الدالة الأصلية يقال أنها تكرارية بشكل غير مباشر.
16. الدالة التكرارية التي يكون فيها البيان الأخير الذي يتم تنفيذه هو الاستدعاء التكراري تسمى دالة تكرارية ذيلية.
17. لتصميم دالة تكرارية يجب عليك القيام بما يلي:
 - أ- استيعاب متطلبات المشكلة.
 - ب- تحديد الظروف القاصرة. على سبيل المثال الشرط القاصر للقائمة يتم تحديده بواسطة عدد العناصر في القائمة.
 - ت- تحديد الحالات الأساسية وتوفير حل مباشر لكل حالة.
 - ث- تحديد الحالات العامة وتوفير حل لكل حالة على أساس صور أصغر منها.
18. حل التراجع الحسابي يحاول إيجاد حلول للمشكلة عن طريق إقامة حلول جزئية ثم التأكد من أن أي حل جزئي لا يخالف متطلبات المشكلة. الحل الحسابي يحاول مد الحل الحسابي نحو الكمال. بالرغم من هذا اذا تقرر أن الحل الجزئي لن يؤدي الى حل أي أن الحل الجزئي سوف ينتهي في طريق مسدود فان الحل الحسابي يتحقق عن طريق ازالة الجزء المضاف حديثاً ثم يحاول امكانيات أخرى.

تمارين

1. ضع علامة صح أم خطأ امام العبارات التالية:
 - أ- كل تعريف تكراري يجب أن يكون له حالة أساسية واحدة أو أكثر.
 - ب- كل دالة تكرارية يجب أن يكون لها حالة أساسية واحدة أو أكثر.
 - ت- الحالة العامة توقف التكرار.
 - ث- في الحالة العامة يتم الحصول على حل المشكلة بشكل مباشر.
 - ج- دائماً تنتج الدالة التكرارية قيمة ما.

2. ما هي الحالة الأساسية؟
3. ما هي الحالة التكرارية؟
4. ما هو التكرار المباشر؟
5. ما هو التكرار الغير مباشر؟
6. ما هو التكرار الذيلي؟
7. انظر الى الدالة التكرارية التالية:

```

int mystery(int number)           //Line 1
{
    if(number == 0)               //Line 2
        return number;           //Line 3
    else                           //Line 4
        return(number + mystery(number - 1)); //Line 5
}

```

- أ- حدد الحالة الأساسية.
- ب- حدد الحالة العامة.
- ت- ما هي القيم الصحيحة التي يمكن تمريرها كعواملات للدالة mystery؟
- ث- اذا كانت mystery (0) استدعاء صحيح ما هي قيمته؟ اذا لم يكن وضح السبب.
- ج- اذا كانت mystery (5) استدعاء صحيح ما هي قيمته؟ اذا لم يكن وضح السبب.
- ح- اذا كانت mystery (-3) استدعاء صحيح ما هي قيمته؟ اذا لم يكن وضح السبب.

8. انظر الى الدالة التكرارية التالية:

```

void funcRec(int u, char v)       //Line 1
{
    if(u == 0)                   //Line 2
        cout<<v;                 //Line 3
    else if(u == 1)              //Line 4
        cout<<static_cast<char>(static_cast<int>(v) + 1); //Line 5
    else                           //Line 6
        funcRec(u - 1, v);       //Line 7
}

```

أجب ما يلي:

- أ- حدد الحالة الأساسية.
- ب- حدد الحالة العامة.
- ت- ما هي مخرجات البيان التالي؟

funcRec (5, 'A');

9. انظر الى الدالة التكرارية التالية:

```

void exercise(int x)
{
    if(x > 0 && x < 10)
    {
        cout<<x<<" ";
        exercise(x+1);
    }
}

```

ما هي مخرجات البيانات التالية؟

أ- exercise (0);

ب- exercise (5);

ت- exercise (10);

ث- exercise (-5);

10. انظر الى الدالة التالية:

```
int test(int x, int y)
{
    if(x == y)
        return x;
    else if(x > y)
        return (x + y);
    else
        return test(x + 1, y - 1);
}
```

ما هي مخرجات البيانات التالية؟

أ. cout<<test (5, 10) <<endl;

ب. cout<<test (3,19) <<endl;

11. انظر الى الدالة التالية:

```
int Func(int x)
{
    if(x == 0)
        return 2;
    else if(x == 1)
        return 3;
    else
        return (Func(x - 1) + Func(x - 2));
}
```

ما هي مخرجات البيانات التالية؟

أ. cout<<func (0) <<endl;

ب. cout<<func (1) <<endl;

ت. cout<<func (2) <<endl

ث. cout<<func (5) <<endl

12. افترض أن intArray مصفوفة من أعداد صحيحة وlength (الطول) يحدد عدد

العناصر في intArray. افترض كذلك أن low وhigh عددين صحيحين حيث $0 \leq$

$low < length$ ، و $low < length < high \leq 0$ ، و $low < length$. هذا يعني أن low وhigh

مؤشرين في intArray. اكتب تعريف تكراري يعكس العناصر في intArray بين low وhigh.

13. اكتب حل حسابي تكراري لضرب عددين صحيحين موجبين س، وص باستخدام جمع متكرر. حدد الحالة الأساسية والحالة التكرارية.

14. انظر الى المسألة التالية: كم عدد الطرق التي يمكن بها اختيار لجنة من أربعة أفراد من مجموعة مكونة من 10 أفراد؟ هناك العديد من المسائل المماثلة الأخرى التي يطلب منك فيها إيجاد عدد الطرق لاختيار مجموعة من العناصر من مجموعة معطاة من العناصر. يمكن ذكر المسألة العامة كما يلي: جد عدد الطرق r التي يمكن بها اختيار أشياء مختلفة من مجموعة من العناصر البالغة n حيث r و n أعداد صحيحة غير سالبة وحيث $r \leq n$. افترض أن $C(n, r)$ تعبر عن عدد الطرق r التي يمكن بها اختيار أشياء مختلفة من مجموعة من العناصر التي تبلغ n . اذن يتم اعطاء $C(n, r)$ بواسطة الصيغة التالية:

$$C(n, r) = \frac{n!}{r! (n-r)!},$$

حيث تعبر علامة التعجب عن دالة المعامل. فضلاً عن هذا $C(n, 0) = C(n, n) = 1$. من المعروف كذلك أن $C(n, r) = C(n-1, r-1) + C(n-1, r)$. أ- اكتب حل حسابي تكراري لتحديد $C(n, r)$ وحدد الحالة الأساسية والحالة العامة. ب- باستخدام حلك الحسابي التكراري حدد $C(5, 3)$ و $C(9, 4)$.

تمارين برمجة

1. اكتب دالة تكرارية تأخذ عدد صحيح غير سالب كمعامل وتولد الجزء التالي من النجوم. اذا كان العدد الصحيح الغير سالب هو 4 اذن فان النجوم المتولدة هي:

```
****
***
**
*
*
**
***
****
```

اكتب كذلك برنامج يحث المستخدم على ادخال عدد الصفوف في المقطع ويستخدم دالة تكرارية لتوليد المقطع. على سبيل المثال تحديد 4 كعدد الصفوف يولد المقطع السابق.

2. اكتب دالة تكرارية لتوليد مقطع من النجوم كما يلي:

```
*  
**  
***  
****  
****  
***  
**  
*
```

اكتب كذلك برنامج يحث المستخدم على ادخال عدد الصفوف في المقطع ويستخدم دالة تكرارية لتوليد المقطع. على سبيل المثال تحديد 4 كعدد الصفوف يولد المقطع السابق.

3. اكتب دالة تكرارية لتوليد مقطع من النجوم كما يلي:

```
*  
* *  
* * *  
* * * *  
* * *  
* *  
*
```

اكتب كذلك برنامج يحث المستخدم على ادخال عدد الصفوف في المقطع ويستخدم دالة تكرارية لتوليد المقطع. على سبيل المثال تحديد 4 كعدد الصفوف يولد المقطع السابق.

4. اكتب دالة تكرارية vowels تنتج عدد الحروف اللينة في مقطع ما واكتب أيضاً برنامج لاختبار ذلك.

5. اكتب دالة تكرارية تجد وتنتج مجموع عناصر مصفوفة int واكتب برنامج لاختبار ذلك.

6. الباليندروم عبارة عن مقطع يقرأ نفس الشيء من الأمام ومن الخلف. على سبيل المثال المقطع "madam" عبارة عن باليندروم. اكتب برنامج يستخدم دالة تكرارية للتحقق مما اذا كان المقطع باليندروم أم لا. يجب أن يشتمل برنامجك على دالة تكرارية منتجة للقيمة تقوم بانتاج true اذا كان المقطع باليندروم وتنتج false بخلاف هذا. لا تستخدم أي متغيرات عالمية بل استخدم المعاملات المناسبة.

7. اكتب برنامج يستخدم دالة تكرارية لطبع مقطع من الخلف ويجب أن يشتمل برنامجك على دالة تكرارية تطبع المقطع من الخلف. لا تستخدم أي متغيرات عالمية بل استخدم المعاملات المناسبة.

8. اكتب دالة تكرارية reverseDigits تتخذ عدد صحيح كمعامل وتنتج العدد مع عكس الأرقام واكتب كذلك برنامج لاختبار دالتك.

9. اكتب دالة تكرارية power تتخذ عددين صحيحين x و y كمعاملات حيث يكون x عدد غير صفري وتنتج x^y . يمكنك استخدام التعريف التكراري التالي لحساب x^y . إذا كان $y \leq 0$ فان:

$$power(x, y) = \begin{cases} 1 & \text{if } y = 0 \\ x & \text{if } y = 1 \\ x * power(x, y - 1) & \text{if } y > 1 \end{cases}$$

إذا كانت $y < 0$ فان:

$$power(x, y) = \frac{1}{power(x, -y)}$$

اكتب كذلك برنامج لاختبار دالتك.

10. (القاسم المشترك الأكبر) باعطاء عددين صحيحين x و y فان الدالة التكرارية التالية تحدد القاسم المشترك الأكبر لكل من x و y ويكتب كما يلي $gcd(x, y)$

$$gcd(x, y) = \begin{cases} x & \text{if } y = 0 \\ gcd(y, x \% y) & \text{if } y \neq 0 \end{cases}$$

لاحظ أن في هذا التعريف يكون % هو mod operator.

اكتب دالة تكرارية gcd تتخذ عددين صحيحين كمعاملات وتنتج القاسم المشترك الأكبر للأعداد واكتب برنامج لاختبار دالتك.

11. اكتب دالة تكرارية لتطبيق الحل الحسابي التكراري للتمرين 12 (عكس عناصر المصفوفة بين مؤشرين) واكتب برنامج لاختبار دالتك.

12. كتب دالة تكرارية لتطبيق الحل الحسابي التكراري للتمرين 13 (ضرب عددين صحيحين باستخدام الجمع المتكرر) واكتب برنامج لاختبار دالتك.

13. اكتب دالة تكرارية لتطبيق الحل الحسابي التكراري للتمرين 14 (تحديد عدد الطرق لاختيار مجموعة أشياء من مجموعة معطاة من الأشياء) واكتب برنامج لاختبار دالتك.

14. في مثال 5-6 "تحويل عدد من عشري الى ثنائي" تعلمت كيفية تحويل عدد عشري الى عدد ثنائي مكافئ. هناك نظامان عدديان وهما نظام ثنائي (القاعدة 2) ونظام الستة عشر (القاعدة 16) يهتمان علماء الحاسب الآلي. في الحقيقة يمكنك في C++ توجيه الحاسب الآلي لتخزين عدد بصيغة ثمانية أو بصيغة ستة عشر. الأرقام في نظام العدد الثماني هي 0، 1، 2، 3، 4،

5، 6، و7. الأرقام في نظام الأرقام ذات القاعدة الستة عشر هي 0، 1، 2، 3، 4، 5، 6، 7، 8، 9، A، B، C، D، E، F. لذلك تكون A في النظام السداسي عشر 10 وتكون B 11 وهكذا. الحل الحسابي لتحويل عدد عشري موجب الى عدد مكافئ بالصيغة الثمانية (أو الستة عشر) يكون هو نفس الحل كما نوقش بالنسبة الى الأعداد الثنائية. هنا نقسم العدد العشري على 8 (بالنسبة الى النظام الثماني) وعلى 16 (بالنسبة الى نظام الستة عشر). افترض أن a_b تمثل العدد a للقاعدة b. على سبيل المثال 75_{10} تعني 75 للقاعدة 10 (هذه صيغة عشرية) و 83_{16} تعني 83 للقاعدة 16 (هذه صيغة الستة عشر). اذن:

$$753_{10} = 1361_8$$

$$753_{10} = 2F1_{16}$$

طريقة تحويل عدد عشري الى القاعدة 2 أو 8 أو 16 يمكن مدها الى أي قاعدة اعتباطية. افترض أنك تريد تحويل عدد عشري الى عدد مكافئ في القاعدة b حيث b تقع بين 2 و36. عند هذا تقوم بقسمة العدد العشري n على b كما هو الحال في الحل الحسابي لتحويل العدد من عشري الى ثنائي.

لاحظ أن الأرقام في القاعدة 20 مثلاً هي 0، 1، 2، 3، 4، 5، 6، 7، 8، 9، A، B، C، D، E، F، G، H، I، وJ.

اكتب برنامج يستخدم دالة تكرارية لتحويل عدد عشري الى قاعدة معطاة b حيث تقع b بين 2 و36. يجب أن يحدد برنامجك المستخدم على ادخال العدد بالصيغة العشرية ثم ادخال القاعدة المطلوبة. اختبر برنامجك على البيانات التالية:

9098 والقاعدة 20

692 والقاعدة 2

753 والقاعدة 16

15. (تحويل العدد من ثنائي الى عشري) لغة الحاسب الآلي المسماة لغة الآلة عبارة عن تتابع من الأصفار والواحدات. عندما تضغط على المفتاح A على لوحة المفاتيح يتم تخزين 01000001 في الحاسب الآلي. كذلك يكون التتابع الجمعي لA في مجموعة رموز ASCII هو 65. في الحقيقة يكون التمثيل الثنائي لA هو 01000001 والتمثيل العشري لA يكون 65.

نظام الترقيم الذي نستخدمه يسمى النظام العشري أو نظام القاعدة 10 ونظام الترقيم الذي نستخدمه الحاسب الآلي يسمى النظام الثنائي أو نظام القاعدة 2. المثال 5-6 يصف كيفية

تحويل عدد عشري الى عدد ثنائي مكافئ له. الهدف من هذا التمرين هو كتابة دالة لتحويل العدد من القاعدة 2 الى القاعدة 10.

لتحويل عدد من القاعدة 2 الى القاعدة 10 نقوم أولاً بايجاد وزن كل جزء في العدد الثنائي. يتم وضع وزن كل جزء في العدد الثنائي من اليمين الى اليسار. وزن rightmost هو صفر. وزن الجزء الواقع على يسار rightmost bit مباشرة هو 1 ووزن الجزء الواقع على يساره مباشرة هو 2 وهكذا. انظر الى العدد الثنائي 1001101. وزن كل جزء يكون كما يلي:

الوزن 0 1 2 3 4 5 6

1 0 1 1 0 0 1

نقوم باستخدام وزن كل جزء لايجاد العدد العشري المكافئ. نقوم لكل جزء بضرب الجزء في 2 لقوة وزنه ثم نجمع جميع الأعداد. بالنسبة الى العدد الثنائي 1001101 يكون العدد العشري المكافئ هو:

$$1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

$$= 64 + 0 + 0 + 8 + 4 + 0 + 1$$

لكتابة برنامج

في العدد الثنائي يجب ان يكون معروف و(2) يتم تحديد الوزن من اليمين الى اليسار. بما اننا لا نعرف مسبقاً كم عدد الأجزاء الموجودة في العدد الثنائي فيجب أن نقوم بمعالجة الأجزاء من اليمين الى اليسار. بعد معالجة جزء ما يمكننا اضافة 1 الى وزنه معطين وزن الجزء على يساره مباشرة. يجب كذلك لأن يتم استخلاص كل جزء من العدد العشري وضربه في 2 الى قوة وزنه. لاستخلاص جزء يمكنك استخدام المعامل mod.

اكتب دالة تقوم بتحويل عدد ثنائي الى عدد عشري مكافئ له وبالإضافة الى هذا اكتب برنامج واخبر دالتك بالنسبة الى القيم التالية: 11000101، و10101010، و11111111، و10000000، و1111100000.

16. الدالة sqrt من الملف الرئيسي cmath يمكن استخدامها لايجاد الجذر التربيعي لعدد حقيقي

غير سالب. باستخدام طريقة نيوتن يمكنك كذلك كتابة حل حسابي لايجاد الجذر التربيعي لعدد حقيقي غير سالب ضمن tolerances معطاة كما يلي:

افترض أن x عدد حقيقي غير سالب وأن a الجذر التربيعي التقريبي للعدد x وأن العدد اليوناني الخامس هو tolerance. ابدأ بـ $x = a$:

أ. اذا كان العدد اليوناني الخامس $|a * a - x| \leq$ اذن a هي الجذر التربيعي للعدد x ضمن tolerance وبخلاف هذا:

ب. استبدل a بـ $(a * a + x) / (2 * a)$ وكرر الخطوة a.

حيث تعبر $|a * a - x|$ عن القيمة المطلقة لـ $a * a - x$.

اكتب دالة تكرارية لتطبيق هذا الحل الحسابي لايجاد الجذر التربيعي لعدد حقيقي غير سالب واكتب أيضاً برنامج لاختبار دالتك.

17. اكتب برنامج لايجاد حلول للغز الثماني ملكات لقيم متنوعة من n . لكي تكون محدداً اختبر برنامجك عندما تكون $n = 4$ و $n = 8$.

18. (برج الفارس) هذا الفصل قام بوصف حل التراجع الحسابي وكيفية استخدام التكرار لتطبيقه. هناك مشكلة أخرى بلوحة الشطرنج يمكن حلها باستخدام حل التراجع الحسابي وهي برج الفارس. باعطائك موضع أولي على اللوحة حدد تتابع الحركات بواسطة فارس يزور كل مربع من لوحة الشطرنج مرة واحدة. على سبيل المثال بالنسبة الى اللوحات المربعة 5×5 أو 6×6 فان شكل 6-17 يوضح تتابع الحركات.

| | | | | |
|----|----|----|----|----|
| 1 | 6 | 15 | 10 | 21 |
| 14 | 9 | 20 | 5 | 16 |
| 19 | 2 | 7 | 22 | 11 |
| 8 | 13 | 24 | 17 | 4 |
| 25 | 18 | 3 | 12 | 23 |

| | | | | | |
|----|----|----|----|----|----|
| 1 | 16 | 7 | 26 | 11 | 14 |
| 34 | 25 | 12 | 15 | 6 | 27 |
| 17 | 2 | 33 | 8 | 13 | 10 |
| 32 | 35 | 24 | 21 | 28 | 5 |
| 23 | 18 | 3 | 30 | 9 | 20 |
| 36 | 31 | 22 | 19 | 4 | 29 |

شكل 6-17: برج الفارس.

يتحرك الفارس عن طريق القفز موضعين اما أفقياً أو رأسياً وموضع واحد في الاتجاه العمودي. اكتب برنامج تراجع تكراري يتخذ من موضع اللوحة الأولي مدخلات له ويحدد تتابع الحركات بواسطة فارس يزور كل مربع من اللوحة مرة واحدة فقط.

الفصل

7

المرصوصات

في هذا الفصل سوف:

- تعرف الكثير عن المرصوصات.
- تدرس عمليات متنوعة للمرصوصات.
- تتعلم كيفية تطبيق المرصوصة كمصفوفة.
- تتعلم كيفية تطبيق المرصوصة كقائمة متصلة.
- تكتشف تطبيقات المرصوصات.
- تتعلم كيفية استخدام المرصوصات لازالة التكرار.
- تصبح على علم بفئة مكتبة القالب المعياري stack.

هذا الفصل يناقش بنية بيانات مفيدة للغاية تسمى ال المرصوصة. التي لها تطبيقات هائلة العدد في علوم الحاسب الآلي.

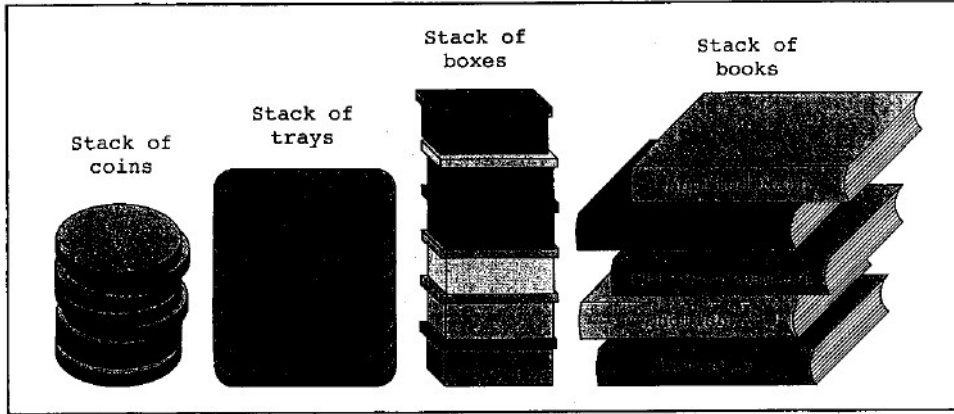
المرصوصات:

افترض أن لديك برنامج له عدة دالات ولكي تكون محدداً افترض أن برنامجك يمتلك الدالات أ، وب، وج، ود. افترض الآن أن الدالة أ تستدعي الدالة ب والدالة ب تستدعي الدالة ج والدالة ج تستدعي الدالة د. عندما تتوقف الدالة د يرجع التحكم الى الدالة ج وعندما تتوقف الدالة ج يعود التحكم الى الدالة ب وعندما تتوقف الدالة ب يعود التحكم الى الدالة أ. أثناء تنفيذ البرنامج كيف تعتقد أن الحاسب الآلي يتتبع استدعاءات الدالة؟ ماذا عن الدالات التكرارية؟ كيف يتتبع الحاسب الآلي الاستدعاءات التكرارية؟ في الفصل رقم 6 قمنا بتصميم دالة تكرارية لطبع قائمة متصلة من الخلف. ماذا اذا كنت تريد كتابة حل حسابي غير تكراري لطبع قائمة متصلة من الخلف؟

هذا القسم يقوم بمناقشة بنية البيانات المسماة المرصوصات. الحاسب الآلي يستخدم المرصوصات لتطبيق استدعاءات الدالة كما يمكنك استخدام المرصوصات لتحويل الحلول الحسابية التكرارية الى حلول حسابية غير تكرارية خاصة الحلول الحسابية التكرارية التي لا تكون تكرارية ذيلية. المرصوصات لها العديد من التطبيقات الأخرى في علوم الحاسب الآلي. بعد تنمية الأدوات اللازمة لتطبيق المرصوصة. سوف ندرس بعض من تطبيقات المرصوصات.

المرصوصة عبارة عن قائمة من عناصر متماثلة حيث يحدث اضافة وحذف العناصر عند نهاية واحد فقط تسمى قمة المرصوصة. على سبيل المثال يمكن في الكافيتيريا ازالة الصينية الثانية الموجودة في مرصوصة من الصواني فقط اذا تمت ازالة أول صينية. مثال آخر للحصول على كتابك المفضل في علوم الحاسب الآلي والذي يقع أسفل كتب الرياضيات والتاريخ الخاصة بك يجب عليك أن تقوم أولاً

بازالة كتب الرياضيات والتاريخ. بعد ازالة هذين الكتابين يصبح كتاب علوم الحاسب الآلي على قمة الكتب – أي العنصر العلوي من المرصوعة. الشكل 7-1 يوضح بعض الأمثلة على المرصوعات.

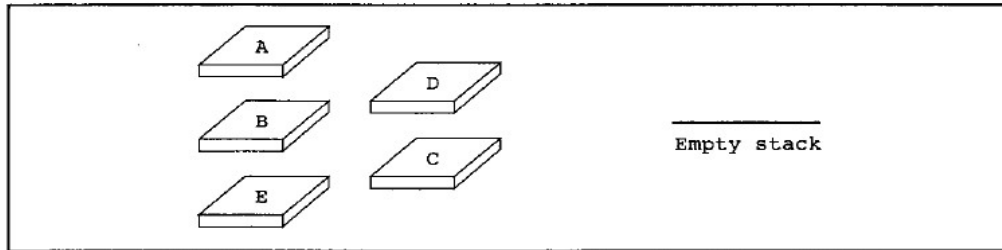


مرصوعة من الكتب مرصوعة من الصناديق مرصوعة من الصواني مرصوعة من العملات
شكل 7-1 : أنواع متنوعة من المرصوعات.

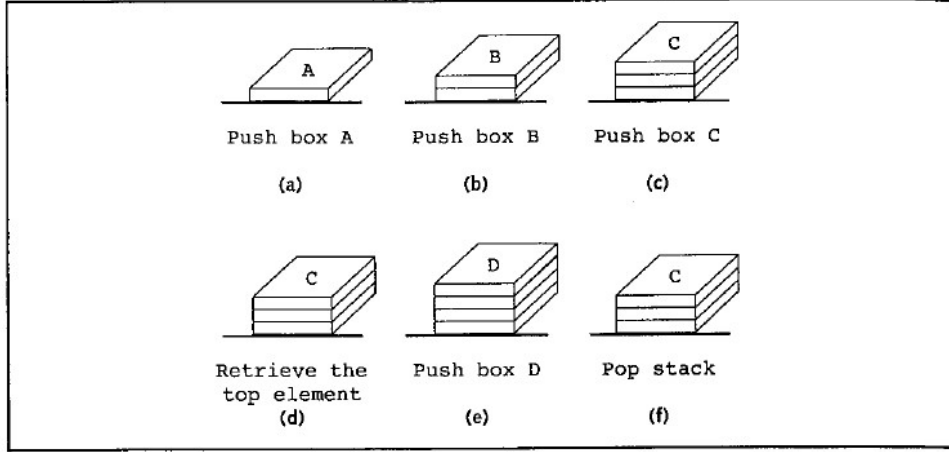
العناصر الموجودة في قاع المرصوعة كانت الأطول في المرصوعة والعنصر العلوي بالمرصوعة هو آخر عنصر تمت اضافته الى المرصوعة. بما أن العناصر تتم اضافتها وحذفها من نهاية واحدة (وهي القمة) اذن العنصر الأخير الذي تتم اضافتها سوف تتم ازالته أولاً. لهذا السبب يطلق على المرصوعة كذلك بنية بيانات الداخل أخيراً يخرج أولاً.

المرصوعة: بنية بيانات يتم فيها اضافة وحذف العناصر من نهاية واحدة فقط وهي بنية بيانات الداخل أخيراً يخرج أولاً.

الآن بعدما علمت ما هي المرصوعة لنقم برؤسة نوع العمليات التي يمكن أدائها على المرصوعة. بما أنه يمكن اضافة عناصر جديدة الى المرصوعة اذن يمكننا اجراء عملية الاضافة المسماة push لاضافة عنصر الى المرصوعة. بالمثل لما أنه يمكن استرجاع و/أو ازالة العنصر العلوي من المصفوفة اذن يمكننا اجراء العملية top لاسترجاع العنصر العلوي من المصفوفة والعملية pop لازالة العنصر العلوي من المصفوفة. الأشكال 7-2 و 7-3 توضح العمليات push و top و pop. العمليات push و top و pop تعمل كما يلي: افترض أن هناك صناديق تقع على الأرض وهناك حاجة الى رصها. مبدئياً تقع جميع الصناديق على الأرض والمرصوعة فارغة.



شكل 7-2: مرصوعة فارغة



شكل 3-7: عمليات المرسوعة.

بعد هذا ندفع الصندوق B الى المرسوعة وبعد عملية الدفع هذه تكون المرسوعة كما هي موضحة في شكل 3-7(ب) ثم ندفع الصندوق C الى المرسوعة وبعد عملية الدفع هذه تكون المرسوعة كما هي موضحة في شكل 3-7(ج). بعد هذا نستعيد العنصر العلوي من المرسوعة وبعد هذه العملية لا تصبح المرسوعة متغيرة وهي موضحة في شكل 3-7(د). ثم نقوم بدفع الصندوق D الى المرسوعة وبعد عملية الدفع هذه تصبح المرسوعة كما هي موضحة في شكل 3-7(هـ). بعد هذا نقوم باسترجاع المرسوعة وبعد هذه العملية تكون المرسوعة كما هي موضحة في شكل 3-7(و). يمكن حذف عنصر وأو استعادته من المرسوعة فقط اذا كان هناك شيئاً ما في المرسوعة ويمكن اضافة عنصر الى المرسوعة فقط اذا كان هناك مساحة. العمليات الناتجة من العمليات push و top و pop هي isFullStack (تتحقق مما اذا كانت المرسوعة ممتلئة) و isEmptyStack (تتحقق مما اذا كانت المرسوعة فارغة). بما أن المرسوعة تظل في تغير مع اضافتنا وازالتنا للعناصر اذن يجب أن تكون المرسوعة فارغة قبل أن نبدأ أولاً في استخدامها. لهذا نحن في حاجة الى عملية أخرى تسمى initializeStack التي تقوم بتهيئة المرسوعة في وضع فارغ. هناك عملية أخرى مفيدة تسمى destroyStack وتقوم هذه العملية عادةً بإزالة جميع العناصر من المرسوعة تاركية اياها في حالة فراغ. لهذا عند تطبيق المرسوعة بنجاح نحتاج على الأقل الى هذه العمليات السبع كما تم وصفها في القسم التالي. قد نحتاج كذلك الى عمليات أخرى على المرسوعة بناءً على التطبيق المحدد.

عمليات المرسوعة:

العمليات الأساسية على المرسوعات هي:

- initializeStack: تهيئ المرسوعة في حالة فراغ.
- destroyStack: تزيل جميع العناصر من المرسوعة تاركة اياها فارغة.

- **isEmptyStack**: تتحقق مما اذا كانت المرصوفة فارغة. اذا كانت المرصوفة فارغة تنتج القيمة true وبخلاف هذا تنتج القيمة false.
 - **IsFullStack**: تتحقق مما اذا كانت المرصوفة ممتلئة. اذا كانت المرصوفة ممتلئة تنتج القيمة true وبخلاف هذا تنتج القيمة false.
 - **push**: تضع عنصر جديد الى قمة المرصوفة ومدخلات هذه العملية تحتوي على المرصوفة والعنصر الجديد. قبل هذه العملية يجب أن تتواجد المرصوفة ويجب ألا تكون ممتلئة.
 - **top**: تنتج العنصر العلوي من المرصوفة وقبل هذه العملية يجب أن تتواجد المرصوفة ويجب ألا تكون فارغة.
 - **pop**: تزيل العنصر العلوي من المرصوفة وقبل هذه العملية يجب أن تتواجد المرصوفة ويجب ألا تكون فارغة.
- الآن نفكر في تطبيق بنية بيانات مرصوفة مجردة. بما أن جميع عناصر المرصوفة تكون من نفس النوع اذن يمكن تطبيق المرصوفة اما كمصفوفة أو كبنية متصلة. هذان التطبيقان مفيدان وتتم مناقشتهم في هذا الفصل.

تطبيق المرصوفات كمصفوفات:

بما أن جميع عناصر المرصوفة من نفس النوع اذن يمكنك استخدام مصفوفة لتطبيق مرصوفة. يمكن وضع أول عنصر من المصفوفة في الجزء الأول من المصفوفة والعنصر الثاني من المرصوفة في الجزء الثاني من المصفوفة وهكذا. قمة المرصوفة هي مؤشر العنصر الأخير الذي تتم اضافته الى المرصوفة.

في هذا التطبيق للمرصوفة يتم تخزين عناصر المرصوفة في مصفوفة والمصفوفة عبارة عن بنية بيانات تناول عشوائي أي أنه يمكنك تناول أي عنصر بالمصفوفة بشكل مباشر. بالرغم من هذا فان المرصوفة من تعريفها عبارة عن بنية بيانات يتم فيها تناول العناصر (استعادتها أو دفعها) عند نهاية واحدة فقط – أي بنية بيانات للداخل أخيراً يخرج أولاً. بهذا يتم تناول عنصر المرصوفة من خلال القمة فقط وليس من خلال القاع أو المنتصف. هذه الخاصية للمرصوفة شديدة الأهمية ويجب ادراكها في البداية.

لنتبع الموضع العلوي بالمصفوفة يمكننا اعلان متغير آخر بساطة يسمى stackTop.

الفئة التالية stackType تقوم بتعريف المرصوفة كنوع بيانات مجرد. باستخدام مؤشر يمكننا تخصيص المصفوفات حركياً لهذا نترك للمستخدم تحديد حجم المصفوفة (أي حجم المرصوفة). نفترض أن الحجم الافتراضي للمرصوفة هو 100. بما أن الفئة stackType لها عنصر بيانات مؤشر (المؤشر الى المصفوفة الذي يقوم بتخزين عناصر المرصوفة) اذن يجب أن نقوم باثاقل معامل الاسناد وادخال مقوم النسخ والمدمر. فضلاً عن هذا نعطي تعريف عام للمرصوفة.

بناءً على التطبيق المحدد يمكننا تمرير نوع عنصر المرسومة عند اعلاننا لهدف المرسومة.

```
template<class Type>
class stackType
{
public:
    const stackType<Type>& operator=(const stackType<Type>&);
    //Overload the assignment operator.
    void initializeStack();
    //Function to initialize the stack to an empty state.
    //Postcondition: stackTop = 0
```

// انقال معامل الاسناد.

// دالة لتهيئة المرسومة في وضع الفراغ.

// شرط

```
bool isEmptyStack();
//Function to determine whether the stack is empty.
//Postcondition: Returns true if the stack is empty;
// otherwise, returns false.
bool isFullStack();
//Function to determine whether the stack is full.
//Postcondition: Returns true if the stack is full;
// otherwise, returns false.
void destroyStack();
//Function to remove all the elements from the stack.
//Postcondition: stackTop = 0

void push(const Type& newItem);
//Function to add newItem to the stack.
//Precondition: The stack exists and is not full.
//Postcondition: The stack is changed and newItem
// is added to the top of stack.
Type top();
//Function to return the top element of the stack.
//Precondition: The stack exists and is not empty.
//Postcondition: If the stack is empty, the program
// terminates; otherwise, the top element
// of the stack is returned.
void pop();
//Function to remove the top element of the stack.
//Precondition: The stack exists and is not empty.
//Postcondition: The stack is changed and the top
// element is removed from the stack.

stackType(int stackSize = 100);
//constructor
//Creates an array of the size stackSize to hold the
//stack elements. The default stack size is 100.
//Postcondition: The variable list contains the base
// address of the array; stackTop = 0; and
// maxStackSize = stackSize.
stackType(const stackType<Type>& otherStack);
//copy constructor
~stackType();
//destructor
//Removes all the elements from the stack.
//Postcondition: The array (list) holding the stack
// elements is deleted.
```

private:

```
int maxStackSize; //variable to store the maximum stack size
int stackTop;     //variable to point to the top of the stack
Type *list;       //pointer to the array that holds the
                  //stack elements
```

```

bool isEmptyStack ( );
// دالة لتحديد ما اذا كانت المرصوفة فارغة.
// شرط تالي: تنتج true اذا كانت المرصوفة فارغة وبخلاف هذا تنتج false.
bool isFullStack ( );
// دالة لتحديد ما اذا كانت المرصوفة ممتلئة.
// شرط تالي: تنتج true اذا كانت المرصوفة ممتلئة وبخلاف هذا تنتج false.
void destroyStack ( );
// دالة لازالة جميع العناصر من المرصوفة.
// شرط تالي: قمة المرصوفة = 0
void push (const Type& newItem);
// دالة لاضافة عنصر جديد للمرصوفة.
// شرط مسبق: المرصوفة متواجدة وليست ممتلئة.
// شرط تالي: تتغير المرصوفة وتتم اضافة عنصر جديد
// الى قمة المرصوفة.
Type top ( );
// دالة لانتاج العنصر العلوي من المرصوفة.
// شرط مسبق: المرصوفة متواجدة وغير فارغة.
// شرط تالي: اذا كانت المرصوفة فارغة يتوقف البرنامج
// وبخلاف هذا يتم انتاج العنصر العلوي من المرصوفة.
void pop ( );
// دالة لازالة العنصر العلوي من المرصوفة.
// شرط مسبق: المرصوفة متواجدة وغير فارغة.
// شرط تالي: تتغير المرصوفة وتتم ازالة العنصر العلوي من المرصوفة.
stackType (int stackSize = 100);
// مقوم
// يخلق مصفوفة حجمها من حجم المرصوفة لحمل عناصر المرصوفة.
// الحجم الافتراضي للمرصوفة هو 100.
// شرط تالي: قائمة المتغير تحتوي على العنوان الرئيسي
// للمصفوفة وقمة المرصوفة = 0 وحجم المرصوفة الأقصى = حجم المرصوفة.
stackType (const stackType<Type>& otherStack);
// مقوم نسخ.
-stackType ( );
// مدمر
// يزيل جميع العناصر من المرصوفة.
// شرط تالي: يتم حذف المصفوفة الحاملة لعناصر المرصوفة.
// خاصة:
int maxStackSize;
// متغير لتخزين حجم المرصوفة الأقصى.
int stackType;
// متغير للإشارة الى قمة المرصوفة.
Type *list;
// مؤشر الى المصفوفة التي تحمل عناصر المرصوفة.

```

```

void copyStack(const stackType<Type>& otherStack);
//Function to make a copy of otherStack.
//Postcondition: A copy of otherStack is created and
//               assigned to this stack.
};

```

```

void copyStack (const stackType<Type>& otherStack);

```

// دالة لعمل نسخة من المرصوعة الأخرى.

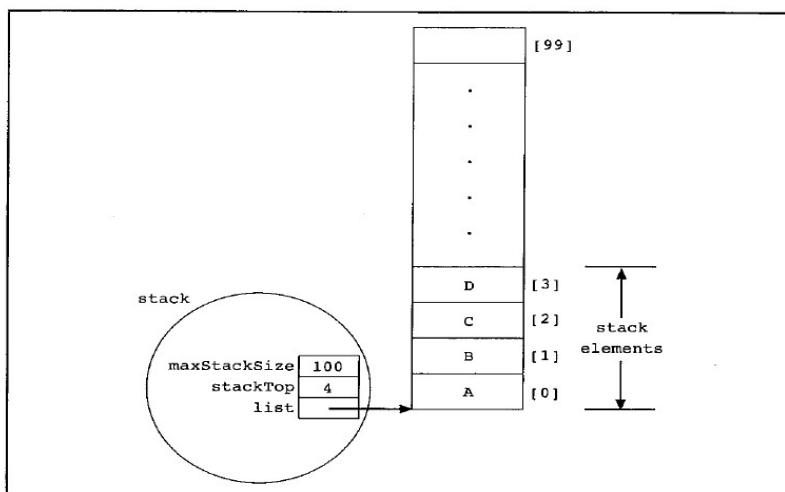
// شرط تالي: يتم عمل نسخة من المرصوعة الأخرى

// ويتم تحديثها لهذه المرصوعة.

بما أن مصفوفات C++ تبدأ بالمؤشر صفر فاننا نحتاج الى التمييز بين قيمة stackTop وموضع المصفوفة المشار اليه بواسطة stackTop. اذا كانت stackTop صفر تكون المرصوعة فارغة واذا لم تكن صفراً اذن فالمرصوعة غير فارغة ويتم اعطاء العنصر العلوي من المرصوعة بواسطة $1 - \text{stackTop}$.

لاحظ أن الدالة copyStack متضمنة كعنصر خاص. هذا لأننا نريد استخدام هذه الدالة فقط لتطبيق مقوم النسخ واثقال معامل الاسناد. لنسخ مرصوعة داخل مرصوعة أخرى يمكن للبرنامج استخدام معامل الاسناد.

الشكل 4-7 يوضح بيئة البيانات هذه حيث stack هدف من النوع stackType. لاحظ أن stackType يمكن أن تتراوح بين صفر الى حجم المرصوعة الأقصى. اذا لم تكن stackTop صفراً فان $1 - \text{stackType}$ يكون مؤشر العنصر stackTop من المصفوفة. افترض أن حجم المرصوعة الأقصى = 100.

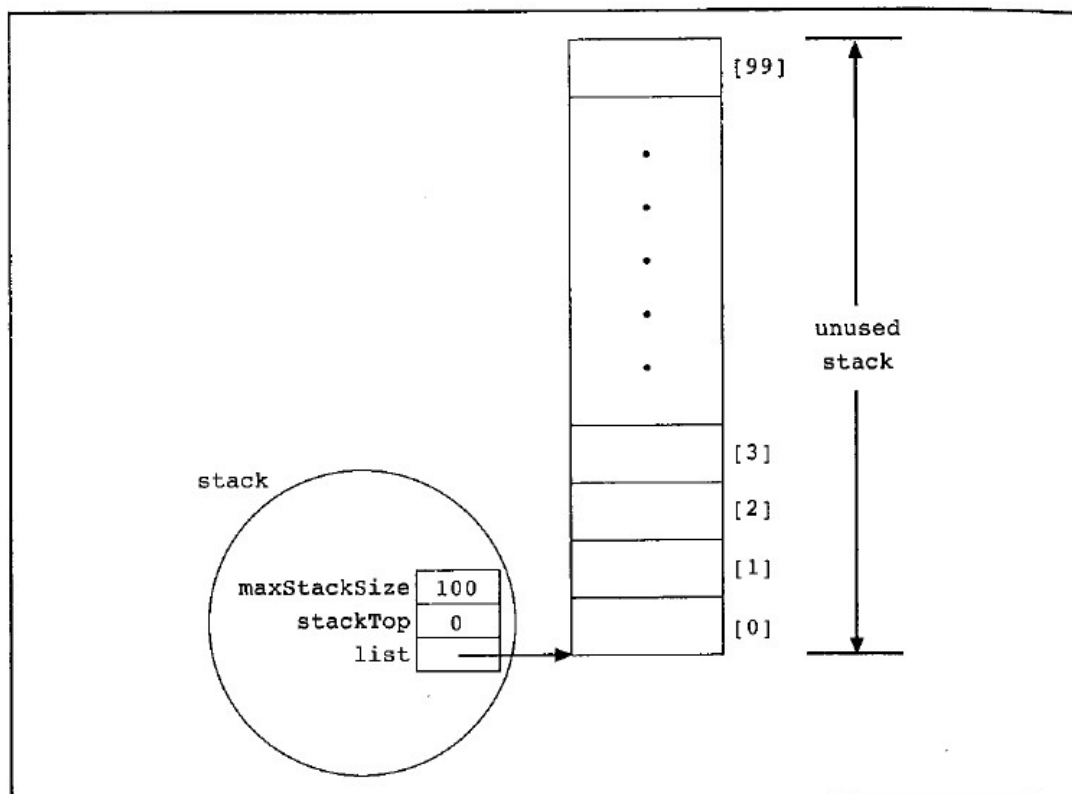


شكل 4-7: نموذج مرصوعة.

لاحظ أن المؤشر list يحتوي على العنوان الرئيسي للمصفوفة (الحاملة لعناصر المرصوعة) – أي عنوان مكون المصفوفة الأول. الأقسام الاحدى عشر القادمة من هذا الفصل تقوم بتعريف دالات عنصر الفئة stackType التي تطبق عمليات المرصوعة.

تهيئة المرصوعة:

لنقم بالنظر الى عملية initializeStack. بما أن قيمة stackTop تشير الى ما اذا كانت المرصوعة فارغة أم لا اذن يمكننا ببساطة تحديد stackop عند صفر لتهيئة المرصوعة. (انظر شكل 5-7).



شكل 5-7: مرصوعة فارغة.

تعريف الدالة initializeStack هو:

```
template<class Type>
void stackType<Type>::initializeStack()
{
    stackTop = 0;
} //end initializeStack
```

تدمير المرصوعة:

في تطبيق المصفوفة للمرصوعة تكون عملية تدمير المرصوعة مماثلة لعملية تهيئة المرصوعة. اذا قمنا بتحديد قيمة stackTop عند صفر فانه يتم تدمير جميع عناصر المرصوعة. بالرغم من أن جميع العناصر لا تزال في المرصوعة (يتم معاملتها كقمامة) الا أن قيمة stackTop تشير الى ما اذا كانت القائمة فارغة. تعريف الدالة destroyStack هو:

```
template<class Type>
void stackType<Type>::destroyStack()
{
    stackTop = 0;
} //end destroyStack
```

المرصوصة الفارغة:

لقد رأينا أن قيمة stackTop تشير الى ما اذا كانت القائمة فارغة أم لا. اذا كانت stackTop صفرا فان المرصوصة فارغة وبخلاف هذا لا تكون فارغة. تعريف الدالة isEmptyStack هو:

```
template<class Type>
bool stackType<Type>::isEmptyStack()
{
    return(stackTop == 0);
} //end isEmptyStack
```

مرصوصة ممتلئة:

بعد هذا ننظر الى العملية isFullStack. وينتج أن المرصوصة تكون ممتلئة اذا كانت stackTop مساوية لحجم المرصوصة الأقصى. تعريف الدالة isFullStack هو:

```
template<class Type>
bool stackType<Type>::isFullStack()
{
    return(stackTop == maxStackSize);
} //end isFullStack
```

الدفع:

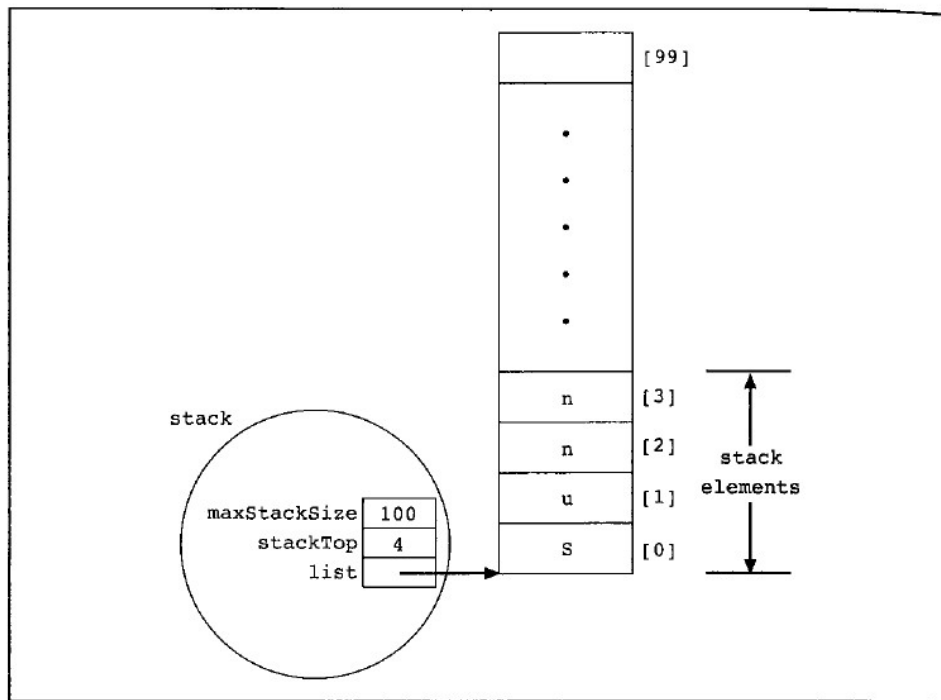
اضافة أو دفع عنصر الى المرصوصة يعتبر عملية على خطوتين. تذكر أن قيمة stackTop تشير الى عدد العناصر في المصفوفة و 1 - stackTop تعطي موضع العنصر stackTop في المرصوصة. لهذا تكون عملية الدفع كما يلي:

أ. تخزين عنصر جديد في مكون المصفوفة المشار اليه بواسطة stackTop.

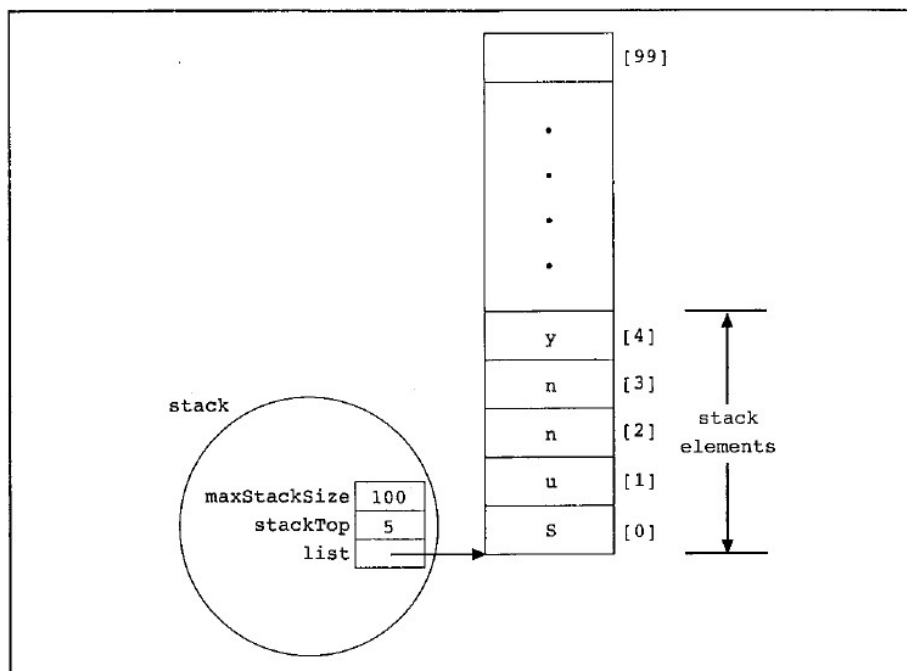
ب. زيادة stackTop.

الأشكال 6-7 و 7-7 توضح عملية الدفع.

افترض أنه قبل عملية الدفع كانت المرصوصة كما هي موضحة في شكل 6-7.



شكل 7-6: المرصوفة قبل دفع y .
 افترض أن العنصر الجديد هو y وبعد عملية الدفع تكون المرصوفة كما هي موضحة في شكل 7-7.



شكل 7-7: المرصوفة بعد دفع y .
 إذا كانت المرصوفة ممتلئة بالطبع لا يمكننا إضافة أية عناصر إلى المرصوفة. لهذا قبل إضافة عنصر إلى المرصوفة نتحقق الدالة $push$ مما إذا كانت القائمة ممتلئة. تعريف الدالة $push$ هو:

```

template<class Type>
void stackType<Type>::push(const Type& newItem)
{
    if(!isFullStack())
    {
        list[stackTop] = newItem;    //add newItem at the top
                                     //of the stack
        stackTop++;                  //increment stackTop
    }
    else
        cerr<<"Cannot add to a full stack."<<endl;
} //end push

```

إذا حاولنا إضافة عنصر جديد إلى مرصوفة ممثلة فان الحالة الناتجة تسمى **فائض**. يمكن التعامل مع تحقق خطأ وجود فائض بعدة طرق منها كما هو موضح أعلاه.

بدلاً من ذلك يمكننا التحقق من الفائض قبل استدعاء الدالة `push` كما هو موضح بعد هذا (بافتراض أن `stack` هدف من النوع `stackType`).

```

if(!stack.isFullStack())
    stack.push(name);

```

إنتاج العنصر العلوي:

العملية `top` تنتج العنصر العلوي من المرصوفة وتعريفها معطى بعد هذا.

```

template<class Type>
Type stackType<Type>::top()
{
    assert(stackTop != 0);           //if the stack is empty,
                                     //terminate the program
    return list[stackTop - 1];       //return the element of the
                                     //stack indicated by
                                     //stackTop - 1
} //end top

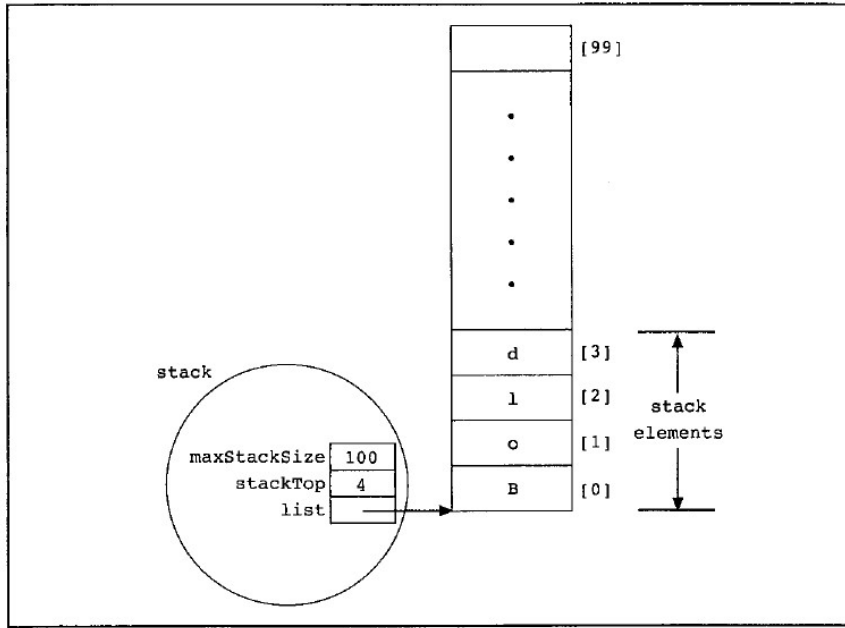
```

الطرح:

لإزالة أو طرح عنصر من المرصوفة نحتاج فقط إلى تقليل `stackTop` بمقدار 1.

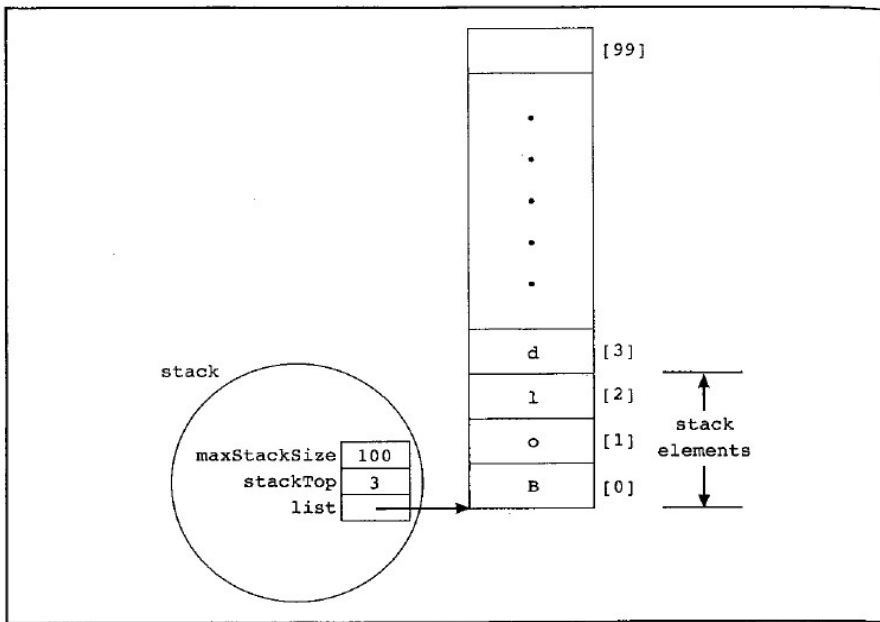
الأشكال 7-8 و 9-7 توضح عملية الطرح.

افترض أن المرصوفة قبل عملية الطرح كانت كما هي موضحة في شكل 7-8.



شكل 7-8: المرصوصة قبل طرح d.

بعد عملية الطرح تكون المرصوصة كما هي موضحة في شكل 7-9.



شكل 7-9: المرصوصة بعد طرح d.

إذا كانت المرصوصة فارغة بالطبع لا يمكننا إزالة أية عناصر من المرصوصة. لهذا قبل إزالة عنصر من المرصوصة نتحقق الدالة pop مما إذا كانت القائمة فارغة. باستخدام الحل الحسابي السابق يكون تعريف الدالة pop هو:

```

template<class Type>
void stackType<Type>::pop()
{
    if(!isEmptyStack())
        stackTop--;           //decrement stackTop
    else
        cerr<<"Cannot remove from an empty stack."<<endl;
} //end pop

```

إذا حاولنا إزالة عنصر من مرصوصة فارغة فإن الحالة الناتجة تسمى **عجز**. يمكن التعامل مع تحقق خطأ وجود عجز بعدة طرق منها كما هو موضح أعلاه. بدلاً من ذلك يمكننا التحقق من العجز قبل استدعاء الدالة pop كما هو موضح بعد هذا (بافتراض أن stack هدف من النوع stackType).

```

if(!stack.isEmptyStack())
    stack.pop();

```

نسخ مرصوصة:

تقوم الدالة copyStack بعمل نسخة من المرصوصة. المرصوصة التي يتم نسخها يتم تمريرها كعامل للدالة copyStack. سوف نقوم في الحقيقة باستخدام هذه الدالة لتطبيق مقوم النسخ واثقال معامل الاسناد. تعريف هذه الدالة هو:

```

template<class Type>
void stackType<Type>::copyStack(const stackType<Type>& otherStack)
{
    delete [] list;
    maxStackSize = otherStack.maxStackSize;
    stackTop = otherStack.stackTop;

    list = new Type[maxStackSize];
    assert(list != NULL);

    //copy otherStack into this stack
    for(int j = 0; j < stackTop; j++)
        list[j] = otherStack.list[j];
} //end copyStack

```

المقوم والمدمر:

الدالات لتطبيق المقوم والمدمر واضحة. المقوم يحدد حجم المرصوصة المحدد بواسطة المستخدم ويضع stackTop عند صفر ويصنع مصفوفة مناسبة يتم تخزين عناصر المرصوصة بها. إذا لم يتم المستخدم بتحديد حجم المصفوفة التي يتم تخزين عناصر المرصوصة بها يقوم المقوم باستخدام القيمة الافتراضية وهي 100 لعمل مصفوفة حجمها 100. يقوم المدمر ببساطة بإزالة تخصيص الذاكرة التي تحتلها المصفوفة (أي المرصوصة) وتضع stackTop عند صفر. تعريفات المقوم والمدمر تكون كما يلي:

```

        //constructor
template<class Type>
stackType<Type>::stackType(int stackSize)
{
    if(stackSize <= 0)
    {
        cerr<<"The size of the array to hold the stack must "
            <<"be positive."<<endl;
        cerr<<"Creating an array of size 100."<<endl;

        maxStackSize = 100;
    }
    else
        maxStackSize = stackSize; //set the stack size to
                                   //the value specified by
                                   //the parameter stackSize

    stackTop = 0; //set stackTop to 0
    list = new Type[maxStackSize]; //create the array to
                                   //hold the stack elements

    assert(list != NULL);
} //end constructor

template<class Type>
stackType<Type>::~~stackType() //destructor
{
    delete [] list; //deallocate the memory occupied by the array
} //end destructor

```

مقوم النسخ:

تذكر أن واحداً من المواقف التي يتم فيها تنفيذ مقوم النسخ هو عندما يتم تمرير أحد الأهداف كمعامل (قيمة) للدالة. انه يقوم بنسخ عناصر البيانات للمعامل الحقيقي داخل عناصر البيانات المتوافقة للمعامل الرسمي وتعريفه يكون:

```

template<class Type>
stackType<Type>::stackType(const stackType<Type>& otherStack)
{
    list = NULL;

    copyStack(otherStack);
} //end copy constructor

```

انقال معامل الاسناد (=):

تذكر أنه بالنسبة الى الفئات التي لديها عناصر بيانات مؤشر يجب أن يتم انقال معامل الاسناد بشكل واضح. تعريف الدالة الخاصة بانقال معامل الاسناد للفئة stackType هو:

```

template<class Type>
const stackType<Type>& stackType<Type>::operator=
    (const stackType<Type>& otherStack)
{
    if(this != &otherStack) //avoid self-copy
        copyStack(otherStack);

    return *this;
} //end operator=

```

تحليل عمليات المرصوصة مماثل لعمليات الفئة `arrayListType` (الفصل 3). لهذا فاننا نقدم موجز فقط في جدول 1-7.

الجدول 1-7 يلخص التركيب الزمني لعمليات الفئة `stackType`.

جدول 1-7: التركيب الزمني لعمليات الفئة `stackType`

| الدالة | التركيب الزمني |
|-------------------------------|----------------|
| <code>isEmptyStack</code> | $O(1)$ |
| <code>isFullStack</code> | $O(1)$ |
| <code>initializeStack</code> | $O(1)$ |
| <code>destroyStack</code> | $O(1)$ |
| المقوم (المقوم) constructor | $O(1)$ |
| <code>top</code> | $O(1)$ |
| <code>Push</code> | $O(1)$ |
| <code>pop</code> | $O(1)$ |
| <code>copyStack</code> | $O(n)$ |
| المدمر (المدمر) Destructor | $O(1)$ |
| Copy constructor (مقوم النسخ) | $O(n)$ |
| انقال معامل الاسناد | $O(n)$ |

ملف المرصوصة الرئيسي:

الآن بعد ما عرفت كيفية تطبيق عمليات المرصوصة يمكنك وضع تعريفات الفئة والدالات لتطبيق عمليات المرصوصة مع عمل الملف الرئيسي للمرصوصة. من أجل الكمال نقوم بعد هذا بتوضيح كيفية عمل الملف الرئيسي. افترض أن اسم الملف الرئيسي الذي يحتوي على تعريف الفئة `stackType` يسمى `myStack.h`. سوف نشير الى هذا الملف الرئيسي في أي برنامج يستخدم مرصوصة.

```
//Header file: myStack.h

#ifndef H_StackType
#define H_StackType

#include <iostream>
#include <cassert>

using namespace std;

//Place the definition of the class template stackType, as given
//previously in this chapter, here.

//Place the definitions of the member functions, as discussed in
//this chapter, here.

#endif
```

مثال 7-1:

قبل أن نعطي مثال برمجة لنقم أولاً بكتابة برنامج بسيط يستخدم الفئة `stackType` ويختبر بعض من عمليات المصفوفة. سوف نقوم من ضمن أشياء أخرى باختبار معامل الاسناد. البرنامج ومخرجاته تكون كما يلي. (نفترض أن تعريف الفئة `stackType` ودالات عناصره موجودة في الملف الرئيسي `myStack.h`.)

```
//Program to test the various operations of a stack
#include <iostream>
#include "myStack.h"

using namespace std;

int main()
{
    stackType<int> intStack(50);
    stackType<int> tempStack;

    intStack.push(23);
    intStack.push(45);
    intStack.push(38);

    tempStack = intStack; //copy intStack into tempStack

    cout<<"tempStack elements: ";

    while(!tempStack.isEmptyStack()) //print tempStack
    {
        cout<<tempStack.top()<<" ";
        tempStack.pop();
    }

    cout<<endl;

    cout<<"The top element of intStack: "<<intStack.top()<<endl;

    return 0;
}
```

المخرجات:

عناصر tempStack: 23 45 38

العنصر العلوي من intStack: 38

من الموصي به أن تقوم بعمل مرور لهذا البرنامج.

مثال برمجة: أعلى GPA:

في هذا المثال نقوم بكتابة برنامج C++ يقرأ ملف بيانات مكون من متوسط درجات كل طالب يليه اسم الطالب ثم يقوم البرنامج بطباعة أعلى متوسط درجات وأسماء جميع الطلاب الذين تلقوا هذا المتوسط للدرجات. البرنامج يسمح ملف المدخلات مرة واحدة فقط.

المدخلات: يقوم البرنامج بقراءة ملف المدخلات المكون من متوسط درجات كل طالب يليه اسم الطالب وعينة البيانات تكون كما يلي:

3.8 ليزا
3.6 جون
3.9 سوزان
3.7 كاثي
3.4 جاسون
3.9 دافيد
3.4 جاك

المخرجات: نتائج البرنامج سوف تكون أعلى متوسط درجات وجميع الأسماء المرتبطة بأعلى متوسط درجات. على سبيل المثال بالنسبة الى البيانات أعلاه يكون أعلى متوسط درجات هو 3.9 والطلاب الحاصلين على متوسط الدرجات هذا هم سوزان ودافيد.

تحليل البرنامج وتصميم الحل الحسابي:

اننا نقرأ متوسط الدرجات الأول واسم الطالب. بما أن هذه البيانات هي أول عنصر نقرأه فانه يكون أعلى متوسط درجات حتى الآن. بعد هذا نقرأ متوسط الدرجات الثاني واسم الطالب. عندها يمكننا مقارنة معدل الدرجات (الثاني) مع أعلى متوسط درجات حتى الآن. تظهر ثلاث حالات:

1. متوسط الدرجات الجديد أكبر من أعلى متوسط درجات حتى الآن. في هذه الحالة نقوم

أ- بتحديث قيمة أكبر متوسط درجات حتى الآن.

ب- تدمير المرصوصة – أي ازالة أسماء الطلاب من المرصوصة.

ت- حفظ اسم الطالب الحائل على أعلى متوسط درجات حتى الآن في المرصوصة.

2. متوسط الدرجات الجديد متساوي مع أعلى متوسط درجات حتى الآن. في هذه الحالة نضيف اسم الطالب الجديد الى المرصوصة.

3. متوسط الدرجات الجديد أقل من أعلى متوسط درجات حتى الآن. في هذه الحالة نهمل اسم الطالب الحاصل على الدرجة.

بعد هذا نقرأ متوسط الدرجات التالي واسم الطالب ونكرر الخطوات من 1 الى 3 ونستمر في هذه العملية حتى نصل الى نهاية الملف.

من هذه المناقشة يتضح أننا في حاجة الى المتغيرات التالية:

```

double GPA; // متغير لحمل متوسط الدرجات الحالي.
double highestGPA; // متغير لحمل أعلى متوسط درجات
newString name; // متغير لحمل اسم الطالب
stackType<newString> stack; // هدف لتطبيق المرصوصة

```

لاحظ أننا نستخدم الفئة newString التي قمنا بتصميمها في الفصل 3 من أجل اسم الطالب ويمكننا استخدام معامل الاسناد لتخزين اسم فس متغير من النوع newString. تتم ترجمة المناقشة السابقة الى الحل الحسابي التالي:

1. اعلن المتغيرات.
2. افتح ملف المدخلات.
3. اذا لم يكن ملف المدخلات موجوداً اخرج من البرنامج.
4. حدد مخرجات الأعداد ذات العلامة العشرية عند صيغة عشرية ثابتة ذات علامة عشرية والأصفار. وقم كذلك بتحديد دقة المنزلتين العشريتين.
5. اقرأ متوسط الدرجات واسم الطالب.
6. أعلى متوسط درجات = متوسط الدرجات
7. قم بتهيئة المرصوصة.
- 8.

```

while (not end of file)
{
    8.1 if (GPA > highestGPA)
    {
        8.1.1 destroyStack(stack);
        8.1.2 push(stack, student name);
        8.1.3 highestGPA = GPA;
    }
    8.2 else
        if(GPA is equal to highestGPA)
            push(stack, student name);
    8.3 Read the GPA and student name;
}

```

9. استخرج أعلى متوسط درجات.
10. اخرج أسماء الطلاب الحاصلين على أعلى متوسط درجات.

قائمة برنامج كاملة:

// البرنامج: أعلى متوسط درجات

```

//Program: Highest GPA
#include <iostream>
#include <iomanip>
#include <fstream>
#include "newString.h"
#include "myStack.h"

using namespace std;

int main()
{
    //Step 1
    double GPA;
    double highestGPA;
    newString name;

    stackType<newString> stack(100);
    ifstream infile;

    infile.open("a:\\Ch7_HighestGPADData.txt"); //Step 2
    if(!infile) //Step 3
    {
        cerr<<"The input file does not exist. "
        <<"Program terminates!"<<endl;
        return 1;
    }

    cout<<fixed<<showpoint; //Step 4
    cout<<setprecision(2); //Step 4

    infile>>GPA>>name; //Step 5
    highestGPA = GPA; //Step 6
    stack.initializeStack(); //Step 7
    while(infile) //Step 8
    {
        if(GPA > highestGPA) //Step 8.1
        {
            stack.destroyStack(); //Step 8.1.1
            if(!stack.isFullStack()) //Step 8.1.2
                stack.push(name);
            highestGPA = GPA; //Step 8.1.3
        }
        else
        {
            if(GPA == highestGPA) //Step 8.2
            {
                if(!stack.isFullStack())
                    stack.push(name);
                else
                {
                    cerr<<"Stack overflow. Program terminates."
                    <<endl;
                    return 1; //exit the program
                }
            }
            infile>>GPA>>name; //Step 8.3
        }
    }

    cout<<"Highest GPA = "<<highestGPA<<endl; //Step 9
    cout<<"The students holding the highest GPA are:"
    <<endl;

    while(!stack.isEmptyStack()) //Step 10
    {
        name = stack.top();
    }
}

```

```

        stack.pop();
        cout<<name<<endl;
    }

    cout<<endl;
    return 0;
}

```

تنفيذ العينة:

ملف المدخلات: (Ch7_HighestGPADData.txt)

| | |
|-----|--------|
| 3.4 | هولت |
| 3.2 | بولت |
| 2.5 | كولت |
| 3.4 | توم |
| 3.8 | رون |
| 3.8 | ميكي |
| 3.6 | بلوتو |
| 3.5 | دونالد |
| 3.8 | سيندي |
| 3.7 | دووم |
| 3.9 | أندي |
| 3.8 | فوكس |
| 3.9 | ميني |
| 2.7 | جوفي |
| 3.9 | دوك |
| 3.4 | داني |

المخرجات:

أعلى متوسط درجات = 3.90
 الطلاب الحاصلين على أعلى متوسط درجات هم:
 دوك
 ميني
 أندي

التطبيق المتصل للمرصوعات:

بما أن حج أي مصفوفة يكون ثابت فانه يمكن في تمثيل المصفوفة (الخطي) للمرصوفة دفع عدد ثابت فقط من العناصر داخل المرصوفة. اذا تجاوز عدد العناصر التي يتم دفعها في برنامج ما حجم المصفوفة فان البرنامج قد يوقف في خطأ ما. يجب علينا التغلب على هذه المشكلات.

لقد رأينا أنه باستخدام متغيرات المؤشر يمكننا تخصيص وازالة تخصيص الذاكرة بشكل حركي ويمكننا باستخدام القوائم المتصلة أن نقوم بتنظيم البيانات حركياً (مثل قائمة مرتبة). بعد هذا نستخدم هذه المفاهيم لتطبيق المرصوفة حركياً .

تذكر أنه في التمثيل الطولي للمرصوصة تشير قيمة stackTop الى عدد العناصر في المرصوصة وتشير قيمة stackTop - 1 الى العنصر العلوي في المرصوصة. يمكننا بمساعدة stackTop أن نقوم بعدة أشياء: ايجاد العنصر العلوي، والتحقق مما اذا كانت المرصوصة فارغة، وهكذا. يتم استخدام stackTop في التمثيل المتصل مثل استخدامها في التمثيل الطولي لتحديد العنصر العلوي للمرصوصة. بالرغم من هذا هناك اختلاف طفيف حيث أنه في الحالة السابقة تقوم stackTop باعطاء مؤشر المصفوفة بينما في الحالة الأخيرة تعطي عنوان (موقع ذاكرة) العنصر العلوي من المرصوصة.

البيانات التالية تقوم بتعريف مرصوصة متصلة كنوع بيانات مجرد:

```
//Definition of the node
template<class Type>
struct nodeType
{
    Type info;
    nodeType<Type> *link;
};

template<class Type>
class linkedStackType
{
public:
    const linkedStackType<Type>& operator=
        (const linkedStackType<Type>&);
    //Overload the assignment operator.
    void initializeStack();
    //Function to initialize the stack to an empty state.
    //Postcondition: The stack elements are removed;
    //                stackTop = NULL.
    bool isEmptyStack();
    //Function to determine whether the stack is empty.
    //Postcondition: Returns true if the stack is empty;
    //                otherwise, returns false.
    bool isFullStack();
    //Function to determine whether the stack is full;
    //Postcondition: Returns false.

    void destroyStack();
    //Function to remove all the elements of the stack,
    //leaving the stack in an empty state.
    //Postcondition: stackTop = NULL

    void push(const Type& newItem);
    //Function to add newItem to the stack.
    //Precondition: The stack exists and is not full.
    //Postcondition: The stack is changed and newItem
    //                is added to the top of stack.
```

// انقال معامل الاسناد.

void initializeStack ();

// دالة لتهيئة المرصوصة في وضع فارغ.

```

// شرط تالي: يتم ازالة عناصر المرصوعة وتكون NULL = stackTop
bool isEmptyStack ( );
// دالة لتحديد ما اذا كانت المرصوعة فارغة.
// شرط تالي: تنتج true اذا كانت المرصوعة فارغة وبخلاف هذا تنتج false.
bool isFullStack ( );
// دالة لتحديد ما اذا كانت المرصوعة ممتلئة.
// شرط تالي: تنتج false.
void destroyStack ( );
// دالة لازالة جميع العناصر من المرصوعة تاركة اياها في وضع فارغ.
// شرط تالي: قمة المرصوعة = 0
void push (const Type& newItem);
// دالة لاضافة عنصر جديد للمرصوعة.
// شرط مسبق: المرصوعة متواجدة وليست ممتلئة.
// شرط تالي: تتغير المرصوعة ويتم اضافة عنصر جديد
// الى قمة المرصوعة.

```

```

Type top();
//Function to return the top element of the stack.
//Precondition: The stack exists and is not empty.
//Postcondition: If the stack is empty, the program
//               terminates; otherwise, the top element
//               of the stack is returned.

void pop();
//Function to remove the top element of the stack.
//Precondition: The stack exists and is not empty.
//Postcondition: The stack is changed and the top element
//               is removed from the stack.

linkedListType();
//default constructor
//Postcondition: stackTop = NULL
linkedListType(const linkedListType<Type>& otherStack);
//copy constructor
~linkedListType();
//destructor
//Postcondition: All the elements of the stack are removed
//               from the stack.

private:
nodeType<Type> *stackTop; //pointer to the stack

void copyStack(const linkedListType<Type>& otherStack);
//Function to make a copy of otherStack.
//Postcondition: A copy of otherStack is created and
//               assigned to this stack.
};

```

```

Type top ( );
// دالة لانتاج العنصر العلوي من المرصوعة.
// شرط مسبق: المرصوعة متواجدة وغير فارغة.

```

```

// شرط تالي: اذا كانت المرصوصة فارغة يتوقف البرنامج
// وبخلاف هذا يتم انتاج العنصر العلوي من المرصوصة.
void pop ( );
// دالة لازالة العنصر العلوي من المرصوصة.
// شرط مسبق: المرصوصة متواجدة و غير فارغة.
// شرط تالي: تتغير المرصوصة وتتم ازالة العنصر العلوي من المرصوصة.
linkedStackType ( );
// مقوم افتراضي
NULL = stackTop : شرط تالي
linkedStackType (const linkedStackType<Type>& otherStack);
// مقوم نسخ.
-linkedStackType ( );
// مدمر
// شرط تالي: يتم حذف جميع عناصر المرصوصة من المرصوصة.
// خاصة:
nodeType<Type> *stackTop; // مؤشر الى المرصوصة.
void copyStack (const linkedStackType<Type>& otherStack);
// دالة لعمل نسخة من otherStack.
// شرط تالي: يتم عمل نسخة من otherStack
// ويتم تحديدها لهذه المرصوصة.

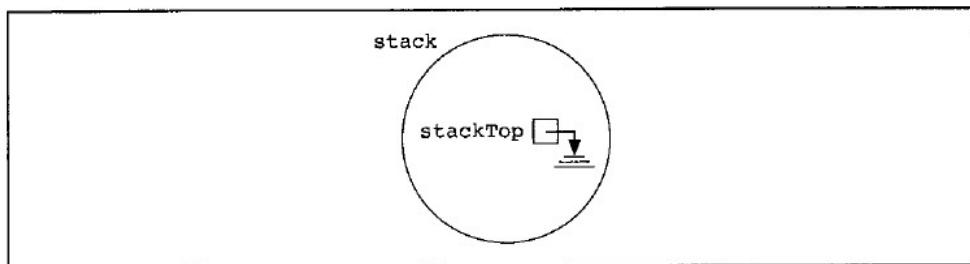
```

في هذا التطبيق المتصل للمرصوصات يتم تخصيص الذاكرة التي تقوم بتخزين عناصر المرصوصة حركياً. لهذا لا تمتلئ المرصوصة أبداً من الناحية المنطقية. بهذا لا يكون من اللازم تطبيق العملية isFullStack لتحديد ما اذا كانت المرصوصة ممتلئة. بالرغم من هذا فان تطبيق المرصوصات كمصفوفات به العملية isFullStack لتحديد ما اذا كانت المرصوصة ممتلئة. مستخدم المرصوصة لا يحتاج الى معرفة تفاصيل تطبيق المرصوصة ولهذا فان تطبيق المرصوصات كقوائم متصلة يشمل كذلك هذه العملية.

المثال التالي يوضح كيفية ظهور المرصوصات الفارغة والغير فارغة.

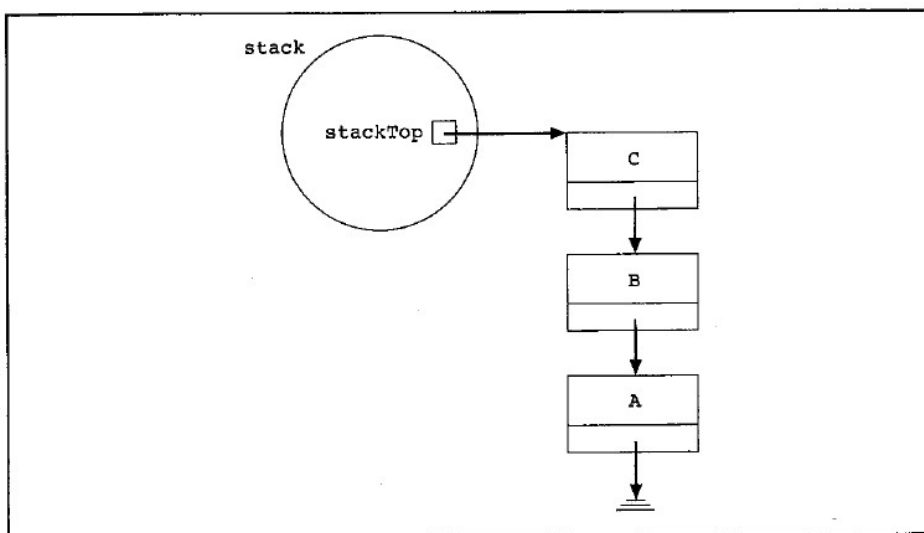
مثال 2-7:

مرصوصة فارغة: افترض أن هذه المرصوعة هدف من النوع `linkedStackType`.
(انظر شكل 10-7).



شكل 10-7: مرصوصة متصلة فارغة

مرصوصة غير فارغة: شكل 11-7 يوضح مرصوصة غير فارغة.



شكل 11-7: مرصوصة متصلة غير فارغة

في الشكل 11-7 العنصر العلوي من المرصوعة هو C وهذا يعني أن العنصر الأخير الذي تم دفعه إلى المرصوعة هو C.

بعد هذا نناقش تعريفات الدالات لتطبيق عمليات المرصوعة المتصلة.

المقوم الافتراضي:

العملية الأولى التي ننظر إليها هي المقوم الافتراضي. يقوم المقوم الافتراضي بتهيئة المرصوعة في وضع فارغ عندما يتم اعلان هدف المرصوعة. لهذا تقوم الدالة بتحديد `stackTop` عند `NULL`. تعريف هذه الدالة هو:

```
template<class Type> //default constructor
linkedStackType<Type>::linkedStackType()
{
    stackTop = NULL;
}
```

تدمير المرصوعة:

في التمثيل المتصل تقوم الدالة destroyStack بعمل أكبر مما تفعله في تمثيل المصفوفة. في تمثيل المصفوفة يتم تدمير المرصوعة ببساطة عن طريق تحديد stackTop عند صفر. في التمثيل المتصل يتم تخصيص ذاكرة لعناصر المرصوعة حركياً. بهذا نحتاج الى تحديد stackTop عند NULL ويجب أن نزيل تخصيص الذاكرة التي تحتلها عناصر المرصوعة وتعريف هذه الدالة هو:

```
template<class Type>
void linkedStackType<Type>::destroyStack()
{
    nodeType<Type> *temp;    //pointer to delete the node

    while(stackTop != NULL) //while there are elements
                            //in the stack
    {
        temp = stackTop;    //set temp to point to
                            //the current node
        stackTop = stackTop->link; //advance stackTop
                                //to the next node
        delete temp;        //deallocate the memory
                            //occupied by temp
    }
} //end destroyStack
```

تهيئة المرصوعة:

عملية تهيئة المرصوعة تعيد تهيئة المرصوعة في وضع فارغ. بما أن المرصوعة قد تحتوي على بعض العناصر وبما أننا نستخدم تطبيق متصل للمرصوعة اذن يجب أن نزيل تخصيص الذاكرة التي تحتلها عناصر المرصوعة. يمكن انجاز هذه المهمة عن طريق استدعاء دالة العنصر destroyStack. لاحظ أن الدالة destroyList أيضاً تحدد stackTop عند NULL.

تعريف الدالة initializeStack هو:

```
template<class Type>
void linkedStackType<Type>:: initializeStack()
{
    destroyStack();
}
```

يتم النظر فيما بعد الى العمليتين isEmptyStack و isFullStack. تكون المرصوعة فارغة اذا كانت stackTop تساوي NULL. بما أن الذاكرة الخاصة بعنصر المرصوعة يتم تخصيصها وازالة تخصيصها حركياً فان المرصوعة لا تكون ممثلة أبداً. (تكون المرصوعة ممثلة اذا نفذت

منا الذاكرة) لهذا دائماً تقوم الدالة `isFullStack` بإنتاج القيمة `false`. تعريفات الدالات لتطبيق هذه العمليات هي:

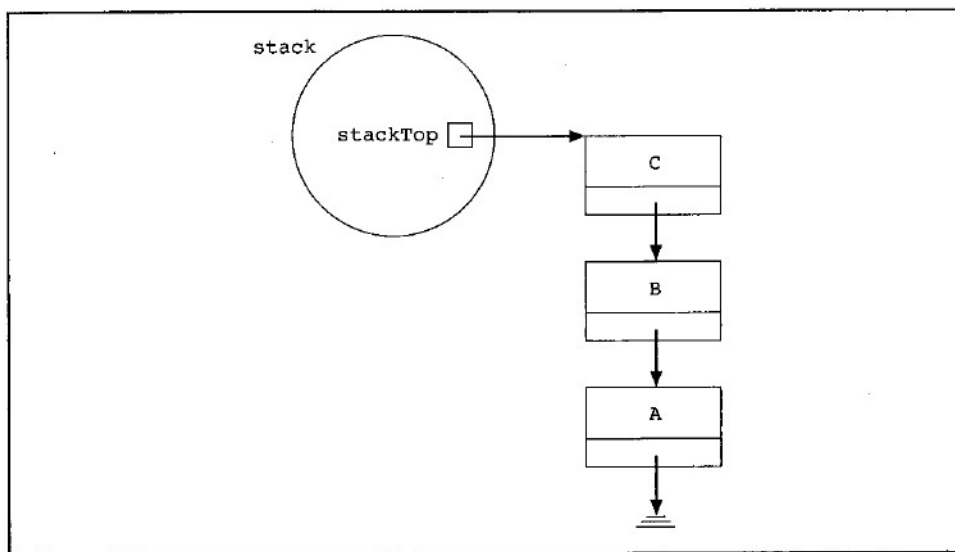
```
template<class Type>
bool linkedStackType<Type>::isEmptyStack()
{
    return(stackTop == NULL);
}

template<class Type>
bool linkedStackType<Type>::isFullStack()
{
    return false;
}
```

بعد هذا ننظر الى العمليات `push` و `top` و `pop`. من الشكل 11-7 يتضح أن العنصر الجديد تتم اضافته (في حالة `push`) الى بداية القائمة المتصلة المشار اليها بواسطة `stackTop`. في حالة `pop` تتم ازالة العقدة المشار اليها بواسطة `stackTop` وفي كلتا الحالتين يتم تحديث قيمة المؤشر `stackTop`.

:Push

انظر الى المرصوفة الموضحة في شكل 12-7.

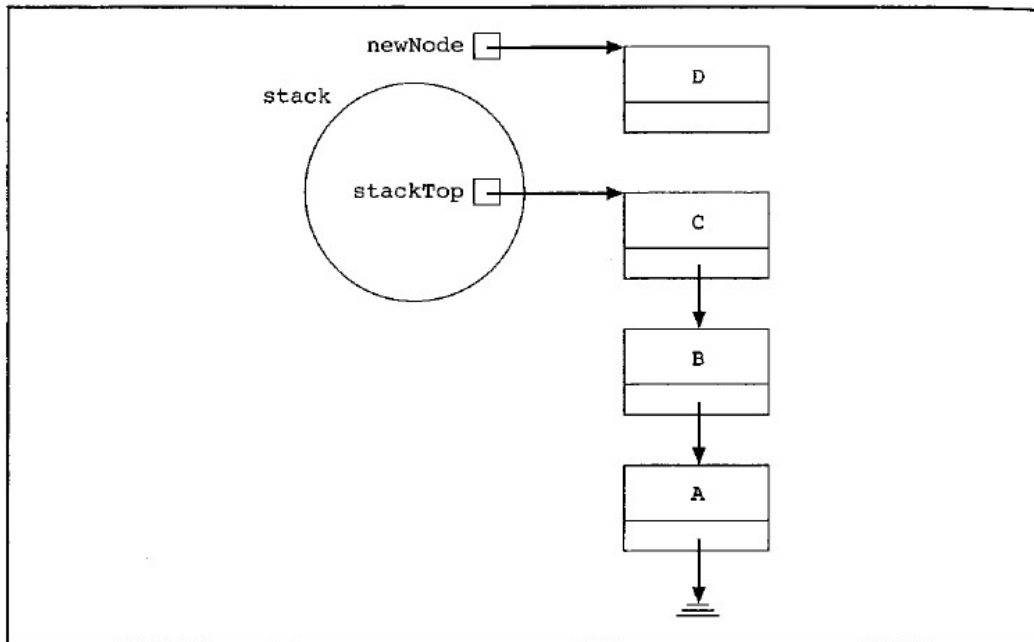


شكل 12-7: مرصوفة قبل عملية الدفع.

افترض أن العنصر الجديد الذي يتم دفعه هو `D`. أولاً نقوم بتخصيص ذاكرة من أجل العقدة الجديدة ثم نقوم بتخزين `D` في العقدة الجديدة وندخل العقدة الجديدة في بداية القائمة. أخيراً نقوم بتحديث قيمة `stackTop` البيانات:

```
newNode = new nodeType<Type>; //create the new node
assert(newNode != NULL); //if unable to allocate memory,
                          //terminate the program
newNode->info = newElement;
```

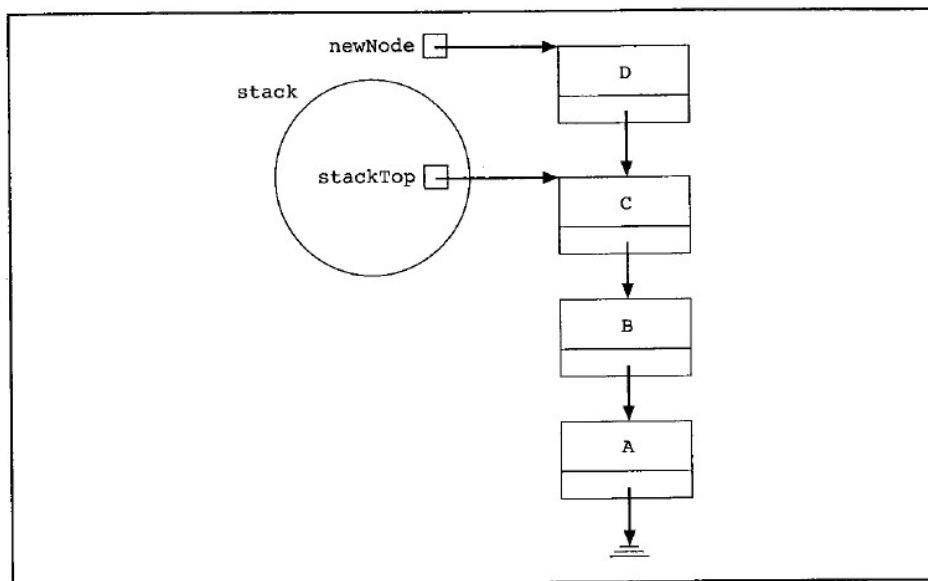
يصنع عقدة ويقوم بتخزين عنوان العقدة في المتغير newNode وتخزين عنصر جديد في حقل info الخاص بالعقدة الجديدة. لهذا يكون لدينا الموقف الموضح في شكل 7-13.



شكل 7-13: المرصوفة وعقدة جديدة

البيان: $newNode \rightarrow link = stackTop$;

يقوم بادخال عقدة جديدة في قمة المرصوفة كما هو موضح في شكل 7-14.

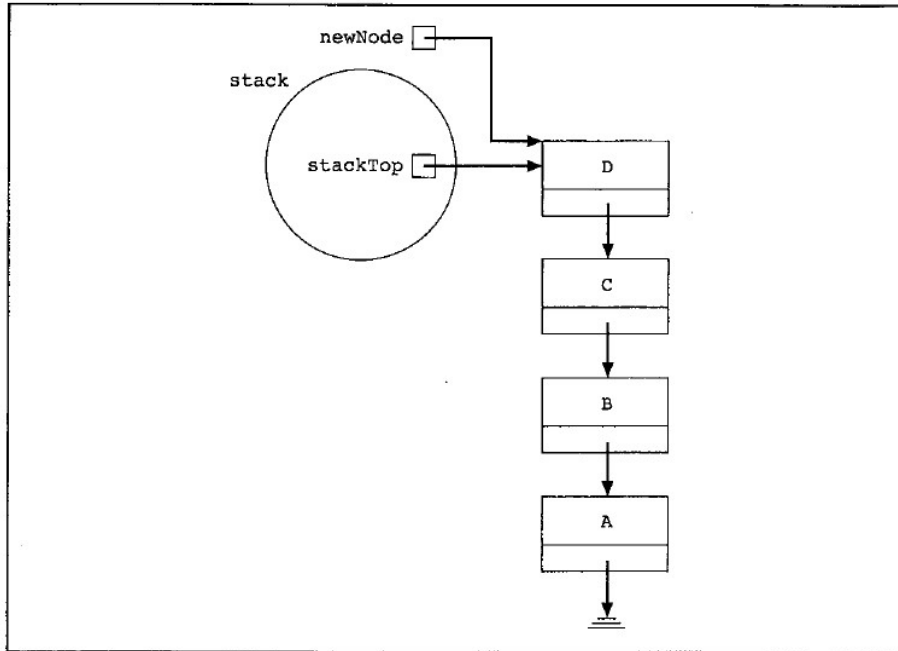


شكل 7-14: المرصوفة بعد تنفيذ البيان $newNode \rightarrow link = stackTop$;

أخيراً يقوم البيان:

stackTop = newNode;

بتحديث قيمة stackTop التي تنتج في الشكل 7-15.



شكل 7-15: المرصوفة بعد تنفيذ البيان stackTop = newNode;

تعريف الدالة push هو:

```
template<class Type>
void linkedStackType<Type>::push(const Type& newElement)
{
    nodeType<Type> *newNode;          //pointer to create the new node

    newNode = new nodeType<Type>; //create the node
    assert(newNode != NULL);

    newNode->info = newElement; //store newElement in the node
    newNode->link = stackTop;    //insert newNode before stackTop
    stackTop = newNode;         //set stackTop to point to the
                                //top node
} //end push
```

لا نحتاج الى التحقق مما اذا كانت المرصوفة ممثلة أم لا قبل أن ندفع عنصر الى المرصوفة لأن في هذا التطبيق لا تصبح المرصوفة ممثلة أبداً .

انتاج العنصر العلوي:

العملية التي يتم بها انتاج العنصر العلوي من المرصوفة واضحة تماماً وتعريفها يكون:

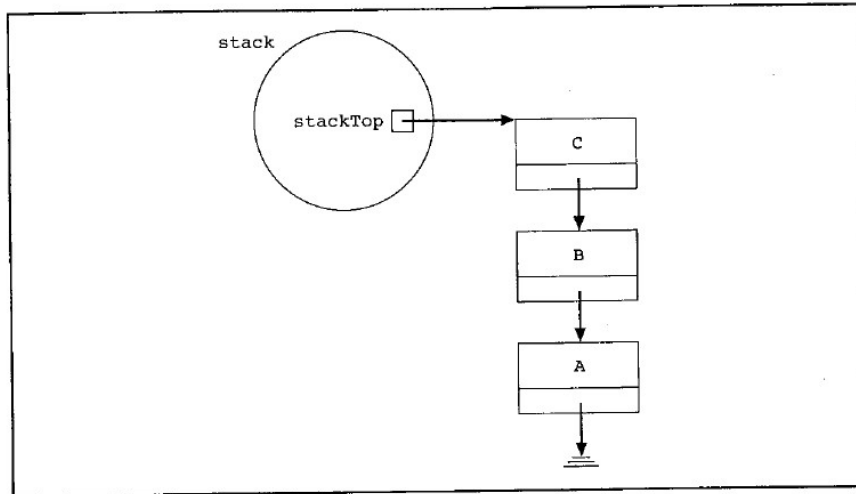
```

template<class Type>
Type linkedStackType<Type>::top()
{
    assert(stackTop != NULL);           //if the stack is empty,
                                        //terminate the program
    return stackTop->info;               //return the top element
}
//end top

```

:pop

الآن ننظر الى عملية pop التي تزيل العنصر العلوي من المرصوصة. انظر الى المرصوصة الموضحة في شكل 16-7.



شكل 16-7: المرصوصة قبل عملية الازالة.

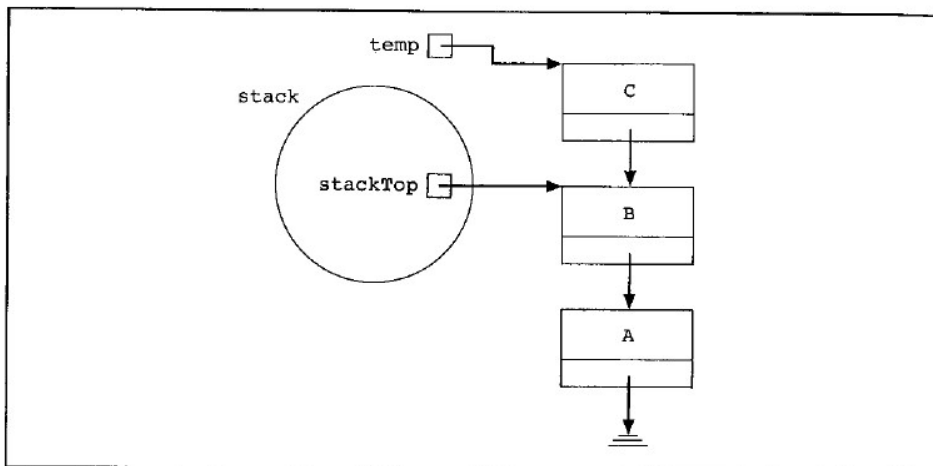
البيان:

temp = stackType;

يجعل temp يشير الى قمة المرصوصة والبيان:

stackTop = stackTop-> link;

يجعل العنصر الثاني بالمرصوصة ب\يصبح العنصر العلوي بالمرصوصة وعندها يكون لدينا الموقف الموضح في شكل 17-7.



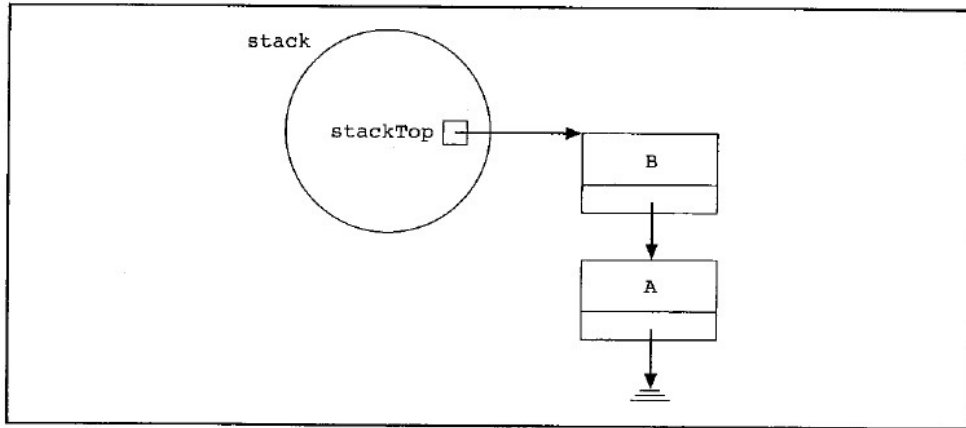
شكل 7-17: المرصوصة بعد تنفيذ البيانان:

stackTop = stackTop-> link; و temp = stackType;

أخيراً يقوم البيان:

delete temp;

بإزالة تخصيص الذاكرة المشار إليها بواسطة temp. الشكل 7-18 يوضح المرصوصة الناتجة.



شكل 7-18: المرصوصة بعد تنفيذ البيان delete temp;

تعريف الدالة pop هو:

```
template<class Type>
void linkedStackType<Type>::pop()
{
    nodeType<Type> *temp;           //pointer to deallocate the memory
    if(stackTop != NULL)             //if stack is nonempty
    {
        temp = stackTop;             //set temp to point to the top node
        stackTop = stackTop->link;    //advance stackTop to the next node
        delete temp;                 //delete the top node
    }
    else
        cerr<<"Cannot remove from an empty stack."<<endl;
} //end pop
```

لقد ناقشنا بالفعل المقوم الافتراضي. لإكمال تطبيق عمليات المرصوصة نحتاج إلى كتابة تعريفات الدالة copyStack ومقوم النسخ، والمدمر، والدالة لإثقال معامل الاسناد. هذه الدالات مماثلة لتلك الدالات التي تمت مناقشتها بالنسبة إلى القوائم المتصلة في الفصل 5 ولهذا سوف يتم تركها كتمرين لك.

تعريف المرصوصة والدالات الخاصة بتطبيق عمليات المرصوصة التي نوقشت من قبل تعريفات عامة. كذلك يجب علينا في حالة تمثيل مصفوفة للمرصوصة في التمثيل الطولي أن نضع تعريف المرصوصة والدالات الخاصة بتطبيق عمليات المرصوصة معاً في ملف (رئيسي). برنامج العميل يمكنه أن يتضمن هذا الملف الرئيسي عبر البيان include. كذلك عندما نعلن هدف مرصوصة يجب

أن نمرر نوع عناصر المرسوعة كعامل للنوع `linkedStackType`. على سبيل المثال يقوم
البيان:

```
linkedStackType<int> stack;
```

بإعلان أن `stack` هدف من النوع `linkedStackType` ونوع عنصر المرسوعة هو `int`. بالمثل
يقوم البيان:

```
linkedStackType<newString> stringStack;
```

بإعلان أن `stringStack` هدف من النوع `linkedStackType` ونوع عنصر المرسوعة هو
`newString`.

المرسوعة كمشتق من الفئة `linkedListType`:

إذا قمنا بمقارنة مرسوعة الدالة `push` مع الدالة `insertFirst` التي تمت مناقشتها بالنسبة إلى القوائم
العامة في الفصل 5 سوف نرى أن الخوارزميات لتطبيق تلك العمليات متشابهة. مقارنة دالات أخرى
مثل `initializeList` و `initializeStack`، و `isEmptyList` و `isEmptyStack` وغيرها تقترح أن
الفئة `linkedStackType` يمكن اشتقاقها من الفئة `linkedListType`. فضلاً عن هذا يمكن تطبيق
الدالات `pop` و `isFullStack` كما تم في القسم السابق.

بعد هذا نقوم بتعريف الفئة `linkedStackType` المستمدة من الفئة `linkedListType`. تعريفات
الدالات الخاصة بتطبيق عمليات المرسوعة معطاة أيضاً .

```

#ifndef H_derivedLinkedStack
#define H_derivedLinkedStack
#include <iostream>
#include "linkedList.h"

using namespace std;

template<class Type>
class linkedStackType: public linkedListType<Type>
{
public:
    void initializeStack();
    bool isEmptyStack();
    bool isFullStack();
    void push(const Type& newItem);
    Type top();
    void pop();
    void destroyStack();
};

template<class Type>
void linkedStackType<Type>::initializeStack()
{
    linkedListType<Type>::initializeList();
}

template<class Type>
bool linkedStackType<Type>::isEmptyStack()
{
    return linkedListType<Type>::isEmptyList();
}

template<class Type>
bool linkedStackType<Type>::isFullStack()
{
    return false;
}

template<class Type>
void linkedStackType<Type>::destroyStack()
{
    linkedListType<Type>::destroyList();
}

template<class Type>
void linkedStackType<Type>::push(const Type& newElement)
{
    linkedListType<Type>::insertFirst(newElement);
}

template<class Type>
Type linkedStackType<Type>::top()
{
    return linkedListType<Type>::front();
}

```

```

template<class Type>
void linkedStackType<Type>::pop()
{
    nodeType<Type> *temp;

    if(first != NULL)
    {
        temp = first;
        first = first->link;
        count--;
        delete temp;
        if(first == NULL)
            last = NULL;
    }
    else
        cerr<<"Cannot remove from an empty stack."<<endl;
}

#endif

```

تطبيق المرصوصات: حاسبة التعبير postfix:

الترميز المعتاد لكتابة تعبيرات رياضية (الترميز الذي تعلمناه في المدرسة الابتدائية) يسمى ترميز **مقحم** الذي يتم فيه كتابة المعامل بين المعاملين. على سبيل المثال في التعبير $a + b$ يكون المعامل a بين المعاملين a و b . في الترميز **المقحم** يكون للمعامل ما يسبقه. هذا يعني أننا نقوم بتقييم التعبيرات من اليسار الى اليمين والضرب والقسمة لهما أسبقية أعلى من الجمع والطرح. إذا أردنا تقييم تعبير ذي ترتيب مختلف يجب أن ندخل أقواس. على سبيل المثال في التعبير $a + b * c$ نقوم أولاً بتقييم $b * c$ باستخدام المعاملين a و b ثم نقيم $+$ باستخدام المعامل a ونتيجة $b * c$. في أوائل الخمسينيات أكتشف لوكاسيفيكز عالم الرياضيات البولندي أنه إذا تمت كتابة معاملات أمام المعاملات (ترميز **مقحم** أو **postfix** مثل $a b +$) أو بعد المعاملات (ترميز **لاحق** أو **postfix** أو **بولندي عكسي** مثل $a b +$) فإنه يمكن حذف الأقواس. على سبيل المثال التعبير:

$$a + b * c$$

في تعبير postfix يكون:

$$a b c * +$$

المثال التالي يوضح تعبيرات مقحمة والتعبيرات postfix المكافئة لها.

مثال 3-7:

الجدول 2-7 يوضح العديد من التعبيرات المقحمة والتعبيرات postfix المكافئة لها.

جدول 2-7: التعبيرات infix والتعبيرات postfix المكافئة لها

| التعبير postfix المكافئ له | التعبير المقدم |
|----------------------------|-----------------------------|
| $a b +$ | $a + b$ |
| $a b c * +$ | $a + b * c$ |
| $a b * c +$ | $a * b + c$ |
| $a b + c *$ | $(a + b) * c$ |
| $a b - c d + *$ | $(a - b) * (c + d)$ |
| $a b + c d e / - * f +$ | $(a + b) * (c - d / e) + f$ |

بعد اكتشاف لوكاسيفيكز بقليل تم ادراك أن الترميز postfix له استخدامات هامة في علوم الحاسب الآلي. في الحقيقة يقوم العديد من المجمعين بترجمة التعبيرات الحسابية الى صورة من ترميز postfix ثم يترجم هذا التعبير postfix الى كود آلي. يمكن تقييم التعبيرات postfix باستخدام الخوارزمية التالية:

تفحص التعبير من اليسار الى اليمين. عندما يتم العثور على معامل عد الى الحصول على العدد اللازم من المعاملات وأداء العملية والاستمرار.

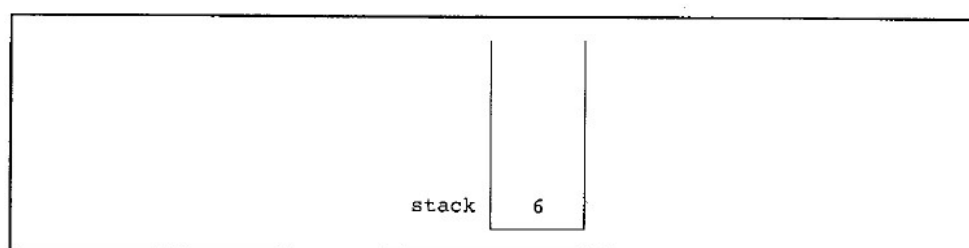
انظر الى التعبير التالي:

$$6 \ 3 \ + \ 2 \ * \ =$$

لنقم بتقييم هذا التعبير باستخدام مرصوصة والخوارزمية السابقة. (في المناقشة التالية نذكر التعبير postfix بعد كل خطوة. التظليل يشير الى الجزء الذي لم تتم معالجته من التعبير.)

$$6 \ 3 \ + \ 2 \ * \ =$$

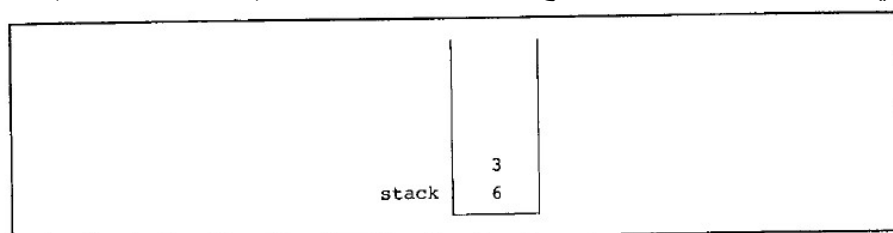
1. اقرأ الرمز الأول 6 الذي هو عبارة عن عدد. ادفع العدد الى المرصوصة (انظر شكل 19-7).



شكل 19-7: المرصوصة بعد دفع 6

$$6 \ 3 \ + \ 2 \ * \ =$$

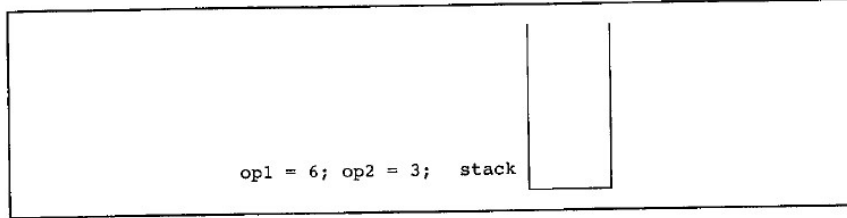
2. اقرأ الرمز التالي 3 وهو عبارة عن عدد. ادفع العدد الى المرصوصة (انظر شكل 20-7).



شكل 7-20: المرصوصة بعد دفع 3

$$6 \ 3 + 2 \ * =$$

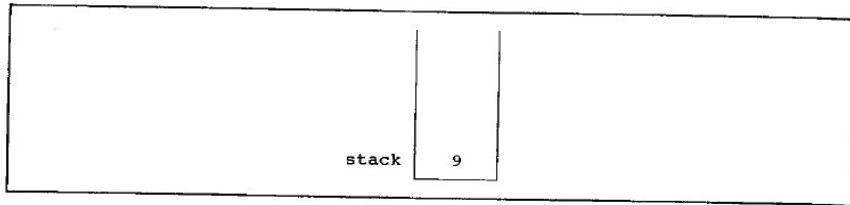
3. اقرأ الرمز التالي + الذي هو عبارة عن معاملة. بما أن المعامل يحتاج الى معاملين يتم تقييمهما استرجع العنصرين العلويين وا طرح المرصوصة مرتين. قم بأداء العملية وضع الناتج داخل المرصوصة (انظر شكلي 7-21 و 7-22).



شكل 7-21: المرصوصة بعد استعادة العنصرين العلويين وا طرح مرتين

$$9 = 3 + 6 = 2 \text{ المعامل} + 1 \text{ المعامل}$$

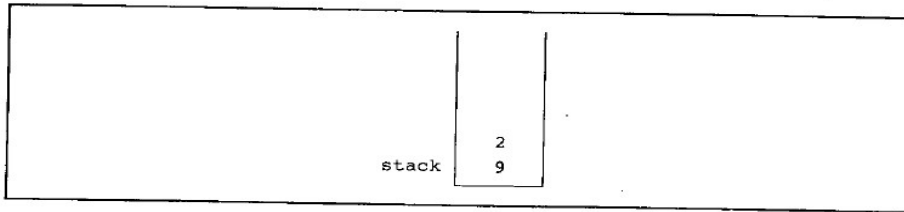
قم بأداء العملية: المعامل 1 + المعامل 2 = 9. ادفع الناتج الى المرصوصة (انظر شكل 7-22).



شكل 7-22: المرصوصة بعد دفع ناتج المعامل 1 + المعامل 2 الذي يساوي 9

$$6 \ 3 + 2 \ * =$$

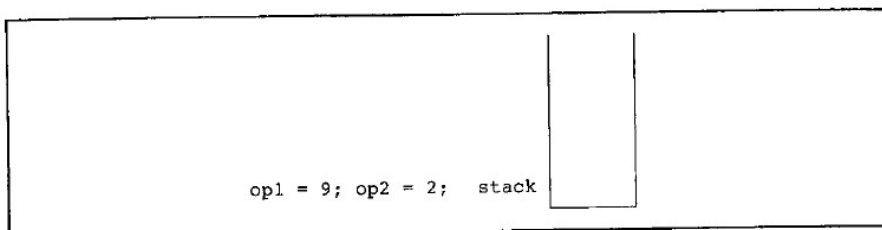
4. اقرأ الرمز التالي 2 وهو عبارة عن رقم وادفه العدد الى المرصوصة (انظر شكل 7-23).



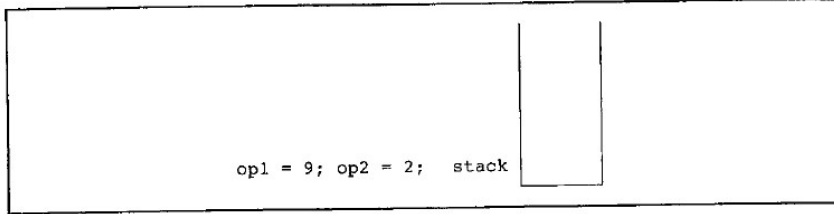
شكل 7-23: المرصوصة بعد دفع 2.

$$6 \ 3 + 2 \ * =$$

5. اقرأ الرمز التالي * الذي هو عبارة عن معاملة. بما أن المعامل يحتاج الى معاملين يتم تقييمهما استرجع العنصرين العلويين وا طرح المرصوصة مرتين. قم بأداء العملية وضع الناتج داخل المرصوصة (انظر شكلي 7-24 و 7-25).

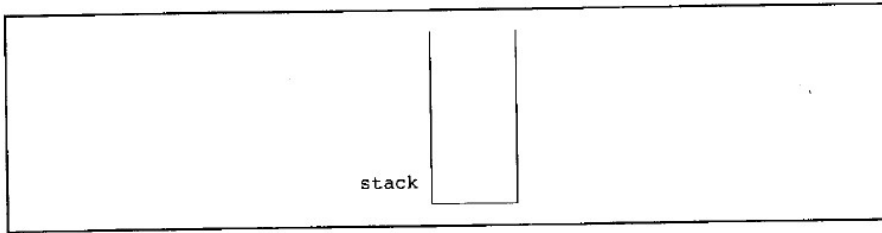


شكل 7-24: المرصوفة بعد استعادة العنصرين العلويين والطرح مرتين
 قم بأداء العملية: المعامل 1 * المعامل 2 = 9 * 2 = 18.
 ادفع الناتج الى المرصوفة (انظر شكل 7-25).



شكل 7-25: المرصوفة بعد دفع ناتج المعامل 1 * المعامل 2 وهو 18
 $6 \ 3 + 2 * =$

6. اقرأ الرمز التالي = وهو علامة يساوي الذي يشير الى نهاية التعبيرات. لهذا اطبع الناتج. ناتج التعبير موجود في المرصوفة ولهذا استرجع العنصر العلوي واطرح المرصوفة واطبع الناتج. بعد عملية الطرح تكون المرصوفة كما هي موضحة في شكل 7-26.



شكل 7-26: المرصوفة بعد طرح العنصر

قيمة التعبيرات $6 \ 3 + 2 * = 18$.

من هذه المناقشة يتضح أنه عندما نقرأ رمز بدلاً من عدد تنشأ الحالات التالية:

1. الرمز الذي نقرأه واحد من الرموز التالية: + أو - أو * أو / أو =.

أ. اذا كان الرمز هو + أو - أو * أو / فان الرمز يكون معامل ولهذا يجب أن نقيمه. بما أن المعامل يتطلب معاملين فان المرصوفة يجب أن يكون لها عنصرين على الأقل وبخلاف هذا يكون هناك خطأ في التعبير.

ب. اذا كان الرمز هو = فان التعبير ينتهي ويجب علينا طبع الاجابة. عند هذه الخطوة يجب أن تحتوي المرصوفة على عنصر واحد فقط وبخلاف هذا سوف يكون هناك خطأ في التعبير.

2. الرمز الذي نقرأه شيء غير + أو - أو * أو / . في هذه الحالة يحتوي التعبير على معامل غير صحيح.

من الواضح كذلك أنه عند مواجهة معامل (عدد) في تعبير ما يتم دفعه الى المرصوفة لأن المعاملات تأتي بعد المعاملات.

انظر الى التعبيرات التالية:

$$(i) \quad 7 \ 6 \ + \ 3 \ ; \ 6 \ - \ =$$

$$(ii) \quad 14 \ + \ 2 \ 3 \ * \ =$$

$$(iii) \quad 14 \ 2 \ 3 \ + \ =$$

التعبير (i) له معامل غير صحيح والتعبير (ii) ليس له معاملات كافية ل + ، والتعبير (iii) له العديد من المعاملات. في حالة التعبير (iii) عندما نواجه علامة يساوي (=) يكون للمرصوفة عنصرين وهذا خطأ لا يمكن اكتشافه حتى نكون مستعدين لطباعة قيمة التعبير.

لجعل المخرجات أسهل في قراءتها نفترض أن التعبيرات postfix تكون في الصيغة التالية:

$$\#6 \ \#3 \ + \ \#2 \ * \ =$$

الرمز # يسبق كل عدد في التعبير وإذا كان الرمز الذي يتم فحصه هو # فإن المدخل التالي يكون عدد (أي معامل). إذا لم يكن الرمز الذي يتم فحصه # فإنه إما يكون معامل (قد يكون غير صحيح) أو علامة يساوي (تشير الى نهاية التعبير). فضلاً عن هذا نفترض أن كل تعبير يحتوي فقط على المعاملات + و - و * و /.

هذا البرنامج يخرج التعبير postfix الكلي مع الاجابة. اذا كان هناك خطأ في التعبير يتم اهمال التعبير وفي هذه الحالة يقوم البرنامج باخراج التعبير مع رسالة خطأ مناسبة. بما أن التعبير قد يحتوي على خطأ ما فيجب علينا ازالة المرصوفة قبل معالجة التعبير التالي. كما يجب تهيئة المرصوفة وهذا يعني أن المرصوفة يجب أن تكون فارغة.

الخوارزمية الرئيسية:

عملاً بالمناقشة السابقة تكون الخوارزمية الرئيسية في الكود الوهمي:

```

read the first ch;
while more data to process
{
    clear the stack;
    output ch;

    while(ch is not = '=') //process each expression
                            // = marks the end of an expression
    {
        switch(ch)
        {
            case '#': read a number;
                      output the number;
                      push the number onto the stack;
                      break;
            default: Assume that ch is an operation
                     evaluate the operation;
        } //end switch

        if no error is found, then
        {
            read next ch;
            output ch;
        }
    } //end while

    if the expression does not contain any error(s), then
        output the result;
    else
        discard the result;

    start processing the next expression;
}

```

Evaluate:

هذه الدالة (إذا أمكن) تقوم بتقييم التعبيرات وهناك حاجة إلى معاملين لتقييم معامل ويتم حفظ المعاملات في المرصوصة. لهذا يجب أن تحتوي المرصوصة على عددين على الأقل. إذا احتوت المرصوصة على أقل من عددين اذن يكون هناك خطأ في التعبير. في هذه الحالة يتم اهمال التعبير بأكمله ويتم طباعة رسالة مناسبة. تقوم هذه الدالة أيضاً بالتحقق من وجود معاملات غير صحيحة وهذه الدالة تكون:

```

if stack is empty
{
    error in the expression
    set expressionOk to false
}
else
{

```

```

retrieve the top element of stack into op2
pop stack
if stack is empty
{
    error in the expression
    set expressionOk to false
}
else
{
    retrieve the top element of stack into op1
    pop stack

    //If the operation is legal, perform the
    //operation and push the result onto stack.
    switch(ch)
    {
        case '+': //Perform the operation and push the result
            //onto stack.
            stack.push(op1 + op2);
            break;
        case '-': //Perform the operation and push the result
            //onto stack.
            stack.push(op1 - op2);
            break;
        case '*': //Perform the operation and push the result
            //onto stack.
            stack.push(op1 * op2);
            break;
        case '/': //If(op2 != 0), perform the operation and
            //push the result onto stack.
            stack.push(op1/op2);

            //Otherwise, report the error.
            //Set expressionOk to false.
            break;
        otherwise operation is illegal
        {
            output an appropriate message;
            set expressionOk to false
        }
    } //end switch
}

```

Discard: هذه الدالة يتم استدعائها عندما يتم اكتشاف خطأ في تعبير ما. انها تقرأ وتكتب بيانات المدخلات فقط حتى يكون المدخل هو = التي تعني نهاية التعبير. الحلقة التالية تحقق ذلك:

```

while(ch != '=')
{
    read ch;
    output ch;
}

```

قائمة البرنامج الكاملة:

```
//Postfix Calculator
#include <iostream>
#include <iomanip>
#include <fstream>
#include "mystack.h"

using namespace std;

void evaluate(ofstream& out, stackType<double>& stack,
             char& ch, bool& expressionOk);
void discard(ifstream& in, ofstream& out, char& ch);

int main()
{
    double num, result;
    bool expressionOk;
    char ch;
    stackType<double> stack(100);
    ifstream infile;
    ofstream outfile;

    infile.open("a:\\Ch7_RpnData.txt");

    if(!infile)
    {
        cerr<<"Cannot open the input file. "
              <<"Program terminates!"<<endl;
        return 1;
    }

    outfile.open("a:\\ch7_RpnOutput.txt");

    outfile<<fixed<<showpoint;
    outfile<<setprecision(2);

    infile>>ch;
    while(infile)
    {
        stack.initializeStack();
        expressionOk = true;
        outfile<<ch;

        while(ch != '=')
        {
            switch(ch)
            {
            case '#': infile>>num;
                      outfile<<num<<" ";
                      if(!stack.isFullStack())
                          stack.push(num);
                      else
                      {

```

```

        cerr<<"Stack overflow. "
        <<"Program terminates!"<<endl;
        return 1;
    }

    break;
default: evaluate(outfile, stack, ch, expressionOk);
} //end switch

if(expressionOk) //if no error
{
    infile>>ch;
    outfile<<ch;
    if(ch != '#')
        outfile<<" ";
}
else
    discard(infile, outfile, ch);
} //end while (ch != '=')

if(expressionOk) //if no error, print the result
{
    if(!stack.isEmptyStack())
    {
        result = stack.top();
        stack.pop();
        if(stack.isEmptyStack())
            outfile<<result<<endl;
        else
            outfile<<" (Error: Too many operands)"<<endl;
    } //end if
    else
        outfile<<" (Error in the expression)"<<endl;
}
else
    outfile<<" (Error in the expression)"<<endl;

outfile<<"_____ "<<endl<<endl;

    infile>>ch; //begin processing the next expression
} //end while

infile.close();
outfile.close();

return 0;
} //end main

```

```

void evaluate(ofstream& out, stackType<double>& stack,
             char& ch, bool& expressionOk)
{
    double op1, op2;
    if(stack.isEmptyStack())
    {
        out<<" (Not enough operands)";
        expressionOk = false;
    }
    else
    {
        op2 = stack.top();
        stack.pop();

        if(stack.isEmptyStack())
        {
            out<<" (Not enough operands)";
            expressionOk = false;
        }
        else
        {
            op1 = stack.top();
            stack.pop();

            switch(ch)
            {
                case '+': stack.push(op1 + op2);
                           break;
                case '-': stack.push(op1 - op2);
                           break;
                case '*': stack.push(op1 * op2);
                           break;
                case '/': if(op2 != 0)
                           stack.push(op1 / op2);
                           else
                           {
                               out<<" (Division by 0)";
                               expressionOk = false;
                           }
                           break;
                default: out<<" (Illegal operator)";
                           expressionOk = false;
            }
            //end switch
        }
        //end else
    }
    //end evaluate
}

void discard(ifstream& in, ofstream& out, char& ch)
{
    while(ch != '=')
    {
        in.get(ch);
        out<<ch;
    }
    //end discard
}

```

تنفيذ العينة:
ملف المدخلات:

```
#35 #27 + #3 * =
#26 #28 + #32 #2 ; - #5 /
#23 #30 #15 * / =
#2 #3 #4 + =
#20 #29 #9 * ; =
#25 #23 - + =
#34 #24 #12 #7 / * + #23 - =
```

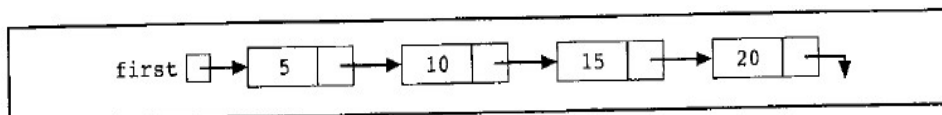
المخرجات:

```
#35.00 #27.00 + #3.00 * = 186.00
-----
#26.00 #28.00 + #32.00 #2.00 ; (Illegal operator) - #5 / = (Error
in the expression)
-----
#23.00 #30.00 #15.00 * / = 0.05
-----
#2.00 #3.00 #4.00 + = (Error: Too many operands)
-----
#20.00 #29.00 #9.00 * ; (Illegal operator) = (Error in the
expression)
-----
#25.00 #23.00 - + (Not enough operands) = (Error in the expression)
-----
#34.00 #24.00 #12.00 #7.00 / * + #23.00 - = 52.14
-----
```

ازالة التكرار: خوارزمية غير تكرارية لطباعة قائمة متصلة من الخلف:

في الفصل السادس استخدمنا التكرار لطباعة قائمة متصلة من الخلف وفي هذا القسم نتعلم كيفية استخدام المرصوصة لتصميم خوارزمية غير تكرارية لطباعة قائمة متصلة من الخلف.

انظر الى القائمة المتصلة الموضحة في شكل 27-7.



شكل 27-7: قائمة متصلة

لطباعة القائمة من الخلف نحتاج أولاً الى الوصول الى العقدة الأخيرة من القائمة وهذا ما يمكننا فعله عن طريق اجتياز القائمة المتصلة بدايةً من العقدة الأولى. بالرغم من هذا فمجرد أن نكون عند العقدة

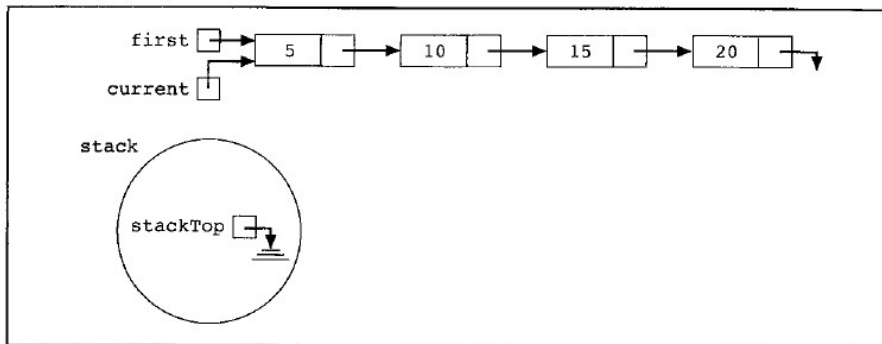
الأخيرة كيف نعود الى العقدة السابقة خاصةً باعطاء هذه الوصلات التي تذهب في اتجاه واحد فقط؟ يمكنك مرة أخرى اجتياز القائمة المتصلة بحالة ايقاف الحلقة المناسبة ولكن هذه الطريقة قد تضيع قدر كبير من زمن الحاسب الآلي خاصةً اذا كانت القائمة كبيرة للغاية. فضلاً عن هذا اذا فعلنا هذا مع كل عقدة في القائمة فقد يتم تنفيذ البرنامج ببطء شديد. لنرى كيفية استخدام المرصوصة بشكل فعال لطباعة القائمة من الخلف.

بعد طباعة info عقدة معينة نحتاج الى الانتقال الى العقدة الواقعة خلف هذه العقدة مباشرةً. على سبيل المثال بعد طباعة 20 نحتاج الى الانتقال الى العقدة المحتوية على 15. لهذا أثناء اجتياز القائمة للانتقال الى العقدة الأخيرة يجب أن نحفظ مؤشر لكل عقدة. على سبيل المثال بالنسبة الى القائمة الموجودة في شكل 7-27 يجب أن نحفظ مؤشر لكل عقدة ذات 5, 10, 15. وبعد طباعة 20 نعود الى العقدة ذات 15 info وبعد طباعة 15 نعود الى العقدة ذات 10 info وهكذا. ينتج من هذا أنه يجب أن نحفظ مؤشرات لكل عقدة في المرصوصة لتطبيق مبدأ الداخل أخيراً يخرج أولاً.

بما أن عدد العقد في القائمة المتصلة لا يكون معلوم عادةً اذن نستخدم التطبيق المتصل للمرصوصة. افترض أن stack هدف من النوع linkedStackType وcurrent مؤشر من نفس النوع مثل المؤشر first. انظر الى البيانات التالية:

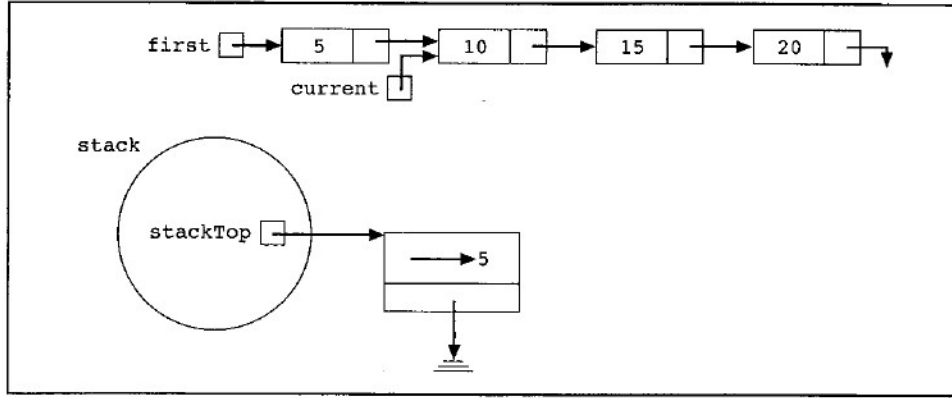
```
current = first;           //Line 1
while(current != NULL)    //Line 2
{
    stack.push(current);   //Line 3
    current = current->link; //Line 4
}
```

بعد تنفيذ البيان في الصف 1 تشير current الى العقدة الأولى (انظر شكل 7-28).



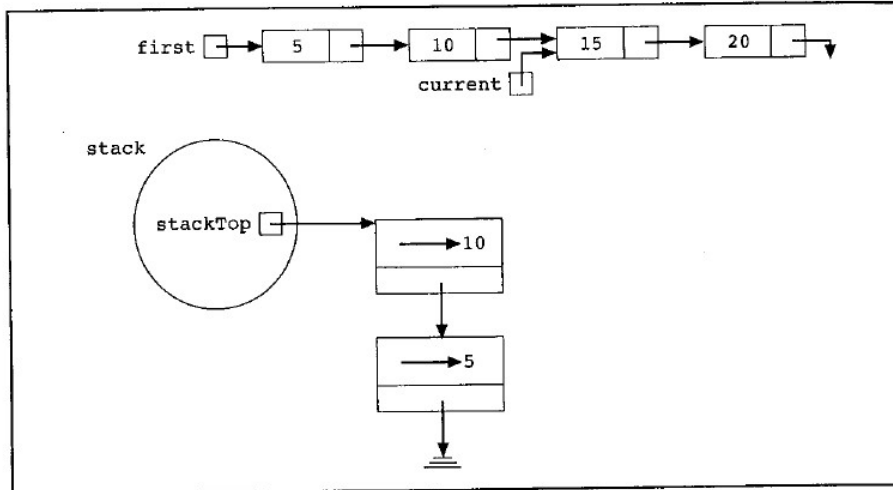
شكل 7-28: القائمة بعد تنفيذ البيان `current = first;`

بما أن `current` ليست `NULL` فانه يتم تنفيذ البيانات في الصفين 3 و4 (انظر شكل 7-29).



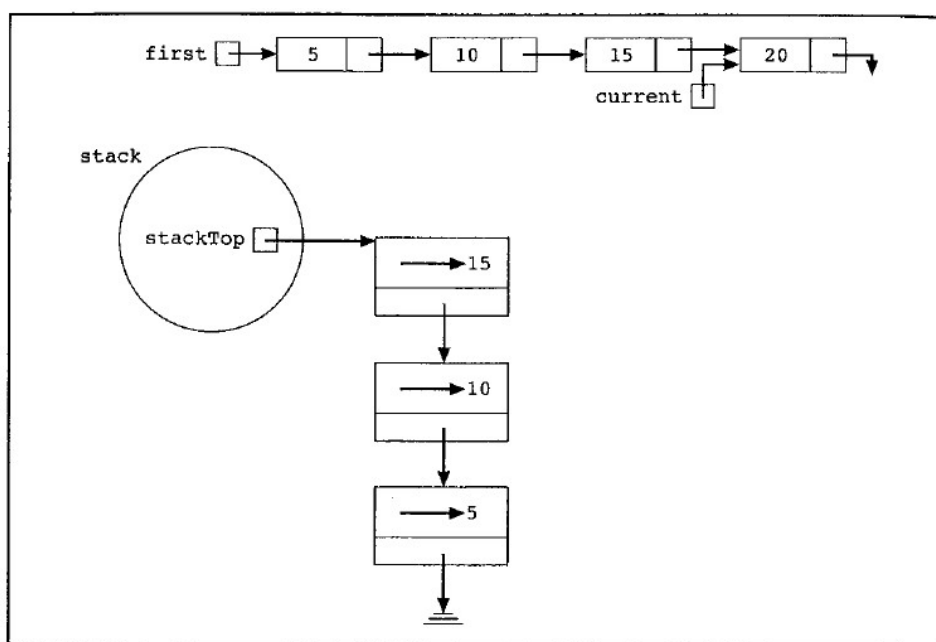
شكل 7-29: القائمة والمرصوصة بعد تنفيذ البيان `stack.push(current);` والبيان `current = current->link;`

بعد تنفيذ البيان في الصف 4 يتم إعادة تقييم وضع الحلقة في الصف 2. بما أن `current` ليست NULL فان الحلقة تقيم `true` ولهذا يتم تنفيذ البيانات في الصفين 3 و4 (انظر شكل 7-30).



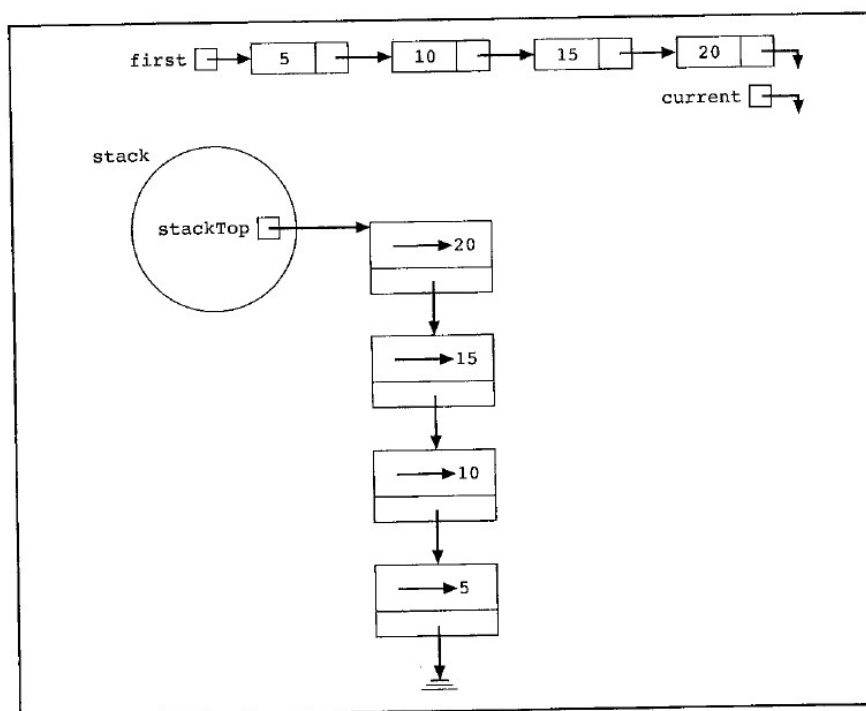
شكل 7-30: القائمة والمرصوصة بعد تنفيذ البيان `stack.push(current);` والبيان `current = current->link;`

بعد تنفيذ البيان في الصف 4 يتم تقييم وضع الحلقة في الصف 2 مرة أخرى. بما أن `current` ليست NULL فان الحلقة تقيم `true` ولهذا يتم تنفيذ البيانات في الصفين 3 و4 (انظر شكل 7-31).



شكل 7-31: القائمة والمرصوصة بعد تنفيذ البيان `stack.push(current);` والبيان `current = current -> link;`

بعد تنفيذ البيان في الصف 4 يتم تقييم وضع الحلقة في الصف 2 مرة أخرى. بما أن `current` ليست NULL فان الحلقة تقيم `true` ولهذا يتم تنفيذ البيانات في الصفين 3 و4 (انظر شكل 7-32).

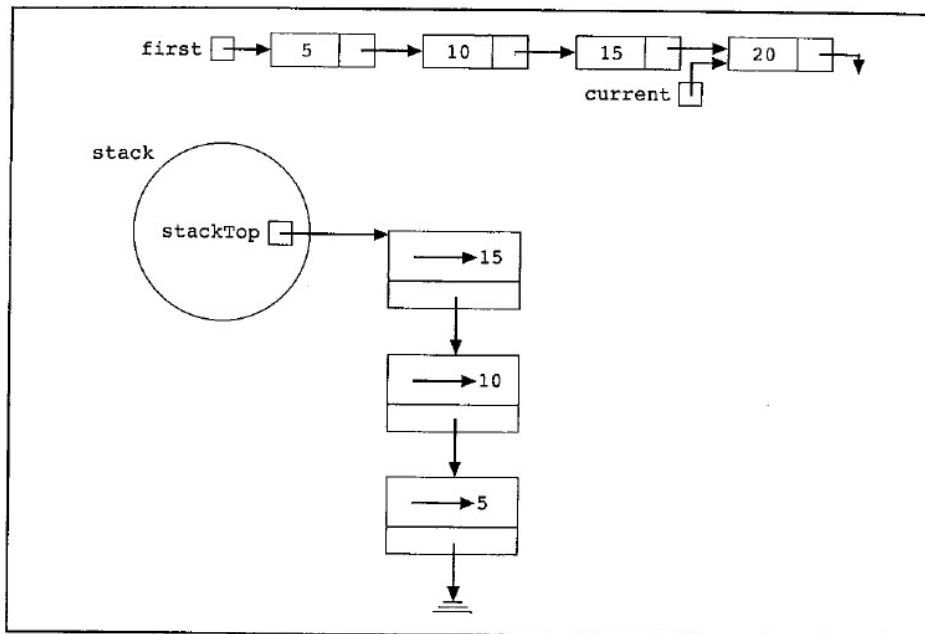


شكل 7-32: القائمة والمرصوفة بعد تنفيذ البيان `current = stack.push(current);` والبيان `current -> link;`

بعد تنفيذ البيان في الصف 4 يتم تقييم وضع الحلقة في الصف 2 مرة أخرى. بما أن `current` تساوي `NULL` فإن الحلقة تقيم `false` وتتوقف الحلقة `while` في الصف 2. من شكل 7-32 ينتج أنه يتم حفظ مؤشر إلى كل عقدة في القائمة المتصلة في المرصوفة ويحتوي العنصر العلوي من المرصوفة على مؤشر إلى العقدة الأخيرة في القائمة وهكذا. لنقم الآن بتنفيذ البيانات التالية:

```
while(!stack.isEmptyStack())           //Line 5
{
    current = stack.top();               //Line 6
    stack.pop();                         //Line 7
    cout<<current->info<<" ";          //Line 8
}
```

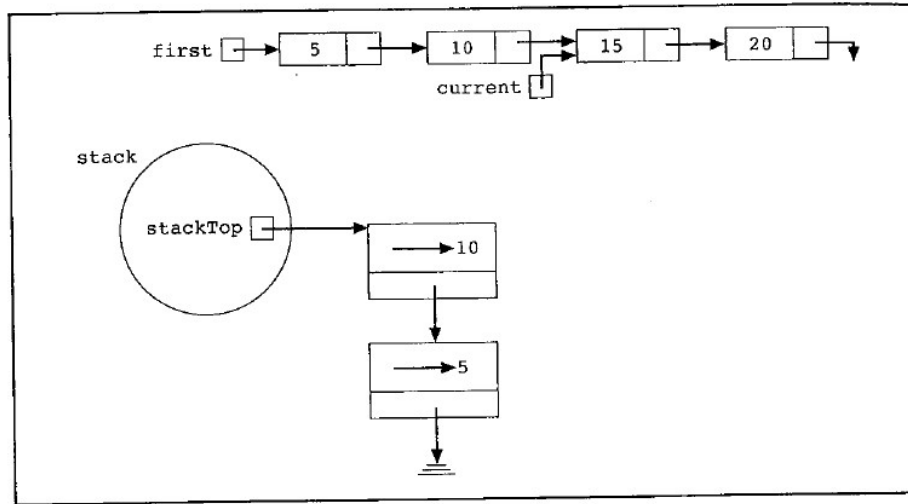
يتم تقييم وضع الحلقة في الصف 5 عند `true` لأن المرصوفة غير فارغة. لهذا يتم تنفيذ البيانات في الصفوف 6 و7 و8. بعد تنفيذ البيان في الصف 6 تشير `current` إلى العقدة الأخيرة وبعد تنفيذ البيان في الصف 7 تتم إزالة العنصر العلوي من المرصوفة (انظر شكل 7-33).



شكل 7-33: القائمة والمرصوفة بعد تنفيذ البيان `current = stack.top ()` والبيان `stack.pop ()`;

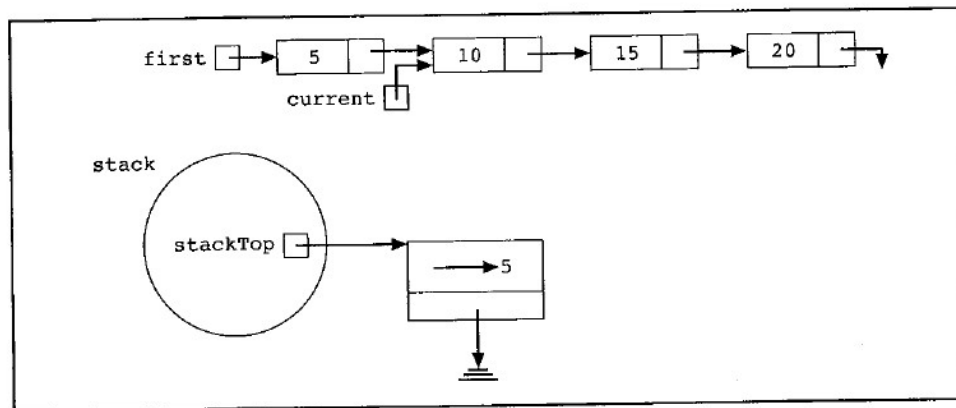
البيان في الصف 8 يخرج info -> current وهو 20 وبعد هذا يتم تقييم وضع الحلقة في الصف 5. بما أن وضع الحلقة يقيم عند true اذن يتم تنفيذ البيانات في الصفوف 6 و 7 و 8. بعد تنفيذ البيانات في الصفوف 6 و 7 ينتج الشكل 34-7.

ازالة التكرار: خوارزمية غير تكرارية لطباعة قائمة متصلة من الخلف



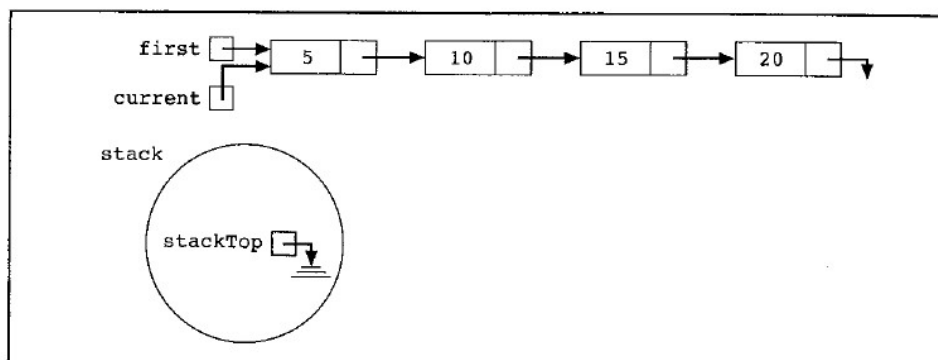
شكل 34-7: القائمة والمرصوفة بعد تنفيذ البيان `current = stack.top ()` والبيان `stack.pop ()`;

البيان في الصف 8 يخرج info -> current وهو 15 وبعد هذا يتم تقييم وضع الحلقة في الصف 5. بما أن وضع الحلقة يقيم عند true اذن يتم تنفيذ البيانات في الصفوف 6 و 7 و 8. بعد تنفيذ البيانات في الصفوف 6 و 7 ينتج الشكل 35-7.



شكل 35-7: القائمة والمرصوفة بعد تنفيذ البيان `current = stack.top ()` والبيان `stack.pop ()`;

البيان في الصف 8 يخرج info -> current وهو 10 وبعد هذا يتم تقييم وضع الحلقة في الصف 5. بما أن وضع الحلقة يقيم عند true اذن يتم تنفيذ البيانات في الصفوف 6 و7 و8. بعد تنفيذ البيانات في الصفوف 6 و7 ينتج الشكل 7-36.



شكل 7-36: القائمة والمرصوفة بعد تنفيذ البيان ($current = stack.top$) والبيان ($stack.pop$);

البيان في الصف 8 يخرج info -> current وهو 5 وبعد هذا يتم تقييم وضع الحلقة في الصف 5. بما أن وضع الحلقة يقيم عند false فان الحلقة while تتوقف وتنتج الحلقة while في الصف رقم خمسة المخرجات التالية:

20 15 10 5

فئة مكتبة القالب المعياري stack (معدل حاوية المصفوفة):

الأقسام السابقة ناقشت بنية البيانات stack بالتفصيل. بما أن المرصوفة بنية بيانات هامة فان مكتبة القالب المعياري تقدم فئة لتطبيق مرصوفة في برنامج. اسم الفئة التي تعرف المرصوفة هي stack واسم الملف الرئيسي المحتوي على تعريف الفئة stack هو stack. تطبيق الفئة stack الموفر بواسطة مكتبة القالب المعياري مماثل للتطبيق الموضح قبل هذا في هذا الفصل. الجدول 7-3 يقوم بتعريف العمليات المتنوعة التي تدعمها فئة حاوية المرصوفة.

جدول 7-3: عمليات على هدف stack

| العملية | التأثير |
|-------------|---|
| size | تنتج العدد الحقيقي من العناصر الموجودة في المرصوفة |
| empty | تنتج true اذا كانت المرصوفة فارغة وبخلاف هذا تنتج false |
| push (item) | تدخل نسخة من العنصر في المرصوفة |
| top | تنتج العنصر العلوي من المرصوفة ولكنها لا تزيل العنصر العلوي من المرصوفة. يتم تطبيق هذه العملية كدالة منتجة للقيمة |
| pop | تزيل العنصر العلوي من المرصوفة |

بالإضافة الى العمليات size و empty و push و top و pop تقدم فئة حاوية المرصوعة معاملات مترابطة للمقارنة بين مرصوعتين. على سبيل المثال يمكن استخدام المعامل المترابط == لتحديد ما اذا كانت المرصوعتان متطابقتين. البرنامج في مثال 4-7 يوضح كيفية استخدام فئة حاوية المرصوعة.

مثال 4-7:

```
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    stack<int> intStack; //Line 1
    intStack.push(16); //Line 2
    intStack.push(8); //Line 3
    intStack.push(20); //Line 4
    intStack.push(3); //Line 5

    cout<<"Line 6: The top element of intStack: "
         <<intStack.top()<<endl; //Line 6
    intStack.pop(); //Line 7

    cout<<"Line 8: After the pop operation, "
         <<"the top element of intStack: "
         <<intStack.top()<<endl; //Line 8
    cout<<"Line 9: intStack elements: "; //Line 9
    while(!intStack.empty()) //Line 10
    {
        cout<<intStack.top()<<" "; //Line 11
        intStack.pop(); //Line 12
    }

    cout<<endl; //Line 13
    return 0;
}
```

المخرجات:

الصف 6: العنصر العلوي من intStack: 3
 الصف 8: بعد عملية الطرح يكون العنصر العلوي من intStack: 20
 الصف 9: عناصر intStack: 16 8 20
 المخرجات السابقة تفسر نفسها والتفاصيل متروكة كتمرين لك.

مراجعة سريعة:

1. المرصوعة عبارة عن بنية بيانات حيث يتم اضافة العناصر وحذفها من نهاية واحدة فقط.

2. المرصوصة عبارة عن بنية بنايات الداخل أخيراً يخرج أولاً.
3. العمليات الرئيسية على المرصوصة هي: دفع عنصر الى المرصوصة، وحذف عنصر من المرصوصة، وانتاج العنصر العلوي من المرصوصة، وتهيئة المرصوصة، وتدمير المرصوصة، والتحقق مما اذا كانت المرصوصة فارغة، والتحقق مما اذا كانت المرصوصة ممتلئة.
4. يمكن تطبيق المرصوصة اما كمصفوفة أو كقائمة متصلة.
5. لا يجب تناول العناصر الموجودة في وسط المرصوصة بشكل مباشر.
6. المرصوصات عبارة عن صور مقيدة من المصفوفات والقوائم المتصلة.
7. الترميز postfix لا يتطلب استخدام أقواس لاجبار أولوية المعامل.
8. في الترميز postfix تتم كتابة العوامل بعد المعاملات.
9. يتم تقييم التعبيرات postfix وفقاً للقواعد التالية:
 - أ- تفحص التعبير من اليسار الى اليمين.
 - ب- اذا تم العثور على عامل حاول الحصول على العدد اللازم من المعاملات وقم بتقييم العامل واكمل.
10. يمكن استخدام فئة مكتبة القالب المعياري stack لتطبيق مرصوصة في برنامج.

تمارين:

1. انظر الى البيانات التالية:

```
stackType<int> stack;
int x, y;
```

وضح ما هي مخرجات الجزء التالي من الكود.

```
x = 4;
stack.push(7);
stack.push(x);
stack.push(x + 5);
y = stack.top();
```

```
stack.pop();
stack.push(x + y);
stack.push(y - 2);
stack.push(3);
x = stack.top();
stack.pop();
cout<<"x = "<<x<<endl;
cout<<"y = "<<y<<endl;

while(!stack.isEmptyStack())
{
    cout<<stack.top()<<endl;
    stack.pop();
}
```

2. انظر الى البيانات التالية:

```
stackType<int> stack;
int x;
```

افترض أن المدخلات هي:

14 45 34 23 10 5 -999

وضح ما هي مخرجات الجزء التالي من الكود.

```
stack.push(5);
cin>>x;
while(x != -999)
{
    if(x % 2 == 0)
    {
        if(!stack.isFullStack())
            stack.push(x);
    }
    else
        cout<<"x = "<<x<<endl;
    cin>>x;
}
cout<<"Stack Elements: ";

while(!stack.isEmptyStack())
{
    cout<<" "<<stack.top();
    stack.pop();
}
cout<<endl;
```

3. قم بتقييم التعبيرات التالية postfix:

أ. $6 \ 4 \ + \ 3 \ * \ 16 \ 4 \ / \ - \ =$

ب. $12 \ 25 \ 5 \ 1 \ / \ / \ * \ 8 \ 7 \ + \ - \ =$

ج. $70 \ 14 \ 4 \ 5 \ 15 \ 3 \ / \ * \ - \ - \ / \ 6 \ + \ =$

د. $3 \ 5 \ 6 \ * \ + \ 13 \ - \ 18 \ 2 \ / \ + \ =$

4. قم بتحويل التعبيرات المقحمة التالية الى ترميز postfix (من أجل الحل الحسابي الذي يحول تعبير مقحم الى تعبير postfix مكافئ له انظر تمرين البرمجة 10 من هذا الفصل).

أ. $(A + B) * (C + D) - E$

ب. $A - (B + C) * D + E / F$

ج. $\{(A + B) / (C - D) + E\} * F - G$

د. $A + B * (C + D) - E / F * G + H$

5. اكتب التعبيرات المقحمة المكافئة للتعبيرات postfix التالية.

أ. $a b * c +$

ب. $a b + c d - *$

ج. $a b - c d *$

6. ما هي مخرجات البرنامج التالي؟

```
#include <iostream>
#include <stack>
#include <string>

using namespace std;

template<class type>
void mystery(stack<type> s, stack<type>& t);

int main()
{
    string list[] = {"Winter", "Spring", "Summer", "Fall",
                     "Cold", "Warm", "Hot"};

    stack<string> s1, s2;
    for(int i = 0; i < 7; i++)
        s1.push(list[i]);

    mystery(s1, s2);
    while(!s2.empty())
    {
        cout<<s2.top()<<" ";
        s2.pop();
    }
    cout<<endl;
    return 0;
}

template<class type>
void mystery(stack<type> s, stack<type>& t)
{
    while(!s.empty())
    {
        t.push(s.top());
        s.pop();
    }
}
```

7. ما هي مخرجات البرنامج التالي؟

```

#include <iostream>
#include <stack>
using namespace std;
void mystery(stack<int> s, stack<int>& t);
int main()
{
    int list[] = {5, 10, 15, 20, 25};
    stack<int> s1, s2;
    for(int i = 0; i < 5; i++)
        s1.push(list[i]);
    mystery(s1, s2);
    while(!s2.empty())
    {
        cout<<s2.top()<<" ";
        s2.pop();
    }
    cout<<endl;
    return 0;
}
void mystery(stack<int> s, stack<int>& t)
{
    while(!s.empty())
    {
        t.push(2 * s.top());
        s.pop();
    }
}

```

8. اكتب تعريف قالب الدالة printListReverse التي تستخدم مرصوصة لطباعة قائمة متصلة بترتيب عكسي. افترض أن هذه الدالة عنصر من الفئة linkedListType التي تم تصميمها في الفصل 5.

9. اكتب تعريف قالب الدالة second التي تتخذ من هدف المرصوصة معامل وتنتج العنصر الثاني من المرصوصة. المرصوصة الأصلية تظل غير متغيرة.

10. اكتب تعريف قالب الدالة clear التي تتخذ من هدف المرصوصة من النوع stack (فئة مكتبة القالب المعياري) وتزيل جميع العناصر من المرصوصة.

تمارين برمجة:

1. اكتب تعريفات الدالة copyStack ومقوم النسخ والمدمر ودالة ائصال معامل الاسناد للفئة linkedStackType واكتب كذلك برنامج لاختبار دالتك.

2. تكون المصفوفتان من نفس النوع اذا كان حجمهما واحد وكانت عناصرهما في المواضع المتوافقة متماثلة. قم بانتقال معامل مترابط == للفئة stackType التي تنتج true اذا كانت المصفوفتان اللتين من نفس النوع متماثلتين وبخلاف هذا تنتج false. اكتب أيضاً تعريف الدالة template لانتقال المعامل == فضلاً عن هذا اكتب برنامج لاختبار ذلك.
3. أعد تمرين البرمجة رقم 2 للفئة linkedStackType. (تعريف الفئة المعطى في هذا الفصل لا يطبق الدالة copyStack والدالات الأخرى المذكورة في تمرين البرمجة رقم 1. برنامجك يستخدم على الأقل المدمر ولهذا اكتب تعريف المدمر قبل تنفيذ البرنامج.)
4. أ. أضف العملية التالية الى الفئة stackType:

```
void reverseStack(stackType<Type> &otherStack);
```

انظر الى البيانات التالية:

```
stackType<int> stack1;
stackType<int> stack2;
```

يقوم البيان:

```
stack1.reverseStack(stack2);
```

بنسخ عناصر stack1 في stack2 بترتيب عكسي. هذا يعني أن العنصر العلوي من stack1 يكون العنصر السفلي من stack2 وهكذا. يتم تدمير المحتويات القديمة من stack2 ولا تتغير stack1.

ب. اكتب تعريف الدالة template لتطبيق العملية reverseStack. و اكتب كذلك برنامج لاختبار الدالة reverseStack.

5. أعد تمارين البرمجة رقم 4، و4ب للفئة linkedStackType و اكتب برنامج لاختبار ذلك. ((تعريف الفئة المعطى في هذا الفصل لا يطبق الدالة copyStack والدالات الأخرى المذكورة في تمرين البرمجة رقم 1. برنامجك يستخدم على الأقل المدمر ولهذا اكتب تعريف المدمر قبل تنفيذ البرنامج.))

6. اكتب برنامج يخرج رسالة مناسبة من أجل رموز التجميع مثل الأقواس والأسوار اذا ناسبها تعبير حسابي. على سبيل المثال يحتوي التعبير $\{8 * (6 - 3) + 25\}$ على رموز تجميع مناسبة.

7. اكتب برنامج يستخدم مرصوفة لطباعة العوامل الرئيسية لعدد صحيح موجب بترتيب تنازلي.

8. تمرين البرمجة 15 في الفصل 6 المسمى تحويل العدد من ثنائي الى عشري يستخدم التكرار لتحويل عدد ثنائي الى عدد عشري مساوي له. اكتب برنامج يستخدم مرصوصة لتحويل عدد ثنائي الى عدد عشري مكافئ له.

9. المثال 4-6 في الفصل 6 المسمى تحويل عدد من عشري الى ثنائي يحتوي على برنامج يقوم باستخدام التكرار لتحويل العدد العشري الى عدد ثنائي مكافئ له. اكتب برنامج يستخدم المرصوصة لتحويل عدد عشري الى عدد ثنائي مكافئ له.

10. (من مقحم الى postfix) اكتب برنامج يقوم بتحويل تعبير مقحم الى تعبير postfix مكافئ له. قواعد تحويل تعبير مقحم الى تعبير postfix تكون كما يلي:

- أ- تفحص التعبير من اليسار الى اليمين ومرور واحد يكفي.
- ب- اذا كان الرمز التالي الذي يتم تفحصه عبارة عن معامل أضفه الى التعبير postfix.
- ت- اذا كان الرمز التالي الذي يتم تفحصه عبارة عن قوس أيسر ادفعه الى المرصوصة.
- ث- اذا كان الرمز التالي الذي يتم تفحصه عبارة عن قوس أيمن ازل والحق جميع الرموز من المرصوصة حتى القوس الأيسر الأحدث. ازل واهمل القوس الأيسر.
- ج- اذا كان الرمز التالي الذي يتم تفحصه عامل:

1- ازل والحق كل عامل بالمصفوفة موجود فوق القوس الأيسر الذي تم

تفحصه حديثاً والذي له أولوية أكبر من أو متساوية مع العامل الجديد الى

التعبير postfix.

2- ادفع العامل الجديد الى المرصوصة.

ح- بعد معالجة المقطع المقحم تماماً ازل والحق كل شئ من المرصوصة الى المقطع

postfix.

في هذا البرنامج عليك النظر الى العوامل الحسابية (الثنائية) التالية: + و- و* و/. قد تفترض أن التعبير الذي تعالجه خالي من الأخطاء.

قم بتصميم فئة تقوم بتخزين المقاطع المقحمة و postfix ويجب أن تتضمن الفئة العمليات التالية:

getInfix: تقوم بتخزين التعبير المقحم.

showInfix: تخرج التعبير المقحم.

showPostfix: تخرج التعبير postfix.

هناك عمليات أخرى قد تحتاجها وهي:

convertToPostfix: تقوم بتحويل التعبير المقحم الى تعبير postfix ويتم تخزين التعبير postfix

الناتج في postfixString.

precedence: تحدد الأولوية بين عاملين. اذا كان العامل الأول أكبر من أو مساوي للعامل الثاني

تنتج القيمة true وبخلاف هذا تنتج false.

ادخل المقومات والمدمر من أجل التهيئة الآلية وإزالة التخصيص الحركي للذاكرة.

اختبر برنامجك على التعبيرات التالية:

$$1. A + B - C;$$

$$2. (A + B) * C;$$

$$3. (A + B) / (C - D);$$

$$4. A + ((B + C) * (E - F) - G) / (H - I);$$

$$5. A + B * (C + D) - E / F * G + H;$$

لكل تعبير يجب أن تكون اجابتك بالصيغة التالية:

$$\text{التعبير المقحم: } A + B - C;$$

$$\text{التعبير postfix: } A B + C -$$

11. أعد البرنامج في قسم تطبيق المرصصات: حاسبة التعبير postfix من هذا الفصل التي

تستخدم فئة مكتبة القالب المعياري stack لتقييم التعبيرات postfix.

12. أعد تمرين البرمجة 10 حتى يستخدم فئة مكتبة القالب المعياري stack لتحويل التعبيرات

المقحمة الى تعبيرات postfix.

الفصل 8 الطوابير

في هذا الفصل سوف:

- تعرف الطوابير.
- تدرس عمليات متنوعة على الطوابير.
- تتعلم كيفية تطبيق الطابور كقائمة متصلة.
- تكتشف تطبيقات الطوابير.
- تدرس فئة مكتبة القالب المعياري queue.
- تكتشف طوابير الأسبقية.

هذا الفصل يناقش بنية بيانات أخرى هامة هي : **الطوابير**. ترميز الطابور في علوم الحاسب الآلي هو نفسه فكرة الطوابير التي اعتدت عليها في الحياة اليومية. هناك طوابير من العملاء في البنك أو في متجر البقالة وطوابير من السيارات تنتظر المرور عبر نقطة تحصيل. بالمثل بما أن الحاسب الآلي يمكنه إرسال طلب طباعة أسرع مما تستطيع الطباعة طبعه فان هناك طابور من المستندات تنتظر غالباً أن تتم طباعتها عند طباعة. القاعدة العامة لمعالجة عناصر في طابور هو أنه يتم خدمة العميل الموجود في مقدمة الطابور وأنه عندما يصل عميل جديد يقف في نهاية الطابور. هذا يعني أن الطابور عبارة عن بنية بيانات الداخل أولاً يخرج أولاً.

الطوابير لها العديد من التطبيقات في علوم الحاسب الآلي. عندما يتم ضبط النظام على مبدأ الداخل أولاً يخرج أولاً يتم استخدام الطوابير. هذا الفصل يناقش واحدة من أكثر التطبيقات المستخدمة للطوابير وهو محاكاة الحاسب الآلي. بالرغم من هذا فاننا نكون أولاً في حاجة الى تطوير الأدوات اللازمة لتطبيق الطابور. الأقسام القليلة التالية تناقش كيفية تصميم فئات لتطبيق الطوابير كنوع بيانات مجرد.

الطوابير:

الطابور عبارة عن مجموعة عناصر من نفس النوع يتم فيه اضافة العناصر في نهاية واحدة تسمى مؤخرة أو جزء خلفي ويتم حذفها من النهاية الأخرى المسماة مقدمة أو أول. على سبيل المثال فكر في طابور من العملاء في مصرف حيث ينتظر العملاء القيام بسحب أو ايداع النقود أو القيام بأعمال أخرى. كل عميل جديد يدخل في الطابور في الجزء الخلفي وعندما يكون الصراف مستعد لعميل جديد يتم خدمة العميل الموجود في مقدمة الطابور.

يتم تناول الجزء الخلفي من الطابور عندما تتم اضافة عنصر جديد الى الطابور ويتم تناول مقدمة الطابور عندما يتم حذف عنصر من الطابور. كما هو الحال في المرصوفة لا يمكن تناول العناصر الموجودة بوسط الطابور حتى اذا كانت عناصر الطابور مخزنة في مصفوفة.

الطابور: بنية بيانات يتم فيها اضافة العناصر في طرف واحد يسمى الجزء الخلفي ويتم حذفها من الطرف الآخر المسمى **المقدمة** أو **الأول** وهي بنية بيانات الداخل أولاً يخرج أولاً.

عمليات الطابور:

من خلال تعريف الطوابير نرى أن العمليتين الرئيسيتين هما الاضافة والحذف ونطلق على عملية الاضافة `addQueue` وعملية الحذف `deleteQueue`. بما أن العناصر لا يمكن حذفها من طابور فارغ ولا يمكن اضافتها الى طابور ممتلئ فاننا نحتاج الى عمليتين لتطبيق عمليات `addQueue` و `deleteQueue` بنجاح وهما: `isEmptyQueue` (تحديد ما اذا كانت القائمة فارغة) و `isFullQueue` (تحدد ما اذا كانت القائمة ممتلئة).

كما نحتاج الى عملية `initializeQueue` لتهيئة الطابور في وضع فارغ و -مثل المرصوفات - نحتاج الى عملية `destroyQueue` لتدمير الطابور تاركة اياه فارغ. فضلاً عن هذا ندخل العمليتين `front` و `back` لاسترجاع العنصر الأول والعنصر الأخير من الطابور كما يتم شرحها في القائمة التالية. لهذا تكون بعض من عمليات الطوابير كما يلي:

- **initializeQueue:** تهيئ الطابور في حالة فراغ.
- **destroyQueue:** تزيل جميع العناصر من الطابور تاركة اياه فارغ.
- **isEmptyQueue:** تتحقق مما اذا كان الطابور فارغ. اذا كان الصف فارغ تنتج القيمة `true` وبخلاف هذا تنتج القيمة `false`.
- **IsFullQueue:** تتحقق مما اذا كان الطابور ممتلئ. اذا كان الصف ممتلئ تنتج القيمة `true` وبخلاف هذا تنتج القيمة `false`.
- **front:** تنتج المقدمة أي أول عنصر من الطابور. قبل هذه العملية يجب أن يتواجد الطابور.
- **back:** تنتج العنصر الأخير من الطابور وقبل هذه العملية يجب أن يتواجد الطابور.
- **addQueue:** تضيف عنصر جديد الى `rear` الطابور وقبل هذه العملية يجب أن يتواجد الطابور وألا يكون ممتلئ.
- **deleteQueue:** تزيل جميع العناصر الأولى من الطابور وقبل هذه العملية يجب أن يتواجد الطابور وألا يكون ممتلئ.

كما في حالة المرصوفة يمكن تخزين الطابور اما في مصفوفة أو في قائمة متصلة. سوف ننظر الى كلا التطبيقين. بما أن العناصر تتم اضافتها الى طرف واحد وتتم ازلتها من الطرف الآخر فاننا نحتاج الى مؤشرين لتتبع مقدمة ومؤخرة الطابور ويسموا `queueFront` و `queueRear`.

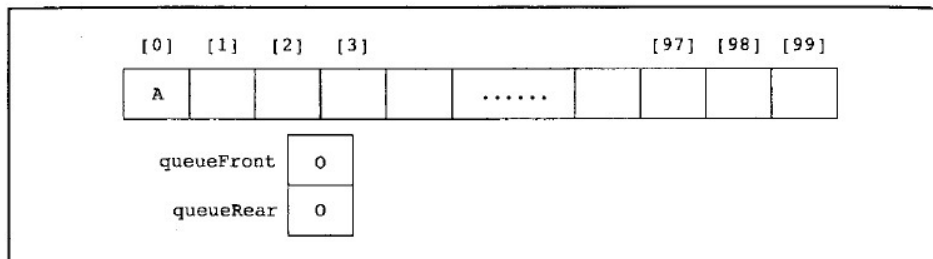
تطبيق الطوابير كمصفوفات:

قبل اعطاء تعريف الفئة لتطبيق الطابور كنوع بيانات مجرد نحتاج الى اتخاذ قرار حول عدد عناصر البيانات اللازمة لتطبيق الطابور. بالطبع نحتاج الى مصفوفة لتخزين عناصر الطابور والمتغير maxSize لتحديد الحجم الأقصى للطابور ولهذا نحتاج الى أربعة عناصر بيانات على الأقل. قبل كتابة الخوارزميات لتطبيق عمليات الطابور نحتاج الى اتخاذ قرار حول كيفية استخدام queueFront و queueRear لتناول عناصر الطابور. كيف تشير مقدمة الطابور ومؤخرة الطابور الى أن الطابور فارغ أو ممتلئ؟ افترض أن queueFront تعطي مؤشر العنصر الأول من الطابور و queueRear تعطي مؤشر العنصر الأخير من الطابور. لاضافة عنصر الى الطابور نقوم أولاً بتقديم queueRear الى موضع المصفوفة التالية ثم نضيف العنصر الجديد في الموضع الذي يشير اليه queueRear. لحذف عنصر من الطابور نقوم أولاً بتقديم queueFront الى العنصر التالي بالطابور. بهذا تتغير queueRear بعد كل عملية addQueue وتتغير queueFront بعد كل عملية deleteQueue.

لنرى ما يحدث عندما تتغير مؤخرة الطابور بعد عملية اضافة الطابور وعندما تتغير مقدمة الطابور بعد عملية حذف الطابور. افترض أن المصفوفة التي تحمل عناصر الطابور حجمها 100. مبدئياً يكون الطابور فارغ. افترض أن queueFront و queueRear تشير مباشرة الى العنصرين الأول والأخير من الطابور على التوالي. بعد العملية:

addQueue ('A');

تكون المصفوفة كما هي موضحة في شكل 8-1.



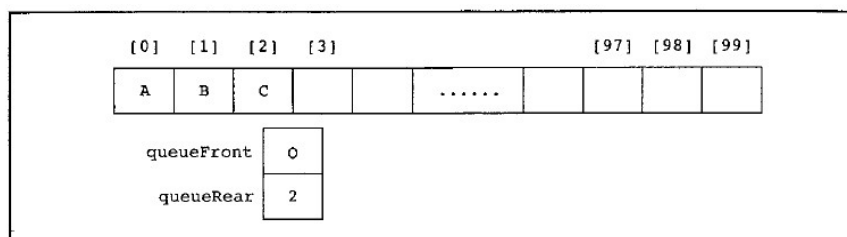
شكل 8-1: الطابور بعد العملية الأولى من addQueue

بعد عمليتان اضافيتان من addQueue:

addQueue ('B');

addQueue ('C');

تكون المصفوفة كما هي موضحة في شكل 8-2.

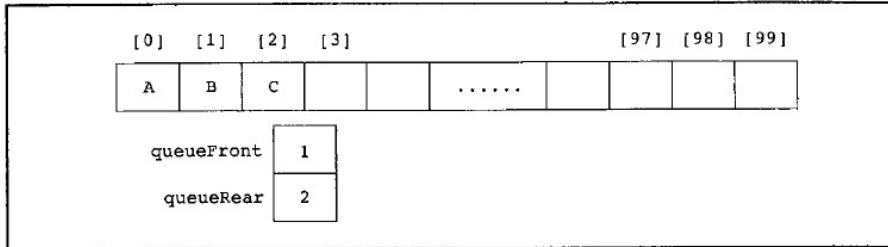


شكل 8-2: الطابور بعد عمليتان اضافيتان من addQueue

انظر الآن الى عملية deleteQueue:

deleteQueue ();

بعد هذه العملية تكون المصفوفة محتوية على الطابور كما هو واضح في شكل 8-3.

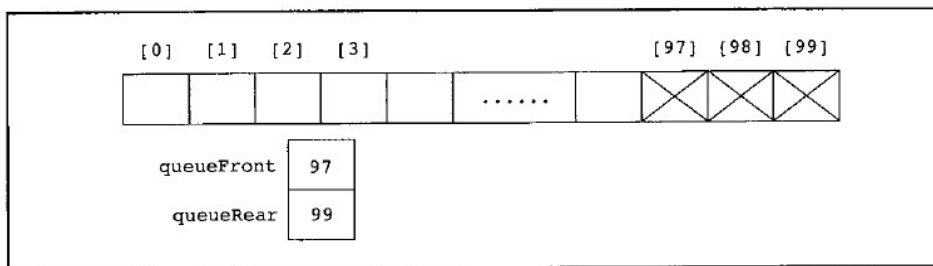


شكل 8-3: الطابور بعد عملية deleteQueue

هل تصميم الطابور هذا سوف يعمل؟ افترض أن A تعبر عن اضافة (أي addQueue) عنصر الى الطابور وتعتبر D عن حذف (أي deleteQueue) عنصر من الطابور. انظر الى التتابع التالي من العمليات:

AAADADADADADADADA...

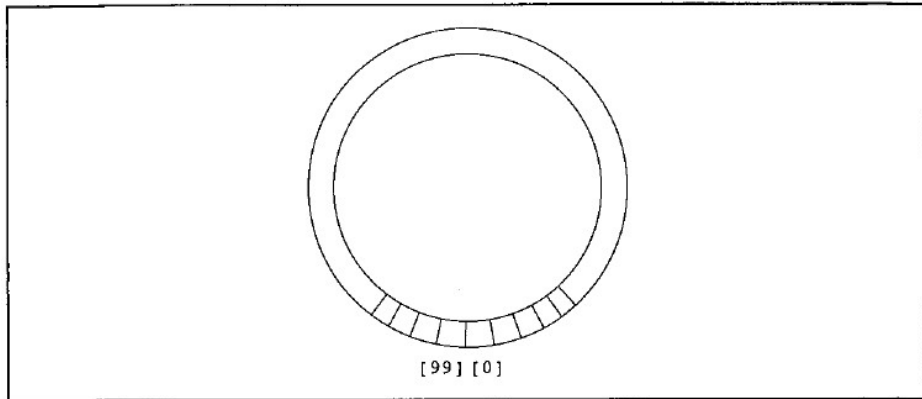
هذا التتابع من العمليات سوف يحدد في النهاية المؤشر queueRear لكي يشير الى موضع المصفوفة الأخير معطياً الانطباع بأن الطابور ممتلئ. بالرغم من هذا فالطابور به عنصرين أو ثلاثة عناصر فقط ومقدمة المصفوفة فارغة (انظر شكل 8-4).



شكل 8-4: الطابور بعد تتابع العمليات AAADADADADADADADA...

هناك حل لهذه المشكلة وهو أنه عندما يفيض الطابور الى المؤخرة (أي تشير queueRear الى موضع المصفوفة الأخير) يمكننا التحقق من قيمة المؤشر queueFront. اذا كانت قيمة queueFront تشير الى أن هناك مساحة في مقدمة المصفوفة اذن عندما تصل المؤخرة الى موضع المصفوفة الأخير يمكننا تحريك جميع عناصر الطابور نحو موضع المصفوفة الأول. هذا الحل يكون جيد اذا كان حجم الطابور صغير للغاية وبخلاف هذا قد يتم تنفيذ البرنامج بشكل أكثر بطء.

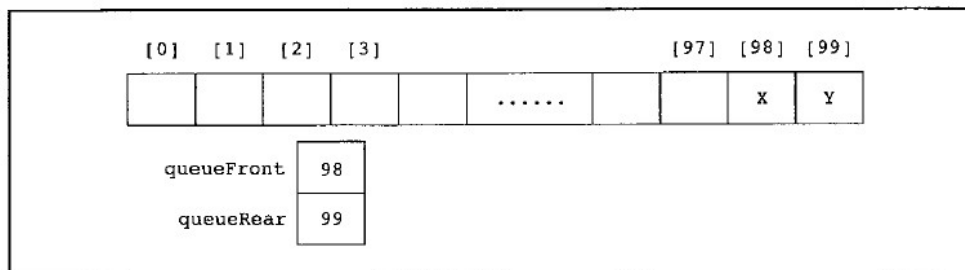
هناك حل آخر لهذه المشكلة وهو افتراض أن المصفوفة دائرية – أي أن موضع المصفوفة الأول يلي موضع المصفوفة الأخير مباشرةً (انظر شكل 5-8).



شكل 5-8: طابور دائري

ننظر الى المصفوفة المحتوية على الطابور الدائري بالرغم من قيامنا برسم أشكال للمصفوفة التي تضم عناصر الطابور كما سبق.

افترض أن لدينا الطابور الموضح في شكل 6-8.

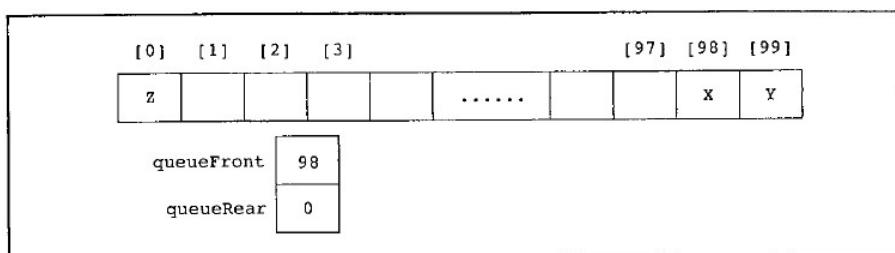


شكل 6-8: طابور به عنصرين عند الموضعين 98 و 99

بعد العملية:

`addQueue ('Z');`

يكون الطابور كما هو موضح في شكل 7-8.



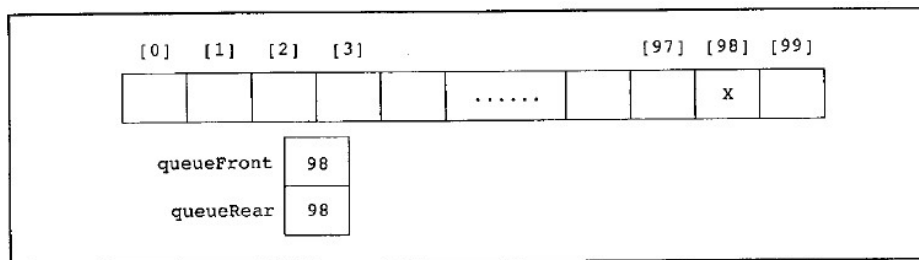
شكل 8-7: الطابور بعد عملية أخرى من addQueue
 بما أن المصفوفة المحتوية على الطابور دائرية اذن يمكننا استخدام البيان التالي لتقديم queueRear الى موضع المصفوفة التالي:

$$\text{queueRear} = (\text{queueRear} + 1) \% \text{maxQueue};$$

اذا كانت $\text{queueRear} < \text{maxQueue} - 1$ اذن $\text{queueRear} + 1 \leq \text{maxQueue} - 1$ وهكذا يكون $(\text{queueRear} + 1) \% \text{maxQueue} = \text{queueRear} + 1$ اذا كانت $\text{queueRear} == \text{maxQueue} - 1$ (أي أن queueRear تشير الى موضع المصفوفة الأخير) اذن $\text{queueRear} + 1 == \text{maxQueue}$ وهكذا $(\text{queueRear} + 1) \% \text{maxQueue} == 0$. في هذه الحالة يتم ضبط queueRear عند صفر وهو موضع المصفوفة الأول. يمكننا استخدام بيان مماثل للبيان السابق لتقديم queueFront الى موضع المصفوفة التالي.

تصميم الصف هذا يبدو أنه يعمل جيداً. قبل أن نكتب الخوارزميات لتطبيق عمليات الطابور انظر الى الحالتين التاليتين.

الحالة 1: افترض أنه بعد عمليات معينة تكون المصفوفة المحتوية على الطابور كما هي موضحة في شكل 8-8.

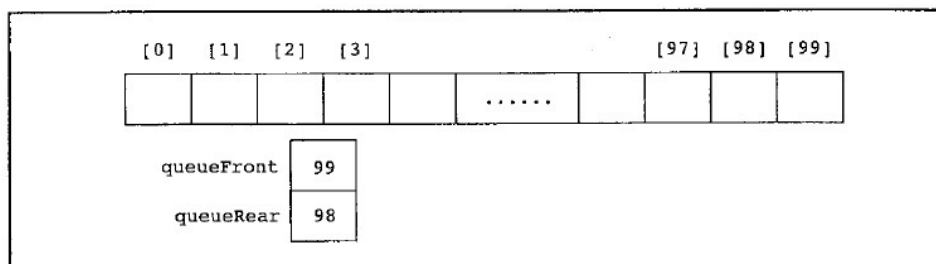


شكل 8-8: طابور ذو عنصر واحد

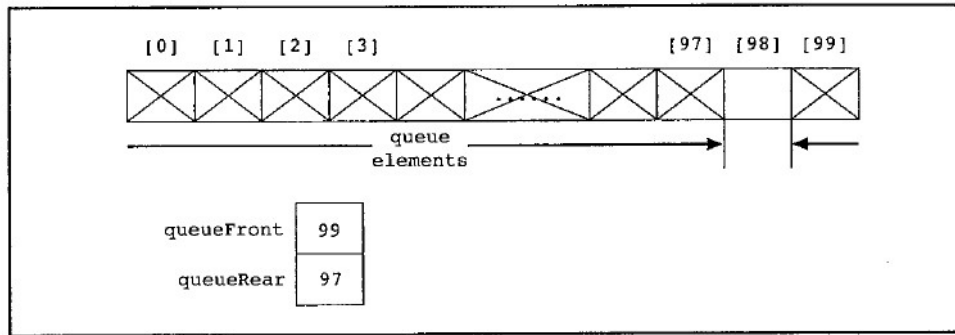
بعد العملية:

deleteQueue ();

تكون المصفوفة الناتجة كما هي موضحة في شكل 8-9.



شكل 8-9: الطابور بعد عملية deleteQueue
 الحالة 2: لننظر الآن الى الطابور الموضح في شكل 8-10.

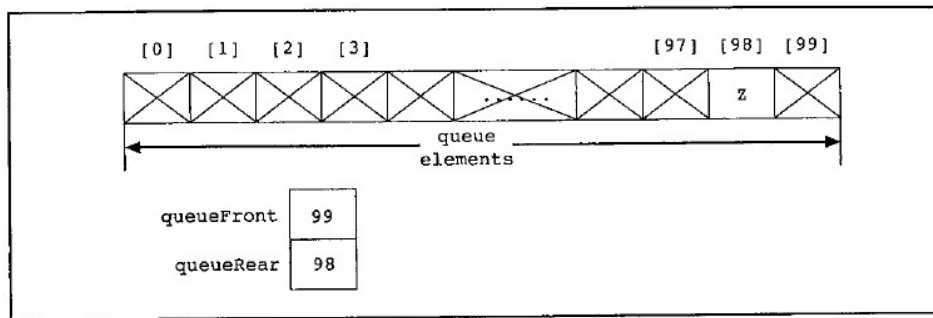


شكل 8-10: طابور به 99 عنصر.

بعد العملية:

addQueue ('z');

تكون المصفوفة الناتجة كما هي موضحة في شكل 8-11.



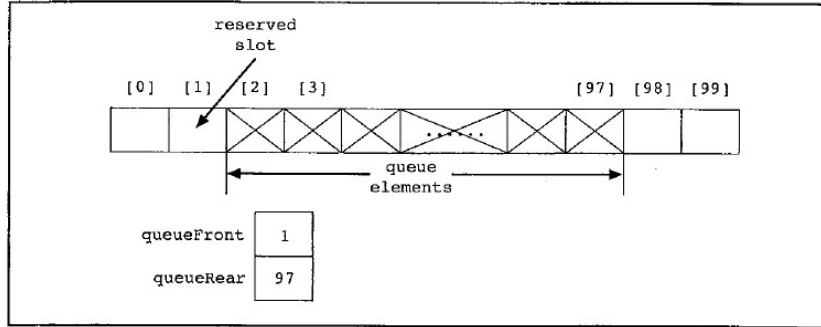
شكل 8-11: الطابور بعد العملية addQueue

المصفوفات في الأشكال 8-9 و 8-11 لها قيم متطابقة بالنسبة الى queueFront و queueRear. بالرغم من هذا فان المصفوفة الناتجة في شكل 8-9 تمثل طابور فارغ بينما المصفوفة الناتجة في شكل 8-11 تمثل طابور ممتلئ. تصميم الطابور الأخير يفرض مشكلة أخرى: التمييز بين طابور فارغ و طابور ممتلئ.

هذه المشكلة لها عدة حلول ممكنة. الحل الأول هو الاحتفاظ بالمتغير count. بالاضافة الى queueFront و queueRear نحتاج الى متغير آخر هو count لتطبيق الطابور. يتم زيادة قيمة count عند اضافة عنصر جديد الى الطابور ويتم تقليلها عند ازالة عنصر من الطابور. في هذه الحالة تقوم الدالات initializeQueue و destroyQueue بتهيئة count عند صفر. هذا الحل يكون مفيد للغاية اذا احتاج مستخدم الطابور كثيراً الى معرفة عدد العناصر في الطابور.

الحل الآخر هو جعل queueFront تشير الى مؤشر موضع المصفوفة الذي يسبق أول عنصر في الطابور بدلاً من مؤشر العنصر الأول (الفعلي). في هذه الحالة بافتراض أن queueRear لا تزال

تشير الى مؤشر العنصر الأخير في الطابور يكون الطابور فارغاً اذا كانت `queueFront ==` في هذا الحل يتم حفظ المقطع المشار اليه بواسطة المؤشر `queueFront` (أي المقطع السابق للعنصر الأول الفعلي). يكون الطابور ممتلئاً اذا كانت المساحة التالية المتاحة هي هذا المقطع الخاص بالمحفوظ المشار اليه بواسطة `queueFront`. أخيراً بما أن موضع المصفوفة المشار اليه بواسطة `queueFront` يتم الاحتفاظ به فارغاً فان حجم المصفوفة اذا كان 100 على سبيل المثال فان هناك 99 عنصر يمكن تخزينها في الطابور (انظر شكل 12-8).



شكل 12-8: المصفوفة الخاصة بتخزين عناصر الطابور وذات المقطع المحفوظ
لنقم بتطبيق الطابور باستخدام الحل الأول. هذا يعني أننا نستخدم المتغير `count` للإشارة الى ما اذا كان الطابور فارغاً أو ممتلئاً.
الفئة التالية تقوم بتعريف الطابور كنوع بيانات مجرد (انظر كذلك شكل 13-8). بما أن المصفوفات يمكن تخصيصها حركياً اذن سوف نترك للمستخدم تحديد حجم المصفوفة لتطبيق الطابور. الحجم الافتراضي للمصفوفة هو 100.

```
template<class Type>
class queueType
{
public:
    const queueType<Type>& operator=(const queueType<Type>&);
    //Overload the assignment operator.

    bool isEmptyQueue();
    //Function to determine whether the queue is empty.
    //Postcondition: Returns true if the queue is empty;
    //                otherwise, returns false.

    bool isFullQueue();
    //Function to determine whether the queue is full.
    //Postcondition: Returns true if the queue is full;
    //                otherwise, returns false.

    void initializeQueue();
    //Function to initialize the queue to an empty state.
    //Postcondition: count = 0; queueFront = 0;
    //                queueRear = maxQueueSize - 1

    const queueType<Type>& operator = (const queueType<Type>&);
```

```

// انقال معامل الاسناد.
bool isEmptyQueue ( );
// دالة لتحديد ما اذا كان الطابور فارغ.
// شرط تالي: تنتج true اذا كان الطابور فارغ وبخلاف هذا تنتج false.
bool isFullQueue ( );
// دالة لتحديد لتحديد ما اذا كان الطابور ممتلئ.
// شرط تالي: تنتج true اذا كان الطابور ممتلئ وبخلاف هذا تنتج false
void initializeQueue ( );
// دالة لتهيئة الطابور في وضع فارغ.
// شرط تالي: count = صفر، queueFront = صفر،
queueRear = maxQueueSize - 1

void destroyQueue();
//Function to remove all the elements from the queue.
//Postcondition: count = 0; queueFront = 0;
//               queueRear = maxQueueSize - 1

Type front();
//Function to return the first element of the queue.
//Precondition: The queue exists and is not empty.
//Postcondition: If the queue is empty, the program
//               terminates; otherwise, the first
//               element of the queue is returned.
Type back();
//Function to return the last element of the queue.
//Precondition: The queue exists and is not empty.
//Postcondition: If the queue is empty, the program
//               terminates; otherwise, the last
//               element of the queue is returned.

void addQueue(const Type& queueElement);
//Function to add queueElement to the queue.
//Precondition: The queue exists and is not full.
//Postcondition: The queue is changed and queueElement
//               is added to the queue.

void deleteQueue();
//Function to remove the first element of the queue.
//Precondition: The queue exists and is not empty.
//Postcondition: The queue is changed and the first
//               element is removed from the queue.

queueType(int queueSize = 100);
//constructor
queueType(const queueType<Type>& otherQueue);
//copy constructor
~queueType();
//destructor

private:
    int maxQueueSize;
    int count;
    int queueFront;
    int queueRear;
    Type *list; //pointer to the array that holds the
                //queue elements
};

```

```
void destroyQueue ( );
// دالة لازالة جميع العناصر من الطابور.
// شرط تالي: count = صفر ، queueFront = صفر ،
queueRear = maxQueueSize - 1
```

```
Type front ( );
// دالة لانتاج العنصر الأول من الطابور.
// شرط مسبق: ا الطابور متواجد وغير فارغ.
// شرط تالي: اذا كان الطابور فارغ يتوقف البرنامج وبخلاف هذا
// يتم انتاج العنصر الأول من ا الطابور.
```

```
Type back ( );
// دالة لانتاج العنصر الأخير من الطابور.
// شرط مسبق: الطابور متواجد وغير فارغ.
// شرط تالي: اذا كان الطابور فارغ يتوقف البرنامج وبخلاف هذا
// يتم انتاج العنصر الأخير من الطابور.
```

```
void addQueue (const Type& queueElement);
// دالة لاضافة عنصر الى الطابور.
// شرط مسبق: الطابور متواجد وغير ممتلئ.
// شرط تالي: يتغير الطابور وتتم اضافة عنصر الى الطابور.
```

```
void deleteQueue ( );
// دالة لحذف العنصر الأول من الطابور.
// شرط مسبق: الطابور متواجد وغير فارغ.
// شرط تالي: يتغير الطابور وتتم ازالة العنصر الأول من الطابور.
```

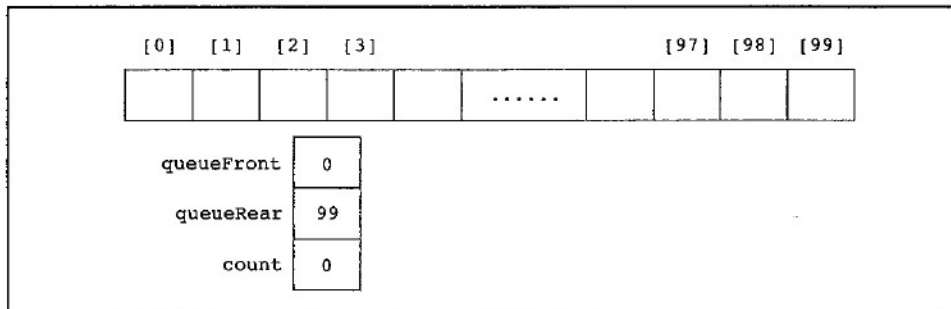
| queueType<Type> |
|---|
| -maxQueueSize: int -count: int -queueFront: int -queueRear: int -*list: Type |
| +operator=(const queueType<Type>&): const queueType<Type>& +initializeQueue(): void +destroyQueue(): void +isEmptyQueue(): bool +isFullQueue(): bool +addQueue(const Type&): void +front(): Type +back(): Type +deleteQueue(): void +queueType(int = 100) +queueType(const queueType<Type>&) +~queueType() |

شكل 8-13: رسم لغة التشكيل الموحدة للفئة queueType.

بعد هذا ننظر الى تطبيق عمليات الطابور.

تهيئة الطابور:

العملية الأولى التي ننظر اليها هي initializeQueue وهذه العملية تقوم بتهيئة الطابور في وضع فارغ وتتم اضافة أول عنصر في موضع المصفوفة الأول. لهذا نقوم بتهيئة queueFront عند صفر و queueRear عند -1 و maxQueueSize و count عند صفر. (انظر شكل 8-14).



شكل 8-14: طابور فارغ

تعريف الدالة initializeQueue هو:

```
template<class Type>
void queueType<Type>::initializeQueue()
{
    queueFront = 0;
    queueRear = maxQueueSize - 1;
    count = 0;
}
```

الطابور الفارغ والطابور الممتلئ:

كما نوقش من قبل يكون الطابور فارغاً اذا كان $count == 0$ ويكون الطابور ممتلئاً اذا كان count $== maxQueueSize$. لهذا دالات تطبيق isEmptyQueue و isFullQueue هي:

```
template<class Type>
bool queueType<Type>::isEmptyQueue()
{
    return (count == 0);
}

template<class Type>
bool queueType<Type>::isFullQueue()
{
    return (count == maxQueueSize);
}
```

تدمير الطابور:

عملية destroyQueue تزيل جميع العناصر من الطابور تاركة اياه في حالة فارغة. بما أن الطابور يتم تخزينه في مصفوفة اذن يمكننا ازالة جميع العناصر من الطابور ببساطة عن طريق اعادة تحديد queueFront و queueRear و count. لذلك في هذا التطبيق من الطابور تكون العملية destroyQueue مثل العملية initializeQueue.

```
template<class Type>
void queueType<Type>::destroyQueue()
{
    queueFront = 0;
    queueRear = maxQueueSize - 1;
    count = 0;
}
```

:front

هذه العملية تنتج العنصر الأول من الطابور واذا لم يكن الطابور فارغاً يتم انتاج عنصر الطابور المشار اليه بواسطة المؤشر queueFront وبخلاف هذا يتوقف البرنامج.

```
template<class Type>
Type queueType<Type>::front()
{
    assert(!isEmptyQueue());
    return list[queueFront];
}
```

:back

هذه العملية تنتج العنصر الأخير من الطابور واذا لم يكن الطابور فارغاً يتم انتاج عنصر الطابور المشار اليه بواسطة المؤشر queueRear وبخلاف هذا يتوقف البرنامج.

```
template<class Type>
Type queueType<Type>::back()
{
    assert(!isEmptyQueue());
    return list[queueRear];
}
```

اضافة طابور:

بعد هذا نطبق العملية addQueue. بما أن queueRear تشير مباشرة الى العنصر الأخير في الطابور اذن لاضافة عنصر جديد الى الطابور نقوم أولاً بتقديم queueRear الى موضع المصفوفة التالي ثم نضيف العنصر الجديد الى موضع المصفوفة الذي تشير اليه queueRear. كما نقوم بزيادة count بمقدار 1 ولهذا تكون الدالة addQueue:

```

template<class Type>
void queueType<Type>::addQueue(const Type& newElement)
{
    if(!isFullQueue())
    {
        queueRear = (queueRear + 1) % maxQueueSize; //use the mod
                                                    //operator to advance queueRear
                                                    //because the array is circular

        count++;
        list[queueRear] = newElement;
    }
    else
        cerr<<"Cannot add to a full queue."<<endl;
}

```

حذف طابور:

لتطبيق العملية deleteQueue نقوم بتقديم queueFront الى عنصر الطابور التالي ولهذا تكون الدالة deleteQueue هي:

```

template<class Type>
void queueType<Type>::deleteQueue()
{
    if(!isEmptyQueue())
    {
        count--;
        queueFront = (queueFront + 1) % maxQueueSize; //use the mod
                                                    //operator to advance queueFront
                                                    //because the array is circular
    }
    else
        cerr<<"Cannot remove from an empty queue."<<endl;
}

```

المقوم والمدمر:

لاكمال تطبيق عمليات الطابور نقوم بعد هذا بالنظر الى تطبيق المقوم والمدمر. المقوم يصنع مصفوفة حجمها يتحدد بواسطة المستخدم ويحدد المتغير maxQueueSize عند حجم المصفوفة. اذا لم يتم المستخدم بتحديد حجم الطابور يستخدم المقوم القيمة الافتراضية وهي 100 لعمل مصفوفة حجمها 100. كما يقوم المقوم بتهيئة queueFront و queueRear للإشارة الى أن الطابور فارغ. تعريف دالة تطبيق المقوم هي:

```

//constructor
template<class Type>
queueType<Type>::queueType(int queueSize)
{
    if(queueSize <= 0)
    {
        cerr<<"The size of the array to hold the queue must "
        <<"be positive."<<endl;
        cerr<<"Creating an array of size 100."<<endl;

        maxQueueSize = 100;
    }
    else
        maxQueueSize = queueSize; //set maxQueueSize to queueSize

    queueFront = 0; //initialize queueFront
    queueRear = maxQueueSize - 1; //initialize queueRear
    count = 0;
    list = new Type[maxQueueSize]; //create the array to
    //hold the queue elements

    assert(list != NULL);
}

```

المصفوفة الخاصة بتخزين عناصر الطابور يتم عملها حركياً ولهذا عند خروج أهداف الطابور من المجال يقوم المدمر ببساطة بإزالة تخصيص الذاكرة التي تحتلها المصفوفة التي تقوم بتخزين عناصر الطابور.

تعريف الدالة لتطبيق المدمر هو:

```

template<class Type>
queueType<Type>::~~queueType() //destructor
{
    delete [] list;
}

```

تطبيق مقوم النسخ واثقال معامل الاسناد يتم تركه كتمرين لك. (تعريفات هذه الدالات مماثلة لتلك الدالات التي تمت مناقشتها بالنسبة الى القوائم المتصلة والمرصوصات).

التطبيق المتصل للمصفوف:

بما أن حجم المصفوفة الخاصة بتخزين عناصر الطابور ثابت حيث يمكن تخزين عدد محدود من عناصر الطابور في المصفوفة. كما أن تطبيق المصفوفة للطابور يقتضي معاملة المصفوفة بطريقة خاصة مع قيم المؤشرين queueFront و queueRear. التطبيق المتصل للطابور يقوم بتبسيط العديد من الحالات الخاصة من تطبيق المصفوفة وبما أن الذاكرة التي تقوم بتخزين عنصر الطابور مخصصة حركياً فإن الطابور لا يكون ممتلئ أبداً. هذا القسم يناقش التطبيق المتصل للطابور.

بما أن العناصر تتم اضافتها في عند طرف واحد هو queueRear ويتم حذفها من الطرف الآخر وهو queueFront فاننا نحتاج الى معرفة مقدمة الطابور ومؤخرة الطابور. لهذا نحتاج الى مؤشرين وهما

queueFront و queueRear للحفاظ على الطابور. الفئة التالية تقوم بتعريف الطابور المتصل كنوع بيانات مجرد.

```
//Definition of the node
template<class Type>
struct nodeType
{
    Type info;
    nodeType<Type> *link;
};

template<class Type>
class linkedQueueType
{
public:
    const linkedQueueType<Type>& operator=
        (const linkedQueueType<Type>&);
    //Overload the assignment operator.

    bool isEmptyQueue();
    //Function to determine whether the queue is empty.
    //Postcondition: Returns true if the queue is empty;
    //                otherwise, returns false.

    (const linkedQueueType<Type>&);
    // انقال معامل الاسناد.
    bool isEmptyQueue ( );
    // دالة لتحديد ما اذا كان الطابور فارغ.
    // شرط تالي: تنتج true اذا كان الطابور فارغ وبخلاف هذا تنتج false.
```

```

bool isFullQueue();
    //Function to determine whether the queue is full.
    //Postcondition: Returns true if the queue is full;
    //                otherwise, returns false.

void destroyQueue();
    //Function to delete all the elements from the queue.
    //Postcondition: queueFront = NULL; queueRear = NULL

void initializeQueue();
    //Function to initialize the queue to an empty state.
    //Postcondition: queueFront = NULL; queueRear = NULL

Type front();
    //Function to return the first element of the queue.
    //Precondition: The queue exists and is not empty.
    //Postcondition: If the queue is empty, the program
    //                terminates; otherwise, the first
    //                element of the queue is returned.

Type back();
    //Function to return the last element of the queue.
    //Precondition: The queue exists and is not empty.
    //Postcondition: If the queue is empty, the program
    //                terminates; otherwise, the last
    //                element of the queue is returned.

void addQueue(const Type& queueElement);
    //Function to add queueElement to the queue.
    //Precondition: The queue exists and is not full.
    //Postcondition: The queue is changed and queueElement
    //                is added to the queue.

void deleteQueue();
    //Function to remove the first element of the queue.
    //Precondition: The queue exists and is not empty.
    //Postcondition: The queue is changed and the first
    //                element is removed from the queue.

linkedQueueType ();
    //default constructor
linkedQueueType(const linkedQueueType<Type>& otherQueue);
    //copy constructor
~linkedQueueType();
    //destructor

private:
    nodeType<Type> *queueFront; //pointer to the front of the queue
    nodeType<Type> *queueRear;  //pointer to the rear of the queue
};

```

bool isFullQueue ();
 // دالة لتحديد لتحديد ما اذا كان الطابور ممتلئ.
 // شرط تالي: تنتج true اذا كان الطابور ممتلئ وبخلاف هذا تنتج false

void destroyQueue ();
 // دالة لازالة جميع العناصر من الطابور.
 // شرط تالي: NULL = queueRear ، NULL = queueFront

void initializeQueue ();
 // دالة لتهيئة الطابور في وضع فارغ.

// شرط تالي: NULL = queueRear ، NULL = queueFront

Type front ();

// دالة لانتاج العنصر الأول من الطابور.
// شرط مسبق: الطابور متواجد وغير فارغ.
// شرط تالي: اذا كان الطابور فارغ يتوقف البرنامج وبخلاف هذا
// يتم انتاج العنصر الأول من الطابور.

Type back ();

// دالة لانتاج العنصر الأخير من الطابور.
// شرط مسبق: الطابور متواجد وغير فارغ.
// شرط تالي: اذا كان الطابور فارغ يتوقف البرنامج وبخلاف هذا
// يتم انتاج العنصر الأخير من الطابور.

void addQueue (const Type& queueElement);

// دالة لاضافة عنصر الى الطابور.
// شرط مسبق: الطابور متواجد وغير ممتلئ.
// شرط تالي: يتغير الطابور وتتم اضافة عنصر الى الطابور.

void deleteQueue ();

// دالة لحذف العنصر الأول من الطابور.
// شرط مسبق: الطابور متواجد وغير فارغ.
// شرط تالي: يتغير الطابور وتتم ازالة العنصر الأول من الطابور.

يتم ترك رسم لغة التشكيل الموحدة للفة linkedQueueType كتمرين لك.

تعريفات الدالات لتطبيق عمليات الطابور معطاة فيما بعد.

يكون الطابور فارغاً اذا كانت queueFront تساوي NULL. يتم تخصيص ذاكرة تخزين عناصر الطابور حركياً ولهذا لا يكون الطابور ممتلئ أبداً ولذلك الدالة لتطبيق العملية isFullQueue تنتج القيمة false. (يكون الصف ممتلئاً فقط اذا نفذت مننا الذاكرة).

العملية destroyQueue تزيل جميع عناصر الطابور تاركة اياه في حالة فارغة. كما ذكر من قبل يتم تخصيص ذاكرة تخزين عناصر الطابور حركياً ولهذا فان هذه العملية تجتاز القائمة المحتوية على الطابور بدءاً من العقدة الأولى وتزيل تخصيص الذاكرة التي تحتلها عناصر الطابور.

تعريفات الدالات لتطبيق العمليات isEmptyQueue و isFullQueue و destroyQueue معطاة فيما بعد:

```

template<class Type>
bool linkedQueueType<Type>::isEmptyQueue()
{
    return(queueFront == NULL);
}

template<class Type>
bool linkedQueueType<Type>::isFullQueue()
{
    return false;
}

template<class Type>
void linkedQueueType<Type>::destroyQueue()
{
    nodeType<Type> *temp;

    while(queueFront!= NULL) //while there are elements left
                               //in the queue
    {
        temp = queueFront;    //set temp to point to
                               //the current node
        queueFront = queueFront->link; //advance queueFront to
                                       //the next node
        delete temp;           //deallocate the memory
                               //occupied by temp
    }
    queueRear = NULL;         //set queueRear to NULL
}

```

العملية initializeQueue تقوم بتهيئة الطابور في حالة فارغة. يكون الطابور فارغاً اذا لم يكن هناك عناصر في الطابور. كما في حالة المرصوعات تقوم الدالة initializeQueue باعادة تهيئة الطابور في حالة فارغة. لاحظ أن المقوم يهيئ الطابور عندما يتم اعلان هدف الطابور. لهذا يجب أن تقوم هذه العملية بازالة جميع العناصر من الطابور.

يمكن انجاز هذه المهمة عن طريق استدعاء الدالة destroyQueue.

```

template<class Type>
void linkedQueueType<Type>::initializeQueue()
{
    destroyQueue();
}

```

العمليات addQueue، front، وback، وdeleteQueue:

عملية addQueue تضيف عنصر جديد في نهاية الطابور. لتطبيق هذه العملية نتناول المؤشر queueRear.

إذا لم يكن الطابور فارغاً تقوم العملية front بإنتاج العنصر الأول من الطابور ولهذا يتم إنتاج عنصر الطابور المشار إليه بواسطة المؤشر queueFront. إذا كان الطابور فارغاً تقوم الدالة front بإيقاف البرنامج.

إذا لم يكن الطابور فارغاً تقوم العملية back بإنتاج العنصر الأخير من الطابور ولهذا يتم إنتاج عنصر الطابور المشار إليه بواسطة المؤشر queueRear. إذا كان الطابور فارغاً تقوم الدالة back بإيقاف البرنامج. بالمثل إذا لم يكن الطابور فارغاً تقوم العملية deleteQueue بإزالة العنصر الأول من الطابور ولهذا تتناول المؤشر queueFront تعريفات الدالات لتطبيق هذه العمليات هي:

```
template<class Type>
void linkedQueueType<Type>::addQueue(const Type& newElement)
{
    nodeType<Type> *newNode;

    newNode = new nodeType<Type>; //create the node
    assert(newNode != NULL);

    newNode->info = newElement; //store the info
    newNode->link = NULL; //initialize the link field to NULL

    if(queueFront == NULL) //if initially the queue is empty
    {
        queueFront = newNode;
        queueRear = newNode;
    }
    else //add newNode at the end
    {
        queueRear->link = newNode;
        queueRear = queueRear->link;
    }
} //end addQueue
```

```

template<class Type>
Type linkedQueueType<Type>::front()
{
    assert(queueFront != NULL);
    return queueFront->info;
}

template<class Type>
Type linkedQueueType<Type>::back()
{
    assert(queueRear != NULL);
    return queueRear->info;
}

template<class Type>
void linkedQueueType<Type>::deleteQueue()
{
    nodeType<Type> *temp;

    if(!isEmptyQueue())
    {
        temp = queueFront;          //make temp point to the first node
        queueFront = queueFront->link; //advance queueFront
        delete temp;                  //delete the first node

        if(queueFront == NULL) //if after deletion the queue is empty,
            queueRear = NULL; //set queueRear to NULL
    }
    else
        cerr<<"Cannot remove from an empty queue."<<endl;
} //end deleteQueue

```

تعريف الدالة لتطبيق المقوم الافتراضي مماثلة لتعريف الدالة initializeQueue. عن خروج هدف الطابور من المجال يقوم المدمر بتدمير الطابور أي أنه يزيل تخصيص الذاكرة التي تحتلها عناصر الطابور. تعريف الدالة لتطبيق المدمر مماثلة لتعريف الدالة destroyQueue. كما أن دالات تطبيق مقوم النسخ واثقال معامل الاسناد مماثلة للدالات المقابلة بالنسبة الى المهام. تطبيق هذه العمليات متروك كتمرين لك.

الطابور المستمد من الفئة linkedListType:

من تعريفات دالات تطبيق عمليات الطابور يتضح أن التطبيق المتصل للطابور مماثل لتطبيق قائمة متصلة تم عملها بأسلوب أمامي (انظر شكل 5). عملية addQueue مماثلة للعملية insertFirst وبالمثل العمليتان initializeQueue و initilaizeList، والعمليتان isEmptyQueue و isEmprtList، والعمليتان destroyQueue و destroyList متطابقتان.

العمليتان deleteQueue و isFullQueue يمكن تطبيقهما كما تم في القسم السابق. المؤشر queueFront هو نفسه المؤشر first والمؤشر queueRear هو نفسه المؤشر last. هذا

التوافق يقترح انه يمكننا استمداد الفئة لتطبيق الطابور من الفئة linkedListType (انظر الفصل 5).

بعد هذا نستمد الفئة linkedQueueType من الفئة linkedListType ونقوم بتطبيق عمليات الطابور باستخدام العمليات المعرفة بالنسبة الى القوائم المتصلة.

```
//Queue derived from the class linkedListType
//Header file: queueLinked.h
#ifndef H_QueueType
#define H_QueueType

#include <iostream>
#include "linkedList.h"

using namespace std;

template<class Type>
class linkedQueueType: public linkedListType<Type>
{
public:
    bool isEmptyQueue();
    bool isFullQueue();
    void destroyQueue();
    void initializeQueue();
    void addQueue(const Type& newElement);
    Type front();
    Type back();
    void deleteQueue();
};

template<class Type>
void linkedQueueType<Type>::initializeQueue()
{
    linkedListType<Type>::initializeList();
}

template<class Type>
void linkedQueueType<Type>::destroyQueue()
{
    linkedListType<Type>::destroyList();
}

template<class Type>
bool linkedQueueType<Type>::isEmptyQueue()
{
    return linkedListType<Type>::isEmptyList();
}
```

```

template<class Type>
bool linkedQueueType<Type>::isFullQueue()
{
    return false;
}

template<class Type>
void linkedQueueType<Type>::addQueue(const Type& newElement)
{
    linkedListType<Type>::insertLast(newElement);
}

template<class Type>
Type linkedQueueType<Type>::front()
{
    return linkedListType<Type>::front();
}

template<class Type>
Type linkedQueueType<Type>::back()
{
    return linkedListType<Type>::back();
}

template<class Type>
void linkedQueueType<Type>::deleteQueue()
{
    nodeType<Type> *temp;

    if(!isEmptyQueue())
    {
        temp = first;           //make temp point to the first node
        first = first->link;    //advance first to the next node
        delete temp;           //delete the first node
        count--;               //decrement count
        if(first == NULL)      //if after deletion the queue is empty,
            last = NULL;       //set last to NULL
    }
    else
        cerr<<"Cannot remove from an empty queue."<<endl;
}

#endif

```

البرنامج في المثال 1-8 يختبر عمليات متنوعة على طابور. انه يستخدم الصورة المتصلة من الطابور المستمد من الفئة `linkedListType`.

مثال 1-8:

```

//Program to test the queue operations
#include <iostream>
#include "linkedList.h"
#include "queueLinked.h"

using namespace std;

int main()
{
    linkedQueueType<int> queue;
    linkedQueueType<int> copyQueue;

    int num;

    cout<<"Queue Operations"<<endl;
    cout<<"Enter numbers ending with -999"<<endl;
    cin>>num;

    while(num != -999)
    {
        queue.addQueue(num); //add an element to the queue
        cin>>num;
    }

    copyQueue = queue;      //copy the queue into copyQueue

    cout<<"Queue contains: ";
    while(!copyQueue.isEmptyQueue())
    {
        cout<<copyQueue.front()<<" ";
        copyQueue.deleteQueue(); //remove an element from
                                   //the queue
    }

    cout<<endl;

    return 0;
}

```

تنفيذ العينة: في هذه العينة يتم تظليل مدخلات المستخدم.

عمليات الطابور

ادخل أعداد تنتهي ب-999

999- 82 11 21 91 28 56 64 76 23

الطابور يتضمن: 999- 82 11 21 91 28 56 64 76 23

فئة مكتبة القالب المعياري queue (مكيف حاوية الطابور)

الأقسام السابقة ناقشت بنية البيانات المسماة الطوابير بالتفصيل. بما أن الطابور عبارة عن بنية بيانات هامة فإن مكتبة القالب المعياري تقدم فئة لتطبيق الطوابير في برنامج. اسم الفئة التي تقوم بتعريف الطابور هي queue واسم الملف الرئيسي المحتوي على تعريف الفئة queue هو queue. الفئة

queue المقدمة من مكتبة القالب المعياري يتم تطبيقها مثل الفئات التي تمت مناقشتها في هذا الفصل. الجدول 1-8 يقوم بتعريف عمليات متنوعة تقدمها فئة حاوية الطابور. جدول 1-8: عمليات على هدف طابور:

| العملية | التأثير |
|-------------|--|
| size | تنتج العدد الحقيقي من العناصر في الطابور |
| empty | تنتج true إذا كان الطابور فارغاً وبخلاف هذا تنتج false |
| push (item) | تدخل نسخة من العنصر الى الطابور |
| front | تنتج العنصر التالي – أي الأول – في الطابور ولكنها لا تزيل العنصر من الطابور. يتم تطبيق هذه العملية كدالة منتجة لقيمة |
| back | تنتج العنصر الأخير في الطابور ولكنها لا تزيل العنصر من الطابور. يتم تطبيق هذه العملية كدالة منتجة لقيمة |
| pop | تزيل العنصر التالي في الطابور |

بالإضافة الى العمليات size و empty و push و front و back و pop تقدم فئة حاوية الطابور عوامل مترابطة للمقارنة بين طابورين. على سبيل المثال يمكن استخدام العامل المترابط == لتحديد ما اذا كان الطابوران متطابقين.

البرنامج في المثال 2-8 يوضح كيفية استخدام فئة حاوية الطابور.

مثال 2-8:

```
#include <iostream>
#include <queue>

using namespace std;

int main()
{
    queue<int> intQueue; //Line 1

    intQueue.push(26); //Line 2
    intQueue.push(18); //Line 3
    intQueue.push(50); //Line 4
    intQueue.push(33); //Line 5
```

```

cout<<"Line 6: The front element of intQueue: "
    <<intQueue.front()<<endl;          //Line 6

cout<<"Line 7: The last element of intQueue: "
    <<intQueue.back()<<endl;          //Line 7

intQueue.pop();                          //Line 8

cout<<"Line 9: After the pop operation, "
    <<"the front element of intQueue: "
    <<intQueue.front()<<endl;          //Line 9

cout<<"Line 10: intQueue elements: ";    //Line 10

while(!intQueue.empty())                //Line 11
{
    cout<<intQueue.front()<<" ";        //Line 12
    intQueue.pop();                    //Line 13
}

cout<<endl;                             //Line 14

return 0;
}

```

المخرجات:

الصف 6: العنصر الأول من intQueue: 26
 الصف 7: العنصر الأخير من intQueue: 33
 الصف 9: بعد عملية الازالة يكون العنصر الأول من intQueue: 18
 الصف 10: عناصر intQueue: 33 50 18

المخرجات السابقة واضحة والتفاصيل متروكة لك كتمرين.

طوابير الأسبقية:

الأقسام السابقة أوضحت كيفية تطبيق طابور في برنامج. استخدام بنية الطابور يضمن أن العناصر تتم معالجتها بالترتيب الذي تم تلقيها به. على سبيل المثال في البيئة البنكية تتم خدمة العميل الذي يصل أولاً وبالرغم من هذا هناك ملقف معينة تحتاج فيها قاعدة الداخل أولاً يخرج أولاً الى أن تستريح نوعاً ما. في بيئة المستشفيات يتم عادةً رؤية المرضى بالترتيب الذي يصلوا به. لهذا يمكنك استخدام الطوابير لضمان أن المرضى تتم رؤيتهم بالترتيب الذي يصلوا به. بالرغم من هذا اذا وصل مريض يعاني من أمراض خطيرة أو تهدد حياته يتم التعامل معه أولاً. بمعنى آخر يأخذ هؤلاء المرضى الأسبقية على المرضى الذين يمكنهم الانتظار مثل هؤلاء المنتظرين لفحصهم السنوي الروتيني. مثال آخر عندما يتم ارسال طلبات الطبع في بيئة مشتركة الى الطابعة هناك برامج تفاعلية تأخذ أسبقية على برامج معالجة الدفعات.

هناك العديد من الحلول الأخرى حيث يتم تحديد بعض من الأسبقية للعملاء. لتطبيق بنية البيانات هذه في برنامج ما نقوم باستخدام أنواع خاصة من الطوابير تسمى **طوابير الأسبقية**. في طابور الأسبقية يتم دفع العملاء أو الوظائف ذات الأسبقية الكبيرة الى مقدمة الطابور. من الطرق الخاصة بتطبيق طابور استخدام قائمة متصلة عادية تحتفظ بالعناصر في صيغة مرتبة من الأسبقية الأعلى الى الأسبقية الأدنى. بالرغم من هذا هناك طريقة فعالة لتطبيق طابور الأسبقية وهي استخدام بنية تشبه الشجرة. (في الفصل 10 نناقش نوع خاص من خوارزمية الترتيب تسمى ترتيب مكس يستخدم بنية تشبه الشجرة لترتيب القائمة). بعد وصف هذه الخوارزمية نناقش كيفية تطبيق طابور الأسبقية بشكل فعال.

فئة مكتبة القالب المعياري `priority_queue`:

مكتبة القالب المعياري تقدم قالب الفئة `priority_queue<elemType>` حيث يتم تحديد نوع بيانات عناصر الطابور بواسطة `elemType`. قالب الفئة هذا يكون موجود في الملف الرئيسي لمكتبة القالب المعياري `queue`. هناك طرق متعددة لتحديد أسبقية عناصر طابور الأسبقية. معيار الأسبقية الافتراضي لعناصر الطابور يستخدم العامل أصغر من $>$. على سبيل المثال يمكن لبرنامج يقوم بتطبيق طابور أسبقية من الأعداد أن يستخدم العامل $>$ لتحديد الأسبقية للأعداد حتى تكون الأعداد الأكبر موجودة دائماً في مقدمة الطابور. اذا صممت فئة خاصة بك لتطبيق عناصر الطابور اذن يمكنك تحديد قاعدتك للأولوية عن طريق ائصال العامل أصغر من $>$ للمقارنة بين العناصر. كما يمكنك تعريف دالة مقارنة لتحديد الأسبقية. تتم مناقشة تطبيق دالات المقارنة في الفصل 13.

استخدام الطوابير: المحاكاة:

الأسلوب الذي يقوم فيه النظام بتشكيل سلوك نظام آخر يسمى **المحاكاة**. على سبيل المثال يقوم المحاكون الماديون بما فيها قنوات التهوية المستخدمة للتجريب مع تصميم هياكل سيارات ومحاكين طائرات لتدريب طيارين الخطوط الجوية. يتم استخدام أساليب المحاكاة عندما يكون من المكلف أو الخطير للغاية اجراء تجارب بأنظمة حقيقية. كما يمكنك تصميم نماذج حاسوبية لدراسة سلوك الأنظمة الحقيقية. (اننا نصف بعض من الأنظمة الحقيقية التي تم تشكيلها بواسطة الحاسب قريباً). محاكاة سلوك التجارب المكلفة أو الخطيرة باستخدام نموذج حاسوبي يكون عادةً أقل تكلفة من استخدام النظام الحقيقي وطريقة جيدة لاكتساب رؤية دون وضع حياة البشر في خطر. فضلاً عن هذا تكون المحاكاة الحاسوبية مفيدة بوجه خاص في الأنظمة المعقدة حيث يكون من الصعب بناء نموذج رياضي. بالنسبة الى هذه الأنظمة يمكن للنماذج الحاسوبية الاحتفاظ بدقة وصفية. في المحاكاة الرياضية يتم استخدام خطوات البرنامج لتشكيل سلوك نظام حقيقي. لنقم بالنظر الى مشكلة كهذه.

مدير مسرح أفلام محلي يسمع شكاوى من الزبائن حول الوقت الذي عليهم الانتظار فيه لشراء التذاكر. المسرح حالياً به موظف خزانة واحد فقط وهناك مسرح آخر يقوم بالتجهيز للفتح في الجوار والمدير

خائف من فقد الزبائن. المدير يريد تعيين موظفي خزانة كافيين حتى لا يجب على الزبون الانتظار طويلاً لشراء تذكرة ولكن المدير لا يريد تعيين موظفي خزانة اضافيين على أساس تجريبي ومن المحتمل أن يضيع الوقت والمال. من أحد الأشياء التي سوف يود المدير معرفتها هي المتوسط الزمني الذي ينتظره الزبون من أجل الخدمة.

المدير يريد شخصاً ما لكتابة برنامج لمحاكاة سلوك المسرح.

في المحاكاة الحاسوبية يتم تمثيل العناصر التي يتم دراستها كبيانات. بالنسبة الى مشكلة المسرح يكون البعض من العناصر هي الزبائن وموظف الخزنة. موظف الخزنة يخدم الزبائن ونحن نريد تحديد المتوسط الزمني لانتظار الزبون. يتم تطبيق التصرفات عن طريق كتابة خوارزميات يتم تطبيقها بلغة البرمجة بمساعدة الدالات. بهذا يتم استخدام الدالات لتطبيق حركات العناصر.

في C++ يمكننا دمج البيانات والعمليات المؤداة على البيانات في وحدة واحدة بمساعدة الفئات. لهذا يمكن تمثيل العناصر كفئات وعناصر بيانات الفئة تصف خصائص العناصر وعناصر الدالة تصف التصرفات المؤداة على هذه البيانات. التغير في نتائج المحاكاة قد يحدث أيضاً اذا قمنا بتغيير قيم البيانات أو تعديل تعريفات الدالات (أي تعديل الخوارزميات التي تطبق التصرفات). الهدف الرئيسي من محاكاة الحاسوب هو اما توليد نتائج توضح أداء نظام قائم أو التنبؤ بأداء نظام مقترح.

في مشكلة المسرح عندما يقوم موظف الخزنة بخدمة زبون يجب أن ينتظر الزبون الآخر. بما أنه يتم خدمة الزبائن عند مجيئهم والأساس القائم على خدمة من يأتي أولاً والطوابير تعتبر طريقة فعالة لتطبيق نظام الداخل أولاً يخرج أولاً فان الطوابير تعتبر بنية بيانات هامة للاستخدام في هذا النوع من المحاكاة الحاسوبية. هذا القسم يصف محاكاة الحاسوب التي تكون فيها الطوابير بنية بيانات أساسية. هذه المحاكاة التي تقوم بتشكيل سلوك الأنظمة تسمى **أنظمة الطوابير** التي تنتظر فيها طوابير من العناصر أن يتم خدمتها بواسطة مقدمي خدمة متنوعين. بمعنى آخر يتكون نظام الطابور من مقدمي الخدمة ومن طوابير العناصر المنتظرة لأن يتم خدمتها.

اننا نتعامل مع مجموعة متنوعة من أنظمة الطوابير على أساس يومي. على سبيل المثال يعتبر متجر البقالة والمصرف أنظمة طوابير. فضلاً عن هذا عندما ترسل طلب طبع الى طابعة تعمل بشبكة مشتركة بين العديد من الأفراد فان طلب الطبع الخاص بك يذهب الى الطابور. طلبات الطبع التي وصلت قبل طلبك بالطبع يتم اتمامها عادةً قبل طلبك ولهذا تعمل الطابعة كمقدم خدمة عندما ينتظر طابور من المستندات لكي يتم طبعه.

تصميم نظام طوابير:

في هذا القسم نوضح نظام طوابير يمكن استخدامه في مجموعة متنوعة من الاستخدامات مثل متجر البقالة، أو البنك، أو مسرح الأفلام، أو الطابعة، أو بيئة حاسوبية يحاول فيها العديد من الأفراد استخدام نفس المعالجات لتنفيذ برامجهم. لوصف نظام الطوابير نستخدم المصطلح **مقدم الخدمة** للعناصر التي

توفر الخدمة. على سبيل المثال في البنك يكون الصراف هو مقدم الخدمة وفي متجر البقالة أو مسرح الأفلام يكون موظف الخزنة هو مقدم الخدمة. اننا نطلق على العنصر الذي يتلقى الخدمة **العميل** ونطلق على زمن الخدمة – الزمن الذي يتم استهلاكه لخدمة العميل – **زمن التعامل**.

بما أن نظام الطوابير مكون من مقدمين خدمة ومن طابور ينتظر العناصر فاننا نقوم بتشكيل نظام يتكون من قيامة من مقدمي الخدمة وطابور منتظر يضم العملاي الذين تتم خدمتهم. العميل الموجود في مقدمة الطابور ينتظر مقدم الخدمة المتوفر التالي. عندما يصبح مقدم الخدمة غير مشغولاً ينتقل العميل الموجود في أول الطابور الى مقدم الخدمة المتفرغ لكي يقوم بخدمته.

عندما يصل العميل الأول يكون جميع مقدمي الخدمة متفرغين ويتحرك العميل الى أول مقدم خدمة. عندما يصل العميل التالي اذا كان مقدم الخدمة موجود يتحرك العميل على الفور الى مقدم الخدمة المتوفر وبخلاف هذا ينتظر العميل في الطابور. لتشكيل نظام الطوابير نحتاج الى معرفة عدد مقدمي الخدمة وزمن وصول العميل المتوقع والزمن بين حالات وصول العملاء وعدد الأحداث التي تؤثر على النظام.

لنقم مرة أخرى بالنظر الى نظام مسرح الأفلام. افترض أن عدد مقدمي الخدمة هو 1 وأن خدمة العميل قد تأخذ زمن متوسط قدره 5 دقائق وأن هناك عميل جديد يصل كل 4 دقائق. أداء النظام يعتمد على عدد مقدمي الخدمة المتوفرين والزمن الذي تستهلكه خدمة العميل وتكرار وصول العملاء. اذا استهلك خدمة العميل وقت طويل للغاية وكان العملاء يصلون كثيراً اذن هناك حاجة الى المزيد من مقدمي الخدمة. يمكن تشكيل هذا النظام كمحاكاة زمنية. في **المحاكاة الزمنية** يتم استخدام الساعة كعداد ويمكن استخدام فقرة قدرها دقيقة واحدة على سبيل المثال عن طريق زيادة العداد بمقدار 1. تجري المحاكاة لقدر ثابت من الزمن. اذا كان هناك حاجة الى اجراء المحاكاة لمدة 100 دقيقة فان العداد يبدأ عند 1 وينطلق الى 100 وهذا يمكن تطبيقه باستخدام حلقة.

بالنسبة الى المحاكاة التي تم وصفها في هذا القسم نريد تحديد متوسط زمن الانتظار لكل عميل. لحساب متوسط زمن الانتظار لكل عميل نحتاج الى جمع زمن انتظار كل عميل ثم نقسم المجموع على عدد العملاء الذين وصلوا. عندما يصل عميل فانه يذهب الى نهاية الطابور ويبدأ زمن انتظار العميل. اذا كان الطابور فارغاً ومقدم الخدمة متفرغ فان العميل تتم خدمته على الفور ولهذا يكون زمن انتظار هذا العميل صفر. على الجانب الآخر اذا كان الطابور غير فارغ أو كان جميع مقدمي الخدمة مشغولين عند وصول العميل فيجب على العميل الانتظار من أجل مقدم الخدمة المتاح ولهذا يبدأ زمن انتظار هذا العميل. يمكننا تتبع زمن انتظار العميل عن طريق استخدام تايمر (ساعة) لكل عميل. عندما يصل عميل يتم ضبط الزمن عند صفر وتتم زيادته بعد كل وحدة من الساعة.

افترض أن متوسط الزمن الذي يستهلكه مقدم الخدمة لخدمة عميل يساوب 5 دقائق. عندما يصبح مقدم الخدمة متفرغ وطابور العملاء المنتظرين غير فارغ يتقدم العميل الموجود في أول الطابور لكي يبدأ

التعامل. لهذا يجب أن نتتبع زمن العميل مع مقدم الخدمة. عندما يصل العميل الى مقدم الخدمة يتم ضبط زمن التعامل عند 5 ويتم تقليله بعد كل وحدة زمنية. عندما يصبح زمن التعامل صفر يتم تحديد أن مقدم الخدمة غير مشغول. من هنا هناك حاجة الى عنصرين لتطبيق محاكاة حاسوبية زمنية لنظام الطوابير وهما العميل ومقدم الخدمة.

بعد هذا وقبل تصميم الخوارزمية الرئيسية لتطبيق هذه المحاكاة نقوم بتصميم فئات لتطبيق كل من هذين العنصرين: العميل ومقدم الخدمة.

العميل:

كل عميل له رقم عميل، وزمن وصول، وزمن انتظار، وزمن تعامل، وزمن رحيل. اذا علمنا أزمنة الوصول والانتظار والتعامل يمكننا تحديد زمن الرحيل عن طريق جمع زمن الوصول وزمن الانتظار وزمن التعامل. لنقم بتسمية فئة تطبيق عنصر العميل `customerType`. ينتج من هذا أن الفئة `customerType` لها أربعة عناصر بيانات وهي: `customerNumber`، `arrivalTime`، `waitingTime`، و `transactionTime` وكل منها من النوع `int`.

العمليات الرئيسية التي يتم اجرائها على هدف من النوع `customerType` تكون كما يلي: تحديد رقم العميل، وزمن الوصول، وزمن الانتظار، وزيادة زمن الانتظار بمقدار وحدة زمنية واحدة، وانتاج زمن الانتظار، وانتاج زمن الوصول، وانتاج زمن التعامل، وانتاج رقم العميل. الفئة التالية `customerType` تقوم بتطبيق العميل كنوع بيانات مجرد (انظر كذلك شكل 8-15)

```

class customerType
{
public:
    customerType(int customerN = 0, int arrvTime = 0, int wTime = 0,
                 int tTime = 0);
    //constructor to initialize the data members
    //according to the parameters
    //In the object declaration if no value is specified,
    //the default is assigned.
    //Postcondition: customerNumber = customerN;
    //                  arrivalTime = arrvTime;
    //                  waitingTime = wTime;
    //                  transactionTime = tTime
    void setCustomerInfo(int customerN, int arrvTime,
                        int wTime, int tTime);
    //Function to set the data members according
    //to the parameters.
    //Postcondition: customerNumber = customerN;
    //                  arrivalTime = arrvTime;
    //                  waitingTime = wTime;
    //                  transactionTime = tTime
    int getWaitingTime() const;
    //Function to return the waiting time of a customer.
    //Postcondition: The value of waitingTime is returned.
    void setWaitingTime(int time);
    //Function to set the waiting time of a customer.
    //Postcondition: waitingTime = time
    void incrementWaitingTime();
    //Function to increment the waiting time.
    //Postcondition: waitingTime++
    int getArrivalTime();
    //Function to return the arrival time of a customer.
    //Postcondition: The value of arrivalTime is returned.
    int getTransactionTime();
    //Function to return the transaction time of a customer.
    //Postcondition: The value of transactionTime is returned.
    int getCustomerNumber();
    //Function to return the customer number.
    //Postcondition: The value of customerNumber is returned.

```

// مقوم لتهيئة عناصر البيانات

// وفقاً للعوامل

// في الان الهدف اذا لم يتم تحديد قيمة يتم تحديد القيمة الافتراضية.

// شرط تالي: رقم العميل = customerN

زمن الوصول = arrvTime

زمن الانتظار = wTime

زمن التعامل = tTime

void setCustomerInfo (int customerN, int arrvTime,
int wTime, int tTime) ;

// دالة لتحديد عناصر البيانات وفقاً للعوامل.

// شرط تالي: رقم العميل = customerN

زمن الوصول = arrivalTime
 زمن الانتظار = wTime
 زمن التعامل = tTime

int getWaitingTime () const;
 // دالة لانتاج زمن انتظار العميل.
 // شرط تالي: يتم انتاج قيمة زمن الانتظار.

void setWaitingTime (int time);
 // دالة لتحديد زمن انتظار العميل.
 // شرط تالي: زمن الانتظار = time

void incrementWaitingTime ();
 // دالة لزيادة زمن الانتظار .
 // شرط تالي: waitingTime++

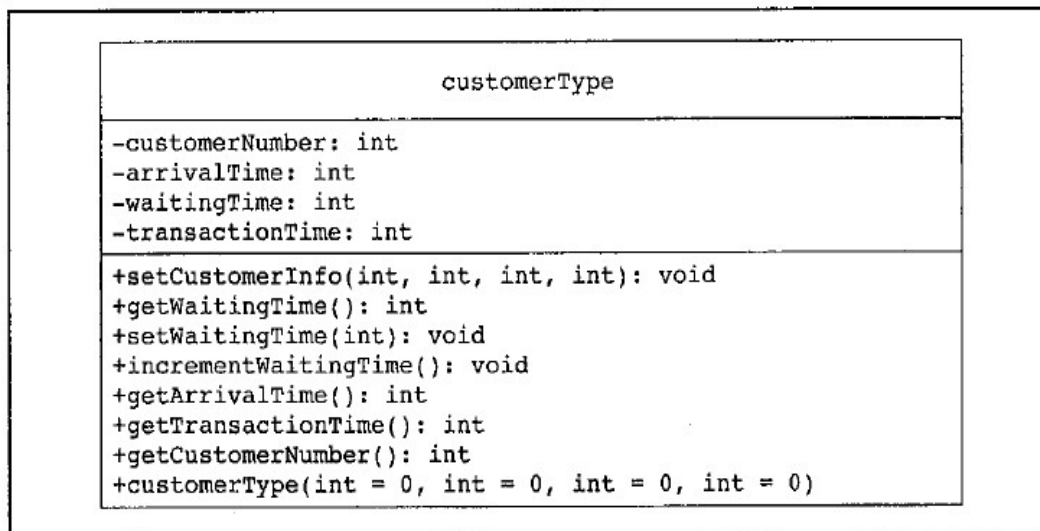
int getArrivalTime ();
 // دالة لانتاج زمن وصول العميل.
 // شرط تالي: يتم انتاج قيمة زمن الوصول.

int getTransactionTime ();
 // دالة لانتاج زمن تعامل العميل.
 // شرط تالي: يتم انتاج قيمة زمن التعامل.

int getCustomerNumber () const;
 // دالة لانتاج رقم العميل.
 // شرط تالي: يتم انتاج قيمة رقم العميل.

```

private:
    int customerNumber;
    int arrivalTime;
    int waitingTime;
    int transactionTime;
};
  
```



شكل 8-15: رسم لغة التشكيل الموحدة للفئة `customerType` بعد هذا نعطي تعريفات دالات العنصر للفئة `customerType`.
الدالة `setCustomerInfo` تستخدم قيم العوامل لتهيئة `customerNumber` و `arrivalTime` و `waitingTime` و `transactionTime` وتعريفها يكون كما يلي:

```
void customerType::setCustomerInfo(int customerN, int arrvTime,
                                   int wTime, int tTime)
{
    customerNumber = customerN;
    arrivalTime = arrvTime;
    waitingTime = wTime;
    transactionTime = tTime;
}
```

تعريف المقوم مماثل لتعريف الدالة `setCustomerInfo`. انها تستخدم قيم العوامل العوامل لتهيئة `customerNumber` و `arrivalTime` و `waitingTime` و `transactionTime`. لجعل تصحيح الأخطاء أسهل نستخدم الدالة `setCustomerInfo` لكتابة تعريف المقوم المعطى فيما بعد.

```
customerType::customerType(int customerN, int arrvTime, int wTime,
                           int tTime)
{
    setCustomerInfo(customerN, arrvTime, wTime, tTime);
}
```

الدالة `getWaitingTime` تنتج زمن الانتظار الحالي. بما أن زمن الانتظار يتم حفظه في عنصر البيانات الخاص `waitingTime` فان الدالة `getWaitingTime` تنتج قيمة زمن الانتظار. تعريف الدالة `getWaitingTime` هو:

```
int customerType::getWaitingTime() const
{
    return waitingTime;
}
```

الدالة `incrementWaitingTime` تزيد قيمة زمن الانتظار وتعريفها هو:

```
void customerType::incrementWaitingTime()
{
    waitingTime++;
}
```

تعريفات الدالات `setWaitingTime` و `getArrivalTime` و `getTransactionTime` و `getCustomerNumber` متروكة كتمرين لك.

مقدم الخدمة:

في أي وحدة زمنية محددة ما أن يكون مقدم الخدمة مشغول واما أن يكون غير مشغول. اننا نستخدم المتغير المقطعي status لتحديد حالة مقدم الخدمة. كل مقدم خدمة له ساهة توقيت وبما أن البرنامج قد يحتاج الى معرفة مقدم الخدمة الذي يخدم العميل فان مقدم الخدمة يقوم بتخزين معلومات العميل الذي يخدمه. لهذا هناك ثلاثة عناصر بيانات مرتبطة بمقدم الخدمة وهي: الحالة من النوع string وزمن التعامل من النوع Int والعميل الحالي من النوع customerType. بعض من العمليات الأساسية التي يجب اجراؤها على مقدم الخدمة تكون كما يلي: التحقق مما اذا كان مقدم الخدمة غير مشغول، وتحديد مقدم الخدمة كغير مشغول، وتحديد مقدم الخدمة كمشغول، وتحديد زمن التعامل (أي الزمن الذي يأخذه خدمة العميل) وانتاج زمن التعامل المتبقي (لتحديد ما اذا كان يجب وضع مقدم الخدمة عن التفرغ) واذا كان مقدم الخدمة مشغولاً بعد كل وحدة زمنية تقليل زمن التعامل بمقدار وحدة زمنية واحدة وغيرها.

الفئة التالية serverType تطبق مقدم الخدمة كنوع بيانات مجرد (انظر شكل 8-16):

```
class serverType
{
public:
    serverType();
    //default constructor
    //Set the values of the data members to their default
    //values.
```

```

        //Postcondition: currentCustomer is initialized by its
        //                  default constructor; status = "free";
        //                  the transaction time is initialized to 0.
bool isFree() const;
    //Function to determine whether a server is free.
    //Postcondition: Returns true if the server is free;
    //                  otherwise, returns false.
void setBusy();
    //Function to set the status of a server to "busy".
    //Postcondition: status = "busy".
void setFree();
    //Function to set the status of a server to "free".
    //Postcondition: status = "free".
void setTransactionTime(int t);
    //Function to set the transaction time according
    //to the parameter t.
    //Postcondition: transactionTime = t
void setTransactionTime();
    //Function to set the transaction time according
    //to the transaction time of the current customer.
    //Postcondition:
    //    transactionTime = currentCustomer.transactionTime
int getRemainingTransactionTime();
    //Function to return the remaining transaction time.
    //Postcondition: The value of the data member
    //                  transactionTime is returned.
void decreaseTransactionTime();
    //Function to decrease the transaction time by 1.
    //Postcondition: transactionTime--
void setCurrentCustomer(customerType cCustomer);
    //Function to set the info of the current customer
    //according to the parameter cCustomer.
    //Postcondition: currentCustomer = cCustomer.
int getCurrentCustomerNumber();
    //Function to return the customer number of the
    //current customer.
    //Postcondition: The value of the data member
    //                  customerNumber of the current customer
    //                  is returned.
int getCurrentCustomerArrivalTime();
    //Function to return the arrival time of the current customer.
    //Postcondition: The value of the data member arrivalTime
    //                  of the current customer is returned.
int getCurrentCustomerWaitingTime();
    //Function to return the current waiting time of the
    //current customer.
    //Postcondition: The value of the data member
    //                  waitingTime of the current
    //                  customer is returned.

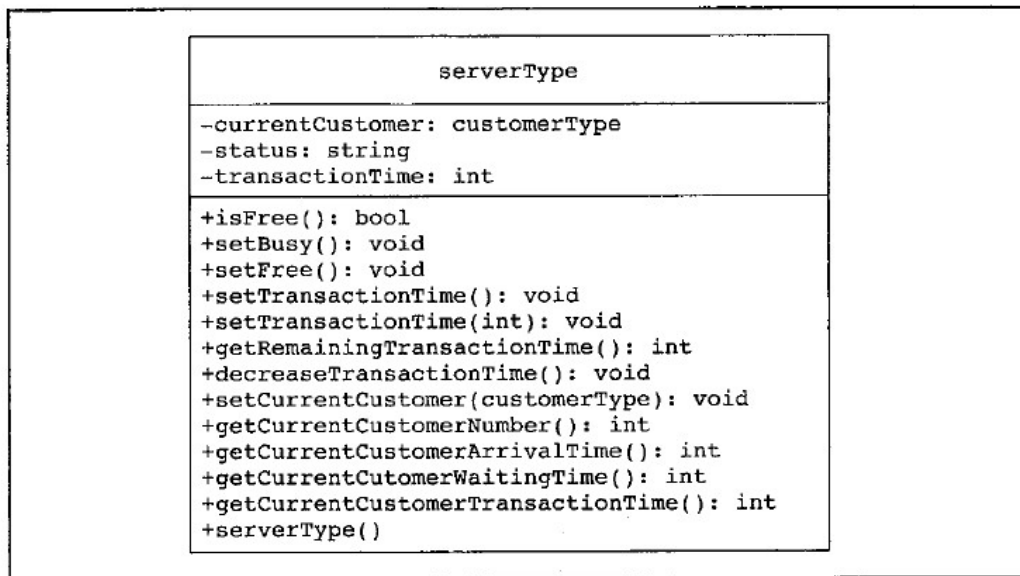
```

```

int getCurrentCustomerTransactionTime();
//Function to return the transaction time of the
//current customer.
//Postcondition: The value of the data member
//                transactionTime of the current customer
//                is returned.

private:
    customerType currentCustomer;
    string status;
    int transactionTime;
};

```



شكل 8-16: رسم لغة التشكيل الموحدة للفئة serverType
تعريفات عناصر بيانات الفئة serverType كما يلي:

```

serverType::serverType()
{
    status = "free";
    transactionTime = 0;
}

bool serverType::isFree() const
{
    return (status == "free");
}

```

```

void serverType::setBusy()
{
    status = "busy";
}

void serverType::setFree()
{
    status = "free";
}

void serverType::setTransactionTime(int t)
{
    transactionTime = t;
}

void serverType::setTransactionTime()
{
    int time;

    time = currentCustomer.getTransactionTime();

    transactionTime = time;
}

void serverType::decreaseTransactionTime()
{
    transactionTime--;
}

```

- نترك تعريفات الدالات التالية كتمرين لك: getRemainingTransactionTime
- setCurrentCustomer – getCurrentCustomerNumber
- getCurrentCustomerArrivalTime – getCurrentCustomerWaitingTime
- getCurrentCustomerTransaction Time

بما أننا نصمم برنامج محاكاة يمكن استخدامه في مجموعة متنوعة من التطبيقات اذن نحن في حاجة الى تصميم فئتين أخرتين: فئة لعمل ومعالجة قائمة من مقدمي الخدمة، وفئة أخرى لعمل ومعالجة طابور العملاء المنتظرين. القسمان التاليان يصفان كل من هاتين الفئتين.

قائمة مقدم الخدمة:

قائمة مقدم الخدمة عبارة عن مجموعة من مقدمي الخدمة مثل صف من صرافين البنك. في أي وقت يكون مقدم الخدمة اما مشغول أو غير مشغول. بالنسبة الى العميل الموجود في مقدمة الطابور نحتاج الى ايجاد مقدم خدمة في القائمة يكون غير مشغول. اذا كان جميع مقدمي الخدمة مشغولين يجب على العميل الانتظار حتى يصبح واحداً من مقدمي الخدمة غير مشغول. لهذا فان الفئة التي تطبق قائمة من مقدمي الخدمة لها عنصري بيانات أحدهما لتخزين عدد مقدمي الخدمة والآخر للاحتفاظ بقائمة مقدمي الخدمة. باستخدام المصفوفات الحركية وبالا اعتماد على عدد مقدمي الخدمة المحدد بواسطة المستخدم يتم عمل قائمة بمقدمي الخدمة أثناء تنفيذ البرنامج.

من العمليات التي يجب إجراؤها على قائمة مقدم الخدمة ما يلي: إنتاج رقم مقدم الخدمة لمقدم خدمة غير مشغول، وعندما يكون العميل مستعد لانتهاء أعماله ومقدم الخدمة متوفر ضبط مقدم الخدمة عند "مشغول"، وعندما تنتهي المحاكاة قد لا يزال بعض مقدمي الخدمة مشغولون لذلك إنتاج عدد مقدمي الخدمة المشغولين بعد كل وحدة زمنية، وتقليل زمن تعامل كل مقدم خدمة بمقدار وحدة زمنية واحدة وإذا أصبح زمن تعامل مقدم الخدمة صفر ضبط مقدم الخدمة عند "غير مشغول". الفئة التالية serverListType تطبق قائمة مقدمي الخدمة كنوع بيانات مجرد (انظر كذلك شكل 8-17):

```
class serverListType
{
public:
    serverListType(int num = 1);
        //constructor to create a list of servers
        //Postcondition: numServers = num
        //    A list of servers, specified by num, is created.
        //    If no value is specified for num, its default
        //    value is assumed.
    ~serverListType();
        //destructor
        //Postcondition: The list of servers is destroyed.
    int getFreeServerID();
        //Function to search the list of servers.
        //Postcondition: If a free server is found, return its ID;
        //    otherwise, return -1.
    int getNumberOfBusyServers();
        //Function to return the number of busy servers.
        //Postcondition: The number of busy servers is returned.
    void setServerBusy(int serverID, customerType cCustomer,
        int tTime);
        //Function to set a server to "busy".
        //Postcondition: To serve the customer specified by
        //    cCustomer, the server specified by serverID is set
        //    to "busy", and the transaction time is set according
        //    to the parameter tTime.
    void setServerBusy(int serverID, customerType cCustomer);
        //Function to set a server to "busy".
        //Postcondition: To serve the customer specified by
        //    cCustomer, the server specified by serverID is
        //    set to "busy", and the transaction time is set
        //    according to the customer's transaction time.
    void updateServers();
        //Function to update the transaction time of each
        //busy server.
        //Postcondition: The transaction time of each busy
        //    server is decremented by one time unit. If the
        //    transaction time of a busy server is reduced to
        //    zero, the server is set to "free" and a message
        //    indicating which customer was served, together
```

serverListType (int num = 1);
 // مقوم لعمل قائمة من مقدمي الخدمة
 // شرط تالي: عدد مقدمي الخدمة = num

// يتم عمل قائمة بمقدمي الخدمة.
// اذا لم يتم تحديد قيمة من أجل num يتم افتراض القيمة الافتراضية.

-serverListType ();

// مدمر
// شرط تالي: يتم تدمير قائمة مقدمي الخدمة.

int getFreaaServerID ();

// دالة لبحث قائمة مقدمي الخدمة.
// شرط تالي: اذا تم العثور على مقدم خدمة غير مشغول
// تنتج هويته وبخلاف هذا تنتج -1.

int detNumberOfBusyServers ();

// دالة لانتاج عدد مقدمي الخدمة المشغولين.
// شرط تالي: يتم انتاج عدد مقدمي الخدمة المشغولين،

void setServerBusy (int serverID, customerType cCustomer,
int tTime);

// دالة لضبط مقدم الخدمة عند "مشغول".
// شرط تالي: لخدمة العميل المحدد بواسطة cCustomer
// يتم ضبط مقدم الخدمة المحدد بواسطة serverID
// عند "مشغول" ويتم ضبط زمن التعامل وفقاً
// للمعامل tTime.

void setServerBusy (int serverID, customerType cCustomer) ;

// دالة لضبط مقدم الخدمة عند "مشغول".
// شرط تالي: لخدمة العميل المحدد بواسطة cCustomer
// يتم ضبط مقدم الخدمة المحدد بواسطة serverID
// عند "مشغول" ويتم ضبط زمن التعامل وفقاً
// لزمن تعامل العميل.

void updateServers ();

// دالة لتحديث زمن تعامل كل مقدم خدمة مشغول.
// شرط تالي: يتم تقليل زمن تعامل كل مقدم خدمة مشغول
// بمقدار وحدة زمنية. اذا تم تقليل زمن تعامل مقدم خدمة
// مشغول الى صفر يتم ضبط مقدم الخدمة عند "حر"
// ورسالة تشير الى أي عميل تتم خدمته مع طباعة زمن رحيل
// العميل على الشاشة.

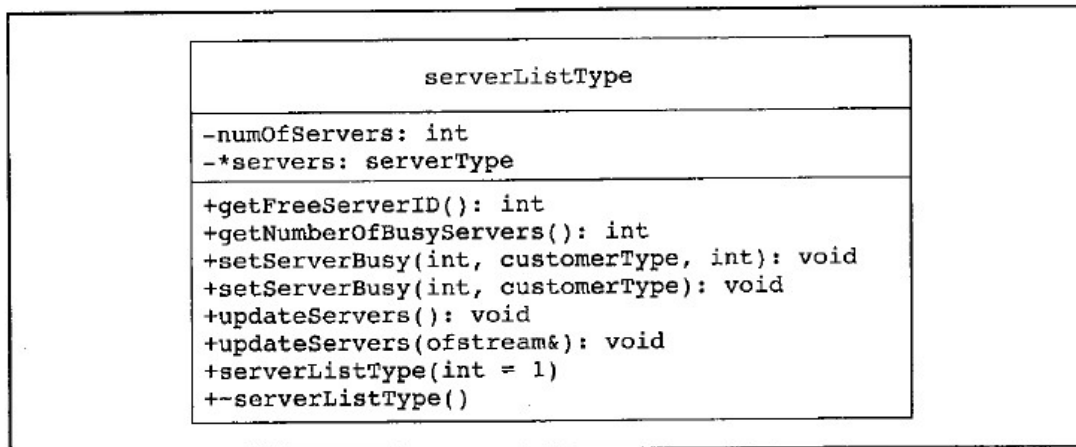
```

// with the customer's departing time, is printed
// on the screen.
void updateServers(ofstream& outFile);
//Function to update the transaction time of each
//busy server.
//Postcondition: The transaction time of each busy
// server is decremented by one time unit. If the
// transaction time of a busy server is reduced to
// zero, the server is set to "free" and a message
// indicating which customer was served, together
// with the customer's departing time, is sent to
// a file specified by outFile.

private:
    int numOfServers;
    serverType *servers;
};

void updateServers (ofstream& outFile) ;
// دالة لتحديث زمن تعامل كل مقدم خدمة مشغول.
// شرط تالي: يتم تقليل زمن تعامل كل مقدم خدمة مشغول
// بمقدار وحدة زمنية. اذا تم تقليل زمن تعامل مقدم خدمة
// مشغول الى صفر يتم ضبط مقدم الخدمة عند "حر"
// ويتم ارسال رسالة تشير الى أي عميل تتم خدمته مع
// زمن رحيل العميل الى ملف يحدده ملف المخرجات.

```



شكل 8-17: رسم لغة التشكيل الموحدة للفئة serverListType

فيما يلي تعريفات دالات عناصر الفئة serverListType. تعريفات المقوم والمدمر كما يلي:

```

serverListType::serverListType(int num)
{
    numOfServers = num;
    servers = new serverType[num]; //create a list of servers
}

serverListType::~serverListType()
{
    delete [] servers;
}

```

الدالة `getFreeServerID` تبحث قائمة مقدمي الخدمة وإذا تم العثور على مقدم خدمة حر تنتج هويته وبخلاف هذا تنتج القيمة -1 التي تشير الى أن جميع مقدمي الخدمة مشغولون. تعريف هذه الدالة يكون:

```
int serverListType::getFreeServerID()
{
    int serverID = -1;

    int i;

    for(i = 0; i < numOfServers; i++)
        if(servers[i].isFree())
        {
            serverID = i;
            break;
        }

    return serverID;
}
```

الدالة `getNumberOfBusyServers` تبحث قائمة مقدمي الخدمة وتحدد عدد مقدمي الخدمة المشغولين ويتم انتاج عدد مقدمي الخدمة المشغولين وتعريف هذه الدالة هو:

```
int serverListType::getNumberOfBusyServers()
{
    int busyServers = 0;

    int i;

    for(i = 0; i < numOfServers; i++)
        if(!servers[i].isFree())
            busyServers++;

    return busyServers;
}
```

الدالة `setServerBusy` تحدد أن مقدم الخدمة "مشغول" وهذه الدالة يتم ائصالها. هوية مقدم الخدمة الذي يتم تحديده كـ "مشغول" يتم تمريرها كعامل لهذه الدالة. دالة تحدد زمن تعامل مقدم الخدمة وفقاً للمعامل `tTime` وأخرى تحدد باستخدام الوقت المخزن في العنصر `cCustomer`.

يتم الحاجة الى زمن التعامل بعد هذا لتحديد متوسط زمن الانتظار. تعريفات هذه الدالات تكون كما يلي:

```

void serverListType::setServerBusy(int serverID,
                                   customerType cCustomer,
                                   int tTime)
{
    servers[serverID].setBusy();
    servers[serverID].setTransactionTime(tTime);
    servers[serverID].setCurrentCustomer(cCustomer);
}

void serverListType::setServerBusy(int serverID,
                                   customerType cCustomer)
{
    int time;

    time = cCustomer.getTransactionTime();

    servers[serverID].setBusy();
    servers[serverID].setTransactionTime(time);
    servers[serverID].setCurrentCustomer(cCustomer);
}

```

بعد هذا ننظر الى تعريف الدالة updateServers. بدءاً من مقدم الخدمة الأول تقوم ببحث قائمة مقدمي الخدمة من أجل مقدمي الخدمة المشغولين. عند العثور على مقدم خدمة مشغول يتم تقليل زمن التعامل بمقدار 1. اذا قل زمن التعامل الى صفر يتم ضبط مقدم الخدمة عند "غير مشغول" واذا قل زمن تعامل مقدم خدمة مشغول الى صفر فان تعامل العميل الذي تتم خدمته بواسطة مقدم الخدمة هذا ينتهي ثم تتم طباعة رسالة تشير الى رقم مقدم الخدمة للعميل، ورقم العميل، وزمن الرحيل. الدالة updateServer يتم ائصالها. هناك دالة ترسل المخرجات الى الشاشة والأخرى ترسل المخرجات الى ملف. تعريفات هذه الدالات تكون كما يلي:

```

void serverListType::updateServers()
{
    int i;

    for(i = 0; i < numOfServers; i++)
        if(!servers[i].isFree())
        {
            servers[i].decreaseTransactionTime();

            if(servers[i].getRemainingTransactionTime() == 0)
            {
                cout<<"Server No: "<<(i + 1)<<" Customer number "
                    <<servers[i].getCurrentCustomerNumber()

```

```

        <<" departed at "<<endl
        <<"          clock unit "
        <<servers[i].getCurrentCustomerArrivalTime()
        + servers[i].getCurrentCustomerWaitingTime()
        + servers[i].getCurrentCustomerTransactionTime()
        <<endl;
        servers[i].setFree();
    }
}

void serverListType::updateServers(ofstream& outFile)
{
    int i;

    for(i = 0; i < numOfServers; i++)
        if(!servers[i].isFree())
        {
            servers[i].decreaseTransactionTime();

            if(servers[i].getRemainingTransactionTime() == 0)
            {
                outFile<<"Server No: "<<(i + 1)<<" Customer number "
                <<servers[i].getCurrentCustomerNumber()
                <<" departed at "<<endl
                <<"          clock unit "
                <<servers[i].getCurrentCustomerArrivalTime()
                + servers[i].getCurrentCustomerWaitingTime()
                + servers[i].getCurrentCustomerTransactionTime()
                <<endl;
                servers[i].setFree();
            }
        }
}

```

انتظار طابور العملاء:

عندما يصل عميل يذهب الى نهاية الطابور وعندما يصبح مقدم خدمة غير مشغول يرحل العميل الموجود في مقدمة الطابور لبدء التعامل. بعد كل وحدة زمنية يتم زيادة زمن انتظار كل عميل في الطابور بمقدار 1. نوع البيانات المجرد queueType الذي تم تصميمه في هذا الفصل يمتلك جميع العمليات اللازمة لتطبيق طابور فيما عدا عملية زيادة زمن انتظار كل عميل في الطابور بمقدار وحدة زمنية واحدة. لهذا نستمد الفئة waitingCustomerQueueType من الفئة

queueType ونضيف العمليات الاضافية لتطبيق طابور العميل.
تعريف الفئة waitingCustomerQueueType يكون:

```

class waitingCustomerQueueType: public queueType<customerType>
{
public:
    waitingCustomerQueueType(int size = 100);
        //constructor
        //Postcondition: The queue is initialized
        //    according to the parameter size. The value of
        //    size is passed to the constructor of queueType.
        //    If no value is specified for size, its default
        //    value is assumed.
    ~waitingCustomerQueueType();
        //destructor
        //Postcondition: The queue is destroyed
    void updateWaitingQueue();
        //Function to increment the waiting time of each
        //customer in the queue by one time unit.
        //Postcondition: The waiting time of each customer in
        //    the queue is incremented by one time unit.
};

```

لاحظ أن الفئة waitingCustomerQueueType مستمدة من الفئة queueType التي تقوم بتطبيق الطابور في مصفوفة. كما يمكنك استمداد الفئة waitingCustomerQueueType من الفئة linkedQueueType التي تقوم بتطبيق الطابور في قائمة متصلة. نترك التفاصيل كتمرين لك. تعريفات دالات العنصر معطاة فيما يحد. تعريفات المقوم والمدمر تكون كما يلي:

```

waitingCustomerQueueType::waitingCustomerQueueType(int size)
    :queueType<customerType>(size)
{
}

waitingCustomerQueueType::~~waitingCustomerQueueType()
{
}

```

الدالة updateWaitingQueue تزيد زمن انتظار كل عميل في الطابور بمقدار وحدة زمنية واحدة. الفئة waitingCustomerQueueType مستمدة من الفئة queueType. بما أن عناصر بيانات queueType خاصة فان الدالة updateWaitingQueue لا يمكنها تناول عناصر الطابور مباشرة. الطريقة الوحيدة لتناول عناصر الطابور هي استخدام العمليات front و deletQueue. بعد زيادة زمن الانتظار يمكن اعادة العنصر الى الطابور باستخدام العملية addQueue. عملية addQueue تقوم بادخال العنصر في نهاية الطابور. اذا أجرينا العمليتين front و deleteQueue وبعدهما عملية addQueue لكل عنصر من الطابور اذن في النهاية يصبح العنصر الأول مرة أخرى العنصر الأول. باعطاء كل من العمليات front و deleteQueue وبعدهما العملية addQueue كيف نقوم بتحديد أن جميع عناصر الطابور تمت معالجتها؟ لا يمكننا استخدام العمليات isEmptyQueue أو isFullQueue على الطابور لأن الطابور لن يكون أبداً فارغاً أو ممتلئاً.

من أحد حلول هذه المشكلة عمل طابور مؤقت. يتم إزالة كل عنصر من الطابور الأصلي ومعالجته وإدخاله إلى الطابور المؤقت. عندما يصبح الطابور الأصلي فارغاً تتم معالجة جميع العناصر في الطابور. يمكننا بعد هذا نسخ العناصر من الطابور المؤقت داخل الطابور الأصلي. بالرغم من هذا فإن هذا الحل يقتضي منا استخدام مساحة ذاكرة إضافية قد تكون مؤثرة. إذا كان الطابور كبيراً إذن يكون هناك حاجة إلى مزيد من زمن الحاسوب لنسخ العناصر من الطابور المؤقت إلى الطابور الأصلي. لننظر إلى حل آخر.

قبل البدء في تحديث عناصر الطابور يمكننا إدخال عميل وهمي زمن الانتظار له -1 مثلاً. أثناء عملية التحديث وعندما نصل إلى العميل صاحب زمن الانتظار -1 يمكننا إيقاف عملية التحديث بدون معالجة هذا العميل. إذا لم نهالج العميل صاحب زمن الانتظار -1 فإن هذا العميل تتم إزالته من الطابور وبعد معالجة جميع عناصر الطابور لا يحتوي الطابور على عناصر إضافية. هذا الحل لا يقتضي منا عمل طابور مؤقت ولهذا لن نحتاج زمن حاسوبي إضافي لنسخ العناصر مرة أخرى داخل الطابور الأصلي. سوف نستخدم هذا الحل لتحديث الطابور. لهذا يكون تعريف الدالة updateWaitingQueue:

```
void waitingCustomerQueueType::updateWaitingQueue()
{
    customerType cust;

    cust.setWaitingTime(-1);
    int wTime = 0;

    addQueue(cust);

    while(wTime != -1)
    {
        cust = front();
        deleteQueue();
        wTime = cust.getWaitingTime();

        if(wTime == -1)
            break;

        cust.incrementWaitingTime();
        addQueue(cust);
    }
}
```

البرنامج الرئيسي:

لأجراء المحاكاة نحتاج أولاً إلى الحصول على المعلومات التالية:

- عدد الوحدات الزمنية التي يجب على المحاكاة أن تجري فيها. افترض أن كل وحدة زمنية قدرها دقيقة واحدة.
- عدد مقدمي الخدمة.
- مقدار الزمن الذي تستهلكه خدمة العميل – أي زمن التعامل.
- الوقت التقريبي بين وصول العملاء.

هذه المعلومات تسمى عوامل المحاكاة. عن طريق تغيير قيم هذه العوامل يمكننا ملاحظة التغيرات في أداء النظام. يمكننا كتابة دالة setSimulationParameters لحث المستخدم على تحديد هذه القيم. تعريف هذه الدالة هو:

```
void setSimulationParameters(int& sTime, int& numOfServers,
                           int& transTime,
                           int& tBetweenCArrival)
{
    cout<<"Enter the simulation time: "<<flush;
    cin>>sTime;
    cout<<endl;

    cout<<"Enter the number of servers: "<<flush;
    cin>>numOfServers;
    cout<<endl;

    cout<<"Enter the transaction time: "<<flush;
    cin>>transTime;
    cout<<endl;

    cout<<"Enter the time between customer arrivals: "<<flush;
    cin>>tBetweenCArrival;
    cout<<endl;
}
```

عندما يصبح مقدم الخدمة حر ويكون طابور العملاء غير فارغ يمكننا نقل العميل الموجود في مقدمة الطابور الى مقدم الخدمة الحر لكي تتم خدمته. فضلاً عن هذا عندما يبدأ العميل في التعامل ينتهي زمن الانتظار. يتم اضافة زمن انتظار العميل الى زمن الانتظار الكلي. الخوارزمية العامة لبدء التعامل (بافتراض أن serverID تعبر عن هوية مقدم الخدمة الحر) هي:

1. قم باستعادة وازالة العميل من مقدمة الطابور.

customer = customerQueue. Front ();

customerQueue. deleteQueue ();

2. قم بتحديث زمن الانتظار الكلي عن طريق جمع زمن انتظار العميل الحالي مع زمن الانتظار الكلي السابق.

totalWait = totalWait + customer. getWaitingTime ()

3. قم بتحديد مقدم الخدمة الحر لبدء التعامل.

serverList. setServerBusy (serverID, customer, transTime);

لاجراء المحاكاة نحتاج الى معرفة عدد العملاء الذين يصلون في وحدة زمنية معينة والزمن الذي يتم استهلاكه في خدمة العميل. اننا نستخدم توزيع بويسون من الاحصاء والذي يقول أن احتمال حدوث الأحداث y في زمن معين يتم الوصول اليها عن طريق الصيغة:

$$P(y) = \frac{\lambda^y e^{-\lambda}}{y!}, y = 0, 1, 2, \dots$$

حيث λ هي القيمة المتوقعة التي تحدث بها عدد y من الأحداث في هذا الزمن. افترض أن العميل في المتوسط يصل كل 4 دقائق. أثناء هذه الفترة المكونة من 4 دقائق يمكن أن يصل العميل في أي دقيقة من هذه الأربعة دقائق. بافتراض وجود احتمال متساوي من كل من الأربعة دقائق تكون القيمة المتوقعة التي يصل فيها العميل في كل من الأربعة دقائق هي $0.25 = 4/1$. بعد هذا نحتاج إلى تحديد ما إذا كان العميل يصل فعلياً في دقيقة محددة أم لا.

الآن هي الاحتمال بعدم حدوث أية أحداث في زمن محدد. من الافتراضات الرئيسية لتوزيع بويسون أن هناك أكثر من عميل سوف يصل في فاصل زمني قصير وهو احتمال مهمل. للبساطة نفترض $e^{-\lambda}$ ن هناك عميل واحد فقط يصل في وحدة زمنية محددة. لهذا نستخدم كنقطة قطعية لتحديد ما إذا كان العميل يصل في وحدة زمنية محددة. افترض أن العميل في المتوسط يصل كل 4 دقائق إذن $\lambda = 0.25$. يمكننا استخدام خوارزمية لتوليد عدد بين صفر وواحد. إذا كانت قيمة العدد الذي يتم إنتاجه $< e^{-0.25}$ افترض أن العميل وصل في وحدة زمنية محددة. على سبيل المثال افترض أن $rNum$ عبارة عن عدد عشوائي حيث $0 \leq rNum \leq 1$. إذا كانت $rNum > e^{-0.25}$ فالعميل وصل في الوحدة الزمنية المحددة.

الآن نوضح الدالة `runSimulation` لتطبيق المحاكاة. افترض أننا نجري المحاكاة لمائة وحدة زمنية والعملاء يصلون في وحدات زمنية 93، و96، و100. متوسط زمن التعامل 5 دقائق – أي 5 وحدات زمنية. من أجل التبسيط افترض أن لدينا مقدم خدمة واحد فقط ويصبح غير مشغول عند الوحدة الزمنية 97 وأن جميع العملاء الذين يصلون قبل الوحدة الزمنية 93 تمت خدمتهم. عندما يصبح مقدم الخدمة غير مشغول في الوحدة الزمنية 97 يبدأ العميل الذي وصل في الوحدة الزمنية 93 في التعامل. بما أن تعامل العميل الذي يصل في الوحدة الزمنية 93 يبدأ في الوحدة الزمنية 97 وإكمال التعامل يأخذ 5 دقائق إذن عندما تنتهي حلقة المحاكاة لا يزال هذا العميل عند مقدم الخدمة. فضلاً عن هذا فإن العميل الذي يصل في الوحدات الزمنية 96 و100 يكون في الطابور. من أجل التبسيط نفترض أنه عندما تنتهي حلقة المحاكاة يتم اعتبار العملاء الموجودين لدى مقدمي الخدمة أنه تمت خدمتهم. الخوارزمية العامة لهذه الدالة هي:

1. قم بإعلان وتهيئة المتغيرات مثل عوامل المحاكاة، ورقم العميل، والساعة، وأزمنة الانتظار الكلية والمتوسطة، وعدد العملاء الذين وصلوا، وعدد العملاء الذين تمت خدمتهم، وعدد العملاء الذين تم تركهم في طابور الانتظار، وعدد العملاء الذين تم تركهم مع مقدمي الخدمة `waitingCustomerQueue`، وقائمة مقدمي الخدمة.

2. الحلقة الأساسية هي:

```
for(clock = 1; clock <= simulationTime; clock++)
{
```

أ- قم بتحديث قائمة مقدم الخدمة لتقليل زمن تعامل كل مقدم خدمة مشغول بمقدار وحدة زمنية واحدة.

ب- إذا لم يكن طابور العملاء فارغاً قم بزيادة زمن انتظار كل عميل بمقدار وحدة زمنية واحدة.

ت- إذا وصل عميل قم بزيادة عدد العملاء بمقدار 1 وأضف العميل الجديد الى الطابور.

ث- إذا كان مقدم الخدمة غير مشغول وكان طابور العملاء غير فارغ قم بإزالة العميل من مقدمة الطابور وارسله الى مقدم الخدمة الحر.

{

3. قم بطباعة النتائج المناسبة ويجب أن تتضمن نتائجك عدد العملاء المتروكين في الطابور، وعدد العملاء الذين لا يزالون مع مقدمي الخدمة، وعدد العملاء الذين وصلوا، وعدد العملاء الذين أكملوا التعامل بالفعل.

بمجرد تصميمك للدالة runSimulation يكون تعريف الدالة main بسيط وواضح لأن الدالة main تستدعي الدالة runSimulation فقط. (انظر تمرين البرمجة 6 في نهاية هذا الفصل).
عندما قمنا باختبار نسختنا من برنامج المحاكاة قمنا بتقديم النتائج التالية. لقد افترضنا أن متوسط زمن التعامل هو 5 دقائق وأن العميل يصل كل 4 دقائق في المتوسط. لقد استخدمنا مولد أرقام عشوائي لانتاج عدد بين 0 و 1 لايجاد ما اذا كان العميل وصل في وحدة زمنية وحددة أم لا.

تنفيذات العينة:

تنفيذ العينة 1:

العميل رقم 1 وصل في الوحدة الزمنية 4

العميل رقم 2 وصل في الوحدة الزمنية 8

مقدم الخدمة رقم: 1 عميل رقم 1 رحل في الوحدة الزمنية 9

العميل رقم 3 وصل في الوحدة الزمنية 9

العميل رقم 4 وصل في الوحدة الزمنية 12

مقدم الخدمة رقم: 1 عميل رقم 2 رحل في الوحدة الزمنية 14

مقدم الخدمة رقم: 1 عميل رقم 3 رحل في الوحدة الزمنية 19

العميل رقم 5 وصل في الوحدة الزمنية 21

مقدم الخدمة رقم: 1 عميل رقم 5 رحل في الوحدة الزمنية 29

العميل رقم 6 وصل في الوحدة الزمنية 37

العميل رقم 7 وصل في الوحدة الزمنية 38

العميل رقم 8 وصل في الوحدة الزمنية 41

مقدم الخدمة رقم: 1 عميل رقم 6 رحل في الوحدة الزمنية 42

العميل رقم 9 وصل في الوحدة الزمنية 43
العميل رقم 10 وصل في الوحدة الزمنية 44
مقدم الخدمة رقم: 1 عميل رقم 7 رحل في الوحدة الزمنية 47
العميل رقم 11 وصل في الوحدة الزمنية 49
العميل رقم 12 وصل في الوحدة الزمنية 51
مقدم الخدمة رقم: 1 عميل رقم 8 رحل في الوحدة الزمنية 52
العميل رقم 13 وصل في الوحدة الزمنية 52
العميل رقم 14 وصل في الوحدة الزمنية 53
العميل رقم 15 وصل في الوحدة الزمنية 54
مقدم الخدمة رقم: 1 عميل رقم 9 رحل في الوحدة الزمنية 57
العميل رقم 16 وصل في الوحدة الزمنية 59
مقدم الخدمة رقم: 1 عميل رقم 10 رحل في الوحدة الزمنية 62
العميل رقم 17 وصل في الوحدة الزمنية 66
مقدم الخدمة رقم: 1 عميل رقم 11 رحل في الوحدة الزمنية 67
العميل رقم 18 وصل في الوحدة الزمنية 71
مقدم الخدمة رقم: 1 عميل رقم 12 رحل في الوحدة الزمنية 72
مقدم الخدمة رقم: 1 عميل رقم 13 رحل في الوحدة الزمنية 77
العميل رقم 19 وصل في الوحدة الزمنية 78
مقدم الخدمة رقم: 1 عميل رقم 14 رحل في الوحدة الزمنية 82
مقدم الخدمة رقم: 1 عميل رقم 15 رحل في الوحدة الزمنية 87
العميل رقم 20 وصل في الوحدة الزمنية 90
مقدم الخدمة رقم: 1 عميل رقم 16 رحل في الوحدة الزمنية 92
العميل رقم 21 وصل في الوحدة الزمنية 92
مقدم الخدمة رقم: 1 عميل رقم 17 رحل في الوحدة الزمنية 97
المحاكاة جرت لمدة 100 وحدة زمنية

عدد مقدمي الخدمة: 1

متوسط زمن التعامل: 5

متوسط فرق زمن الوصول بين العملاء: 4

زمن الانتظار الكلي: 269

عدد العملاء الذين أنهوا التعامل: 17

عدد العملاء الذين تم تركهم عند مقدمي الخدمة: 1

عدد العملاء الذين تم تركهم في الطابور: 3

متوسط زمن الانتظار: 12.8!

*****انهاء المحاكاة*****

تنفيذ العينة 2:

- العميل رقم 1 وصل في الوحدة الزمنية 4
- العميل رقم 2 وصل في الوحدة الزمنية 8
- مقدم الخدمة رقم: 1 عميل رقم 1 رحل في الوحدة الزمنية 9
- العميل رقم 3 وصل في الوحدة الزمنية 9
- العميل رقم 4 وصل في الوحدة الزمنية 12
- مقدم الخدمة رقم: 1 عميل رقم 2 رحل في الوحدة الزمنية 13
- مقدم الخدمة رقم: 1 عميل رقم 3 رحل في الوحدة الزمنية 14
- مقدم الخدمة رقم: 1 عميل رقم 4 رحل في الوحدة الزمنية 18
- العميل رقم 5 وصل في الوحدة الزمنية 21
- مقدم الخدمة رقم: 1 عميل رقم 5 رحل في الوحدة الزمنية 26
- العميل رقم 6 وصل في الوحدة الزمنية 37
- العميل رقم 7 وصل في الوحدة الزمنية 38
- العميل رقم 8 وصل في الوحدة الزمنية 41
- مقدم الخدمة رقم: 1 عميل رقم 6 رحل في الوحدة الزمنية 42
- مقدم الخدمة رقم: 1 عميل رقم 7 رحل في الوحدة الزمنية 43
- العميل رقم 9 وصل في الوحدة الزمنية 43
- العميل رقم 10 وصل في الوحدة الزمنية 44
- مقدم الخدمة رقم: 1 عميل رقم 8 رحل في الوحدة الزمنية 47
- مقدم الخدمة رقم: 2 عميل رقم 9 رحل في الوحدة الزمنية 48
- العميل رقم 11 وصل في الوحدة الزمنية 49
- العميل رقم 12 وصل في الوحدة الزمنية 51
- مقدم الخدمة رقم: 1 عميل رقم 10 رحل في الوحدة الزمنية 52
- العميل رقم 13 وصل في الوحدة الزمنية 52
- العميل رقم 14 وصل في الوحدة الزمنية 53
- مقدم الخدمة رقم: 2 عميل رقم 11 رحل في الوحدة الزمنية 54
- العميل رقم 15 وصل في الوحدة الزمنية 54

مقدم الخدمة رقم: 1 عميل رقم 12 رحل في الوحدة الزمنية 57
مقدم الخدمة رقم: 2 عميل رقم 13 رحل في الوحدة الزمنية 59
العميل رقم 16 وصل في الوحدة الزمنية 59
مقدم الخدمة رقم: 1 عميل رقم 14 رحل في الوحدة الزمنية 62
مقدم الخدمة رقم: 2 عميل رقم 15 رحل في الوحدة الزمنية 64
العميل رقم 17 وصل في الوحدة الزمنية 66

مقدم الخدمة رقم: 1 عميل رقم 16 رحل في الوحدة الزمنية 67
مقدم الخدمة رقم: 2 عميل رقم 17 رحل في الوحدة الزمنية 71
العميل رقم 18 وصل في الوحدة الزمنية 71
مقدم الخدمة رقم: 1 عميل رقم 18 رحل في الوحدة الزمنية 76
العميل رقم 19 وصل في الوحدة الزمنية 78
مقدم الخدمة رقم: 1 عميل رقم 19 رحل في الوحدة الزمنية 83
العميل رقم 20 وصل في الوحدة الزمنية 90
العميل رقم 21 وصل في الوحدة الزمنية 92
مقدم الخدمة رقم: 1 عميل رقم 20 رحل في الوحدة الزمنية 95
مقدم الخدمة رقم: 2 عميل رقم 21 رحل في الوحدة الزمنية 97
المحاكاة جرت لمدة 100 وحدة زمنية

عدد مقدمي الخدمة: 2

متوسط زمن التعامل: 5

متوسط فرق زمن الوصول بين العملاء: 4

زمن الانتظار الكلي: 20

عدد العملاء الذين أنهوا التعامل: 21

عدد العملاء الذين تم تركهم عند مقدمي الخدمة: 0

عدد العملاء الذين تم تركهم في الطابور: 0

متوسط زمن الانتظار: 0.95

***** انتهاء المحاكاة *****

تنفيذ العينة 3: (في هذه المخرجات يتم حذف تفاصيل مخرجات أزمنة وصول ورحيل العملاء لحفظ المساحة).

العميل رقم 1 وصل في الوحدة الزمنية 4

العميل رقم 2 وصل في الوحدة الزمنية 8
مقدم الخدمة رقم: 1 عميل رقم 1 رحل في الوحدة الزمنية 9
العميل رقم 3 وصل في الوحدة الزمنية 9
العميل رقم 4 وصل في الوحدة الزمنية 12
مقدم الخدمة رقم: 1 عميل رقم 2 رحل في الوحدة الزمنية 14
مقدم الخدمة رقم: 1 عميل رقم 3 رحل في الوحدة الزمنية 19
العميل رقم 5 وصل في الوحدة الزمنية 21
مقدم الخدمة رقم: 1 عميل رقم 4 رحل في الوحدة الزمنية 24
مقدم الخدمة رقم: 1 عميل رقم 5 رحل في الوحدة الزمنية 29
العميل رقم 6 وصل في الوحدة الزمنية 37
العميل رقم 7 وصل في الوحدة الزمنية 38
العميل رقم 8 وصل في الوحدة الزمنية 41

مقدم الخدمة رقم: 1 عميل رقم 6 رحل في الوحدة الزمنية 42
العميل رقم 9 وصل في الوحدة الزمنية 43
العميل رقم 10 وصل في الوحدة الزمنية 44

...

المحاكاة جرت لمدة 100 وحدة زمنية

عدد مقدمي الخدمة: 1

متوسط زمن التعامل: 5

متوسط فرق زمن الوصول بين العملاء: 4

زمن الانتظار الكلي: 8008

عدد العملاء الذين أنهوا التعامل: 197

عدد العملاء الذين تم تركهم عند مقدمي الخدمة: 1

عدد العملاء الذين تم تركهم في الطابور: 15

متوسط زمن الانتظار: 37.60

*****انتهاء المحاكاة*****

تنفيذ العينة 4: (في هذه المخرجات يتم حذف تفاصيل مخرجات أزمنة وصول ورحيل العملاء لحفظ المساحة).

العميل رقم 1 وصل في الوحدة الزمنية 4

العميل رقم 2 وصل في الوحدة الزمنية 8

مقدم الخدمة رقم: 1 عميل رقم 1 رحل في الوحدة الزمنية 9
العميل رقم 3 وصل في الوحدة الزمنية 9
العميل رقم 4 وصل في الوحدة الزمنية 12
مقدم الخدمة رقم: 2 عميل رقم 2 رحل في الوحدة الزمنية 13
مقدم الخدمة رقم: 1 عميل رقم 3 رحل في الوحدة الزمنية 14
مقدم الخدمة رقم: 3 عميل رقم 4 رحل في الوحدة الزمنية 17
العميل رقم 5 وصل في الوحدة الزمنية 21
مقدم الخدمة رقم: 1 عميل رقم 5 رحل في الوحدة الزمنية 26
العميل رقم 6 وصل في الوحدة الزمنية 37
العميل رقم 7 وصل في الوحدة الزمنية 38
العميل رقم 8 وصل في الوحدة الزمنية 41
مقدم الخدمة رقم: 1 عميل رقم 6 رحل في الوحدة الزمنية 42
مقدم الخدمة رقم: 2 عميل رقم 7 رحل في الوحدة الزمنية 43
العميل رقم 9 وصل في الوحدة الزمنية 43
العميل رقم 10 وصل في الوحدة الزمنية 44
مقدم الخدمة رقم: 3 عميل رقم 8 رحل في الوحدة الزمنية 46
...

المحاكاة جرت لمدة 100 وحدة زمنية

عدد مقدمي الخدمة: 3

متوسط زمن التعامل: 5

متوسط فرق زمن الوصول بين العملاء: 4

زمن الانتظار الكلي: 13

عدد العملاء الذين أنهوا التعامل: 212

عدد العملاء الذين تم تركهم عند مقدمي الخدمة: 1

عدد العملاء الذين تم تركهم في الطابور: 0

متوسط زمن الانتظار: 0.06

***** انتهاء المحاكاة *****

مراجعة سريعة

1. الطابور عبارة عن بنية بيانات يتم فيها اضافة العناصر في طرف واحد ويتم ازلتها من الطرف الآخر.
2. الطابور عبارة عن بنية بيانات الداخل أولاً يخرج أولاً .
3. العمليات الرئيسية على الطابور هي تهيئة الطابور، وتدمير الطابور، والتحقق مما اذا كان الطابور فارغ، والتحقق مما اذا كان الطابور ممتلئ، واطافة عنصر الى الطابور، وازالة عنصر من الطابور.
4. يمكن تطبيق الطابور اما كمصفوفة أو كقائمة متصلة.
5. لا يجب تناول العناصر الموجودة بمنصف الطابور مباشرة .
6. اذا كان الطابور غير فارغ فان الدالة front تنتج العنصر الأول من الطابور والدالة back تنتج العنصر الأخير من الطابور.
7. الطوابير عبارة عن صور محدودة من المصفوفات والقوائم المتصلة.

تمارين

1. افترض أن الطابور عبارة عن هدف queueType وحجم المصفوفة التي تطبق الطابور 100 وافترض كذلك أن قيمة queueFront هي 50 وقيمة queueRear هي 99؟
أ- ما هي قيم queueFront وqueueRear بعد اضافة عنصر الى الطابور؟
ب- ما هي قيم queueFront وqueueRear بعد ازالة عنصر من الطابور؟
2. افترض أن الطابور عبارة عن هدف queueType وحجم المصفوفة التي تطبق الطابور 100 وافترض كذلك أن قيمة queueFront هي 99 وقيمة queueRear هي 25؟
أ- ما هي قيم queueFront وqueueRear بعد اضافة عنصر الى الطابور؟
ب- ما هي قيم queueFront وqueueRear بعد ازالة عنصر من الطابور؟
3. افترض أن الطابور عبارة عن هدف queueType وحجم المصفوفة التي تطبق الطابور 100 وافترض كذلك أن قيمة queueFront هي 225 وقيمة queueRear هي 75؟
أ- ما هي قيم queueFront وqueueRear بعد اضافة عنصر الى الطابور؟
ب- ما هي قيم queueFront وqueueRear بعد ازالة عنصر من الطابور؟
4. افترض أن الطابور عبارة عن هدف queueType وحجم المصفوفة التي تطبق الطابور 100 وافترض كذلك أن قيمة queueFront هي 99 وقيمة queueRear هي 25؟
أ- ما هي قيم queueFront وqueueRear بعد اضافة عنصر الى الطابور؟
ب- ما هي قيم queueFront وqueueRear بعد ازالة عنصر من الطابور؟

5. افترض أن الطابور يتم تطبيقه كمصفوفة ذات المقطع الخاص المحفوظ كما تم توضيحه في هذا الفصل. وافترض كذلك أن حجم المصفوفة التي تطبق الطابور 100. اذا كانت قيمة queueFront هي 50 فما هو موضع أول عنصر بالطابور؟

6. افترض أن الطابور يتم تطبيقه كمصفوفة ذات المقطع الخاص المحفوظ كما تم توضيحه في هذا الفصل. وافترض كذلك أن حجم المصفوفة التي تطبق الطابور 100. اذا كانت قيمة queueFront هي 74 وقيمة queueRear هي 99.

ت- ما هي قيم queueFront و queueRear بعد اضافة عنصر الى الطابور؟

ث- ما هي قيم queueFront و queueRear بعد ازالة عنصر من الطابور؟ ما هو وضع عنصر الطابور المزال؟

7. انظر الى البيانات التالية:

```
queueType<int> queue;  
int x, y;
```

وضح ما هي مخرجات الجزء التالي من الكود.

```
queue.initializeQueue();  
x = 4;  
y = 5;  
queue.addQueue(x);  
queue.addQueue(y);  
x = queue.front();  
queue.deleteQueue();  
queue.addQueue(x + 5);  
queue.addQueue(16);  
queue.addQueue(x);  
queue.addQueue(y - 3);
```

```
cout<<"Queue Elements: ";
```

```
while(!queue.isEmptyQueue())  
{  
    cout<<" "<<queue.front();  
    queue.deleteQueue();  
}
```

```
cout<<endl;
```

8. انظر الى البيانات التالية:

```
stackType<int> stack;  
queueType<int> queue;  
int x;
```

افترض أن المخرجات هي:

15 28 14 22 64 35 19 32 7 11 13 30 -999

وضح ما هي مخرجات الجزء التالي من الكود.

```

stack.initializeStack();
queue.initializeQueue();
stack.push(0);
queue.addQueue(0);
cin>>x;

while(x != -999)
{
    switch(x % 4)
    {
        case 0: stack.push(x);
                break;
        case 1: if(!stack.isEmptyStack())
                {
                    cout<<"Stack Element = "<<stack.top()<<endl;
                    stack.pop();
                }
                else
                    cout<<"Sorry, the stack is empty."<<endl;
                break;
        case 2: queue.addQueue(x);
                break;
        case 3: if(!queue.isEmptyQueue())
                {
                    cout<<"Queue Element = "<<queue.front()<<endl;
                    queue.deleteQueue();
                }
                else
                    cout<<"Sorry, the queue is empty."<<endl;
                break;
    }
    cin>>x;
}

cout<<"Stack Elements: ";
while(!stack.isEmptyStack())
{
    cout<<stack.top()<<" ";
    stack.pop();
}

cout<<endl;
cout<<"Queue Elements: ";
while(!queue.isEmptyQueue())
{
    cout<<queue.front()<<" ";
    queue.deleteQueue();
}

cout<<endl;

```

9. ما هي مخرجات البرنامج التالي؟ (في كود C++ يكون الطابور فئة مكتبة القالب المعياري queue).

```
queue<int> q;
int x, y;

x = 2;
q.push(8);
q.push(x + 3);
y = q.front();
q.push(2 * y);
q.pop();
x = q.front();
q.push(q.front());

while(!q.empty())
{
    cout<<q.front()<<" ";
    q.pop();
}

cout<<endl;
```

10. ماذا تفعل الدالة التالية؟

```
void mystery(queue<int>& q)
{
    stack<int> s;

    while(!q.empty())
    {
        s.push(q.front());
        q.pop();
    }

    while(!s.empty())
    {
        q.push(2 * s.top());
        s.pop();
    }
}
```

11. اكتب تعريف قالب الدالة moveNthFront التي تتخذ من الطابور والعديد الصحيح الموجب n عوامل لها. الدالة تحرك العنصر رقم n من الصف الى المقدمة. ترتيب العناصر المتبقية يظل كما هو. على سبيل المثال افترض أن

queue = {5, 11, 34, 67, 43, 55}

و $n = 3$.

بعد استدعاء الدالة moveNthFront تكون:

queue = {34, 5, 11, 67, 43, 55}

12. اكتب قالب الدالة reverseStack التي تتخذ من هدف المرصوفة وهدف الطابور التي تكون عناصرها من نفس النوع عوامل لها. الدالة reverseStack تستخدم الطابور لعكس عناصر المرصوفة.

13. اكتب قالب الدالة reverseQueue التي من هدف المرصوفة وهدف الطابور التي تكون عناصرها من نفس النوع عوامل لها. الدالة reverseQueue تستخدم المرصوفة لعكس عناصر الطابور.

14. أضف العملية queueCount الى الفئة queueType (تطبيق المصفوفة للطوابير) التي تنتج عدد العناصر في الطابور. اكتب تعريف قالب الدالة لتطبيق هذه العملية.

تمارين البرمجة

1. اكتب تعريفات الدالات لاثقال معامل الاسناد ومقوم النسخ للفئة queueType واكتب كذلك برنامج لاختبار هذه العمليات.

2. اكتب تعريفات الدالات لاثقال معامل الاسناد ومقوم النسخ للفئة linkedqueueType واكتب كذلك برنامج لاختبار هذه العمليات.

3. تطبيق المصفوفة للطوابير المعطى في هذا الفصل يستخدم مقطع المصفوفة – مقطع محفوظ – للتمييز بين طابور فارغ وطابور ممتلئ. اكتب تطبيق المصفوفة للطوابير كنوع بيانات مجرد حيث يتم استخدام المتغير count لمتتبع عدد العناصر في الطابور. اكتب تعريفات عناصر الدالة لتصميم هذا الطابور. واكتب كذلك برنامج لاختبار عمليات متنوعة على الطابور.

4. اكتب برنامج يقرأ صف من النص ويغير كل حرف كبير الى حرف صغير ويضع كل حرف في طابور وفي مرصوفة. البرنامج يجب أن يحدد بعد هذا ما اذا كان خط النص يقرأ من الناحيتين أم لا.

5. تطبيق الطابور في مصفوفة كما هو موضح في هذا الفصل يستخدم المتغير count لتحديد ما اذا كان الطابور فارغ أو ممتلئ. يمكنك أيضاً استخدام المتغير count لانتاج عدد العناصر في الطابور.

انظر تمرين رقم 8 من هذا الفصل. من الجهة الأخرى لا تقوم الفئة linkedQueueType باستخدام هذا المتغير لتحديد عدد العناصر في الطابور. أعد تعريف الفئة linkedQueueType عن طريق

اضافة المتغير count لمتتبع عدد العناصر في الطابور. قم بتعديل تعريفات الدالات addQueue وdeleteQueue حسب الحاجة. وقم كذلك باضافة الدالة queueCount لانتاج عدد العناصر في

الطابور وبالإضافة الى هذا اكتب برنامج لاختبار عمليات متنوعة على الفئة التي عرفت.

6. أ. اكتب تعريفات الدالات setWaitingTime وgetArrivalTime، و

getTransactionTime وgetCustomerNumber من الفئة customerType التي تم

تعريفها في القسم "استخدام الطوابير: المحاكاة".

ب. اكتب تعريفات الدالات `getRemainingTransactionTime`، `setCurrentCustomer`،
و `getCurrentCustomerNumber`، و `getCurrentCustomerArrivalTime`،
و `getCurrentCustomerWaitingTime`، و `getCurrentCustomerTransactionTime` من
الفئة `serverType` التي تم تعريفها في القسم "استخدام الطوابير: المحاكاة".

ج. اكتب تعريف الدالة `runSimulation` لاكمال تصميم برنامج محاكاة الحاسوب في القسم "استخدام
الطوابير: المحاكاة". اختبر تشغيل برنامجك لمجموعة متنوعة من البيانات. فضلاً عن هذا استخدم
مولد عدد عشوائي لتحديد ما اذا كان العميل وصل في وحدة زمنية محددة.

7. أعد عمل برنامج المحاكاة لهذا الفصل حتى يستخدم فئة مكتبة القالب المعياري `queue` للاحتفاظ
بقائمة العملاء المنتظرين.

الفصل

9

خوارزميات البحث

في هذا الفصل سوف:

- تتعلم خوارزميات البحث المتنوعة.
- تستكشف كيفية تطبيق خوارزميات البحث التتبعي والثنائي.
- تصبح على علم بالحدود الأدنى على خوارزميات البحث القائمة على المقارنة.
- تتعلم المزج.

الفصل 3 وصف كيفية تنظيم البيانات داخل ذاكرة الحاسوب باستخدام المصفوفة وكيفية أداء العمليات الأساسية على هذه البيانات. الفصل 5 قام بعد ذلك بوصف كيفية تنظيم البيانات باستخدام القوائم المتصلة. أهم عملية يتم إجراؤها على القائمة هي خوارزمية البحث. باستخدام خوارزمية البحث يمكنك:

- تحديد ما إذا كان هناك عنصر محدد في القائمة.
- إذا كانت البيانات منظمة بوجه خاص (على سبيل المثال مفروزة) إيجاد الموقع في القائمة حيث يمكن إدخال عنصر جديد.
- إيجاد موقع عنصر يتم حذفه.

لهذا يكون أداء خوارزمية البحث هام للغاية. إذا كان البحث بطيئاً فإنه يأخذ قدر كبير من زمن الحاسوب لإنجاز مهمتك وإذا كان البحث سريعاً يمكنك إنجاز مهمتك بسرعة.

خوارزميات البحث:

الفصول 3 و5 قاموا بوصف كيفية تطبيق خوارزمية البحث التتبعي. هذا الفصل يناقش خوارزميات بحث أخرى ويقوم بتحليل الخوارزميات. تحليل الخوارزميات يجعل المبرمج قادراً على تحديد الخوارزمية التي يتم استخدامها من أجل تطبيق محدد. قبل وصف هذه الخوارزميات لنقم بعمل الملاحظات التالية.

هناك عنصر خاص مرتبط بكل عنصر في مجموعة البيانات يقوم بتحديد العنصر في مجموعة البيانات. على سبيل المثال إذا كان لديك مجموعة بيانات مكونة من سجلات الطالب فإن هوية الطالب تحدد كل طالب في مدرسة معينة. هذا العنصر الفريد من العنصر يسمى **مفتاح العنصر**. مفاتيح العنصر في مجموعة البيانات يتم استخدامها في عمليات مثل البحث والفرز والإدخال والحذف. على

سبيل المثال عندما نقوم ببحث مجموعة البيانات من أجل عنصر محدد نقوم بمقارنة مفتاح العنصر الذي نبحث عنه مع مفاتيح العناصر في مجموعة البيانات.

كما لوحظ من قبل يقوم هذا الفصل بالاضافة الى وصف خوارزميات البحث والفرز بتحليل هذه الخوارزميات. في تحليل الخوارزمية تشير مقارنات المفاتيح الى مقارنة مفتاح عنصر البحث مع مفتاح عنصر في القائمة. فضلاً عن هذا يشير عدد مقارنات المفتاح الى عدد المرات التي يتم بها مقارنة مفتاح العنصر (في خوارزميات مثل البحث والفرز) مع مفاتيح العناصر في القائمة.

في الفصل 3 قمنا بتصميم وتطبيق الفئة `arrayListType` لتطبيق القائمة والعمليات الرئيسية عليها. بما أن هذا الفصل يشير الى هذه الفئة فاننا نعطي تعريفها هنا من أجل سهولة الاشارة:

```
template<class elemType>
class arrayListType
{
public:
    const arrayListType<elemType>&
        operator=(const arrayListType<elemType>&);
    //Overload the assignment operator.

    bool isEmpty();
    //Function to determine whether the list is empty.
    //Postcondition: Returns true if the list is empty;
    //                otherwise, returns false.
    bool isFull();
    //Function to determine whether the list is full.
    //Postcondition: Returns true if the list is full;
    //                otherwise, returns false.
    int listSize();
    //Function to determine the number of elements in list.
    //Postcondition: Returns the value of the length.
    int maxListSize();
    //Function to determine the size of the list.
    //Postcondition: Returns the value of maxSize.
```

```

void print() const;
    //Function to output the elements of the list.
    //Postcondition: Elements of the list are output on the
    //                standard output device.
bool isItemAtEqual(int location, const elemType& item);
    //Function to determine whether item is the same as the
    //item in the list at the position specified by location.
    //Postcondition: Returns true if list[location] is the
    //                same as item; otherwise, returns false.
void insertAt(int location, const elemType& insertItem);
    //Function to insert an item in the list at the
    //location specified by location. The item to be inserted
    //and the list are passed as parameters to the function.
    //Postcondition: Starting at location, the elements of the
    //                list are shifted down, list[location] =
    //                insertItem; and length++
    //    If the list is full or location is out of range,
    //    an appropriate message is displayed.
void insertEnd(const elemType& insertItem);
    //Function to insert an item at the end of the list.
    //The parameter insertItem specifies the item to be inserted.
    //Postcondition: list[length] = insertItem; and length++
    //    If the list is full, an appropriate message is
    //    displayed.
void removeAt(int location);
    //Function to remove the item from the list at the position
    //specified by location.
    //Postcondition: The list element at list[location] is removed
    //    and the length is decremented by 1.
    //    If location is out of range, an appropriate message
    //    is displayed.
void retrieveAt(int location, elemType& retItem);
    //Function to retrieve the element from the list at the
    //position specified by location.
    //Postcondition: retItem = list[location]
    //    If location is out of range, an appropriate message
    //    is displayed.
void replaceAt(int location, const elemType& repItem);
    //Function to replace the element in the list at the
    //position specified by location. The item to be replaced
    //is specified by the parameter repItem.
    //Postcondition: list[location] = repItem
    //    If location is out of range, an appropriate message
    //    is displayed.
void clearList();
    //Function to remove all the elements from the list.
    //After this operation, the size of the list is zero.
    //Postcondition: length = 0

```

```

int seqSearch(const elemType& item);
//Function to search the list for a given item.
//Postcondition: If item is found, returns the location
//                in the array where item is found;
//                otherwise, returns -1.
void insert(const elemType& insertItem);
//Function to insert the item specified by the parameter
//insertItem at the end of the list. However, first the
//list is searched to see whether the item to be inserted
//is already in the list.
//Postcondition: list[length] = insertItem; and length++
//    If the item is already in the list or the list is
//    full, an appropriate message is displayed.
void remove(const elemType& removeItem);
//Function to remove an item from the list. The parameter
//removeItem specifies the item to be removed.
//Postcondition: If removeItem is found in the list, it is
//                removed from the list and length is
//                decremented by one.

arrayListType(int size = 100);
//constructor
//Creates an array of the size specified by the parameter
//size. The default array size is 100.
//Postcondition: list points to the array; length = 0;
//                and maxSize = size

arrayListType(const arrayListType<elemType>& otherList);
//copy constructor

~arrayListType();
//destructor
//Deallocates the memory occupied by the array.

protected:
    elemType *list;        //array to hold the list elements
    int length;            //variable to store the length of the list
    int maxSize;           //variable to store the maximum size of
                           //the list
};

```

البحث التتابعي:

تم توضيح البحث التتابعي (المسمى البحث الطولي كذلك) في القوائم المبنية على المصفوفات في الفصل 3 وتمت تغطية البحث التتابعي على القوائم المتصلة في الفصل 5. البحث التتابعي يعمل كما يعمل في كل من القوائم المبنية على المصفوفات والقوائم المتصلة دائماً يبدأ البحث من العنصر الأول في القائمة ويستمر حتى يتم العثور على العنصر في القائمة أو حتى يتم بحث القائمة بأكملها.

بما أننا مهتمون بأداء البحث التتابعي (أي تحليل هذا النوع من البحث) فأننا نقوم بعد هذا باعطاء خوارزمية البحث التتابعي للقوائم المبنية على المصفوفات (كما تم وصفها في الفصل 3). إذا تم العثور

```

template<class elemType>
int arrayListType<elemType>::seqSearch(const elemType& item)
{
    int loc;
    bool found = false;

    for(loc = 0; loc < length; loc++)
        if(list[loc] == item)
        {
            found = true;
            break;
        }

    if(found)
        return loc;
    else
        return -1;
} //end seqSearch

```

على عنصر البحث يتم انتاج مؤشره (أي موقعه في المصفوفة). إذا لم يكن البحث ناجحاً يتم انتاج -1. لاحظ أن البحث التتابعي التالي لا يتطلب أن تكون عناصر القائمة مرتبة ترتيباً محدداً.

خوارزمية البحث التتابعي المعطاة هنا تستخدم بنية تحكم تكرارية (حلقة for) للمقارنة بين عنصر البحث وعناصر القائمة. كما يمكنك كتابة خوارزمية تكرارية لتطبيق خوارزمية البحث التتابعي. (انظر تمرين البرمجة 1 في نهاية هذا الفصل).

تحليل البحث التتابعي:

يقوم هذا القسم بتحليل أداء خوارزمية البحث التتابعي في كل من الحالة الأسوأ والحالة المتوسطة. يتم تنفيذ البيانات الموجودة قبل وبعد الحلقة مرة واحدة فقط ومن هنا تحتاج الى زمن قليل من الحاسوب. البيانات الموجودة في الحلقة for هي البيانات التي يتم تكرارها عدة مرات. بالنسبة الى كل تكرار للحلقة تتم مقارنة عنصر البحث مع عنصر في القائمة ويتم تنفيذ القليل من البيانات الأخرى بما فيها بعض المقارنات الأخرى. بوضوح تتوقف الحلقة بمجرد أن يتم العثور على عنصر البحث في القائمة. لهذا يرتبط تنفيذ البيانات الأخرى في الحلقة بشكل مباشر بنتيجة مقارنة المفتاح. كما قد يقوم المبرمجون المختلفون بتطبيق نفس الخوارزمية بطريقة مختلفة بالرغم من أن عدد مقارنات المفاتيح سوف يكون واحد عادةً. سرعة الحاسوب يمكنها أن تؤثر بسهولة على الزمن الذي تأخذه الخوارزمية للأداء ولكن ليس عدد مقارنات المفتاح.

لهذا عند تحليل خوارزمية البحث نقوم بعد مقارنات المفتاح لأن هذا العدد يعطينا أكثر المعلومات فائدة. فضلاً عن هذا يمكن تطبيق معيار احصاء عدد مقارنات المفاتيح بشكل جيد ومتساوي على خوارزميات بحث أخرى.

افترض أن طول القائمة L هو n ونريد تحديد عدد مقارنات المفتاح التي يتم عملها بواسطة البحث التتابعي عندما يتم بحث القائمة L من أجل عنصر محدد.

إذا لم يكن عنصر البحث موجوداً في القائمة فإننا نقوم بمقارنة عنصر البحث مع كل عنصر في القائمة صانعين عدد n من المقارنات. هذه الحالة حالة غير ناجحة.

افترض أن عنصر البحث موجود في القائمة. إذن يعتمد عدد مقارنات المفتاح على مكان عنصر البحث في القائمة. إذا كان عنصر البحث هو العنصر الأول من القائمة L فإننا نقوم بعمل مقارنة واحدة وهذه هي أفضل حالة. على الجانب الآخر إذا كان عنصر البحث هو آخر عنصر في القائمة فإن الخوارزمية تصنع عدد n من المقارنات وتكون هذه هي الحالة الأسوأ. من غير المحتمل أن تحدث الحالات الأفضل والأسوأ في كل مرة نقوم فيها بتطبيق البحث التتابعي على القائمة L لذلك سوف يكون من الأكثر نفعاً إذا استطعنا تحديد السلوك المتوسط للخوارزمية. هذا يعني أننا نحتاج إلى تحديد متوسط عدد مقارنات المفتاح التي تقوم بها خوارزمية البحث في الحالة الناجحة.

من أجل تحديد متوسط عدد المقارنات في الحالة الناجحة من خوارزمية البحث التتابعي:

1. التفكير في جميع الحالات المحتملة.

2. إيجاد عدد المقارنات بالنسبة إلى كل حالة.

3. جمع عدد المقارنات وقسمتها على عدد الحالات.

إذا كان عنصر البحث المسمى الهدف هو أول عنصر في القائمة فإنه سوف يكون هناك حاجة إلى مقارنة واحدة فقط وإذا كان الهدف هو العنصر الثاني من القائمة فيكون هناك حاجة إلى مقارنتين. بالمثل إذا كان الهدف هو العنصر رقم k فب القائمة k فيكون هناك حاجة إلى عدد k من المقارنات. نفترض أن العنصر يمكن أن يكون أي عنصر في القائمة أي أن هناك احتمال متساوي بين جميع عناصر القائمة أن تكون الهدف. افترض أن هناك عدد n من العناصر في القائمة. التعبير التالي يعطي متوسط عدد المقارنات:

$$\frac{1 + 2 + \dots + n}{n}$$

من المعروف أن:

$$1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

لهذا فإن التعبير التالي يعطي متوسط عدد المقارنات التي يصنعها البحث التتابعي في الحالة الناجحة:

هذا لأن $\frac{1 + 2 + \dots + n}{n} = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$ نابعي يبحث نصف القائمة وينتج من هذا أن القائمة إذا كان $n = 500.000$ نقوم بعمل 500.000 مقارنة. كنتيجة لذلك لا يكون

البحث التتابعي ذو كفاءة في القوائم الكبيرة.

القوائم المرتبة:

تكون القائمة مرتبة إذا كانت عناصرها مرتبة وفقاً لبعض المعايير. عادةً تكون عناصر القائمة مرتبة ترتيباً تصاعدياً. هناك العديد من العمليات التي يمكن إجراؤها على القائمة المرتبة وهي مشابهة للعمليات المؤداة على القائمة الكيفية. على سبيل المثال عمليات تحدد ما إذا كانت القائمة فارغة أو ممثلة، وتحديد طول القائمة، وطباعة القائمة، ومسح القائمة هي نفسها العمليات المؤداة على القائمة المرتبة والقائمة الغير مرتبة. لهذا ومن أجل تعريف قائمة مرتبة كنوع بيانات مجرد فانه يمكننا باستخدام آلية التوريث أن نستمد الفئة لتطبيق القوائم المرتبة من الفئة `arrayListType` التي نوقشت في القسم السابق. بناءً على ما إذا كان هناك استخدام محدد للقائمة يمكن تخزينه في المصفوفة أو قائمة متصلة نقوم بتعريف فئتين:

الفئة التالية `orderedArrayType` تقوم بتعريف قائمة مرتبة مخزنة في مصفوفة كنوع بيانات مجرد:

```
template<class elemType>
class orderedArrayListType: public arrayListType<elemType>
{
public:
    orderedArrayListType(int size = 100);
    //constructor

    ...
    //We will add the necessary members as needed.

private:
    //We will add the necessary members as needed.
}
```

الفصل 5 قام بتعريف الفئة التالية لتطبيق القوائم المتصلة المرتبة:

```
template<class elemType>
class orderedLinkedListType: public linkedListType<elemType>
{
public:
    ...

}
```

بحث ثنائي:

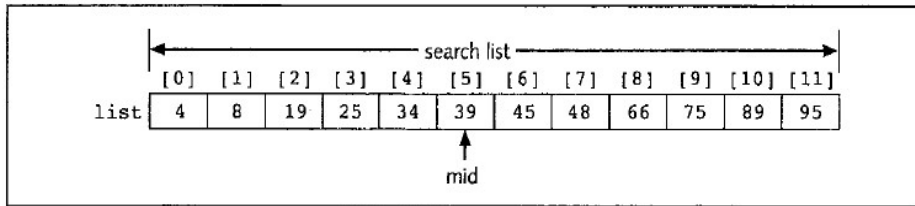
كما ترى البحث الثنائي ليس فعال بالنسبة الى القوائم الكبيرة لأن البحث التتابعي يقوم ببحث نصف القائمة في المتوسط. لهذا نوضح خوارزمية بحث أخرى تسمى **البحث الثنائي** وهي سريعة للغاية. بالرغم من هذا يمكن إجراء البحث الثنائي فقط على القوائم المرتبة. لهذا نفترض أن القائمة مرتبة. خوارزمية البحث الثنائي تستخدم أسلوب "اقسم وتغلب" لبحث القائمة. أولاً تتم مقارنة عنصر البحث مع العنصر الأوسط بالقائمة. إذا كان عنصر البحث أقل من العنصر الأوسط بالقائمة فاننا نحصر البحث على النصف الأول من القائمة وبخلاف هذا نبحث النصف الثاني من القائمة.

انظر الى القائمة المرتبة التي يبلغ طولها 12 في شكل 9-1.

| | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|
| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] |
| list | 4 | 8 | 19 | 25 | 34 | 39 | 45 | 48 | 66 | 75 | 89 | 95 |

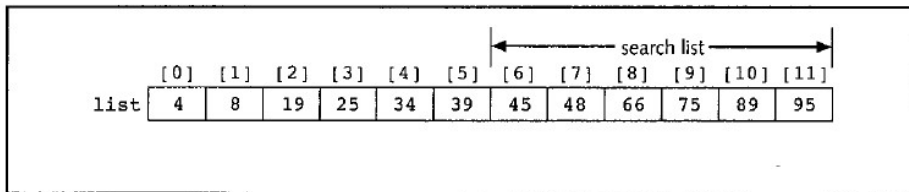
شكل 9-1: قائمة طولها 12

افترض أننا نريد تحديد ما اذا كان 75 موجود في القائمة. مبدئياً تكون القائمة بأكملها هي قائمة البحث (انظر شكل 9-2).



شكل 9-2: قائمة البحث [0] ... list [11]

أولاً نقارن 75 مع العنصر الأوسط في القائمة وهو [5] list (وبساوي 39). بما أن $75 \neq 39$ وأن $75 < 39$ list [5] إذن يمكننا قصر بحثنا على القائمة [6]...list [11] كما هو موضح في شكل 9-3.



شكل 9-3: قائمة البحث [6] ... list [11]

يتم تكرار هذه العملية الآن على القائمة [6]...list [11] وهي قائمة طولها 6. بما أننا في حاجة الى تحديد العنصر الأوسط من القائمة بشكل متكرر فان خوارزمية البحث الثنائي يتم تطبيقها عادةً بالنسبة الى القوائم المبنية على المصفوفات. لتحديد العنصر الأوسط بالقائمة نقوم بجمع مؤشر البداية first ومؤشر النهاية last من قائمة البحث ثم نقوم بالقسمة على 2 لحساب مؤشرها. هذا يعني:

$$mid = \frac{first + last}{2}$$

مبدئياً first = صفر و (بما أن مؤشر المصفوفة في C++ يبدأ عند صفر ويعبر الطول عن عدد العناصر في القائمة) last = الطول - 1.

دالة C++ تقوم بتطبيق خوارزمية البحث الثنائي. اذا تم العثور على عنصر البحث في القائمة يتم انتاج مكانه واذا لم يكن العنصر موجوداً في القائمة فسوف يتم انتاج -1.

```

template<class elemType>
int orderedArrayListType<elemType>::binarySearch
                                   (const elemType& item)
{
    int first = 0;
    int last = length - 1;
    int mid;

    bool found = false;

    while(first <= last && !found)
    {
        mid = (first + last) / 2;

        if(list[mid] == item)
            found = true;
        else
            if(list[mid] > item)
                last = mid - 1;
            else
                first = mid + 1;
    }

    if(found)
        return mid;
    else
        return -1;
} //end binarySearch

```

في خوارزمية البحث الثنائي نقوم بعمل مقارنتين مفتاح في كل مرة عبر الحلقة. الاستثناء الوحيد هو في الحالة الناجحة حيث يتم عمل مقارنة واحدة فقط في المرة الأخيرة خلال الحلقة. ان خوارزمية البحث الثنائي كما هي موضحة من قبل تستخدم بنية تحكم تكرارية (الحلقة while) للمقارنة بين عنصر البحث وعناصر القائمة. كما يمكنك كتابة خوارزمية تكرارية لتطبيق خوارزمية البحث الثنائي. (انظر تمرين البرمجة 2 في نهاية هذا الفصل). المثال التالي يوضح كيفية عمل خوارزمية البحث الثنائي.

مثال 9-1:

انظر الى القائمة المعطاة في شكل 9-4.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|
| list | 4 | 8 | 19 | 25 | 34 | 39 | 45 | 48 | 66 | 75 | 89 | 95 |

شكل 9-4: قائمة مرتبة للبحث الثنائي

حجم هذه القائمة يبلغ 12 أي أن طولها يبلغ 12. الجدول 9-1 يوضح قيم first و last و middle في كل مرة خلال الحلقة. كما أنه يوضح عدد المرات التي يتم فيها مقارنة العنصر بعنصر في القائمة في كل مرة خلال الحلقة.

افترض أننا نقوم بالبحث عن العنصر 89.

جدول 9-1: قيم first و last و middle وعدد المقارنات لعنصر البحث 89

| المكرر | first | last | mid | List [mid] | عدد المقارنات |
|--------|-------|------|-----|------------|---------------|
| 1 | 0 | 11 | 5 | 39 | 2 |
| 2 | 6 | 11 | 8 | 66 | 2 |
| 3 | 9 | 11 | 10 | 89 | 1 |

يتم العثور على العنصر في الموقع 10 والعدد الكلي للمقارنات هو 5. بعد هذا لنقم بالبحث القائمة من أجل العنصر 34. كما في الجدول 9-1 فإن الجدول 9-2 يوضح قيم first و last و middle كل مرة خلال الحلقة. كما أنه يوضح عدد المرات التي تتم بها مقارنة العنصر مع عنصر في القائمة كل مرة خلال الحلقة.

الجدول 9-2: قيم first و last و middle وعدد المقارنات لعنصر البحث 34

| المكرر | first | last | mid | List [mid] | عدد المقارنات |
|--------|-------|------|-----|------------|---------------|
| 1 | 0 | 11 | 5 | 39 | 2 |
| 2 | 0 | 4 | 2 | 19 | 2 |
| 3 | 3 | 4 | 3 | 25 | 2 |
| 4 | 4 | 4 | 4 | 34 | 1 |

تم العثور على العنصر في الموقع 4 وإجمالي عدد المقارنات هو 7. لنقم الآن بالبحث عن العنصر 22 كما هو موضح في الجدول 9-3.

الجدول 9-3: قيم first و last و middle وعدد المقارنات لعنصر البحث 22

| المكرر | first | last | mid | List [mid] | عدد المقارنات |
|--------|-------|------|---------------------------------|------------|---------------|
| 1 | 0 | 5 | 5 | 39 | 2 |
| 2 | 0 | 2 | 2 | 19 | 2 |
| 3 | 3 | 3 | 3 | 25 | 2 |
| 4 | 3 | 4 | الحلقة تتوقف (لأن last > first) | | |

هذا بحث غير ناجح وإجمالي عدد المقارنات هو 6.

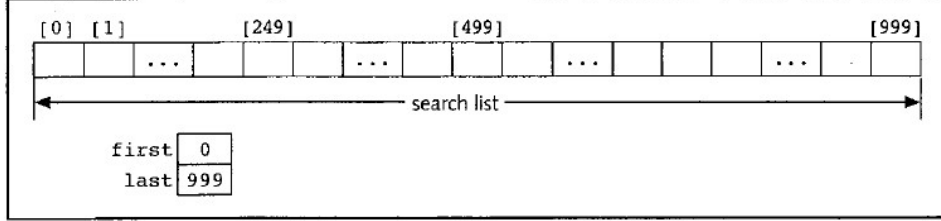
أجراء البحث الثنائي:

افترض أن L عبارة عن قائمة مرتبة مكونة من 1000 عنصر وأنت تريد تحديد ما إذا كان العنصر x موجود في L. بما أن L قائمة مرتبة اذن يمكنك تطبيق خوارزمية البحث الثنائي للبحث عن x. افترض أن L كما هي موضحة في شكل 9-5.

| | | | | |
|-----|-----|-------|-------|-------|
| [0] | [1] | [249] | [499] | [999] |
| | | ... | | ... |

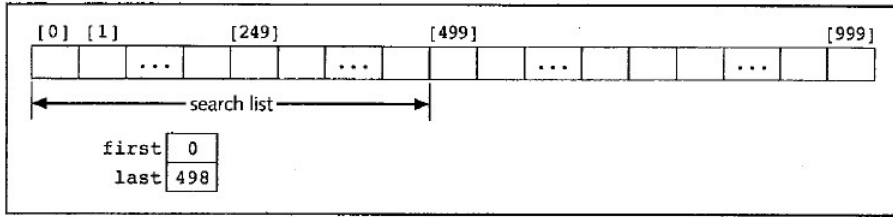
شكل 5-9: القائمة L

التكرار الأول من الحلقة loop يبحث عن x في $L[0] \dots L[999]$ وهي قائمة مكونة من 1000 عنصر. هذا التكرار للحلقة while يقارن x مع $L[499]$. (انظر شكل 6-9).



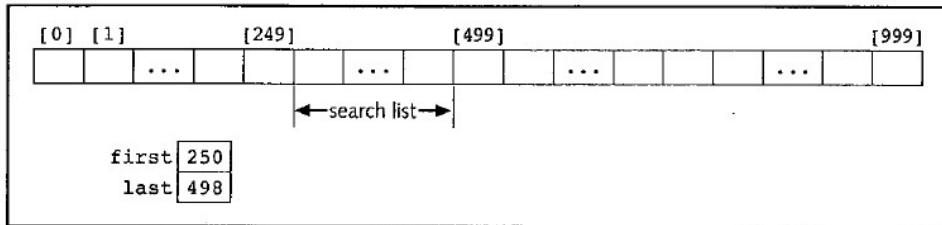
شكل 6-9: قائمة بحث

افترض أن $x \neq L[499]$. إذا كانت $x < L[499]$ اذن المكرر التالي من الحلقة while يبحث عن x في $L[0] \dots L[498]$ وبخلاف هذا تبحث الحلقة while عن x في $L[500] \dots L[999]$. افترض أن $x < L[499]$. اذن المكرر التالي من الحلقة while يبحث عن x في $L[0] \dots L[489]$ وهي قائمة من 499 عنصر كما هو موضح في شكل 7-9.



شكل 7-9: قائمة البحث بعد المكرر الأول.

المكرر الثاني للحلقة while يقارن x مع $L[249]$. افترض مرة أخرى أن $x \neq L[249]$. افترض كذلك أن $x > L[249]$. المكرر التالي من الحلقة while يبحث عن x في $L[250] \dots L[498]$ وهي قائمة من 249 عنصر كما هو موضح في شكل 8-9.



شكل 8-9: قائمة البحث بعد المكرر الثاني.

ينتج من هذه الملاحظات أن: كما، مكرر للحلقة while يقطع حجم قائمة البحث بالنصف. بما أن 1000 $\approx 2^{10} = 1024$ فان الحلقة while لها 11 مكرر على الأكثر لتحديد ما اذا كان العنصر x موجود

في L. (الرمز \approx يعبر عن "تقريباً يساوي"). بما أن كل مكرر من الحلقة while يقوم بعمل مقارنتين – أي تتم مقارنة x مرتين مع عناصر L – البحث الثنائي يقوم بعمل 22 مقارنة على الأكثر لتحديد ما إذا كان x موجود في L. على النقيض تذكر أن البحث التتابعي يقوم في المتوسط بعمل 500 مقارنة لتحديد ما إذا كان x موجود في L.

للحصول على فكرة أفضل عن مدى سرعة البحث الثنائي بالمقارنة مع البحث التتابعي افترض أن L قائمة حجمها 1,000,000. بما أن $1000000 \approx 1048576 = 2^{20}$ إذن يسج أن الحلقة while في البحث الثنائي لها على الأكثر 21 مكرر لتحديد ما إذا كان عنصر ما موجود في L.

كل مكرر من الحلقة while يقوم بعمل مقارنات لعنصرين (مفتاح) ولهذا من أجل تحديد ما إذا كان العنصر موجود في L فإن البحث الثنائي يقوم بعمل 42 مقارنة عنصر على الأقل. على العكس يقوم البحث التتابعي في المتوسط بعمل 500,000 مقارنة عنصر لتحديد ما إذا كان العنصر موجود في L. لاحظ أن:

$$40 = 2 * 20 = 2 * \log_2 2^{20} = 2 * \log_2 (1048576) \approx 2 * \log_2 (1000000)$$

بوجه عام إذا كانت L قائمة مرتبة حجمها n فإن البحث الثنائي من أجل تحديد وجود العنصر في L من عدمه يقوم بعمل مقارنات مفتاح تبلغ على الأقل $2 * \log_2 n + 2$

البحث الغير ناجح:

في حالة البحث الفاشل يمكن توضيح أنه بالنسبة الى قائمة طولها n يقوم البحث الثنائي بعمل $2 * \log_2 (n + 1)$ -ة تقريباً .

البحث الناجح:

في حالة البحث الناجح يمكن توضيح أنه بالنسبة الى قائمة طولها n يقوم البحث الثنائي في المتوسط بعمل $2 * \log_2 n - 4$ key

الآن بعد علمنا بكيفية البحث بشكل فعال في قائمة مرتبة ومخزنة في مصفوفة لنقم بمناقشة كيفية ادخال عنصر داخل قائمة مرتبة.

الادخال في قائمة مرتبة:

افترض أن لديك قائمة مرتبة وأنت تريد ادخال عنصر في القائمة. بعد الادخال يجب أن تكون القائمة الناتجة مرتبة. الفصل 5 قام بوصف كيفية ادخال عنصر في قائمة متصلة مرتبة. هذا القسم يصف كيفية ادخال عنصر في قائمة مرتبة مخزنة في مصفوفة.

لتخزين العنصر في القائمة المرتبة يجب أن نقوم أولاً بإيجاد المكان الذي يتم ادخال العنصر فيه في القائمة ثم نقوم بتحريك عناصر القائمة موضع واحد الى أسفل بالمصفوفة لتوفير مساحة للعنصر لكي يتم ادخاله ثم ندخل العنصر. بما أن القائمة مرتبة ومخزنة في مصفوفة إذن يمكننا استخدام خوارزمية مماثلة لخوارزمية البحث الثنائي لإيجاد المكان الذي يتم ادخال العنصر فيه في القائمة. يمكننا عند ذلك استخدام الدالة insertAt (من الفئة ArrayListType) لادخال العنصر. (لاحظ أننا لا يمكننا استخدام


```

        insertAt(mid, item);
    }
}
} //end insertOrd

```

بالمثل يمكنك كتابة دالة لازالة عنصر من قائمة مرتبة. انظر تمرين البرمجة 6 في نهاية هذا الفصل. اذا قمنا باضافة خوارزمية البحث الثنائي وخوارزمية insertOrd الى الفئة orderedArrayListType فان تعريف هذه الفئة يكون:

```

template<class elemType>
class orderedArrayListType: public arrayListType<elemType>
{
public:
    void insertOrd(const elemType&);
    int binarySearch(const elemType& item);
    orderedArrayListType(int size = 100);
};

```

الجدول 4-9 يلخص تحليل الخوارزمية الخاص بخوارزمية البحث التي نوقشت من قبل. الجدول 4-9: عدد المقارنات لقائمة طولها n.

| الخوارزمية | بحث ناجح | بحث غير ناجح |
|------------|----------|--------------|
| بحث تتابعي | *** | *** |
| بحث ثنائي | *** | *** |

الحد الأدنى على خوارزميات البحث القائمة على المقارنة:

خوارزميات البحث التتابعي والثنائي تبحث القائمة عن طريق مقارنة العنصر المستهدف مع عناصر القائمة. لهذا السبب يطلق على هذه الخوارزميات **خوارزميات بحث قائم على المقارنة**. أقسام سابقة من هذا الفصل أوضحت أن البحث التتابعي يكون ترتيبه n والبحث الثنائي ترتيبه حيث n هو حجم القائمة. $\log_2 n$ الواضح هو: هل يمكننا استنباط خوارزمية بحث ترتيبها أصغر من ؟ قبل أن نقوم باجابة هذا السؤال $\log_2 n$ أولاً على الحد الأدنى على عدد المقارنات بالنسبة الى خوارزميات البحث القائمة على المقارنة.

نظرية رياضية: لتكن L قائمة حجمها $n > 1$. افترض أن عناصر L مرتبة. اذا كانت SRH(n) تعبر عن العدد الأدنى من المقارنات اللازمة فانه في الحالة الأسوأ وباستخدام خوارزمية قائمة على المقارنة

لتمييز ما اذا كان العنصر x موجود في n فان $SRH(n) \geq \log_2(n + 1)$

نتيجة: خوارزمية البحث الثنائي هي خوارزمية الحالة الأسوأ لحل مشكلات البحث عن طريق أسلوب المقارنة.

من هذه النتائج ينتج أننا إذا أردنا تصميم خوارزمية بحث ترتبها أقل من $\log_2 n$ يمكن أن تكون قائمة على المقارنة.

البعثة:

قامت أقسام سابقة من هذا الفصل بمناقشة خوارزميتان للبحث: الثنائي والتتابعي. في البحث الثنائي يجب أن تكون البيانات مرتبة وفي البحث التتابعي لا يكون هناك حاجة إلى أن تكون البيانات في أي ترتيب معين. لقد قمنا كذلك بتحليل كلا من هاتين الخوارزميتين وأوضحنا أن البحث التتابعي ترتبته n والبحث الثنائي ترتبته $\log_2 n$ حيث n هو طول $\log_2 n$ سؤال الواضح هو: هل يمكننا إقامة خوارزمية بحث ترتبها أصغر من $\log_2 n$ ؟ تذكر أن كل $\log_2 n$ ميات البحث الثنائي والتتابعي عبارة عن خوارزميات قائمة على المقارنة. لقد حصلنا على حد أدنى على خوارزميات البحث القائمة على المقارنة يوضح أن الخوارزميات البحثية القائمة على المقارنة تكون على الأقل بالترتيب. لهذا إذا أردنا إقامة خوارزمية بحث يكون $\log_2 n$ قل من $\log_2 n$ فإنها لا يمكن أن تكون قائمة على المقارنة. هذا $\log_2 n$ ف خوارزمية يكون ترتبها في المتوسط 1.

القسم السابق أوضح أنه بالنسبة إلى الخوارزميات القائمة على المقارنة يحقق البحث الثنائي الحد الأدنى. بالرغم من هذا يتطلب البحث الثنائي من البيانات أن تكون منظمة أي يجب أن تكون مرتبة. خوارزمية البحث التي نصفها الآن تسمى **البعثة** وهي تتطلب أيضاً أن تكون البيانات مرتبة. في البعثة يتم ترتيب البيانات بمساعدة جدول يسمى **جدول البعثة** ويتم تخزينه في مصفوفة. لتحديد ما إذا كان هناك عنصر خاص له مفتاح مثل x في الجدول فإننا نقوم بتطبيق الدالة h المسماة **دالة البعثة** على المفتاح X أي أننا نحسب $h(X)$ التي تقرأ h من x . الدالة h عبارة عن دالة حسابية و $h(X)$ تعطي عنوان العنصر في جدول البعثة. افترض أن حجم جدول البعثة هو m . إذن $0 \leq h(X) < m$. لهذا ومن أجل تحديد ما إذا كان العنصر صاحب المفتاح X في الجدول أم لا فإننا ننظر إلى المدخلات $HT[h(x)]$ في جدول البعثة. بما أن عنوان العنصر يتم حسابه بمساعدة دالة ما إذن ينتج أن العناصر يتم تخزينها في غير ترتيب معين. قبل الاستمرار في هذه المناقشة لنقم بالنظر إلى الأسئلة التالية:

■ كيف نختار دالة بعثة؟

■ كيف نقوم بتنظيم البيانات بمساعدة جدول البعثة؟

أولاً نناقش كيفية تنظيم البيانات في جدول البعثة.

هناك طريقتان يتم بهما ترتيب البيانات بمساعدة جدول البعثة. في الطريقة الأولى يتم تخزين البيانات داخل جدول البعثة أي في مصفوفة. في الطريقة الثانية يتم تخزين البيانات في قوائم متصلة ويكون جدول البعثة عبارة عن مصفوفة من المؤشرات إلى تلك القوائم المتصلة. كل طريقة لها مميزاتها

وعيوبها ونحن نناقش كلتا الطريقتين بالتفصيل. مع هذا نقوم أولاً بتقديم بعض من المصطلحات المستخدمة في هذا القسم.

جدول البعثة HT يكون مقسم عادةً الى مقادير b وهي $HT[0], HT[1], \dots, HT[b-1]$. كل مقدار قادر على احتواء عدد r من العناصر. ينتج من هذا أن $br = m$ حيث m هو حجم HT. بوجه عام تكون $1 = r$ وبهذا يمكن لكل مقدار أن يحتوي على عنصر واحد.

دالة البعثة h ترسم المفتاح X على عدد صحيح t وهذا يعني أن $h(X) = t$ حيث $0 \leq h(X) \leq b-1$ حان X_1 و X_2 بحيث $X_2 \neq X_1$ تسمى مرادفات اذا كان $h(X_1) = h(X_2)$ مفتاح X ، $h(X) = t$. اذا كان المقدار t ممتلئ نقول أنه حدث فائض. لتكن X_1 و X_2 مفتاحان غير متطابقان أي أن $X_2 \neq X_1$. اذا كان $h(X_2) = h(X_1)$ فاننا نقول أنه حدث تعارض. اذا كانت $1 = r$ أي أن حجم المقدار 1 فان الفائض والتعارض يحدثان في آن واحد. عند اختيار دالة بعثة الأهداف الرئيسية تكون:

■ اختيار دالة بعثة سهل حسابها.

■ تقليل عدد التعارضات الى أدنى حد.

بعد هذا ننظر الى بعض الأمثلة على دالات البعثة.

افترض أن $Htsize$ يعبر عن حجم جدول البعثة أي حجم المصفوفة التي تضم جدول البعثة. نفترض أن حجم المقدار هو 1 وبهذا يمكن لكل مقدار أن يضم عنصر واحد ولهذا يحدث الفائض والتعارض في وقت واحد.

دالات بعثة: بعض الأمثلة:

يتم توضيح العديد من دالات البعثة في الكتب وهنا نوضح بعض من دالات البعثة المستعملة بشكل شائع.

متوسط المربع: في هذه الطريقة يتم حساب دالة البعثة h عن طريق تربيع المحدد ثم استخدام العدد المناسب من الأجزاء من منتصف المربع للحصول على عنوان المقدار. بما أن الأجزاء المتوسطة من المربع تعتمد عادةً على جميع الرموز فمن المتوقع أن هناك مفاتيح مختلفة سوف تنتج عناوين بعثة مختلفة ذات احتمالية مرتفعة حتى اذا كانت بعض من الرموز متماثلة.

التقطيع: في التقطيع يتم تقسيم المفتاح X الى أجزاء بحيث تكون جميع الأجزاء ذات طول واحد فيما عدا الأجزاء الأخيرة. يتم بعد ذلك جمع الأجزاء بطريقة مناسبة للحصول على عنوان البعثة.

القسمة (حساب وحدات): في هذه الطريقة يتم تحويل المفتاح X الى عدد صحيح ثم i_x قسمة هذا العدد الصحيح على حجم جدول البعثة للحصول على الباقي معطياً عنوان X في جدول البعثة وهذا (في C++) يعني:

$$h(X) = i_x \% HTSize;$$

افترض أن كل مفتاح عبارة عن مقطع. دالة C++ التالية تستخدم طريقة القسمة لحساب عنوان المفتاح:

```
int hashFunction(char *key, int keyLength)
{
    int sum = 0;

    for(int j = 0; j <= keyLength; j++)
        sum = sum + static_cast<int>(key[j]);

    return (sum % HTSize);
} //end hashFunction
```

حل التعارض:

كما لوحظ من قبل فإن دالة البعثة التي نخترها لا يجب فقط أن تكون سهلة الحساب ولكن من الأكثر رغبة أن يتم تقليل عدد حالات التعارض الى أدنى حد ممكن. بالرغم من هذا فإن التعارضات لا يمكن تجنبها فعلياً. لهذا فإننا في البعثة يجب أن نتضمن خوارزميات لمعالجة التعارضات. يتم تصنيف أساليب حل التعارض الى فئتين هما: **معالجة مفتوحة** (تسمى أيضاً **بعثة مغلقة**) و **تسلسل** (يسمى أيضاً **بعثة مفتوحة**). في المعالجة المفتوحة يتم تخزين البيانات داخل جدول البعثة وفي التسلسل يتم تنظيم البيانات في قوائم متصلة ويكون جدول البعثة عبارة عن مصفوفة من مؤشرات الى القوائم المتصلة. نناقش أولاً حل التعارض بواسطة المعالجة المفتوحة.

حل التعارض: المعالجة المفتوحة:

كما وضح من قبل يتم تخزين البيانات في المعالجة المفتوحة داخل جدول البعثة ولهذا فإن $h(X)$ تعطي لكل مفتاح X المؤشر في المصفوفة حيث من المحتمل تخزين العنصر ذو المفتاح X . يمكن تطبيق المعالجة المفتوحة بعدة طرق. بعد هذا نصف بعضاً من الطرق الشائعة لتطبيق المعالجة المفتوحة.

البحث الطولي:

افترض أن العنصر ذي المفتاح X سوف يتم ادخاله في جدول البعثة. اننا نستخدم دالة البعثة لحساب المؤشر $h(X)$ لهذا العنصر في جدول البعثة. افترض أن $h(X) = t$ اذن $0 \leq h(X) \leq HTSize - 1$. اذا كان $HT[t]$ فارغاً نقوم بتخزين هذا العنصر داخل مقطع المصفوفة هذا. افترض أن $HT[t]$ مشغولة بالفعل بعنصر آخر اذن لدينا تعارض. في البحث الطولي نقوم بدءاً من t يبحث المصفوفة تتابعياً للعثور على مقطع المصفوفة التالي المتاح.

نفترض في البحث الطولي أن المصفوفة دائرية ولهذا اذا كان الجزء الأدنى من المصفوفة ممثلاً يمكننا الاستمرار في البحث في الجزء الأعلى من المصفوفة. يمكن انجاز ذلك بسهولة عن طريق

استخدام المعامل mod. هذا يعني أننا بدءاً من t نتفقد مواقع المصفوفة $t, (t+1) \% HTSize, (t+2) \% HTSize, \dots, (t+j) \% HTSize$. هذا يطلق عليه تتابع بحثي. يتم الوصول الى مقطع المصفوفة التالي عن طريق:

$$(h(X) + j) \% HTSize$$

حيث j البحث رقم j .

الكود التالي يطبق البحث الطولي:

```
hIndex = hashFunction(insertKey);
found = false;

while(HT[hIndex] != emptyKey && !found)
    if(HT[hIndex].key == key)
        found = true;
    else
        hIndex = (hIndex + 1) % HTSize;

if(found)
    cerr<<"Duplicate items are not allowed."<<endl;
else
    HT[hIndex] = newItem;
```

من تعريف البحث الطولي نرى أن البحث الطولي سهل التطبيق. بالرغم من هذا فإن البحث الطولي يسبب **التجميع** أي أن المزيد والمزيد من المفاتيح الجديدة من المحتمل أن يتم بعثرتها الى مقاطع المصفوفة التي تكون مشغولة بالفعل.

على سبيل المثال انظر الى جدول البعثة الذي حجمه 20 والموضح في شكل 9-9.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| | | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

شكل 9-9: جدول بعثة حجمه 20

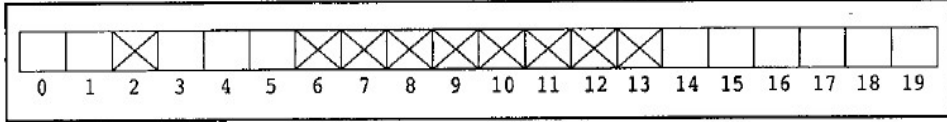
مبدئياً جميع مواضع المصفوفة متاحة. بما أن جميع مواضع المصفوفة متاحة اذن فإن احتمالاً بحث أي موضع يكون (20/1). افترض أنه بعد ترتيب بعض من العناصر يكون جدول البعثة كما هو موضح في شكل 9-10.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| | | X | | | | X | X | X | | X | X | X | X | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

شكل 9-10: جدول بعثرة حجه 20 وبه مواضع معينة مشغولة

في هذا الشكل يشير علامة اكس الى أن مقطع المصفوفة مشغول. المقطع 9 سوف يتم شغله بعد هذا اذا كان عنوان البعثرة بالنسبة الى المفتاح التالي كان 6 أو 7 أو 8 أو 9. لهذا يكون احتمال أن يتم شغل المقطع 9 فيما بعد هو 20/4. بالمثل في جدول البعثرة يكون احتمال شغل الموضع 14 من المصفوفة هو 20/5.

انظر الآن الى جدول البعثرة في شكل 9-11.



شكل 9-11: جدول بعثرة حجمه 20 وبه مواضع معينة مشغولة

في جدول البعثرة هذا يكون احتمال أن يصبح موضع المصفوفة رقم 14 مشغولاً فيما بعد 20/9 بينما يكون احتمال أن يصبح موضع المصفوفة رقم 15 أو 16 أو 17 فيما بعد 20/1. اننا نرى أن العناصر تميل الى التجمع وهذا قد يزيد من طول البحث. لهذا يتسبب البحث الطولي في التجميع. هذا التجميع يسمى **تجميع أولي**.

من احدى طرق تحسين البحث الطولي هو تجاوز مواضع المصفوفة بمقدار ثابت لنقل c بدلاً من 1. في هذه الحالة يكون عنوان البعثرة هو:

$$HTSize \% (h(X) + I * c)$$

اذا كانت $c = 2$ ، و $h(X) = 2k$ أي أن $h(X)$ تكون زوجية حيث تتم زيارة مواضع المصفوفة الزوجية فقط. بالمثل اذا كانت $c = 2$ ، و $h(X) = 1 + 2k$ فان هذا يعني أن $h(X)$ فردي وأنه تتم زيارة مواضع المصفوفة الفردية فقط. لزيارة جميع مواضع المصفوفة يجب أن تكون c أكبر من HTSize.

بحث عشوائي:

هذه الطريقة تستخدم مولد أعداد عشوائية للعثور على المقطع التالي المتاح. المقطع رقم I في التتابع البحثي هو:

$$HTSize \% (h(X) + i * c)$$

حيث r_i هو القيمة i في التبديل العشوائي للأعداد من 1 الى $1 - HTSize$. جميع عمليات الادخال والبحث تستخدم نفس التتابع من الأعداد العشوائية.

المثال 9-2 يوضح كيفية عمل تتابع البحث باستخدام البحث العشوائي.

مثال 9-2:

افترض أن حجم جدول البعثة هو 101 وبالنسبة الى المفاتيح X_1 و X_2 تكون $h(X_1) = 26$ و $h(X_2) = 35$. افترض كذلك أن $r_1 = 2$, $r_2 = 5$, $r_3 = 8$ تتابع البحثي ل X_1 يمتلك العناصر 26، 28، 31، و 34. بالمثل يمتلك التتابع البحثي ل X_2 العناصر 35، 37، 40، و 43.

اعادة البعثة:

في هذه الطريقة اذا حدث تعارض مع دالة البعثة h نستخدم سلسلة من دالات البعثة h_1 و h_2 ... و h_s وهذا يعني أنه اذا حدث التعارض عند $h(X)$ فانه يتم فحص مقاطع المصفوفة

$$h_i(X), 1 \leq h_i(X) \leq s$$

البحث التربيعي:

افترض أنه تتم بعثة العنصر ذو المفتاح X عند t أي أن $t = h(X)$ و $0 \leq t \leq HTSize - 1$. افترض كذلك أن الموضع t مشغول بالفعل. في البحث التربيعي نقوم بدءاً من الموضع t ببحث المصفوفة طويلاً عند المواضع $(t + 1) \% HTSize, (t + 2) \% HTSize = (t + 4) \% HTSize, (t + 3) \% HTSize = (t + 9) \% HTSize, \dots, (t + i^2) \% HTSize$ هو : $t, (t + 1) \% HTSize, (t + 2) \% HTSize, (t + 3) \% HTSize, \dots, (t + i^2) \% HTSize$. المثال 3-9 يوضح كيفية عمل التتابع البحثي باستخدام البحث التربيعي.

مثال 3-9:

افترض أن حجم جدول البعثة هو 101 وبالنسبة الى المفاتيح X_1 و X_2 و X_3 تكون $h(X_1) = 25$ و $h(X_2) = 96$ و $h(X_3) = 34$. اذن يكون التتابع البحثي ل X_1 هو 25، 26، 29، و 34، و 41، وهكذا. التتابع البحثي ل X_2 يكون 96، 97، 100، و 4، و 11، وهكذا. (لاحظ أن $32 + 96$) $4 = 101 \% 105 = 101 \% 4$ التتابع البحثي ل X_3 هو 34، 35، 38، و 43، و 50، و 59 وهكذا. بالرغم من أن العنصر 34 من التتابع البحثي ل X_3 هو نفسه العنصر الرابع من التتابع البحثي ل X_1 الا أن التتابع البحثي لكليهما يكون مختلف بعد العنصر 34.

بالرغم من أن البحث التربيعي يقلل التجميع الا أننا لا نعرف ما اذا كان يبحث جميع المواضع في الجدول أم لا. في الحقيقة انه لا يبحث جميع المواضع في الجدول. مع هذا عندما يكون $HTSize$ رئيسي يقوم البحث التربيعي ببحث حوالي نصف الجدول قبل تكرار التتابع البحثي. لنقم باثبات هذه الملاحظة:

افترض أن $HTSize$ رئيسي وأن $0 \leq i < j \leq HTSize$:

$$(t + i^2) \% HTSize = (t + j^2) \% HTSize.$$

هذا يشير ضمناً الى أن:

$$HTSize \mid (j^2 - i^2) \text{ or } HTSize \mid (j - i)(j + i)$$

(هنا الرمز | يعني القسمة. على سبيل المثال أ | ب تعني أن أ تقسم ب) بما أن HTSize رئيسي فأننا نحصل على:

$$HTSize \mid (j - i) \text{ or } HTSize \mid (j + i)$$

الآن بما أن $0 < i - j < HTSize$ اذن HTSize لا تقسم $(j - i)$. من هنا $HTSize \mid (j + i)$. وهذا يشير ضمناً الى أن $j + i \geq HTSize$ وهكذا:

$$j \geq \frac{HTSize}{2}$$

من هنا يقوم البحث التربيعي ببحث نصف الجدول قبل تكرار التتابع البحثي. وينتج من هذا أنه اذا كان حجم HTSize كبيراً على الأقل ضعفي عدد العناصر اذن يمكننا حل جميع التعارضات. بما أن بحث نصف الجدول يعتبر عدد معقول من حالات البحث فأننا نفترض بعد عمل حالات البحث هذه أن الجدول ممتلئ ونوقف الادخال (والبحث). (يمكن أن يحدث هذا عندما يكون الجدول نصف ممتلئ وعملياً من النادر أن يحدث هذا الا اذا كن الجدول ممتلئ تقريباً). بعد هذا نوضح كيفية توليد التتابع البحثي. لاحظ أن:

$$2^2 = 1 + (2 \cdot 2 - 1)$$

$$3^2 = 1 + 3 + (2 \cdot 3 - 1)$$

$$4^2 = 1 + 3 + 5 + (2 \cdot 4 - 1)$$

⋮

$$i^2 = 1 + 3 + 5 + 7 + \dots + (2 \cdot i - 1), i \geq 1.$$

ينتج اذن أن:

$$(t + i^2) \% HTSize = (t + 1 + 3 + 5 + 7 + \dots + (2 \cdot i - 1)) \% HTSize$$

انظر الى التتابع البحثي:

$$t, (t + 1) \% HTSize, (t + 2^2) \% HTSize, (t + 3^2) \% HTSize, \dots, (t + i^2) \% HTSize$$

كود C++ التالي يحسب البحث رقم i حيث $(t + i^2) \% HTSize$

```
int inc = 1;
int pCount = 0;

while(p < i)
{
    t = (t + inc) % HTSize;
    inc = inc + 2;
    pCount++;
}
```

الكود الافتراضي التالي يطبق البحث التربيعي (افترض أن HTSize رئيسي)

```
int pCount;
int inc;
int hIndex;

hIndex = hashFunction(insertKey);

pCount = 0;
inc = 1;

while(HT[hIndex] is not empty
    && HT[hIndex] is not the same as the insert item
    && pCount < HTSize / 2)
{
    pCount++;
    hIndex = (hIndex + inc) % HTSize;
    inc = inc + 2;
}

if(HT[hIndex] is empty)
    HT[hIndex] = newItem;

else
    if(HT[hIndex] is the same as the insert item)
        cerr<<"Error: No duplicates are allowed."<<endl;
    else
        cerr<<"Error: The table is full. "
            <<"Unable to resolve the collision."<<endl;
```

كل من البحث العشوائي والتربيعي يمنع التجميع الأولي. بالرغم من هذا اذا تمت بعثرة مفاتيح غير متطابقين X_1 و X_2 الى نفس موضع المنزل أي أن $h(X_1) = h(X_2)$ اذن يتم اتباع التتابع البحثي ذاته لكل من المفاتيح لأن البحث العشوائي والبحث التربيعي دالات للمواضع المنزلية وليس المفتاح الأصلي. ينتج من هذا أنه اذا تسببت دالة البعثرة في تجميع في موضع منزلي معين فان التجميع يستمر في ظل هذه البحوثات. هذا يطلق عليه **التجميع الثانوي**.

من احدى طرق حل التجميع الثانوي استخدام البحث الطولي مع قيمة زيادة دالة المفتاح. هذا يسمى **البعثرة المزدوجة**. في البعثرة المزدوجة اذا حدث تعارض عند $h(X)$ يتم توليد التتابع البحثي

باستخ $(h(X) + i * h'(X)) \% HTSize$

حيث h' دالة البعثرة الثانية.

المثال 4-9 يوضح كيفية عمل التتابع البحثي باستخدام بعثرة مزدوجة.

مثال 4-9:

افترض أن حجم جدول البعثرة 101 وبالنسبة الى المفاتيح X_1 و X_2 يكون $h(X_1) = 35$ و $h(X_2) = 83$ وافترض أيضاً أن $h'(X_1) = 3$ و $h'(X_2) = 6$. اذن التتابع البحثي ل X_1 يكون 35، 38،

و41، و44، و47، وهكذا ويكون التابع البحثي لـ X2 83، و89، و95، و0، و6، وهكذا. (لاحظ أن $0 = 101 \% 101 = 101 \% (6 * 3 + 83)$)

الحذف: معالجة مفتوحة:

افترض أن هناك عنصر لنقل R يتم حذفه من جدول البعثة HT يجب أن نقوم أولاً بالعثور على مؤشر R في جدول البعثة. لايجاد مؤشر R نقوم بتطبيق المعيار ذاته الذي تم تطبيقه على R عندما تم ادخال R في جدول البعثة. لنقم بافتراض أنه بعد ادخال R تم ادخال عنصر آخر R' في جدول البعثة وكان موضع R و R' واحد. التابع البحثي لـ R موجود في التابع البحثي لـ R' لأنه تم ادخال R' في جدول البعثة بعد R. افترض أننا نحذف R ببساطة عن طريق تحديد مقطع المصفوفة المحتوي على R كمقطع فارغ. اذا ظل موضع المصفوفة هذا فارغ اذن فأتثناء البحث عن R' واتباع تتابعها البحثي يتوقف البحث عن هذا الموضع الفارغ من المصفوفة. هذا يعطي انطباعاً بأن R' ليست في الجدول وهذا غير صحيح بالطبع. لا يمكن حذف العنصر R ببساطة بواسطة تحديد موضعه كموضع فارغ من جدول البعثة.

من احدى طرق حل هذه المشكلة عمل مفتاح خاص يتم تخزينه في مفاتيح العناصر التي يتم حذفها. المفتاح الخاص في أي مقطع يشير الى أن مقطع المصفوفة هذا متاح لعنصر جديد يتم ادخاله. بالرغم من هذا فان البحث لا يجب أن يتوقف عند هذا الموضع أثناء البحث. هذا لسوء الحظ يجعل خوارزمية الحذف بطيئة ومعقدة.

هناك حل آخر وهو استخدام مصفوفة أخرى لنقل indexStatusList من النوع int ومن نفس حجم جدول البعثة كما يلي: تهيئة كل موضع من indexStatusList عند صفر مشيراً الى أن الموضع المقابل في جدول البعثة فارغ. عند اضافة عنصر الى جدول البعثة في الموضع i نضبط indexStatusList [i] عند 1. عند حذف عنصر من جدول البعثة عند الموضع k فاننا نضبط indexStatusList [k] عند 1. لهذا يكون كل مدخل في المصفوفة indexStatusList اما 1- أو صفر أو 1.

على سبيل المثال افترض أن لديك جدول البعثة الموضح في الشكل 9-12.

| indexStatusList | | HashTable | |
|-----------------|---|-----------|--------|
| [0] | 1 | [0] | Mickey |
| [1] | 1 | [1] | Goofy |
| [2] | 0 | [2] | |
| [3] | 1 | [3] | Grumpy |
| [4] | 0 | [4] | |
| [5] | 1 | [5] | Balto |
| [6] | 1 | [6] | Ducky |
| [7] | 0 | [7] | |
| [8] | 1 | [8] | Minnie |
| [9] | 0 | [9] | |

شكل 9-12: جدول البعثة وindexStatusList

في الشكل 9-12 تكون مواضع جدول البعثة 0، و1، و3، و6، و5، و8 مشغولة. افترض أنه تمت إزالة المدخلات الموجودة في المواضع 3 و6. لازالة هذه المدخلات من جدول البعثة نقوم بتخزين 1- في المواضع 3 و6 في المصفوفة indexStatusList (انظر شكل 9-13).

| indexStatusList | | HashTable | |
|-----------------|----|-----------|--------|
| [0] | 1 | [0] | Mickey |
| [1] | 1 | [1] | Goofy |
| [2] | 0 | [2] | |
| [3] | -1 | [3] | Grumpy |
| [4] | 0 | [4] | |
| [5] | 1 | [5] | Balto |
| [6] | -1 | [6] | Ducky |
| [7] | 0 | [7] | |
| [8] | 1 | [8] | Minnie |
| [9] | 0 | [9] | |

شكل 9-13: جدول البعثة وindexStatusList بعد ازالة المدخلات بالمواضع 3 و6

البعثة: التطبيق باستخدام البحث التربيعي:

هذا القسم يصف بايجاز كيفية تصميم فئة كنوع بيانات مجرد لتطبيق البعثة باستخدام البحث التربيعي. لتطبيق البعثة نستخدم مصفوفتين. مصفوفة يتم استخدامها لتخزين البيانات والأخرى indexStatusList كما تم وصفها في القسم السابق ويتم استخدامها للإشارة الى ما اذا كان الموضع في جدول البعثة حر أم مشغول أم تم استخدامه من قبل. قالب الفئة التالي يقوم بتعريف البعثة كنوع بيانات مجرد:

```
template<class elemType>
class hashT
{
public:
    void insert(int hashIndex, const elemType& rec);
        //Function to insert an item in the hash table.
        //The first parameter specifies the initial hash index
        //of the item to be inserted.
        //The item to be inserted is specified by the parameter rec.
        //Postcondition: If an empty position is found in the
        // hash table, rec is inserted and the length is
        // incremented by one; otherwise, an appropriate error
        // message is displayed.

    void search(int& hashIndex, const elemType& rec, bool& found);
        //Function to determine whether the item specified by the
        //parameter rec is in the hash table.
        //The parameter hashIndex specifies the initial hash index
        //of rec.
```

```

void insert (int hashIndex, const elemType& rec);
// دالة لادخال عنصر في جدول البعثة.
// المعامل الأول يحدد مؤشر البعثة الأولي للعنصر الذي يتم ادخاله.
// يتم تحديد العنصر الذي يتم ادخاله بواسطة المعامل rec.
// شرط تالي: اذا تم العثور على موضع فارغ في جدول البعثة
// يتم ادخال rec ويزيد الطول بمقدار واحد وبخلاف هذا
// يتم عرض رسالة خطأ.

void search (int& hashIndex, const elemType& rec, bool& found);
// دالة لتحديد ما اذا كان العنصر المحدد بواسطة
// المعامل rec موجود في جدول البعثة.
// المعامل hashIndex يحدد مؤشر البعثة الأولي لrec.

//Postcondition: If rec is found, found is set to true and
// hashIndex specifies the position where rec is found;
// otherwise, found is set to false.

bool isItemAtEqual(int hashIndex, const elemType& rec);
//Function to determine whether the item specified by the
//parameter rec is the same as the item in the hash table
//at position hashIndex.
//Postcondition: Returns true if HTable[hashIndex] == rec;
// otherwise, returns false.

void retrieve(int hashIndex, elemType& rec);
//Function to retrieve the item at position hashIndex.
//Postcondition: If the table has an item at position
// hashIndex, it is copied into rec.

void remove(int hashIndex, const elemType& rec);
//Function to remove an item from the hash table.
//Postcondition: Given the initial hashIndex, if rec
// is found in the table it is removed; otherwise,
// an appropriate error message is displayed.

void print() const;
//Function to output the data.

hashT(int size = 101);
//constructor
//Postcondition: Create the arrays HTable and
// indexStatusList; initialize the array
// indexStatusList to 0; length = 0; HTSize =
// size; and the default array size is 101.

~hashT();
//destructor
//Postcondition: Array HTable and indexStatusList
// are deleted.

private:
elemType *HTable; //pointer to the hash table
int *indexStatusList; //pointer to the array indicating
//the status of a position in the
//hash table
int length; //number of items in the hash table
int HTSize; //maximum size of the hash table
};

```

// شرط تالي: اذا تم العثور على rec يتم ضبط found عند true
 // وhashIndex يحدد الموضع الذي تم فيه العثور على rec
 // وبخلاف هذا يتم ضبط found عند false.

bool isItemAtEqual (int hashIndex, const elemType& rec);
 // دالة لتحديد ما اذا كان العنصر المحدد بواسطة
 // المعامل rec هو نفس العنصر الموجود في جدول البعثة
 // عند الموضع hashIndex.
 // شرط تالي: تنتج true اذا كانت rec == Htable[hashIndex]
 // وبخلاف هذا تنتج false.

void retrieve (int hashIndex, const elemType& rec);
 // دالة لاستعادة النسر عند الموضع hashIndex.
 // شرط تالي: اذا كان الجدول به عنصر عند الموضع
 // hashIndex يتم نسخه داخل rec.

void remove (int hashIndex, const elemType& rec);
 // دالة لازالة عنصر من جدول البعثة.
 // شرط تالي: باعطاء hashIndex الأولي اذا تم العثور
 // على rec يتم حذفه وبخلاف هذا
 // يتم عرض رسالة خطأ مناسبة.

void print () const;
 // دالة لايخراج البيانات.

hashT (int size = 101);
 // مقوم

// شرط تالي: عمل المصفوفات Htable وindexStatusList
 // وتهيئة المصفوفة indexStatusList عند صفر، والطول = صفر
 // وHTSize = size وحجم المصفوفة الافتراضي 101.

~ hashT ();
 // مدمر

// شرط تالي: يتم حذف المصفوفتين Htable وindexStatusList

خاصة:

| | |
|-----------------------|------------------------------------|
| elemType *Htable; | // مؤشر الى جدول البعثة. |
| int *indexStatusList; | // مؤشر الى المصفوفة التي تشير الى |
| | // الى حالة الموضع في جدول البعثة. |
| int length; | // عدد العناصر في جدول البعثة. |
| int HTSize; | // الحجم الأقصى لجدول البعثة. |

اننا نعطي تعريف الدالة insert فقط ونترك الدالات الأخرى كتمرين لك.

تعريف الدالة insert باستخدام البحث التربيعي يكون كما يلي:

```
template<class elemType>
void hashT<elemType>::insert(int hashIndex, const elemType& rec)
{
    int pCount;
    int inc;

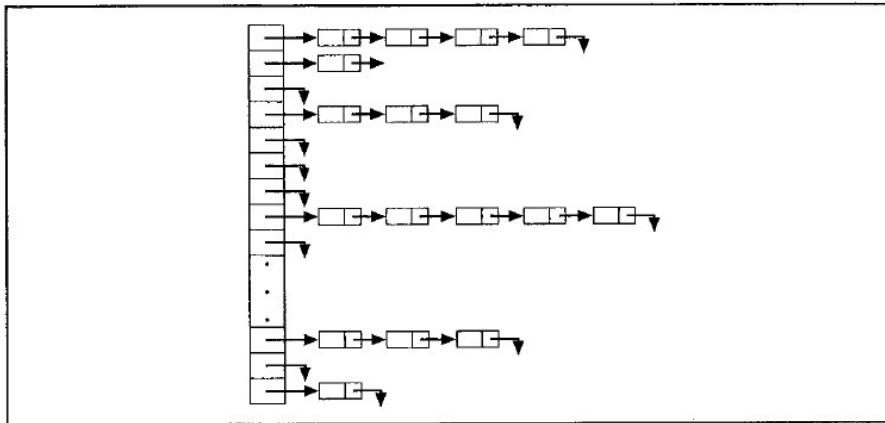
    pCount = 0;
    inc = 1;

    while(indexStatusList[hashIndex] == 1
        && HTable[hashIndex] != rec
        && pCount < HTSize / 2)
    {
        pCount++;
        hashIndex = (hashIndex + inc) % HTSize;
        inc = inc + 2;
    }

    if(indexStatusList[hashIndex] != 1)
    {
        HTable[hashIndex] = rec;
        indexStatusList[hashIndex] = 1;
        length++;
    }
    else
    {
        if(HTable[hashIndex] == rec)
            cerr<<"Error: No duplicates are allowed."<<endl;
        else
            cerr<<"Error: The table is full. "
                <<"Unable to resolve the collision."<<endl;
    }
}
```

حل التعارض: التسلسل (بعثرة مفتوحة):

في التسلسل يكون جدول البعثرة HT عبارة عن مصفوفة من المؤشرات (انظر شكل 14-9). لهذا فان لكل j حيث $0 \leq j \leq \text{HTSize} - 1$ يكون $\text{HT}[j]$ مؤشر الى قائمة متصلة. حجم جدول البعثرة HTSize يكون أصغر من أو يساوي عدد العناصر.



شكل 14-9: جدول بعثرة متصل

ادخال العنصر والتعارض:

بالنسبة الى كل مفتاح X (في العنصر) نقوم أولاً بإيجاد t حيث $H(X) = t$ حيث $0 \leq t \leq HTSize-1$. العنصر ذو هذا المفتاح يتم ادخاله في القائمة المتصلة (التي قد تكون فارغة) المشار اليها بواسطة $HT[t]$. ينتج من هذا أنه بالنسبة الى المفاتيح الغير متطابقة $X1$ و $X2$ اذا كانت $h(X1) = h(X2)$ فانه يتم ادخال العناصر ذات المفاتيح $X1$ و $X2$ في نفس القائمة المتصلة ولهذا يتم التعامل مع التعارض بسرعة وبفاعلية. (يمكن ادخال عنصر جديد في بداية القائمة المتصلة لأن البيانات في القائمة المتصلة لا يكون لها ترتيب محدد).

البحث:

افترض أننا نريد تحديد ما اذا كان العنصر R ذي المفتاح X موجود في جدول البعثة أم لا. كالمعتاد نقوم أولاً بحساب $h(X)$. افترض أن $t = h(X)$. اذن القائمة المتصلة المشار اليها بواسطة $HT[t]$ يتم بحثها بالتتابع.

الحذف:

لحذف عنصر مثل R من جدول البعثة نقوم أولاً ببحث جدول البعثة للعثور على المكان الذي يتواجد فيه R في القائمة المتصلة. بعد هذا نقوم بتعديل المؤشرات عند المواضع المناسبة ونزيل تخصيص الذاكرة التي يشغلها R .

الفائض:

بما أن البيانات يتم تخزينها في قوائم متصلة فان الفائض لا يعد مصدر قلق لأنه يتم ازالة تخصيص مساحة الذاكرة لتخزين البيانات بطريقة حركية. فضلاً عن هذا لا يعد هناك حاجة الى أن يكون حجم جدول البعثة أكبر من عدد العناصر. اذا كان حجم جدول البعثة أصغر من عدد العناصر فان بعض من القوائم المتصلة تحتوي على أكثر من عنصر واحد. بالرغم من هذا وبدالة بعثة جيدة لا يزال يكون متوسط طول القائمة المتصلة صغير ولهذا يكون البحث ذو كفاءة.

مزاي التسلسل:

من اقامة جدول البعثة باستخدام التسلسل نرى أن ادخال وحذف العنصر واضح. اذا كانت دالة البعثة جيدة يتم بعثة القليل من المفاتيح الى نفس الموضع. بهذا تكون القائمة المتصلة قصيرة في المتوسط وهذا يتسبب في طول بحثي أقصر. اذا كان حجم العنصر كبير فانه يحتفظ بمقدار كبير من المساحة. على سبيل المثال افترض أن هناك 1000 عنصر وكل عنصر يحتاج الى 10 كلمات للتخزين. (عادةً الكلمة الواحدة تعادل 4 بايت). افترض كذلك أن كل مؤشر يحتاج الى كلمة واحدة للتخزين. اذن نحن نحتاج الى 1000 كلمة من أجل جدول البعثة و 10000 كلمة من أجل العناصر، و 1000 كلمة من أجل الوصلة في كل عقدة. لهذا يكون هناك حاجة الى اجمالي 12000 كلمة من مساحة التخزين لتطبيق التسلسل. من الجهة الأخرى اذا استخدمنا البحث التربيعي واذا كان حجم جدول البعثة ضعف عدد العناصر فاننا نحتاج الى 20000 كلمة من التخزين.

مساوئ التسلسل:

إذا كان حجم العنصر صغير يتم تضييع قدر معقول من المساحة. على سبيل المثال افترض أن هناك 1000 عنصر وأن كل عنصر يحتاج إلى كلمة واحدة من التخزين. إذن التسلسل يحتاج إجمالي 3000 كلمة. من الجهة الأخرى ومع البحث التربيعي إذا كان حجم جدول البعثة ضعف عدد العناصر فإنه يكون هناك حاجة إلى 2000 كلمة فقط من أجل جدول البعثة. كذلك إذا كان حجم جدول البعثة ثلاثة أضعاف عدد العناصر إذن في البحث التربيعي تنتشر المفاتيح بشكل معقول. هذا يتسبب في تعارضات أقل وهكذا يكون البحث سريع.

تحليل البعثة:

لتكن:

$$\alpha = \frac{\text{Number of records in the table}}{HTSize}$$

إذن α تسمى عامل التحميل.

اثباتات النتائج التالية حول تحليلات البعثة متروكة كتمرين لك.

البحث الطولي:

متوسط عدد المقارنات للبحث الناجح والبحث الغير ناجح تكون كما يلي:

1. البحث الناجح

$$\frac{1}{2} \left\{ 1 + \frac{1}{1-\alpha} \right\}$$

2. البحث الغير ناجح

$$\frac{1}{2} \left\{ 1 + \frac{1}{(1-\alpha)^2} \right\}$$

البحث التربيعي:

متوسط عدد المقارنات للبحث الناجح والبحث الغير ناجح تكون كما يلي:

1. البحث الناجح

$$\frac{-\log_2(1-\alpha)}{\alpha}$$

2. البحث الغير ناجح

$$\frac{1}{1-\alpha}$$

التسلسل:

متوسط عدد المقارنات للبحث الناجح والبحث الغير ناجح تكون كما يلي:

1. البحث الناجح

$$1 + \frac{\alpha}{2}$$

2. البحث الغير ناجح

$$\alpha$$

مراجعة سريعة

1. القائمة عبارة عن مجموعة من العناصر من نفس النوع.

2. طول القائمة هو عدد العناصر في القائمة.

3. المصفوفة أحادية البعد عبارة عن مكان مناسب لتخزين ومعالجة القوائم.

4. خوارزمية البحث التتابعي تبحث القائمة عن عنصر محدد بدءاً من العنصر الأول في القائمة وتستمر لكي تقارن عنصر البحث مع العناصر في القائمة حتى يتم العثور على العنصر أو حتى لا يتم ترك عناصر في القائمة تتم مقارنتها.

5. في المتوسط تقوم خوارزمية البحث التتابعي يبحث نصف القائمة.

6. بالنسبة الى قائمة طولها n في بحث ناجح وفي المتوسط يقوم البحث التتابعي بعمل:

$$4. \frac{n+1}{2} = O(n)$$

7. البحث التتابعي غير فعال بالنسبة الى القوائم الكبيرة.

8. البحث الثنائي أسرع كثيراً من البحث التتابعي.

9. البحث الثنائي يقتضي أن تكون عناصر القائمة مرتبة.

10. بالنسبة الى قائمة طولها 1024 يحتاج البحث الثنائي للبحث عن عنصر في القائمة الى ملا يزيد

عن 11 مكرر من الحلقة وبهذا لا يزيد عن 22 مقارنة.

11. بالنسبة الى قائمة طولها n يقوم البحث الثنائي في البحث الناجح بعمل $2\log_2 n - 4 = O(\log_2 n)$ مقارنة مفتاح في المتوسط.

12. لتكن L قائمة حجمها $n > 1$. افترض أن تاعناصر في L مرتبة. اذا كان $SRH(n)$ هو العدد الأدنى من المقارنات اللازمة فانه في الحالة الأسوأ وباستخدام خوارزمية قائمة على المقارنة لادراك ما اذا كان العنصر x موجود في L اذن:

$$SRH(n) \geq \log_2(n + 1)$$

13. خوارزمية البحث الثنائي هي خوارزمية الحالة الأسوأ الفضاي لحل مشكلات البحث عن طريق استخدام أسلوب المقارنة.

14. لاقامة خوارزمية بحث ترتيبها لا يقل عن $\log_2 n!$ أن تكون قائمة على المقارنة.

15. في البعثة يتم تنظيم البيانات بمساعدة جدول يسمى جدول البعثة ويرمز اليه بالرمز HT ويتم تخزين جدول البعثة في مصفوفة.

16. لتحديد ما اذا كان هناك عنصر محدد مفتاحه هو X في جدول البعثة نقوم بتطبيق الدالة $h(X)$. الدالة h دالة حسابية و $h(X)$ تعطي عنوان العنصر في جدول البعثة.

17. في البعثة بما أن عنوان العنصر يتم حسابه بمساعدة دالة اذن العناصر يتم تخزينها بدون ترتيب محدد.

18. مفتاحان X_1 و X_2 بحيث $X_2 \neq X_1$ يسميان مرادفان اذا كانت $h(X_2) = h(X_1)$.

19. لتكن X مفتاح و $t = h(X)$. اذا كان المقدار t ممتلئاً نقول أنه حدث فائض.

20. لتكن X_1 و X_2 مفتاحان غير متطابقان. اذا كانت $h(X_2) = h(X_1)$ نقول أن هناك تعارض حدث. اذا كانت $r = 1$ أي أن حجم المقدار هو 1 يحدث فائض وتعارض في نفس الوقت.

21. يتم تصنيف أساليب حل الاعتراض الى فئتين: المعالجة المفتوحة (تسمى أيضاً بعثة مغلقة) والتسلسل (سمى أيضاً بعثة مفتوحة).

22. في المعالجة المفتوحة يتم تخزين البيانات داخل جدول البعثة.

23. في التسلسل يتم تنظيم البيانات في قوائم متصلة ويكون جدول البعثة عبارة عن مصفوفة من مؤشرات الى القوائم المتصلة.

24. في البحث الطولي اذا حدث تعارض عند الموضع t نقوم ببحث المصفوفة تتابعياً من الموضع t لاجاد مقطع المصفوفة التالي المتاح.

25. في البحث الطولي نفترض أن المصفوفة دائرية وأنه اذا كان الجزء الأدنى من المصفوفة ممتلئاً يمكننا الاستمرار في البحث في الجزء العلوي من المصفوفة. اذا حدث تعارض عند الموضع

t فإننا نتفقد مواضع المصفوفة بدءاً من t وهي t، و $HTSize \% (t + 1)$ ، و $t + 2$ ($HTSize \% (t + j)$ ،، هذا يطلق عليه التتابع البحثي.

26. البحث الطولي يسبب تجميع يطلق عليه تجميع أولي.

27. في البحث العشوائي يتم استخدام مولد اعداد عشوائية لاجاد المقطع التالي المتاح.

28. في اعادة البعثة اذا حدث تعارض مع دالة التعارض h نستخدم سلسلة من دالات البعثة.

29. في البحث الترتيبي اذا حدث تعارض عند الموضع فإننا نبحث المصفوفة طويلاً بدءاً من

الموضع t في المواضع $HTSize \% (t + 1)$ ، و $HTSize \% (t + 4)$ ، $HTSize \% (t + 2^2)$

، و $HTSize \% (t + 9)$ ، $HTSize \% (t + 3^2)$ و $HTSize \% (t + i^2)$ بع البحثي

هو: $HTSize \% (t + 1)$ ، $HTSize \% (t + 2^2)$ ، $HTSize \% (t + 3^2)$

$$HTSize \% (t + t^2)$$

30. كل من البحث العشوائي والتربيعي تقضي على التجميع الأولي. بالرغم من هذا اذا تمت بعثرة مفتاحان غير متطابقان X_1 و X_2 الى نفس الموضع أي أن $h(X_2) = h(X_1)$ فانه يتم نفس اتباع التتابع البحثي لكلا المفتاحين. هذا لأن البحث العشوائي والبحث التربيعي دالات للموضع وليس المفتاح الأصلي. اذا تسببت دالة البعثرة في تجميع عند موضع معين فان التجميع يظل أسفل حالات البحث هذه. هذا يسمى تجمع ثانوي.

31. من إحدى طرق حل التجمع الثانوي استخدام البحث الطولي حيث تكون قيمة الزيادة دالة للمفتاح. هذا يسمى بعثرة مزدوجة. في البعثرة المزدوجة اذا حدث تعارض عند $h(X)$ يتم توليد التتابع البحثي باستخدام القاعدة:

$$HTSize \% (h(X) + i * h'(X))$$

حيث h' هي دالة البعثرة الثانية.

32. في المعالجة المفتوحة عندما يتم حذف عنصر لا يمكن تحديد موضعه في المصفوفة كموضع فارغ.

33. في التسلسل وبالنسبة لكل مفتاح X (في العنصر) نقوم أولاً بإيجاد $t = h(X)$ حيث $0 \leq t \leq HTSize - 1$. العنصر صاحب هذا المفتاح يتم ادخاله في القائمة المتصلة (التي قد تكون فارغة) المشار اليها بواسطة $HT[t]$.

34. في التسلسل وبالنسبة الى المفاتيح الغير متطابقة X_1 و X_2 اذا كانت $h(X_2) = h(X_1)$ يتم ادخال العناصر ذات المفاتيح X_1 و X_2 في نفس القائمة المتصلة.

35. في التسلسل ومن أجل حذف عنصر R من جدول البعثرة نقوم أولاً يبحث جدول البعثرة للعثور على المكان الذي يوجد به R في القائمة المتصلة ثم نقوم بتعديل المؤشرات عند المواقع المناسبة وازالة تخصيص الذاكرة التي يشغلها R .

36. ليكن:

$$\alpha = \frac{\text{Number of records in the table}}{HTSize}$$

37. البحث الطولي: متوسط عدد المقارنات:

$$\frac{1}{2} \left\{ 1 + \frac{1}{1 - \alpha} \right\} \quad \text{البحث الناجح:}$$

$$\frac{1}{2} \left\{ 1 + \frac{1}{1 - \alpha} \right\} \quad \text{البحث الغير ناجح:}$$

38. البحث التربيعي: متوسط عدد المقارنات:

$$\frac{-\log_2(1-\alpha)}{\alpha} \quad \text{البحث الناجح:}$$

$$\frac{1}{1-\alpha} \quad \text{البحث الغير ناجح:}$$

39. التسلسل: متوسط عدد المقارنات:

$$1 + \frac{\alpha}{2} \quad \text{البحث الناجح:}$$

$$\alpha \quad \text{البحث الغير ناجح:}$$

التمارين:

1. ضع علامة صح أم خطأ أمام العبارات التالية:
 - أ- البحث التتابعي للقائمة يفترض أن القائمة مرتبة ترتيب تصاعدي.
 - ب- البحث الثنائي للقائمة يفترض أن القائمة مرتبة.
 - ت- البحث الثنائي أسرع على القوائم المرتبة وأبطأ على القوائم الغير مرتبة.
 - ث- البحث الثنائي أسرع على القوائم الكبيرة ولكن البحث التتابعي أسرع على القوائم الصغيرة.
2. انظر الى القائمة التالية:

100 28 57 46 98 32 45 63

باستخدام البحث التتابعي كما تم توضيحه في هذا الفصل كم عدد المقارنات اللازمة لايجاد ما اذا كانت العناصر التالية موجودة في القائمة؟ (تذكر أننا نقصد بالمقارنات مقارنات العناصر وليس مقارنات المؤشر).

أ- 90

ب- 57

ت- 63

ث- 120

3. انظر الى القائمة التالية:

110 98 92 85 68 55 49 45 17 10 2

باستخدام البحث الثنائي كما تم توضيحه في هذا الفصل كم عدد المقارنات اللازمة لايجاد ما اذا كانت العناصر التالية موجودة في القائمة؟ أوضح قيم first و last و middle وعدد المقارنات بعد كل مكرر للحلقة.

أ- 25

ب- 49

ت- 98

ث- 99

4. اكتب تعريف الفئة `orderedArrayListType` التي تطبق خوارزميات البحث لقوائم مبنية على المصفوفات كما نوقش في هذا الفصل.

5. افترض أن حجم جدول البعثة هو 150 وحجم المقدار هو 5. كم عديد المقادير في جدول البعثة وكم عدد العناصر التي يمكن للمقدار ضمها؟

6. وضح كيف يتم حل التعارض باستخدام البحث الطولي.

7. وضح كيف يتم حل التعارض باستخدام البحث التريبيعي.

8. ما هي البعثة المزدوجة؟

9. افترض أن حجم جدول البعثة هو 101 والعناصر يتم ادخالها في الجدول باستخدام البحث التريبيعي وافترض كذلك أنه يتم ادخال عنصر جديد في الجدول وعنوانه في جدول البعثة هو 30. اذا كان الموضع 30 في جدول البعثة مشغولاً والأربعة مواضع التالية المعطاة بواسطة التتابع البحثي مشغولة كذلك. قم بتحديد المكان الذي يتم ادخال العنصر فيه في الجدول.

10. افترض أن حجم جدول البعثة هو 101 وافترض كذلك أنه يتم ادخال مفاتيح معينة مؤشرات لها 15، و101، و116، و0، و172 بهذا الترتيب داخل جدول بعثة فارغ مبدئياً. باستخدام حساب الوحدات اوجد المؤشرات في جدول البعثة اذا:

أ- تم استخدام البحث الطولي.

ب- م استخدام البحث التريبيعي.

11. افترض أنه يتم ادخال 50 مفتاح داخل جدول بعثة فارغ مبدئياً باستخدام البحث التريبيعي. ما هو حجم جدول البعثة اللازم لضمان أن يتم حل جميع التعارضات؟

12. افترض أنه يتم ازالة عنصر من جدول البعثة تم تطبيقه باستخدام بحث طولي أو تريبيعي. لماذا لم تحدد موضع العنصر الذي يتم حذفه بأنه فارغ؟

13. ما هي مزايا البعثة المفتوحة؟

14. أعطى مثال رقمي لتوضيح أن حل التعارض باستخدام البحث التريبيعي أفضل من التسلسل.

15. أعطى مثال رقمي لتوضيح أن حل التعارض باستخدام التسلسل أفضل من البحث التريبيعي.

16. افترض أن حجم جدول البعثة هو 1001 والجدول به 850 عنصر. ما هي دالة التحميل؟

17. افترض أن حجم جدول البعثة هو 1001 والجدول به 500 عنصر. في المتوسط كم عدد المقارنات التي يتم عملها لتحديد ما اذا كان العنصر موجود في القائمة اذا:

أ- تم استخدام البحث الطولي؟

ب- تم استخدام البحث التريبيعي.

ت- تم استخدام التسلسل؟

18. افترض أن هناك 550 عنصر يتم تخزينهم في جدول بعثرة. اذا كان هناك حاجة الى ثلاث مقارنات مفاتيح في المتوسط لتحديد ما اذا كان العنصر موجود في الجدول. ما هو حجم جدول البعثرة الذي يجب أن يكون عليه اذا:

أ- تم استخدام البحث الطولي؟

ب- تم استخدام التسلسل؟

ت-

تمارين برمجة

1. (البحث التتابعي التكراري) خوارزمية البحث التكراري المعطاة في هذا الفصل غير تكرارية. قم بكتابة وتطبيق نسخة تكرارية من خوارزمية البحث التتابعي.

2. (البحث الثنائي التكراري) خوارزمية البحث الثنائي المعطاة في هذا الفصل غير تكرارية. قم بكتابة وتطبيق نسخة تكرارية من خوارزمية البحث الثنائي. اكتب أيضاً نسخة من خوارزمية بحث تتابعي يمكن تطبيقها على القوائم المرتبة. أضف هذه العملية الى الفئة `orderedArrayListType` للقوائم المبنية على المصفوفات. فضلاً عن هذا اكتب برنامج اختبار لاختبار الخوارزمية الخاص بك.

3. خوارزمية البحث التتابعي كما هي معطاة في هذا الفصل لا تفترض أن القائمة مرتبة. لهذا فانها تعمل بنفس الشكل بالنسبة الى كل من القوائم المرتبة والغير مرتبة. بالرغم من هذا اذا كانت عناصر القائمة مرتبة يمكنك نوعاً ما تحسين أداء خوارزمية البحث التتابعي. على سبيل المثال اذا لم يكن عنصر البحث موجود في القائمة يمكنك ايقاف البحث بمجرد أن تجد عنصر في القائمة أكبر من عنصر البحث. اكتب الدالة `seqOrdSearch` لتطبيق نسخة من خوارزمية البحث التتابعي على القوائم المرتبة. أضف هذه الدالة الى الفئة `orderedArrayListType` واكتب برنامج لاختبارها.

4. اكتب برنامج لايجاد عدد المقارنات باستخدام البحث الثنائي وخوارزمية البحث التتابعي كما يلي:

افترض أن `list` عبارة عن مصفوفة مكونة من 1000 عنصر.

أ- استخدم مولد أعداد عشوائية لملأ القائمة.

ب- استخدم أي خوارزمية ترتيب لترتيب القائمة. يمكنك استخدام الدالة `insertOrd` لادخال جميع العناصر في القائمة مبدئياً.

ت- ابحث القائمة من أجل بعض العناصر كما يلي:

1) استخدم خوارزمية البحث الثنائي لبحث القائمة. (قد تحتاج الى تعديل الخوارزمية

المعطاة في هذا الفصل لاحصاء عدد المقارنات).

(2) استخدم خوارزمية البحث الثنائي لبحث القائمة والانتقال الى البحث التتابعي عند تقليل

حجم قائمة البحث الى أقل من 15. (استخدم خوارزمية البحث التتابعي لقائمة مرتبة).

ث- اطبع عدد المقارنات للخطوات ت.ا وت.2. اذا تم العثور على العنصر في القائمة اطبع موضعه.

5. اكتب برنامج لاختبار الدالة insertOrd التي تدخل عنصر داخل قائمة مرتبة قائمة على مصفوفة.

6. اكتب الدالة removeOrd التي تحذف عنصر من قائمة مرتبة قائمة على مصفوفة. العنصر الذي يتم حذفه يتم تمريره كمعامل لهذه الدالة. بعد حذف العنصر يجب أن تكون القائمة الناتجة مرتبة مع عدم وجود مواضع فارغة بالمصفوفة بين العناصر. أضف هذه الدالة الى الفئة orderedArrayListType و اكتب برنامج لاختبارها.

7. اكتب تعريفات الدالات search، isItemAtEqual، retrieve، remove، و print، والمقوم، والمدمر للفئة hashT كما نوقش في القسم "البعثة: التطبيق باستخدام البحث التريبيعي" من هذا الفصل. اكتب أيضاً برنامج لاختبار عمليات بعثة متنوعة.

8. أ. من بعض ما تمتلكه أي ولاية في الولايات المتحدة اسمها، وعاصمتها، ومساحتها، وعام انضمامها الى الاتحاد، وأمر انضمامها الى الاتحاد. قم بتصميم الفئة stateData لتتبع معلومات الولاية. يجب أن تتضمن فئتك دالات مناسبة للتحكم في بيانات الولاية مثل الدالات setStateInfo و getStateInfo وغيرها. قم كذلك باثقال معاملات مترابطة للمقارنة بين ولايتين بأسمائهما. لسهولة الادخال والاخراج قم باثقال معاملات التدفق.

ب. استخدم الفئة hashT كما تم وصفها في القسم "البعثة: التطبيق باستخدام البحث التريبيعي" التي تستخدم البحث التريبيعي لحل التعارض لعمل جدول بعثة لتتبع معلومات كل ولاية. استخدم اسم الولاية كمفتاح لتحديد عنوان البعثة. قد تفترض أن اسم الولاية عبارة عن مقطع لا يزيد عن 15 رمز. قم باختبار برنامجك عن طريق البحث عن وازالة ولايات معينة من جدول البعثة. قد تستخدم دالة البعثة التالية لتحديد عنوان بعثة عنصر ما:

```
int hashFunc(string name)
{
    int i, k, sum;
    int len;
    i = 0;
    sum = 0;
    len = name.length();
    for(k = 0; k < 15 - len; k++)
        name = name + ' '; //increase the length of the name
                                //to 15 characters
    for(k = 0; k < 5; k++)
    {
        sum = sum + static_cast<int>(name[i]) * 128 * 128
                + static_cast<int>(name[i + 1]) * 128
                + static_cast<int>(name[i + 2]);
        i = i + 3;
    }
    return sum % HTSize;
}
```

الفصل

10

خوارزميات الترتيب

في هذا الفصل سوف:

- تتعلم خوارزميات ترتيب متنوعة.
- تكتشف كيفية تطبيق خوارزميات الاختيار، والادخال، والدمج السريع، والترتيب المكس.
- تكتشف كيفية أداء خوارزميات الترتيب التي نوقشت في هذا الفصل.
- تتعلم كيفية تطبيق طوابير الأسبقية.

الفصل 9 ناقش خوارزميات البحث على القوائم. البحث التتابعي لا يفترض أن البيانات مرتبة أي ترتيب معين وبالرغم من هذا فإن هذا البحث كما لوحظ لن يعمل بكفاءة بالنسبة الى القوائم الكبيرة. على النقيض البحث الثنائي سريع للغاية بالنسبة الى القوائم المبنية على المصفوفات ولكنه يحتاج الى أن تكون البيانات مرتبة. بما أن البحث الثنائي يقتضي أن تكون البيانات مرتبة وأداؤه جيد في القوائم المبنية على المصفوفة فإن هذا الفصل يقوم بالتركيز على خوارزميات الترتيب.

خوارزميات البحث:

هناك العديد من خوارزميات البحث في الكتب وفي هذا الفصل نناقش بعض من الخوارزميات الشائعة الاستخدام. للمقارنة بين أداء هذه الخوارزميات نقدم كذلك بعض من التحليل لهذه الخوارزميات. يمكن تطبيق خوارزميات البحث هذه على القوائم المبنية على المصفوفات أو على القوائم المتصلة. سوف نقوم بتحديد ما اذا كانت الخوارزمية تم عملها من أجل القوائم القائمة على المصفوفة أو القوائم المتصلة.

الدالات المطبقة لخوارزميات البحث هذه موجودة في العناصر العامة للفئة المرتبطة بها. (على سبيل المثال هناك عناصر الفئة `orderedArrayListType` بالنسبة الى القوائم المبنية على المصفوفات). بالقيام بهذا يكون للخوارزميات تناول مباشر لعناصر القائمة.

افترض أن نوع اختيار خوارزمية الترتيب (الذي تم وصفه في القسم التالي) سوف يتم تطبيقه على القوائم المبنية على المصفوفات. البيانات التالية توضح كيفية تضمين نوع اختيار كعنصر من الفئة

`:orderedArrayListType`

```
template<class elemType>
class orderedArrayListType: public arrayListType
{
public:
    void selectionSort();
    ...
};
```

ترتيب الاختيار: القوائم المبنية على المصفوفة:

خوارزمية ترتيب الاختيار تقوم بترتيب القائمة عن طريق اختيار أصغر عنصر في (الجزء الغير مرتب من) القائمة ثم تنقل هذا العنصر الأصغر الى قمة القائمة (الغير مرتبة). أول مرة نضع العنصر الأصغر في القائمة كلها والمرة الثانية نضع العنصر لأصغر في القائمة بدءاً من العنصر الثاني في القائمة وهكذا. خوارزمية نوع الاختيار الموصوفة هنا مصممة من أجل القوائم المبنية على المصفوفات.

على سبيل المثال افترض أن لديك القائمة الموضحة في شكل 1-10.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 16 | 30 | 24 | 7 | 25 | 62 | 45 | 5 | 65 | 50 |

شكل 1-10: قائمة من 10 عناصر

مبدئياً تكون القائمة كلها غير مرتبة ولهذا نوجد أصغر عنصر في القائمة وهو الموجود في الموضع 7 كما هو موضح في الشكل 2-10.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 16 | 30 | 24 | 7 | 25 | 62 | 45 | 5 | 65 | 50 |

smallest
↓
[7]

← unsorted list →

شكل 2-10: أصغر عنصر بالقائمة الغير مرتبة

بما أن هذا هو أصغر عنصر اذن يجب نقله الى الموضع صفر ولهذا نقوم بمبادلة 16 (أي [0] list) مع 5 (أي [7] list) كما هو موضح في شكل 3-10.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 16 | 30 | 24 | 7 | 25 | 62 | 45 | 5 | 65 | 50 |

swap
↔

← unsorted list →

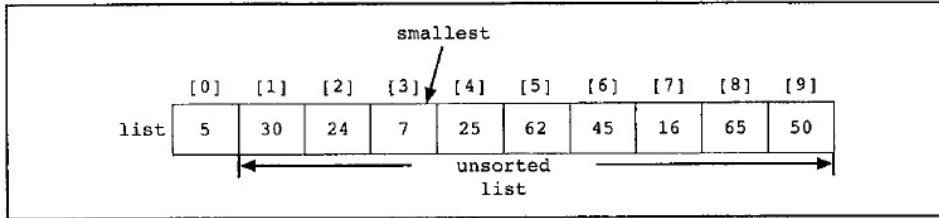
شكل 3-10: تبادل العناصر [0] list و [7] list.

بعد تبادل هذه العناصر يوضح الشكل 4-10 القائمة الناتجة.

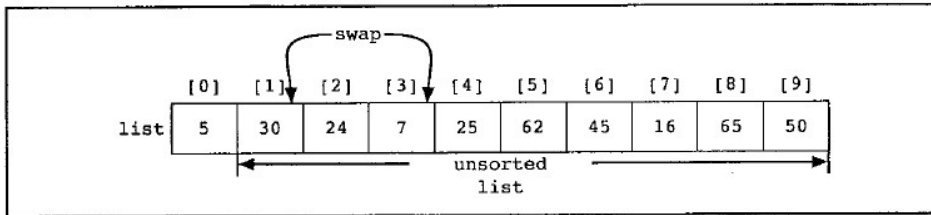
| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 5 | 30 | 24 | 7 | 25 | 62 | 45 | 16 | 65 | 50 |

← unsorted list →

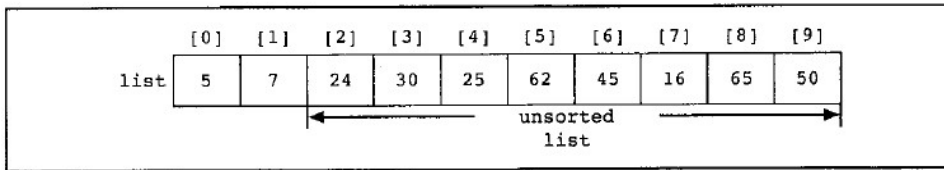
شكل 10-4: القائمة بعد تبادل list [0] و list [7]
الآن القائمة الغير مرتبة هي list [9]...list [1]. بعد هذا نوجد العنصر الأصغر في الجزء الغير مرتب من القائمة وهو موجود في الموضع 3 كما هو موضح في شكل 10-5.



شكل 10-5: أصغر عنصر في الجزء الغير مرتب من القائمة
بما أن العنصر الأصغر في القائمة الغير مرتبة موجود في الموضع 3 اذن يجب نقله الى الموضع 1. هذا يعني أننا نقوم بتبادل 7 (أي list [3]) مع 30 (أي list [1]) كما هو موضح في شكل 10-6.



شكل 10-6: تبادل list [1] مع list [3]
بعد تبادل list [1] مع list [3] الشكل 10-7 يوضح القائمة الناتجة.



شكل 10-7: القائمة بعد تبادل list [1] مع list [3]
الآن القائمة الغير مرتبة هي list [9] ... list [2]. نقوم بتكرار هذه العملية الخاصة بإيجاد (موضع) أصغر عنصر في الجزء الغير مرتب من القائمة ونقله الى بداية القائمة الغير مرتبة. بهذا تتضمن خوارزمية الترتيب الاختياري الخطوات التالية.

في الجزء الغير مرتب من القائمة:

أ- اوجد موضع أصغر عنصر.

ب- قم بنقل العنصر الأصغر الى بداية القائمة الغير مرتبة.

مبدئياً القائمة بأكملها $list[0] \dots list[length - 1]$ تكون هي القائمة الغير مرتبة. بعد تنفيذ الخطوات أ وب مرة واحدة تكون القائمة الغير مرتبة هي $list[1] \dots list[length - 1]$. بعد تنفيذ الخطوات أ وب مرة أخرى تكون القائمة الغير مرتبة هي $list[2] \dots list[length - 1]$ وهكذا. يمكننا تتبع الجزء الغير مرتب من القائمة وتكرار الخطوات أ وب بمساعدة الحلقة for كما يلي:

```
for(index = 0; index < length - 1; index++)
{
    a. Find the location, smallestIndex, of the smallest element in
       list[index]...list[length - 1].
    b. Swap the smallest element with list[index]. That is, swap
       list[smallestIndex] with list[index].
}
```

المرة الأولى خلال الحلقة نضع أصغر عنصر في $list[0] \dots list[length - 1]$ ونستبدل هذا العنصر الأصغر مع $list[0]$. المرة الثانية خلال الحلقة نضع أصغر عنصر في $list[1] \dots list[length - 1]$ ونستبدل هذا العنصر الأصغر مع $list[1]$ وهكذا. هذه العملية تستمر حتى يكون طول القائمة الغير مرتبة 1. (لاحظ أن القائمة التي طولها 1 تكون مرتبة). ينتج اذن أنه لتطبيق خوارزمية ترتيب الاختيار نحتاج الى تطبيق الخطوات أ وب.

باعطاء مؤشر البداية first ومؤشر النهاية last للقائمة تقوم دالة C++ التالية بانتاج مؤشر أصغر عنصر في $list[first] \dots list[last]$:

```
template<class elemType>
int orderedArrayListType<elemType>::minLocation(int first, int last)
{
    int loc, minIndex;
    minIndex = first;

    for(loc = first + 1; loc <= last; loc++)
        if(list[loc] < list[minIndex])
            minIndex = loc;

    return minIndex;
} //end minLocation
```

باعطاء المواضع في القائمة للعناصر التي يتم استبدالها تقوم دالة C++ التالية بتبادل هذه العناصر:

```
template<class elemType>
void orderedArrayListType<elemType>::swap(int first, int second)
{
    elemType temp;

    temp = list[first];
    list[first] = list[second];
    list[second] = temp;
} //end swap
```

يمكننا الآن اكمال تعريف الدالة selectionSort.

```

template<class elemType>
void orderedArrayListType<elemType>::selectionSort()
{
    int loc, minIndex;
    for(loc = 0; loc < length - 1; loc++)
    {
        minIndex = minLocation(loc, length - 1);
        swap(loc, minIndex);
    }
}

```

تعريف الفئة orderedArrayListType لتطبيق خوارزمية ترتيب الاختيار يكون كما يلي:

```

template<class elemType>
class orderedArrayListType: public arrayListType<elemType>
{
public:
    void insertOrd(const elemType&);
    int binarySearch(const elemType& item);
    void selectionSort();

    orderedArrayListType(int size = 100);

private:
    void swap(int first, int second);
    int minLocation(int first, int last);
};

```

مثال 1-10:

البرنامج التالي يختبر خوارزمية ترتيب الاختيار. نفترض أن تعريف الفئة orderedArrayListType.h هو في الملف الرئيسي

```

#include <iostream>
#include "orderedArrayListType.h"

using namespace std;

int main()
{
    orderedArrayListType<int> list;           //Line 1
    int num;                                  //Line 2

    cout<<"Line 3: Enter numbers ending with -999"
        <<endl;                               //Line 3

    cin>>num;                                 //Line 4
}

```

```

while(num != -999)                                //Line 5
{
    list.insert(num);                               //Line 6
    cin>>num;                                       //Line 7
}

cout<<"Line 8: The list before sorting:"<<endl;    //Line 8
list.print();                                       //Line 9
cout<<endl;                                         //Line 10

list.selectionSort();                               //Line 11

cout<<"Line 12: The list after sorting:"<<endl;    //Line 12
list.print();                                       //Line 13
cout<<endl;                                         //Line 14

return 0;
}

```

تنفيذ العينة: في تنفيذ هذه العينة يتم تظليل مدخلات المستخدم.

الصف 3: ادخل أعداد منتهية ب-999

999- 66 32 5 79 36 56 78 12 23 67 34

الصف 8: القائمة قبل الترتيب:

66 32 5 79 36 56 78 12 23 67 34

الصف 12: القائمة بعد الترتيب:

79 78 67 66 56 36 34 32 23 12 5

بالنسبة الى الجزء الأكبر فان المخرجات السابقة تفسر نفسها. لاحظ أن البيان في الصف 6 تستدعي الدالة insert من الفئة arrayListType هي الفئة الأساسية للفئة orderedArrayListType. يالمثل تقوم البيانات في الصفين 9 و 13 باستدعاء الدالة print من الفئة arrayListType. البيان في الصف 11 يستدعي الدالة selectionSort من الفئة orderedArrayListType لترتيب القائمة.

1. يمكن كذلك تطبيق الترتيب بالاختيار عن طريق اختيار أكبر عنصر في (الجزء الغير مرتب من) القائمة ونقله الى قاع القائمة. يمكنك تطبيق هذه الصيغة من الترتيب بالاختيار بسهولة عن طريق تعديل البيان if في الدالة minLocation وتميرير المعاملات المناسبة الى الدالة المقابلة والدالة swap عندما يتم استدعاء هذه الدالات في الدالة selectionSort.
2. يمكن كذلك تطبيق الترتيب بالاختيار على القوائم المتصلة. الخوارزمية العامة واحدة والتفاصيل متروكة كتمرين لك. انظر تمرين البرمجة 1 في نهاية هذا الفصل.

تحليل: الترتيب بالاختيار:

في حالة خوارزمية البحث (الفصل 9) كان اهتمامنا الوحيد هو عدد مقارنات المفاتيح. خوارزمية البحث تقوم بعمل مقارنات مفتاح وتقوم كذلك بنقل البيانات. لهذا في تحليل خوارزمية الترتيب ننظر الى عدد مقارنات المفتاح وكذلك عدد حركات البيانات. لننظر الى الترتيب بالاختيار. افترض أن طول القائمة هو n . الدالة swap تقوم بعمل ثلاث اسنادات للعنصر ويتم تنفيذها عدد $n - 1$ ومن هنا يكون عدد اسنادات العنصر $3(n - 1)$.

مقارنات المفتاح يتم عملها بواسطة الدالة minLocation. بالنسبة الى قائمة طولها k تقوم الدالة minLocation بعمل مقارنات مفاتيح تبلغ $1 - k$. كذلك يتم تنفيذ الدالة minLocation مرات تبلغ $1 - n$ (بالدالة selectionSort). المرة الأولى تجد الدالة minLocation مؤشر العنصر الأصغر في القائمة بأكملها ثم تقوم بعمل $1 - n$ مقارنة. المرة الثانية تقوم الدالة minLocation بإيجاد مؤشر أصغر عنصر في القائمة الفرعية التي طولها $1 - n$ ولهذا تقوم بعمل مقارنات تبلغ $2 - n$ وهكذا. من هنا يكون عدد مقارنات المفتاح كما يلي:

$$\begin{aligned} (n-1) + (n-2) + \dots + 2 + 1 &= \frac{n(n-1)}{2} \\ &= \frac{1}{2}n^2 - \frac{1}{2}n \\ &= \frac{1}{2}n^2 + O(n) \\ &= O(n^2). \end{aligned}$$

ينتج من هذا أنه اذا كانت $n = 1000$ فان عدد مقارنات المفتاح التي تقوم خوارزمية الترتيب بالاختيار

$$\frac{1}{2}(1000)^2 - \frac{1}{2}(1000) = 499500 \approx 500000 \text{ بعمله هو}$$

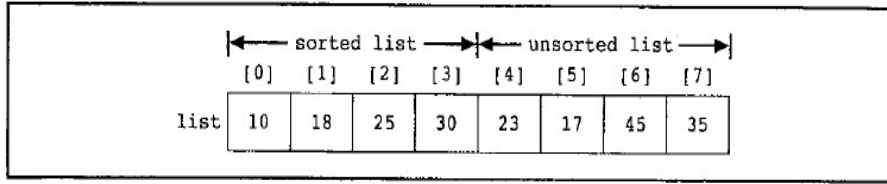
الترتيب بالادخال: القوائم المبنية على المصفوفة:

القسم السابق وصف وحل خوارزمية الترتيب بالاختيار وتم ايضاح أنه اذا كانت $n = 1000$ فان عدد مقارنات المفتاح يكون تقريباً 500.000 وهذا رقم مرتفع. هذا القسم يصف خوارزمية الترتيب المسماة الترتيب بالادخال والتي تحاول تحسين – أي تقليل – عدد مقارنات المفتاح. خوارزمية الترتيب بالادخال تقوم بترتيب القائمة عن طريق نقل كل عنصر الى مكانه الصحيح. انظر الى القائمة المعطاة في شكل 10-8.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 10 | 18 | 25 | 30 | 23 | 17 | 45 | 35 |

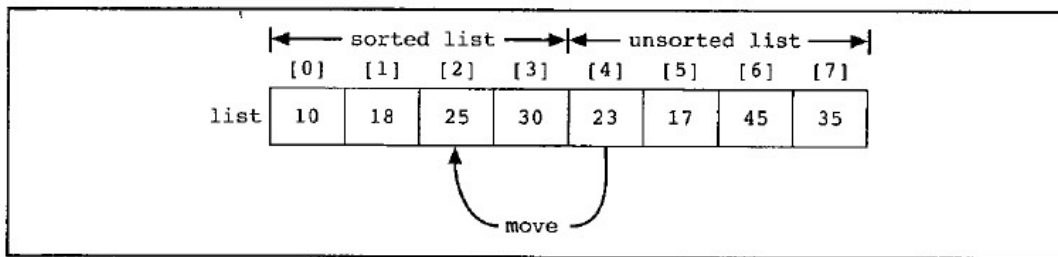
شكل 10-8: قائمة

طول القائمة هو 8 وفي هذه القائمة تكون العناصر list [0] و list [1] و list [2] و list [3] مرتبة. هذا يعني أن list [0] ... list [3] مرتبة (انظر شكل 10-9).



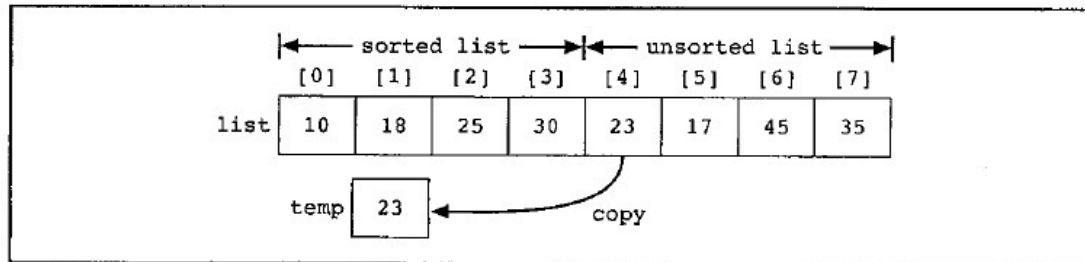
شكل 10-9: الجزء المرتب والجزء الغير مرتب من القائمة

بعد هذا ننظر الى العنصر [4] list وهو العنصر الأول في القائمة الغير مرتبة. بما أن $list[3] < list[4]$ اذن نحتاج الى تحريك العنصر [4] list الى مكانه المناسب. ينتج من هذا أن العنصر [4] list يجب نقله الى [2] list (انظر شكل 10-10).



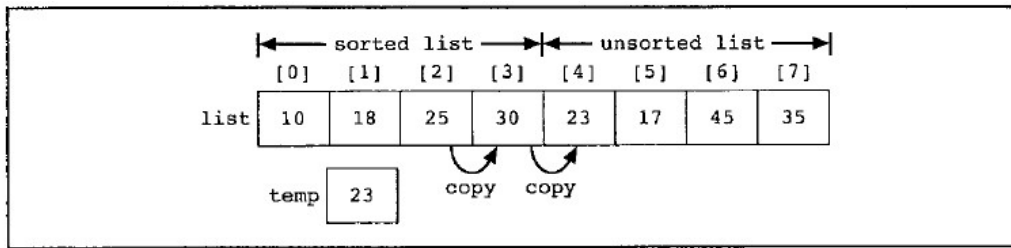
شكل 10-10: نقل [4] list بداخل [2] list

لنقل [4] list داخل [2] list نقوم أولاً بنسخ [4] list داخل temp وهي مساحة ذاكرة مؤقتة (انظر شكل 10-11).

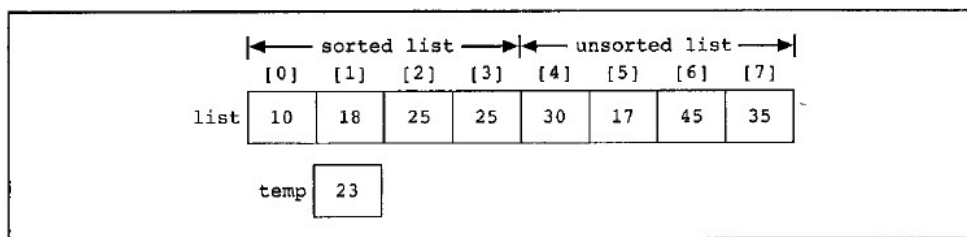


شكل 10-11: نسخ [4] list داخل temp.

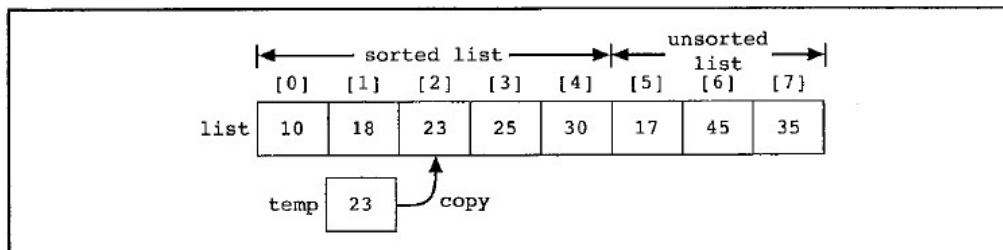
بعد ذلك نقوم بنسخ [3] list داخل [4] list ثم [2] List داخل [3] list (انظر شكل 10-12).



شكل 10-12: القائمة قبل نسخ list [3] داخل list [4] ثم list [2] داخل list [3]
 بعد نسخ list [3] داخل list [4] ثم list [2] داخل list [3] تكون القائمة كما هي موضحة في شكل 10-13.



شكل 10-13: القائمة بعد نسخ list [3] داخل list [4] ثم list [2] داخل list [3]
 الآن نقوم بنسخ temp داخل list [2] يوضح القائمة الناتجة.



شكل 10-14: القائمة بعد نسخ temp داخل list [2]

الآن list [0] ... list [4] مرتبة والقائمة من list [5] ... list [7] غير مرتبة. اننا نكرر هذه العملية على القائمة الناتجة عن طريق نقل العنصر الأول من القائمة الغير مرتبة الى القائمة المرتبة في المكان المناسب.

من هذه المناقشة نرى أن أثناء مرحلة الترتيب يتم تقسيم المصفوفة المحتوية على القائمة الى قائمتين فرعيتين علوية وسفلية. العناصر في القائمة الفرعية العلوية مرتبة والعناصر في القائمة الفرعية السفلية يتم نقلها الى القائمة الفرعية العلوية في أماكنها المناسبة عنصر واحد في كل مرة. اننا نستخدم مؤشر ليكن firstOutOfOrder للإشارة الى العنصر الأول في القائمة الفرعية السفلية وهذا يعني أن

firstOutOfOrder يعطي مؤشر العنصر الأول في الجزء الغير مرتب من المصفوفة. مبدئياً تتم تهيئة firstOutOfOrder عند 1.

تتم ترجمة هذه المناقشة الى الخوارزمية الوهمية التالية:

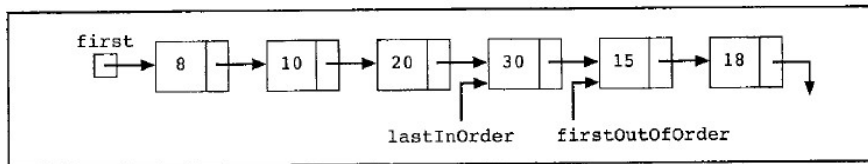
```
for(firstOutOfOrder = 1; firstOutOfOrder < length; firstOutOfOrder++)
    if(list[firstOutOfOrder] is less than list[firstOutOfOrder - 1])
    {
        copy list[firstOutOfOrder] into temp
        initialize location to firstOutOfOrder
        do
        {
            a. move list[location - 1] one array slot down
            b. decrement location by 1 to consider the next element
              of the sorted portion of the array
        }
        while(location > 0 && the element in the upper sublist at
              location - 1 is greater than temp)
    }
    copy temp into list[location]
```

لنقم بتتبع تنفيذ هذه الخوارزمية على القائمة المعطاة في شكل 10-15.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 13 | 7 | 15 | 8 | 12 | 30 | 3 | 20 |

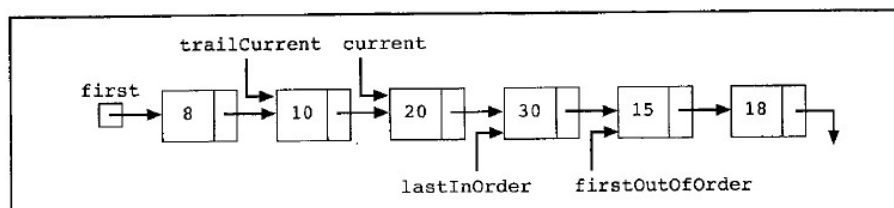
شكل 10-15: قائمة غير مرتبة

طول هذه القائمة هو 8 أي أن length = 8. نقوم بتهيئة firstOutOfOrder عند 1 (انظر الشكل 10-16).

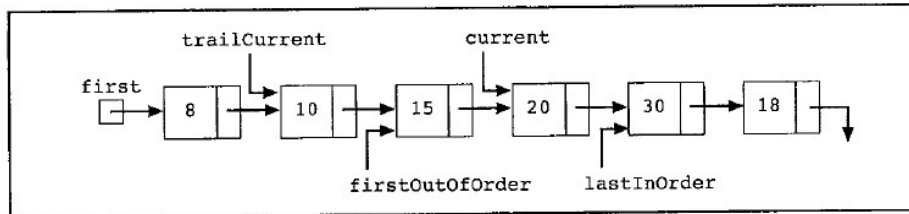


شكل 10-27: القائمة المتصلة و المؤشران lastInOrder و firstOutOfOrder

بما أن info -> firstOutOfOrder أكبر من info -> first فاننا نبحث القائمة ليجاد المكان الذي يتم نقل firstOutOfOrder اليه. كما في الحالة 2 نستخدم المؤشرين trailCurrent و current لاجتياز القائمة. بالنسبة الى هذه القائمة تنتهي هذه المؤشرات عند العقد كما هو موضح في شكل 10-28.



شكل 10-28: القائمة المتصلة والمؤشران `current` و `trailCurrent`.
الآن يتم نقل `firstOutOfOrder` بين `current` و `trailCurrent`. لهذا نقوم بتعديل الوصلات اللازمة
ونحصل على القائمة كما هي موضحة في شكل 10-29.



شكل 10-29: القائمة المتصلة بعد نقل `firstOutOfOrder` بين `current` و `trailCurrent`.

نقوم الآن بكتابة دالة C++ وهي linkedInsertionSort لتطبيق الخوارزمية السابقة.

```

template<class elemType>
void orderedLinkedListType<elemType>::linkedInsertionSort()
{
    nodeType<elemType> *lastInOrder;
    nodeType<elemType> *firstOutOfOrder;
    nodeType<elemType> *current;
    nodeType<elemType> *trailCurrent;

    lastInOrder = first;

    if(first == NULL)
        cerr<<"Cannot sort an empty list."<<endl;
    else
        if(first->link == NULL)
            cout<<"The list is of length 1. "
                <<"It is already in order."<<endl;
        else
            while(lastInOrder->link != NULL)
            {
                firstOutOfOrder = lastInOrder->link;
                if(firstOutOfOrder->info < first->info)
                {
                    lastInOrder->link = firstOutOfOrder->link;
                    firstOutOfOrder->link = first;
                    first = firstOutOfOrder;
                }
                else
                {
                    trailCurrent = first;
                    current = first->link;
                    while(current->info < firstOutOfOrder->info)
                    {
                        trailCurrent = current;
                        current = current->link;
                    }

                    if(current != firstOutOfOrder)
                    {
                        lastInOrder->link = firstOutOfOrder->link;
                        firstOutOfOrder->link = current;
                        trailCurrent->link = firstOutOfOrder;
                    }
                    else
                        lastInOrder = lastInOrder->link;
                }
            }
        }
    }
}

```

تترك لك كتارين كتابة برنامج لاختبار خوارزمية الترتيب بالادخال. انظر تمارين البرمجة 2 و3 في نهاية هذا الفصل.

تحليل: الترتيب بالادخال:

يمكن توضيح أن متوسط عدد مقارنات المفتاح ومتوسط عدد اسنادات العنصر في خوارزمية الترتيب بالادخال هو:

$$\frac{1}{4}n^2 + O(n) = O(n^2)$$

الجدول 1-10 يلخص سلوك خوارزميات الترتيب بالاختيار والادخال.

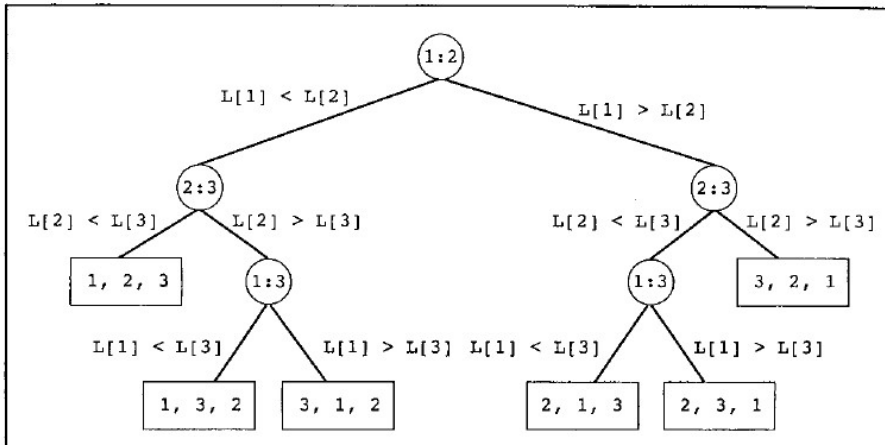
جدول 1-10: متوسط سلوك الحالة لخوارزميات الترتيب بالاختيار والادخال لقائمة طولها n.

| الخوارزمية | عدد المقارنات | عدد التبادلات |
|-------------------|---------------|---------------|
| الترتيب بالاختيار | **** | **** |
| الترتيب بالادخال | **** | **** |

الحد الأدنى على خوارزميات الترتيب القائمة على المقارنة:

الأقسام السابقة ناقشت خوارزميات الترتيب بالاختيار والادخال ولاحظ أن متوسط سلوك هذه الخوارزميات $O(n^2)$ من هاتين الخوارزميتين عبارة عن خوارزميات مبنية على المقارنة أي أنه يتم ترتيب القوائم عن طريق مقارنة مفاتيحهم. قبل مناقشة أي خوارزميات ترتيب إضافية أخرى لنقم بمناقشة السيناريو الأفضل لخوارزميات الترتيب المبنية على المقارنة.

يمكننا تتبع تنفيذ الخوارزمية المبنية على المقارنة عن طريق استخدام شكل يسمى **شجرة المقارنة**. لتكن L قائمة بها عدد n من العناصر حيث $n > 0$. بالنسبة إلى أي j و k حيث $1 \leq j, k \leq n$ تكون اما $L[j] < L[k]$ أو $L[j] > L[k]$. بما أن كل مقارنة للمفاتيح لها نتيجتين فان شجرة المقارنة عبارة عن شجرة ثنائية. أثناء رسم هذه الشكل نرسم كل مقارنة كدائرة تسمى **عقدة**. يتم تسمية العقدة $j:k$ التي تمثل مقارنة $L[j]$ مع $L[k]$. اذا كانت $L[j] < L[k]$ اتبع الفرع الأيسر وبخلاف هذا اتبع الفرع الأيمن. الشكل 10-30 يوضح شجرة المقارنة لقائمة طولها 3. (في شكل 10-30 المستطيل المسمى ورقة يمثل الترتيب النهائي للعقد).



شكل 10-30: شجرة مقارنة لترتيب ثلاثة عناصر

اننا نطلق على العقدة العليا في الشكل العقدة الجذرية. الخط المستقيم الذي يصل بين العقدتين يسمى فرع وتتابع الأفرع من العقدة x الى عقدة أخرى y يسمى مسار من x الى y. هناك تبديل فريد لعناصر القائمة L مرتبط بكل مسار من الجذر الى الورقة. هذا التفرد ينتج لأن خوارزمية الترتيب تقوم فقط بنقل البيانات وتصنع مقارنات. فضلاً عن هذا فان حركة البيانات على أي مسار من الجذر الى احدى الأوراق متماثلة بصرف النظر عن المدخلات الأولية. بالنسبة الى قائمة بها عدد n من العناصر و $n > 0$ هناك تبديلات مختلفة $n!$ قد يحدث أن يكون أي واحد من هذه التبديلات $n!$ الترتيب الصحيح للقائمة L وبهذا يجب أن تحتوي شجرة المقارنة على عدد $n!$ على الأقل من الأوراق.

لنقم بالنظر الى الحالة الأسوأ لجميع خوارزميات الترتيب القائمة على المقارنة. اننا نذكر النتيجة التالية بدون دليل.

النظرية: لتكن L قائمة بها عدد n من العناصر المستقلة. أي خوارزمية ترتيب تقوم بترتيب L بواسطة مقارنة المفاتيح فقط تقوم في أسوأ حالاتها بعمل $O(n \log_2 n)$ الى الأقل. كما تم التحليل في الأقسام السابقة تكون كل من خوارزميات الترتيب بالاختيار وبالادخال من الترتيب. بقية هذا الفصل $O(n^2)$ ش خوارزميات الترتيب التي تكون في المتوسط بالترتيب $O(n \log_2 n)$

ترتيب سريع: قوائم مبنية على المصفوفة:

لاحظ القسم السابق أن الحد الأدنى على الخوارزميات المبنية على المقارنة تكون $O(n \log_2 n)$ من خوارزميات الترتيب بالاختيار وبالادخال التي نوقشت من قبل في هذا الفصل تكون $O(n^2)$. هذا القسم والقسمان التاليان يناقشوا خوارزميات الترتيب التي يكون ترتيبها الخوارزمية $O(n \log_2 n)$ رزمية الترتيب السريع. خوارزمية الترتيب السريع تستخدم أسلوب القسمة والتغلب لترتيب القائمة. يتم تقسيم القائمة الى قائمتين فرعيتين ثم يتم ترتيب القائمتين الفرعيتين ويتم دمجها في قائمة واحدة بطريقة تجعل القائمة المدمجة مرتبة. بهذا تكون الخوارزمية العامة هي:
إذا (كان حجم القائمة أكبر من 1)

- أ- قم بتقسيم القائمة الى قائمتين فرعيتين مثل قائمة فرعية سفلية وقائمة فرعية علوية.
- ب- قم بعمل ترتيب سريع للقائمة الفرعية السفلية.
- ت- قم بعمل ترتيب سريع للقائمة الفرعية العلوية.
- ث- ادمج القائمة الفرعية السفلية المرتبة والقائمة الفرعية العلوية المرتبة.

}

بعد تقسيم القائمة الى قائمتين فرعيتين – قائمة فرعية سفلية lowerSublist وقائمة فرعية علوية upperSublist – يتم ترتيب هاتين القائمتين الفرعيتين باستخدام خوارزمية الترتيب السريع. بمعنى آخر نستخدم التكرار لتطبيق خوارزمية الترتيب السريع.

خوارزمية الترتيب السريع التي تم وصفها هنا هي من أجل القوائم المبنية على المصفوفة. يمكن عمل خوارزمية القوائم المتصلة بأسلوب مماثل وهو متروك كتمرين لك. انظر تمرين البرمجة 4 في نهاية هذا الفصل.

في خوارزمية الترتيب السريع يتم تقسيم القائمة بطريقة تجعل دمج القائمة الفرعية السفلية والقائمة الفرعية العلوية تافهاً. لهذا في الترتيب السريع يتم القيام بجميع أعمال الترتيب في تقسيم القائمة. بما أن جميع عمل الترتيب يحدث أثناء تقسيم القائمة اذن نقوم أولاً بوصف اجراءات التقسيم بالتفصيل. لتقسيم القائمة الى قائمتين فرعيتين نقوم أولاً باختيار عنصر من القائمة يسمى **محور**. يتم استخدام المحور لتقسيم القائمة الى قائمتين فرعيتين هما: القائمة الفرعية السفلية والقائمة الفرعية العلوية. العناصر الموجودة في القائمة الفرعية السفلية تكون أصغر من المحور والعناصر في القائمة الفرعية العلوية تكون أكبر من المحور. على سبيل المثال انظر الى القائمة في شكل 10-31.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| list | 45 | 82 | 25 | 94 | 50 | 60 | 78 | 32 | 92 |

شكل 10-31: القائمة قبل التقسيم

هناك عدة طرق لتحديد المحور. بالرغم من هذا فان المحور يتم اختياره على أمل أن تكون كل من القائمة الفرعية السفلية والقائمة الفرعية العلوية ذات حجم متساوي تقريباً. لأهداف توضيحية لنقم باختيار العنصر الأوسط من القائمة كمحور. اجراء التقسيم الذي نوضحه يقسم القائمة باستخدام المحور كعنصر أوسط وهو في حالتنا 50 كما هو موضح في شكل 10-32.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
|------|--------------|-----|-----|-----|--------------|-----|-----|-----|-----|
| list | 32 | 25 | 45 | 50 | 82 | 60 | 78 | 94 | 92 |
| | lowerSublist | | | | upperSublist | | | | |

شكل 10-32: القائمة بعد التقسيم

من الشكل 10-32 يتضح أن بعد تقسيم القائمة الى قائمة فرعية سفلية وقائمة فرعية علوية يكون المحور في المكان الصحيح. بهذا بعد ترتيب القائمة الفرعية السفلية والعلوية يكون دمج القائمتين الفرعيتين المرتبتين شديد السهولة.

خوارزمية التقسيم تكون كما يلي (نفترض أنه يتم اختيار المحور كعنصر أوسط بالقائمة):

1. حدد المحور واستبدله مع العنصر الأول من القائمة.

افترض أن المؤشر smallIndex يشير إلى العنصر الأخير الأصغر من المحور. تتم تهيئة المؤشر smallIndex عند العنصر الأول من القائمة.

2. بالنسبة إلى باقي العناصر في القائمة (بدءاً من العنصر الثاني):

إذا كان العنصر الحالي أصغر من المحور

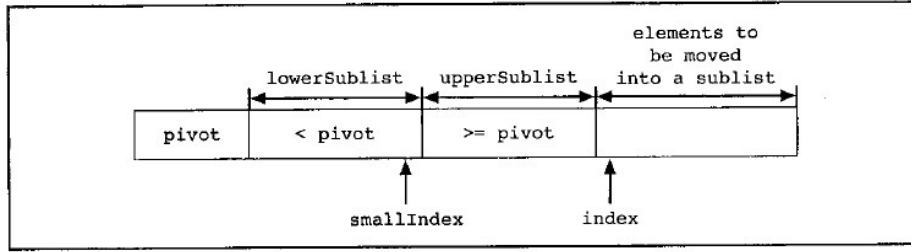
أ- قم بزيادة smallIndex.

ب- استبدل العنصر الحالي بعنصر المصفوفة المشار إليه بواسطة smallIndex.

3. استبدل العنصر الأول وهو المحور مع عنصر المصفوفة المشار إليه بواسطة smallIndex.

يمكن تطبيق الخطوة 2 باستخدام الحلقة for مع الحلقة بدءاً من العنصر الثاني من القائمة.

الخطوة 1 تحدد المحور وتنقل المحور إلى موضع المصفوفة الأول. أثناء تنفيذ الخطوة 2 يتم ترتيب عناصر القائمة كما هو موضح في شكل 10-33. (افترض أن اسم المصفوفة المحتوية على عناصر القائمة هو list).



شكل 10-33: القائمة أثناء تنفيذ الخطوة 2

كما هو موضح في شكل 10-33 المحور في موضع المصفوفة الأول. العناصر في القائمة الفرعية السفلية أصغر من المحور والعناصر في القائمة الفرعية العلوية أكبر من أو تساوي المحور. المتغير smallIndex يحتوي على مؤشر العنصر الأخير من القائمة الفرعية السفلية والمتغير index يحتوي على مؤشر العنصر التالي الذي يكون هناك حاجة إلى نقله إلى القائمة الفرعية السفلية أو إلى القائمة الفرعية العلوية. كم وضح في الخطوة 2 إذا كان العنصر التالي من القائمة (أي list[index]) أصغر من المحور نقوم بتقديم smallIndex إلى موضع المصفوفة التالي ونستبدل list[index] مع

list[smallIndex]. بعد هذا نوضح الخطوة 2.

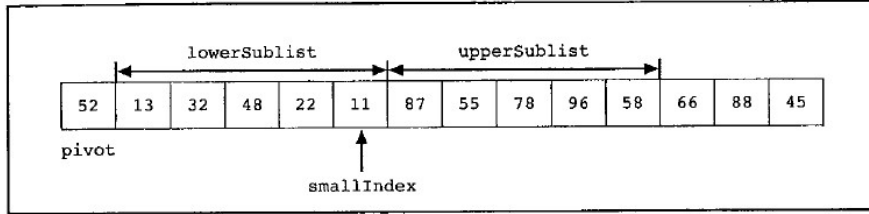
افترض أن القائمة كما هي معطاة في شكل 10-34.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|
| 32 | 55 | 87 | 13 | 78 | 96 | 52 | 48 | 22 | 11 | 58 | 66 | 88 | 45 |

شكل 10-34: القائمة قبل الترتيب

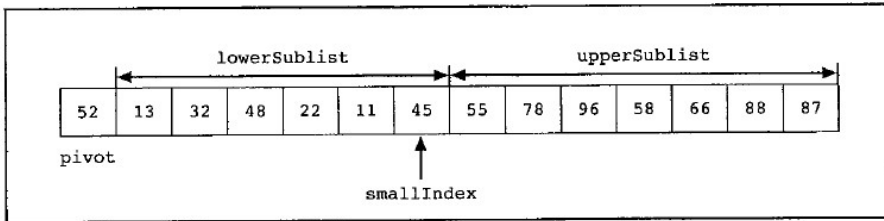
شكل 10-38: القائمة قبل تحريك 58 الى داخل قائمة فرعية

بالنسبة الى القائمة في شكل 10-38 فان list [index] تكون 58 وهي أكبر من المحور. لهذا يتم نقل list [index] الى داخل القائمة الفرعية العلوية وهذا يتم عمله عن طريق ترك 58 في موضعه وزيادة حجم القائمة الفرعية العلوية بمقدار واحد الى موضع المصفوفة التالي. بعد نقل 58 الى داخل القائمة الفرعية العلوية تكون القائمة كما هي موضحة في شكل 10-39.



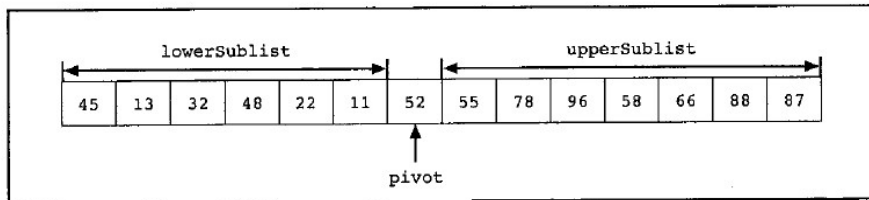
شكل 10-39: القائمة بعد نقل 58 الى داخل القائمة الفرعية العلوية.

بعد نقل جميع العناصر الأصغر من المحور الى داخل القائمة الفرعية السفلية وجميع العناصر الأكبر من المحور الى داخل القائمة الفرعية العلوية (أي أنه بعد تنفيذ الخطوة 2 كليةً) يوضح الشكل 10-40 القائمة الناتجة.



شكل 10-40: عناصر القائمة بعد الترتيب في القائمتين الفرعيتين العلوية والسفلية.

بعد هذا نقوم بتنفيذ الخطوة 3 وننقل 52 وهو المحور الى المكان المناسب في القائمة ويتم انجاز هذا عن طريق استبدال 52 مع 45. القائمة الناتجة تكون كما هي موضحة في شكل 10-41.



شكل 10-41: القائمة بعد تبادل 52 مع 45

كما هو موضح في شكل 10-41 تقوم الخطوات 1 و 2 و 3 في الخوارزمية السابقة بتقسيم القائمة الى قائمتين فرعيتين. العناصر الأقل من المحور تكون في القائمة الفرعية السفلية والعناصر الأكبر من أو تساوي المحور تكون في القائمة الفرعية العلوية.

لتقسيم القائمة الى القائمتين الفرعيتين العلوية والسفلية نحتاج الى تتبع العنصر الأخير فقط من القائمة الفرعية السفلية والعنصر التالي من القائمة الذي يحتاج الى أن يتم نقله الى داخل اما القائمة الفرعية السفلية أو القائمة الفرعية العلوية. في الحقيقة تكون القائمة الفرعية العلوية بين المؤشرين smallIndex و index.

الآن نكتب الدالة partition لتطبيق خوارزمية التقسيم السابقة وبعد اعادة ترتيب عناصر القائمة تقوم الدالة بانتاج موقع المحور حتى يمكننا مواقع بداية ونهاية القوائم الرئيسية. بما أن الدالة partition عنصر من الفئة فان له تناول مباشر الى المصفوفة المحتوية على القائمة. بهذا نحتاج لتقسيم القائمة الى تمرير مؤشرات البداية والنهاية للقائمة فقط.

```
template<class elemType>
int orderedArrayListType<elemType>::partition(int first, int last)
{
    elemType pivot;

    int index, smallIndex;

    swap(first, (first + last) / 2);

    pivot = list[first];
    smallIndex = first;

    for(index = first + 1; index <= last; index++)
        if(list[index] < pivot)
        {
            smallIndex++;
            swap(smallIndex, index);
        }

    swap(first, smallIndex);

    return smallIndex;
}
```

كما يمكنك أن ترى من تعريف الدالة partition هناك حاجة الى تبادل عناصر محددة من القائمة. الدالة التالية swap تحقق هذه المهمة (لاحظ أن هذه الدالة swap من نفس نوع الدالة المعطاة من قبل في هذا الفصل من أجل خوارزمية الترتيب بالاختيار).

```
template<class elemType>
void orderedArrayListType<elemType>::swap(int first, int second)
{
    elemType temp;

    temp = list[first];
    list[first] = list[second];
    list[second] = temp;
}
```

بمجرد تقسيم القائمة الى قائمة فرعية سفلية وقائمة فرعية علوية نقوم مرة أخرى بتطبيق طريقة الترتيب السريع لترتيب القائمتين الفرعيتين. بما أن كلا القائمتين الفرعيتين يتم ترتيبهما باستخدام نفس خوارزمية الترتيب السريع فان أسهل طريقة لتطبيق هذه الخوارزمية هي استخدام التكرار. لهذا يقوم هذا القسم باعطاء الصورة التكرارية من خوارزمية الترتيب السريع.

كما وضح من قبل بعد اعادة ترتيب عناصر القائمة تقوم الدالة partition بانتاج مؤشر المحور حتى يمكن تحديد مؤشرات البداية والنهاية للقوائم الفرعية. باعطاء مؤشرات البداية والنهاية للقائمة تقوم الدالة recQuickSort بتطبيق الصورة التكرارية من خوارزمية الترتيب السريع:

```
template<class elemType>
void orderedArrayListType<elemType>::recQuickSort(int first, int last)
{
    int pivotLocation;
    if(first < last)
    {
        pivotLocation = partition(first, last);
        recQuickSort(first, pivotLocation - 1);
        recQuickSort(pivotLocation + 1, last);
    }
}
```

أخيراً نكتب دالة الترتيب السريع quickSort التي تستدعي الدالة recQuickSort على القائمة الأصلية.

```
template<class elemType>
void orderedArrayListType<elemType>::quickSort()
{
    recQuickSort(0, length - 1);
}
```

نترك لك كتمرين كتابة برنامج لاختبار خوارزمية الترتيب السريع. انظر تمرين البرمجة 5 في نهاية هذا الفصل.

تحليل: الترتيب السريع:

جدول 2-10 يلخص سلوك خوارزمية الترتيب السريع لقائمة طولها n . (اثباتات تلك النتائج بعيدة عن مجال هذا الكتاب)

جدول 2-10: تحليل خوارزمية الترتيب السريع لقائمة طولها n .

| عدد المقارنات | عدد التبادلات | |
|---------------|---------------|-----------------|
| *** | *** | الحالة المتوسطة |
| *** | *** | الحالة الأسوأ |

الترتيب بالدمج: قوائم متصلة مبنية على قوائم:

القسم السابق ناقش خوارزمية الترتيب السريع وذكر أن سلوك الحالة المتوسطة من الترتيب السريع هي $O(n \cdot \log_2 n)$ إذا فان سلوك الحالة الأسوأ للترتيب السريع يكون $O(n^2)$. هذا القسم

يصف خوارزمية الترتيب التي يكون سلوكها دائماً $O(n \cdot \log_2 n)$.

ان خوارزمية الترتيب بالدمج مثله مثل خوارزمية الترتيب السريع تستخدم أسلوب فرق تسد لترتيب قائمة. تقوم خوارزمية الترتيب بالدمج كذلك بتقسيم القائمة الى قائمتين فرعيتين وترتيب القوائم الفرعية ثم دمج القوائم الفرعية المرتبة في قائمة واحدة مرتبة. يقوم هذا القسم بوصف خوارزمية الترتيب

بالدمج بالنسبة الى القوائم المتصلة. نترك لك كتمرين عمل خوارزمية الترتيب بالدمج لقوائم مبنية على المصفوفات وهذا يمكن عمله عن طريق استخدام الأساليب التي تم وصفها من أجل القوائم المتصلة. خوارزميات الترتيب بالدمج والترتيب السريع تختلف في كيفية تقسيمها للقائمة. كما نوقش من قبل يقوم الترتيب السريع أولاً باختيار عنصر في القائمة يسمى محور ثم يقسم القائمة حتى تكون العناصر في قائمة فرعية أصغر من المحور والعناصر في قائمة فارعية أخرى أكبر من المحور. على النقيض يقوم الترتيب السريع بتقسيم القائمة الى قائمتين فرعيتين ذات حجم متساوي تقريباً. على سبيل المثال انظر الى القائمة التي تكون عناصرها كما يلي:

القائمة: 38 30 48 62 45 18 28 35

خوارزمية الترتيب بالدمج تقوم بتقسيم هذه القائمة الى قائمتين فرعيتين كما يلي:

القائمة الفرعية الأولى: 45 18 28 35

القائمة الفرعية الثانية: 38 30 48 62

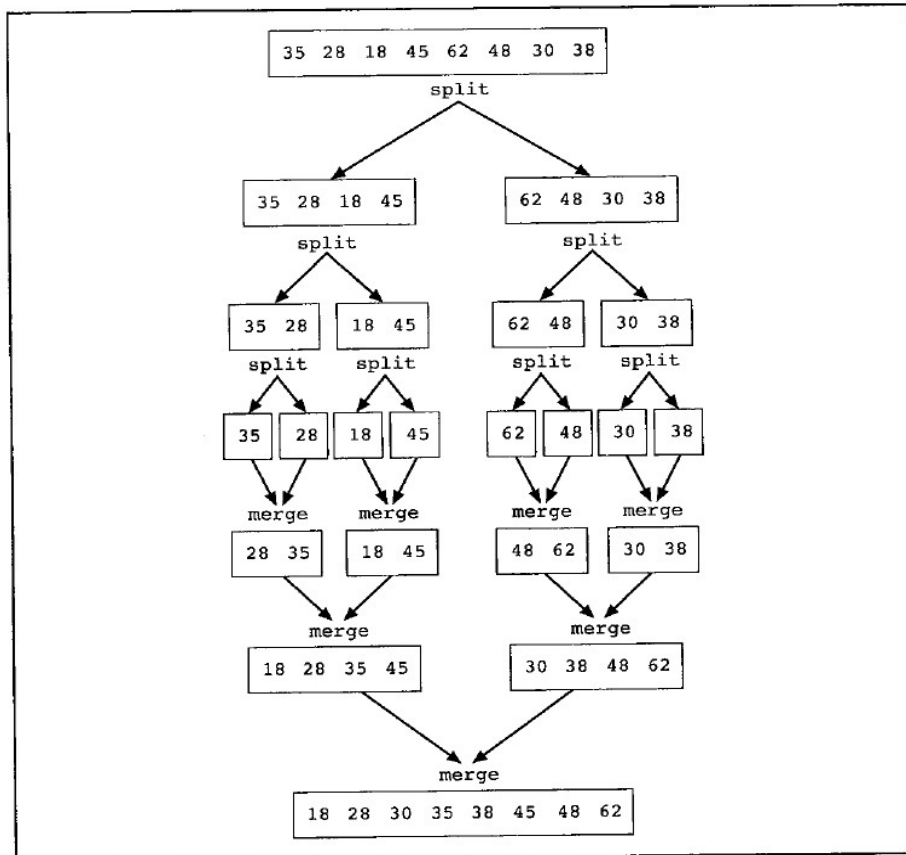
يتم ترتيب القائمتين الفرعيتين باستخدام نفس الخوارزمية (أي الترتيب بالدمج) التي تم استخدامها على القائمة الأصلية. افترض أننا قمنا بترتيب القائمة الفرعية. افترض أن:

القائمة الفرعية الأولى: 45 35 28 18

القائمة الفرعية الثانية: 62 48 38 30

بعد هذا تقوم خوارزمية الترتيب بالدمج بدمج أي اتحاد القوائم الفرعية المرتبة في قائمة واحدة مرتبة.

الشكل 10-42 يوضح عملية الترتيب بالدمج.



شكل 10-42: خوارزمية الترتيب بالدمج

من شكل 10-42 يتضح أن في خوارزمية الترتيب بالدمج يتم القيام بغالبية عمل الترتيب في دمج القوائم الفرعية المرتبة.

الخوارزمية العامة للترتيب بالدمج تكون كما يلي:

إذا كانت القائمة حجمها أكبر من 1

}

أ. اقسام القائم الى قائمتين فرعيتين.

ب. قم بترتيب القائمة الفرعية الأولى بالدمج.

ت. قم بترتيب القائمة الفرعية الثانية بالدمج.

ث. قم بدمج القائمة الفرعية الأولى والقائم الفرعية الثانية.

{

كما لوحظ من قبل بعد تقسيم القائمة الى قائمتين فرعيتين – القائمة الفرعية الأولى والقائمة الفرعية الثانية – يتم ترتيب هاتين القائمتين الفرعيتين باستخدام خوارزمية الترتيب بالدمج. بمعنى آخر نستخدم التكرار لتطبيق خوارزمية الترتيب بالدمج.

بعد هذا نصف الخوارزمية اللازمة من أجل:

■ تقسيم القائمة الى قائمتين فرعيتين حجمهما متساوي تقريباً .

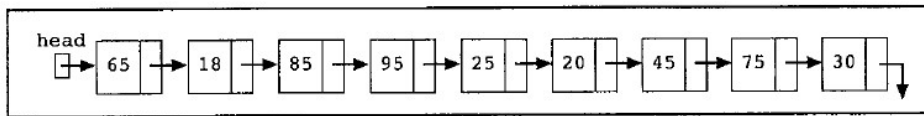
■ ترتيب كلتا القائمتين بالدمج.

■ دمج القائمتين المرتبتين.

التقسيم:

بما أن البيانات يتم تخزينها في قائمة متصلة اذن فنحن لا نعلم طول القائمة. فضلاً عن هذا القائمة المتصلة ليست ببنية بيانات تتناول عشوائي ولهذا من أجل تقسيم القائمة الى قائمتين فرعيتين فنحن نحتاج الى ايجاد العقدة الوسطى للقائمة.

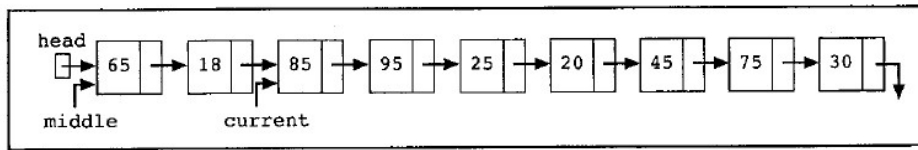
انظر الى القائمة في شكل 10-43.



شكل 10-43: قائمة متصلة غير مرتبة

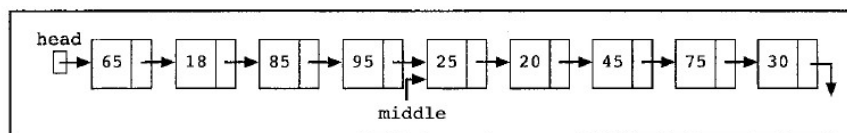
لايجاد منتصف القائمة نجتاز القائمة بمؤشرين – مثل middle و current. المؤشر middle تتم تهيئته عند أول عقدة بالقائمة. بما أن هذه القائمة بها أكثر من عقدتين فاننا نقوم بتهيئة المؤشر current عند العقدة الثالثة. (تذكر أننا نرتب القائمة فقط اذا كان بها أكثر من عنصر واحد لأن القائمة التي

حجمها 1 تكون بالفعل مرتبة. فضلاً عن هذا اذا كانت القائمة بها عقدتان فقط نحدد current عند NULL). انظر الى القائمة الموضحة في شكل 10-44.



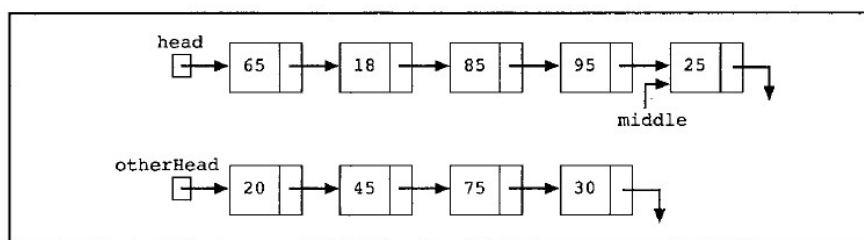
شكل 10-44: middle و current قبل اجتياز القائمة

في كل مرة نقدم فيها middle عقدة واحدة نقوم بتقديم current عقدة واحدة. بعد تقديم current عقدة واحدة اذا كانت current ليست صفراً فاننا نقوم مرة أخرى بتقديم current عقدة واحدة. هذا يعني أنه في الغالبية كلما تقدمت middle عقدة واحدة تتقدم current عقدتين. أخيراً تصبح current قيمتها NULL وتشير middle الى العقدة الأخيرة من القائمة الفرعية الأولى. على سبيل المثال يشير middle الى العقدة ذات البيان 25 (انظر شكل 10-45).



شكل 10-45: middle بعد اجتياز القائمة

من السهل الآن تقسيم القائمة الى قائمتين فرعيتين. أولاً نقوم باستخدام وصلة middle بتحديد مؤشر للعقدة التي تلي middle. ثم نحدد الوصلة middle عند NULL. الشكل 10-46 يوضح القوائم الفرعية الناتجة.



شكل 10-46: القائمة بعد تقسيمها الى قائمتين

تتم ترجمة هذه المناقشة الى دالة C++ التالية divideList:

```
template<class elemType>
void orderedLinkedListType<elemType>::divideList
(nodeType<elemType>* first1,
 nodeType<elemType>* &first2)
{
    nodeType<elemType>* middle;
    nodeType<elemType>* current;

    if(first1 == NULL)    //the list is empty
        first2 = NULL;
```

```

else
    if(first1->link == NULL) //the list has only one node
        first2 = NULL;
    else
    {
        middle = first1;
        current = first1->link;
        if(current != NULL) //the list has more than two nodes
            current = current->link;

        while(current != NULL)
        {
            middle = middle->link;
            current = current->link;
            if(current != NULL)
                current = current->link;
        } //end while

        first2 = middle->link; //first2 points to the first
                               //node of the second sublist
        middle->link = NULL;   //set the link of the last node
                               //of the first sublist to NULL

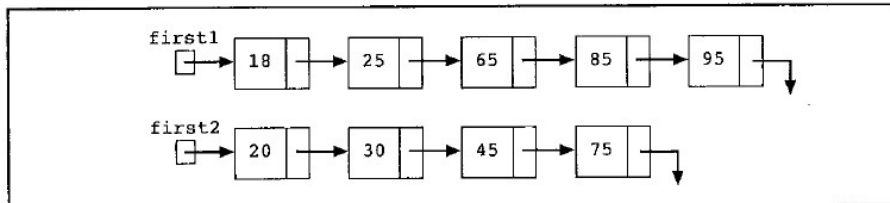
    } //end else
} //end divideList

```

الآن بعدما علمنا كيفية تقسيم القائمة الى قائمتين فرعيتين ذات حجم متساوي تقريباً نقوم بعد هذا بالتركيز على دمج القوائم المرتبة. تذكر أن غالبية عمل الترتيب في الترتيب بالدمج يتم القيام به في دمج القوائم الفرعية المرتبة.

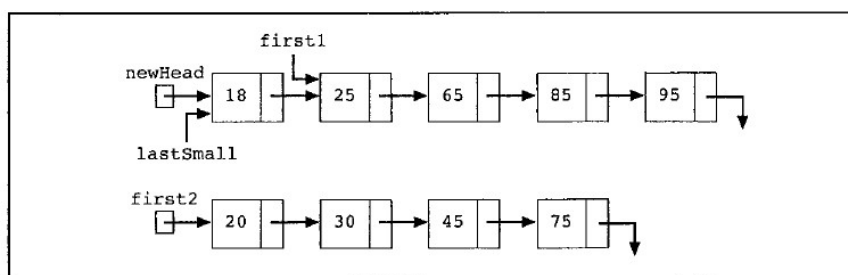
الدمج:

بمجرد أن يتم ترتيب القوائم الفرعية تكون الخطوة التالية في خوارزمية الترتيب بالدمج دمج القوائم الفرعية المرتبة. يتم دمج القوائم الفرعية المرتبة في قائمة مرتبة عن طريق مقارنة عناصر القوائم الفرعية ثم تعديل مؤشر العقد ذات أصغر بيان. لنقم بتوضيح هذا الاجراء على القوائم الفرعية الموضحة في شكل 10-47. افترض أن first1 يشير الى العقدة الأولى من القائمة الفرعية الأولى ويشير first2 الى العقدة الأولى من القائمة الفرعية الثانية.

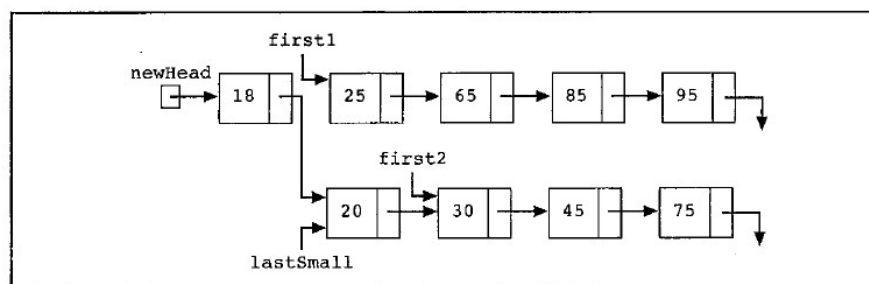


شكل 10-47: القوائم الفرعية قبل الدمج

أولاً نقارن بيان العقدة الأولى لكل من القائمتين الفرعيتين لتحديد العقدة الأولى للقائمة المدمجة. نقوم بتحديد مؤشر newHead للإشارة إلى العقدة الأولى من القائمة المدمجة. كما نقوم باستخدام مؤشر lastSmall لتتبع العقدة الأخيرة من القائمة المدمجة. بعد هذا يتقدم المؤشر الرئيسي للقائمة الفرعية ذات العقدة الأصغر إلى العقدة التالية من هذه القائمة الفرعية. الشكل 10-48 يوضح القوائم الفرعية للشكل 10-47 بعد تحديد newHead و lastSmall وتقديم first1.



شكل 10-48: القوائم الفرعية بعد تحديد newHead و lastSmall وتقديم first1
 في الشكل 10-48 يشير first1 إلى العقدة الأولى من القائمة الفرعية الأولى التي سوف يتم دمجها مع القائمة الفرعية الثانية. لهذا نقوم مرة أخرى بمقارنة العقد المشار إليها بواسطة first1 و first2 ونعدل وصلة العقدة الأصغر والعقدة الأخيرة من القائمة المدمجة لنقل العقدة الأصغر إلى نهاية القائمة المدمجة. بالنسبة إلى القوائم الفرعية الموضحة في شكل 10-48 يكون لدينا الشكل 10-49 بعد تعديل الوصلات اللازمة.



شكل 10-49: القائمة المدمجة بعد وضع العقدة ذات البيان 20 في نهاية القائمة المدمجة

اننا نستمر في هذه العملية بالنسبة الى العناصر المتبقية من هاتين القائمتين الفرعيتين. في كل مرة نقوم فيها بنقل عقدة الى القائمة المدمجة نقوم بتقديم اما first1 أو first2 الى العقدة التالية. في النهاية تصبح first1 أو first2 قيمتها NULL. اذا أصبحت first1 تساوي NULL فان القائمة الفرعية الأولى يتم ارهاقها أولاً وبهذا نلحق العقد الأخرى من القائمة الفرعية الثانية في نهاية القائمة المدمجة جزئياً. اذا أصبحت first2 قيمتها NULL فان القائمة الفرعية الثانية يتم ارهاقها أولاً وبهذا نلحق العقد الأخرى من القائمة الفرعية الأولى في نهاية القائمة المدمجة جزئياً.

بعد هذه المناقشة يمكننا كتابة دالة C++ المسماة mergeList لدمج القائمتين الفرعيتين المرتبتين. المؤشر الرئيسي للقوائم الفرعية يتم تمريرها كمعاملات للدالة mergeList.

```
template<class elemType>
nodeType<elemType>* orderedLinkedListType<elemType>::mergeList
(nodeType<elemType>* first1,
 nodeType<elemType>* first2)
{
    nodeType<elemType> *lastSmall; //pointer to the last node of
                                //the merged list
    nodeType<elemType> *newHead; //pointer to the merged list

    if(first1 == NULL) //the first sublist is empty
        return first2;
    else
        if(first2 == NULL) //the second sublist is empty
            return first1;
        else
        {
            if(first1->info < first2->info) //compare the first nodes
            {
                newHead = first1;
                first1 = first1->link;
                lastSmall = newHead;
            }
            else
                if(first2->info < first1->info)
                {
                    newHead = first2;
                    first2 = first2->link;
                    lastSmall = newHead;
                }

            while(first1 != NULL && first2 != NULL)
            {
                if(first1->info < first2->info)
                {
                    lastSmall->link = first1;
                    lastSmall = lastSmall->link;
                    first1 = first1->link;
                }
            }
        }
    }
}
```

```

        else
        {
            lastSmall->link = first2;
            lastSmall = lastSmall->link;
            first2 = first2->link;
        }
    } //end while

    if(first1 == NULL) //first sublist is exhausted first
        lastSmall->link = first2;
    else //second sublist is exhausted first
        lastSmall->link = first1;

    return newHead;
}
} //end mergeList

```

أخيراً نقوم بكتابة الدالة التكرارية للترتيب بالدمج `recMergeSort` التي تستخدم الدالات `divideList` و `mergeList` لترتيب القائمة. المؤشر الرئيسي للقائمة التي يتم ترتيبها يتم تمريره كمعامل للدالة `recMergeSort`.

```

template<class elemType>
void orderedLinkedListType<elemType>::recMergeSort
    (nodeType<elemType>* &head)
{
    nodeType<elemType> *otherHead;

    if(head != NULL) //if the list is not empty
        if(head->link != NULL) //if the list has more than one node
        {
            divideList(head, otherHead);
            recMergeSort(head);
            recMergeSort(otherHead);
            head = mergeList(head, otherHead);
        }
    } //end recMergeSort

```

يمكننا الآن اعطاء تعريف الدالة `mergeSort` التي يجب تضمينها كعنصر عام للفئة `orderedLinkedListType`. (لاحظ أن الدالات `divideList` و `merge` و `reMergeSort` يمكن ضمها كعناصر خاصة للفئة `orderedLinkedListType` لأن هذه الدالات يتم استخدامها فقط لتطبيق الدالة `mergeSort`). الدالة `mergeSort` تقوم ببساطة باستدعاء الدالة `recMergeSort` وتقوم بتمرير المؤشر `first` الى هذه الدالة. تعريف الدالة `mergeSort` هو:

```

template<class elemType>
void orderedLinkedListType<elemType>::mergeSort()
{
    recMergeSort(first);
} //end mergeSort

```

نترك كتمرين لك كتابة برنامج لاختبار خوارزمية الترتيب بالدمج. انظر تمرين البرمجة 8 في نهاية هذا الفصل.

تحليل: الترتيب بالدمج:

افترض أن L قائمة بها عدد n من العناصر حيث $n > 0$. لتكن $A(n)$ تعبر عن عدد مقارنات المفتاح في الحالة العادية وتعبر $W(n)$ عن عدد مقارنات المفتاح في الحالة الأسوأ لترتيب L . يمكن توضيح أن:

$$A(n) = n * \log_2 n - 1.26n = O(n * \log_2 n)$$

$$W(n) = n * \log_2 n - (n-1) = O(n * \log_2 n)$$

الترتيب بالتكدس: القوائم المبنية على مصفوفة:

في فصل سابق قمنا بوصف خوارزمية الترتيب السريع للقوائم المتجاورة أي القوائم المبنية على مصفوفة. لقد لاحظنا أن الترتيب السريع في المتوسط يكون من النوع $O(n * \log_2 n)$ ثم من هذا في الحالة الأسوأ يكون الترتيب السريع من النوع $O(n^2)$. ذا القسم بوصف خوارزمية أخرى وهي الترتيب بالتكدس للقوائم المبنية على المصفوفة. هذه الخوارزمية من النوع $O(n * \log_2 n)$ حتى في الحالة الأسوأ ولهذا تتغلب على الحالة الأسوأ من الترتيب السريع.

تعريف: التكدس هو قائمة كل عنصر من عناصر الذي يحتوي على مفتاح مثل المفتاح في العنصر عند الموضع k في القائمة يكون على الأقل كبير مثل المفتاح في العنصر عند الموضع $2k + 1$ (إذا وجد) و $2k + 2$ (إذا وجد).

تذكر أن في $C++$ يبدأ مؤشر المصفوفة عند صفر ولهذا العنصر الموجود عند الموضع k هو في الحقيقة العنصر رقم $k + 1$ من القائمة.

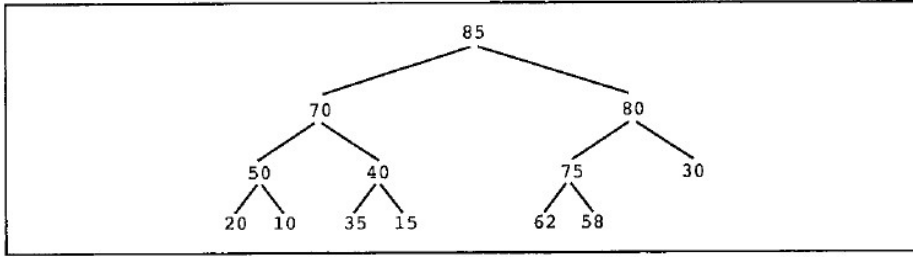
انظر الى القائمة في شكل 10-50.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|
| 85 | 70 | 80 | 50 | 40 | 75 | 30 | 20 | 10 | 35 | 15 | 62 | 58 |

شكل 10-50: قائمة عبارة عن تكدس

يمكن التحقق من أن القائمة في شكل 10-50 عبارة عن تكدس. باعطائك تكدس يمكنك إقامة شجرة ثنائية كاملة كما يلي: العقدة الجذرية للشجرة هي العنصر الأول من القائمة والطفل الأيسر للجذر هو العنصر الثاني في القائمة والطفل الأيسر من العقدة الجذرية هو العنصر الثالث من القائمة. بهذا فانه بالنسبة الى العقدة k بوجه عام والتي هي العنصر رقم $1 - k$ من القائمة يكون طفلها الأيسر هو العنصر رقم $2k$ (إذا وجد) بالقائمة الذي يكون في الموضع $1 - 2k$ في القائمة والطفل الأيمن هو العنصر رقم $1 + 2k$ (إذا وجد) من القائمة والموجود في الموضع $2k$ في القائمة.

الرسم في شكل 10-51 يمثل الشجرة الثنائية الكاملة المتوافقة مع القائمة في شكل 10-50



شكل 10-51: الشجرة الثنائية الكاملة المتوافقة مع القائمة في شكل 10-50

الشكل 10-51 يوضح أن القائمة الموجودة في شكل 10-50 موجودة في تكديس. في الحقيقة ومن أجل توضيح خوارزمية الترتيب بالتكديس سوف نقوم دائماً برسم الشجرة الثنائية الكاملة المتوافقة مع القائمة.

نقوم الآن بوصف خوارزمية الترتيب بالتكديس.

الخطوة الأولى في خوارزمية الترتيب بالتكديس هي تحويل القائمة الى كومة تسمى buildHeap وبعد تحويلنا المصفوفة الى كومة تبدأ مرحلة الترتيب.

بناء كومة:

هذا القسم يصف خوارزمية بناء كومة.

الخوارزمية العامة تكون كما يلي: افترض أن length يعبر عن طول القائمة. لتكن $\text{index} = \text{length} / 2 - 1$. إذن $\text{list}[\text{index}]$ هو العنصر الأخير في القائمة لا يكون ورقة وهذا يعني أن هذا العنصر له طفل واحد على الأقل. بهذا تكون العناصر $\text{list}[\text{index} + 1] \dots \text{list}[\text{length} - 1]$ عبارة عن أوراق.

أولاً نقوم بتحويل الشجرة الفرعية ذات العقدة الجذرية $\text{list}[\text{index}]$ الى كومة. لاحظ أن هذه الشجرة الفرعية لها ثلاثة عقد على الأكثر. بعد هذا نقوم بتحويل الشجرة الفرعية ذات العقدة الجذرية $\text{list}[\text{index} - 1]$ الى كومة وهكذا.

لتحويل شجرة فرعية الى كومة نؤدي الخطوات التالية: افترض أن $\text{list}[\text{a}]$ هي العقدة الجذرية من الشجرة الفرعية وأن $\text{list}[\text{b}]$ هي الطفل الأيسر و $\text{list}[\text{c}]$ اذا وجد يكون الطفل الأيمن من $\text{list}[\text{a}]$. قارن $\text{list}[\text{b}]$ مع $\text{list}[\text{c}]$ لتحديد الطفل الأكبر. اذا لم تتواجد $\text{list}[\text{c}]$ فان $\text{list}[\text{b}]$ تكون الطفل الأكبر. افترض أن largerIndex تشير الى الطفل الأكبر (لاحظ أن largerIndex اما يكون b أو c) قارن $\text{list}[\text{a}]$ مع $\text{list}[\text{largerIndex}]$. اذا كانت $\text{list}[\text{a}] < \text{list}[\text{largerIndex}]$ إذن استبدل $\text{list}[\text{a}]$ مع $\text{list}[\text{largerIndex}]$ وبخلاف هذا تكون الشجرة الفرعية ذات العقدة الجذرية $\text{list}[\text{a}]$ كومة بالفعل.

افترض أن $\text{list}[\text{a}] < \text{list}[\text{largerIndex}]$ ونقوم باستبدال العناصر $\text{list}[\text{a}]$ مع $\text{list}[\text{largerIndex}]$. بعد عمل هذه المبادلة قد لا تكون الشجرة الفرعية ذات العقدة الجذرية

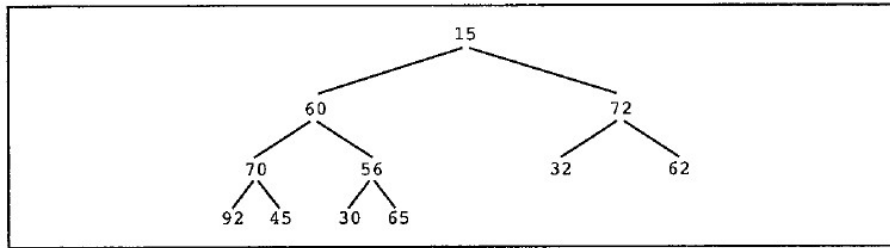
[largerIndex] list في كومة. اذا كان الوضع هكذا نكرر الخطوتين 1 و 2 على الشجرة الفرعية ذات العقدة الجذرية [largerIndex] list ونستمر في هذه العملية حتى يتم تخزين الكومات في الشجرة الفرعية أو حتى نصل الى شجرة فرعية فارغة.

هذه الخطوة يتم تطبيقها باستخدام حلقة يتم وصفها عندما نكتب الخوارزمية.
انظر الى القائمة في شكل 10-52 ولنطلق عليها list.

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| list | 15 | 60 | 72 | 70 | 56 | 32 | 62 | 92 | 45 | 30 | 65 |

شكل 10-52: المصفوفة list

الشكل 10-53 يوضح الشجرة الثنائية الكاملة المتوافقة مع القائمة في شكل 10-52.



شكل 10-53: شجرة ثنائية كاملة متوافقة مع القائمة في شكل 10-52

لتسهيل هذه المناقشة عندما نقول العقدة 56 فاننا نقصد العقدة ذات البیان 56.

هذه القائمة بها 11 عنصر ولهذا يكون طول القائمة هو 11. لتحويل المصفوفة الى كومة نبدأ عند عناصر القائمة $56 = [4]$ وأطفال $[4]$ هم $[4 * 2 + 1]$ و $[4 * 2 + 2]$ أي $[9]$ و $[10]$ list. في القائمة السابقة تتواجد كل من $[9]$ و $[10]$ list. لتحويل الشجرة ذات العقدة الجذرية $[4]$ list نؤدي الخطوات التالية:

1. ايجاد العنصر الأكبر من $[9]$ و $[10]$ list أي الطفل الأكبر من $[4]$ list. في هذه الحالة

تكون $[10]$ list أكبر من $[9]$ list.

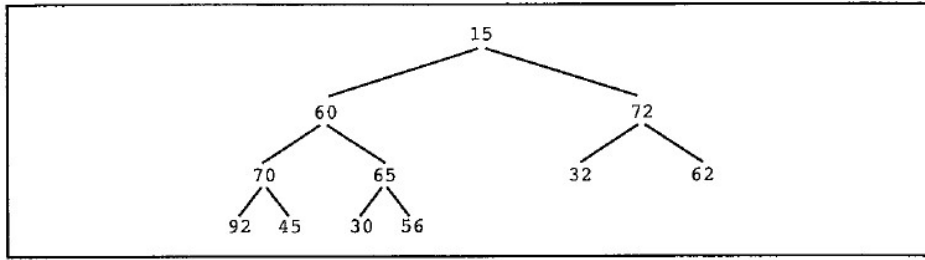
2. المقارنة بين الطفل الأكبر ذات العقدة الأبوية. اذا كان الطفل الأكبر أكبر من الأب نقوم

بإستبدال الطفل الأكبر بالأب. بما أن $[10] < [4]$ List فاننا نستبدل $[4]$ list مع

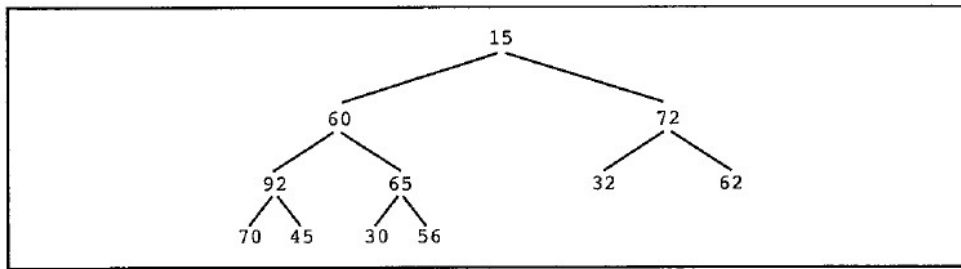
$[10]$.

3. بما أن $[10]$ list ليس لها شجرة فرعية اذن لا يتم تنفيذ الخطوة 3.

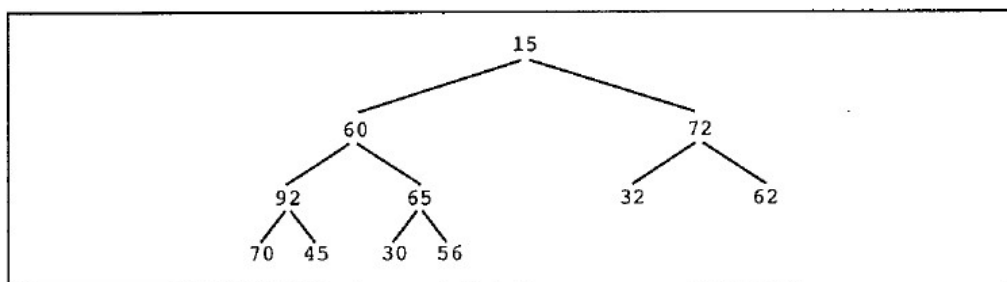
الشكل 10-54 يوضح الشجرة الثنائية الناتجة.



شكل 10-54: الشجرة الثنائية بعد استبدال list [4] مع list [10]
 بعد هذا ننظر الى الشجرة الفرعية ذات العقدة الجذرية [3] list وهي 70 ونقوم بتكرار الخطوات
 الثلاث المعطاة من قبل للحصول على الشجرة الثنائية الكاملة كما هي معطاة في شكل 10-55. (لاحظ
 أن هنا أيضاً لا يتم تنفيذ الخطوة 3)

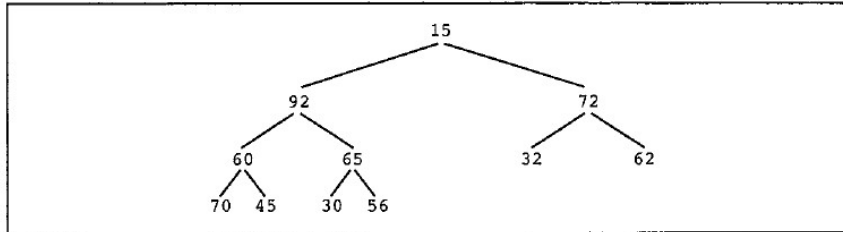


شكل 10-55: الشجرة الثنائية بعد تكرار الخطوات 1 و 2 و 3 عند العقدة الجذرية [3] list
 الآن ننظر الى الشجرة الفرعية ذات العقدة الجذرية [2] list وهي 72 ونقوم بتطبيق الخطوات الثلاث
 المعطاة من قبل. شكل 10-56 يوضح الشجرة الثنائية الناتجة. (لاحظ أن في هذه الحالة وبما أن الأب
 أكبر من كلا الطفلين فإن هذه الشجرة الفرعية عبارة بالفعل عن كومة.)

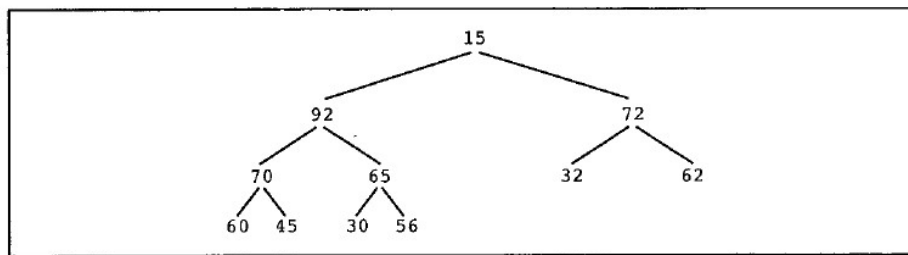


شكل 10-56: الشجرة الثنائية بعد تكرار الخطوات 1 و 2 و 3 عند العقدة الجذرية [2] list

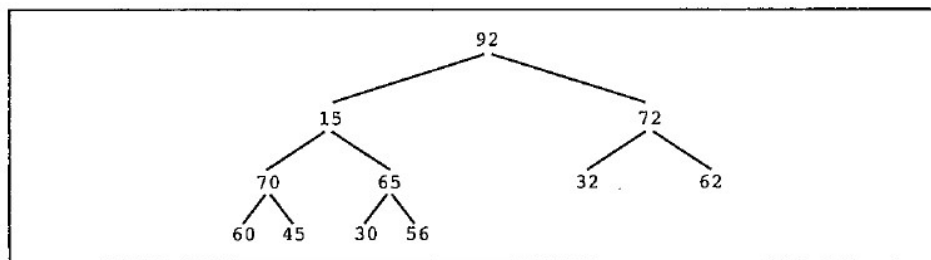
بعد هذا ننظر الى الشجرة الفرعية ذات العقدة الجذرية [1] list وهي 60. أولاً نقوم بتطبيق الخطوات 1 و 2. بما أن $list[1] = 60 < list[3] = 92$ فاننا نستبدل [1] list مع [3] list للحصول على الشجرة المعطاة في شكل 10-57.



شكل 10-57: الشجرة الثنائية بعد استبدال [1] list مع [3] list بالرغم من هذا وبعد استبدال [1] list مع [3] list لا تعد الشجرة الفرعية ذات العقدة الجذرية list [3] التي تساوي 60 كومة. لهذا يجب علينا اعادة الكومة في هذه الشجرة الفرعية. للقيام بهذا نقوم بتطبيق الخطوة 3 ونوجد الأطفل الأكبر من 60 ونستبدله مع 60. بعد هذا نحصل على الشجرة الفرعية كما هي معطاة في شكل 10-58.

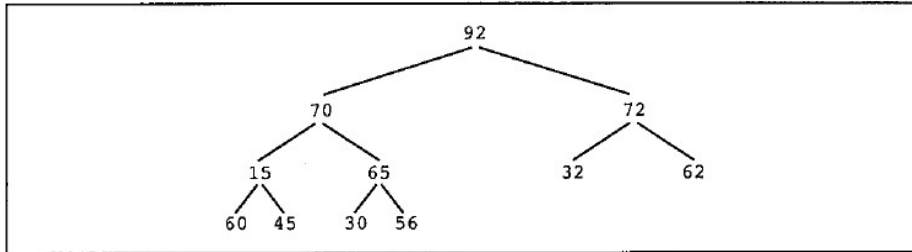


شكل 10-58: الشجرة الثنائية بعد استعادة الكومة عند [3] list مرة أخرى تكون الشجرة الفرعية ذات العقدة الجذرية [1] list التي تساوي 92 في كومة (انظر شكل 10-58). في النهاية ننظر الى الشجرة ذات العقدة الفرعية [0] list التي تساوي 15 ونكرر الخطوات الثلاث السابقة للحصول على الشجرة الثنائية الموضحة في شكل 10-59.

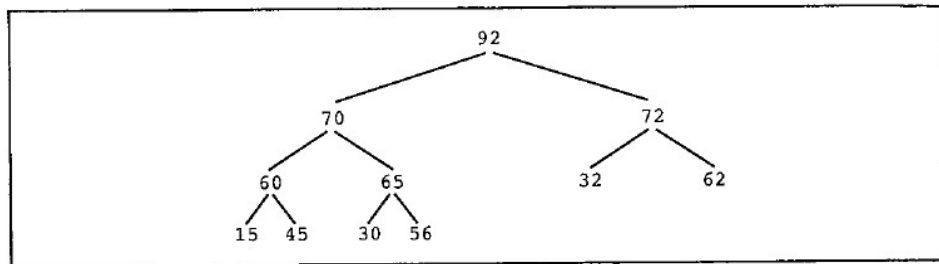


شكل 10-59: الشجرة الثنائية بعد تطبيق الخطوات 1 و 2 على [0] list

نرى أن الشجرة الفرعية ذات العقدة الجذرية [1] list التي تساوي 5 لم تعد في كومة ولهذا يجب أن نقوم بتطبيق الخطوة 3 لاستعادة الكومة في هذه الشجرة الفرعية. (هذا يقتضي منا تكرار الخطوتان 1 و 2 على الشجرة الفرعية ذات العقدة الجذرية [1] list.) اننا نستبدل [1] list بالطفل الأكبر [3] list الذي يساوي 70. بعد هذا نحصل على الشجرة الثنائية للشكل 10-60.



شكل 10-60: الشجرة الثنائية بعد تطبيق الخطوات 1 و 2 على [1] list الشجرة الفرعية ذات العقدة الجذرية [3] list التي تساوي 15 ليست في كومة ولهذا يجب أن نقوم باستعادة الكومة في هذه الشجرة الفرعية. للقيام بهذا نقوم بتطبيق الخطوات 1 و 2 على الشجرة الفرعية ذات العقدة الجذرية [3] list. نستبدل [3] list بالطفل الأكبر [7] list الذي يساوي 60. الشكل 10-61 يوضح الشجرة الثنائية الناتجة.



شكل 10-61: الشجرة الثنائية بعد استعادة الكومة عند [3] list الشجرة الثنائية الناتجة (في شكل 10-61) تكون في كومة ولهذا فان القائمة المقابلة لهذه الشجرة الثنائية الكاملة في كومة.

لها فاننا بوجه عام ننظر الى الشجرة الفرعية بدءاً من المستوى الأدنى من اليمين الى اليسار ونقوم بتحويل الشجرة الفرعية الى كومة كما يلي: اذا كانت العقدة الجذرية للشجرة الفرعية أصغر من الطفل الأكبر فاننا نستبدل العقدة الجذرية بالطفل الأكبر. بعد تبادل العقدة الجذرية مع الطفل الأكبر يجب ان نستعيد الكومة في الشجرة الفرعية التي تم استبدال عقدها الجذرية.

افترض أن low تحتوي على مؤشر العقدة الجذرية للشجرة الفرعية وأن high تحتوي على مؤشر العنصر الأخير في القائمة. الكومة التي تتم استعادتها في الشجرة الفرعية لها جذور عند [low] list. تتم ترجمة المناقشة السابقة الى خوارزمية C++ التالية:

```

int largeIndex = 2 * low + 1;    //the index of the left child
while(largeIndex <= high)
{
    if(largeIndex < high)
        if(list[largeIndex] < list[largeIndex + 1])
            largeIndex = largeIndex + 1; //the index of the larger child
    if(list[low] > list[largeIndex])    //the subtree is already in
                                        //a heap
        break;
    else
    {
        swap(list[low], list[largeIndex]); //Line **
        low = largeIndex; //go to the subtree to further
                           //restore the heap
        largeIndex = 2 * low + 1;
    } //end else
} //end while

```

البيان swap في الصف المسمى ** Line يستبدل الأب بالطفل الأكبر. بما أن البيان swap يقوم بعمل ثلاث اسنادات عنصر لاستبدال محتويات متغيرين فان في كل مرة خلال الحلقة يتم عمل ثلاث اسنادات عنصر. الحلقة while تنقل عقدة الأل الى مكان في الشجرة حتى تكون الشجرة الفرعية الناتجة ذات العقدة الجذرية [low] list في كومة. يمكننا بسهولة تقليل عدد الاسنادات في كل مرة خلال الحلقة من ثلاثة الى واحد عن طريق أولاً ترتيب العقدة الجذرية في موضع مؤقت مثل temp. بعد هذا في كل مرة خلال الحلقة تتم مقارنة الطفل الأكبر مع temp. اذا كان الطفل الأكبر أكبر من temp نقوم بنقل الطفل الأكبر الى العقدة الجذرية للشجرة الفرعية تحت الدراسة.

بعد هذا نصف الدالة heapify التي تستعيد الكومة في شجرة فرعية عن طريق عمل اسناد عنصر واحد في كل مرة خلال الحلقة. يتم تمرير كل من مؤشر العقدة الجذرية للقائمة ومؤشر آخر عنصر بالقائمة الى هذه الدالة.

```

template<class elemType>
void orderedArrayListType<elemType>::heapify(int low, int high)
{
    int largeIndex;
    elemType temp = list[low]; //copy the root node of the subtree
    largeIndex = 2 * low + 1; //index of the left child
    while(largeIndex <= high)
    {
        if(largeIndex < high)
            if(list[largeIndex] < list[largeIndex + 1])
                largeIndex = largeIndex + 1; //index of the
                                                //largest child

        if(temp > list[largeIndex]) //subtree is already in a heap
            break;
        else
        {
            list[low] = list[largeIndex]; //move the larger child
                                            //to the root
            low = largeIndex; //go to the subtree to
                               //restore the heap
            largeIndex = 2 * low + 1;
        }
    }
    list[low] = temp; //insert temp into the tree, that is, list
} //end heapify

```

بعد هذا نستخدم الدالة heapify لتطبيق الدالة buildHeap لتحويل القائمة الى كومة.

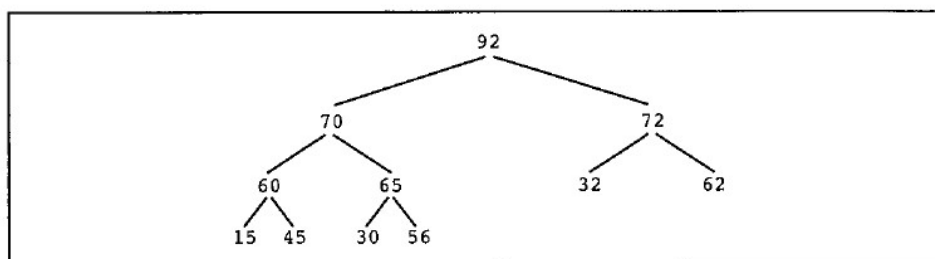
```

template<class elemType>
void orderedArrayListType<elemType>::buildHeap()
{
    int index;
    for(index = length / 2 - 1; index >= 0; index--)
        heapify(index, length - 1);
}

```

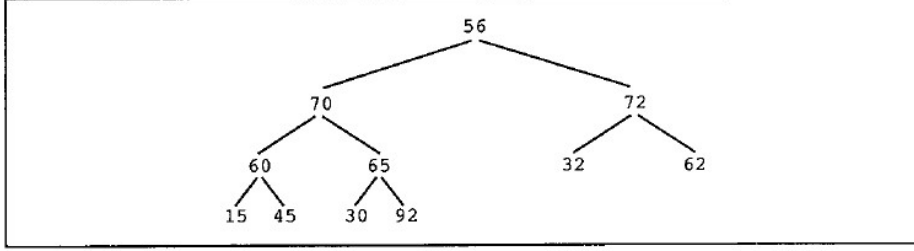
الآن نوضح خوارزمية الترتيب بالتكدس.

افترض أن القائمة في كومة. انظر الى الشجرة الثنائية الكاملة التي تمثل القائمة كما هي معطاة في شكل 10-62.



شكل 10-62: كومة

بما أن هذه عبارة عن كومة فإن العقدة الجذرية هي أكبر عنصر في الشجرة وهذا يعني أنها أكبر عنصر في القائمة. لهذا يجب نقلها إلى نهاية القائمة. نقوم باستبدال العقدة الجذرية للشجرة بأي العنصر الأول بالقائمة مع آخر عقدة في الشجرة (وهي آخر عنصر في القائمة). بعد هذا نحصل على الشجرة الثنائية كما هي موضحة في شكل 10-63.

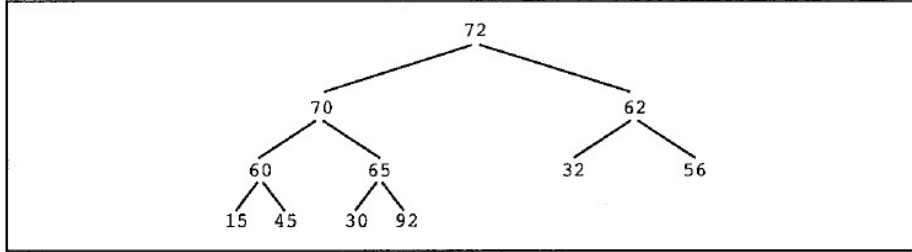


شكل 10-63: الشجرة الثنائية بعد نقل العقدة الجذرية إلى النهاية

بما أن العنصر الأكبر موجود الآن في مكانه الصحيح فإننا ننظر إلى العناصر المتبقية في القائمة أي العناصر [0] list ... [9] list. الشجرة الثنائية الكاملة التي تمثل هذه القائمة لم تعد كومة ولهذا يجب علينا استعادة الكومة في هذا الموضع من الشجرة الثنائية الكاملة. نستخدم الدالة heapify لاستعادة الكومة. الاستدعاء لهذه الدالة هو:

heapify (list, 0, 9);

بهذا نحصل على الشجرة الثنائية الموضحة في شكل 10-64.



شكل 10-64: الشجرة الثنائية بعد تنفيذ البيان heapify (list, 0, 9);

نكرر هذه العملية للشجرة الثنائية الكاملة المقابلة لعناصر القائمة [0] list ... [9] List. نقوم باستبدال [0] list مع [9] list ثم استعادة الكومة في الشجرة الثنائية الكاملة المقابلة لعناصر القائمة [0] list ... [8] list. ونكمل هذه العملية.

دالة C++ التالية تصف هذه الخوارزمية:

```

template<class elemType>
void orderedArrayListType<elemType>::heapSort()
{
    int lastOutOfOrder;
    elemType temp;

    buildHeap();

    for(lastOutOfOrder = length - 1; lastOutOfOrder >= 0;
        lastOutOfOrder--)
    {
        temp = list[lastOutOfOrder];
        list[lastOutOfOrder] = list[0];
        list[0] = temp;
        heapify(0, lastOutOfOrder - 1);
    } //end for
} //end heapSort

```

نترك كتابة برنامج لاختبار خوارزمية الترتيب بالتكدس كتمرين لك. انظر تمرين البرمجة 9 في نهاية هذا الفصل.

تحليل: الترتيب بالتكدس:

افترض أن L قائمة بها عدد n من العناصر حيث $n > 0$. في أسوأ حالة يكون عدد مقارنات المفتاح في خوارزمية الترتيب بالتكدس لترتيب القائمة L (عدد المقارنات في heapsort وعدد المقارنات في buildHeap) هو $2n \log_2 n + O(n)$.
 وفي أسوأ حالة لترتيب L يكون $n \log_2 n + O(n)$ عدد المقارنات التي تقوم بها خوارزمية الترتيب بالتكدس من أجل ترتيب القائمة L هو $O(n \log_2 n)$.

في الحالة العادية من خوارزمية الترتيب السريع يكون عدد مقارنات المفتاح $1.39n \log_2 n + O(n)$ وعدد التبادلات يكون $0.69n \log_2 n + O(n)$ كل تبادل عبارة عن ثلاث اسنادات فان عدد اسنادات العنصر في الحالة العادية من خوارزمية الترتيب السريع يكون على الأقل $1.39n \log_2 n + O(n)$.
 ينتج من هذا أنه بالنسبة الى مقارنات المفتاح تكون الحالة العادية من خوارزمية الترتيب السريع أفضل نوعاً ما من الحالة الأسوأ من خوارزمية الترتيب بالتكدس. من الجهة الأخرى وبالنسبة الى اسنادات العنصر تكون الحالة العادية من خوارزمية الترتيب السريع أسوأ نوعاً ما من الحالة الأسوأ من خوارزمية الترتيب بالتكدس. بالرغم من هذا فان الحالة الأسوأ من خوارزمية الترتيب السريع تكون $O(n^2)$. لقد أوضحت الدراسات التجريبية أن خوارزمية الترتيب بالتكدس تأخذ عادةً ضعف طول خوارزمية الترتيب السريع ولكنها تتجنب الاحتمال الضعيف للأداء الضعيف.

طوابير الأسبقية (مرة أخرى):

قدم الفصل 8 طوابير الأسبقية. تذكر أن في طابور الأسبقية يتم دفع العملاء أو الوظائف ذات الأسبقية الأكبر الى مقدمة الطابور. الفصل 8 ذكر أننا سوف نناقش تطبيق طوابير الأسبقية بعد وصف خوارزمية الترتيب بالتكديس. للتبسيط نفترض أن أسبقية عناصر الطابور يتم تحديدها باستخدام العوامل المترابطة.

في الكومة يكون العنصر الأكبر من القائمة هو دائماً العنصر الأول من القائمة. بعد ازالة العنصر الأكبر من القائمة تقوم الدالة heapify باستعادة الكومة في القائمة. للتأكد من أن العنصر الأكبر من طابور الأسبقية هو دائماً العنصر الأول من الطابور يمكننا تطبيق طوابير الأسبقية ككومات. يمكننا كتابة خوارزميات مماثلة للخوارزميات المستخدمة في الدالة heapify لادخال (عملية addQueue) عنصر في طابور الأسبقية وازالة (عملية deleteQueue) عنصر من الطابور. يقوم القسمان التاليان بوصف تلك الخوارزميات.

ادخال عنصر في طابور الأسبقية:

بافتراض أن طابور الأسبقية يتم تطبيقه ككومة نؤدي الخطوات التالية:

1. ادخال العنصر الجديد في أول موضع متاح في القائمة. (هذا يضمن أن المصفوفة التي تضم القائمة عبارة عن شجرة ثنائية كاملة.)

2. بعد ادخال العنصر الجديد في الكومة قد لا تعد القائمة كومة. لهذا لاستعادة الكومة:

while (أب المدخل الجديد أصغر من المدخل الجديد)

استبدال الأب بالمدخل الجديد.

لاحظ أن استعادة الكومة قد تتسبب في نقل المدخل الجديد الى العقدة الجذرية.

ازالة عنصر من طابور الأسبقية:

بافتراض أن طابور الأسبقية يتم تطبيقه ككومة نؤدي الخطوات التالية لازالة العنصر الأول من طابور الأسبقية:

1. نسخ العنصر الأخير من القائمة داخل موضع المصفوفة الأول.

2. تقليل طول القائمة بمقدار 1.

3. استعادة الكومة في القائمة.

يمكن تطبيق العمليات الأخرى الخاصة بطوابير الأسبقية بنفس الطريقة التي يتم تطبيقها على الطوابير. نترك تطبيق طوابير الأسبقية كتمرين لك.

تمرين برمجة: نتائج الانتخابات:

الانتخابات الرئاسية لمجلس طلاب جامعتك المحلية على وشك البدء وبسبب السرية يريد رئيس لجنة الانتخابات أداء التصويت بالحاسوب. الرئيس يبحث عن شخص ما للقيام بكتابة برنامج لتحليل البيانات وتقديم الفائز. لنقم بكتابة برنامج لمساعدة لجنة الانتخابات.

الجامعة بها أربع تقسيمات رئيسية وكل تقسيم به العديد من الأقسام. من أجل الانتخابات تمت تسمية الأربعة تقسيمات كما يلي المنطقة 1، والمنطقة 2، والمنطقة 3، والمنطقة 4. كل قسم في كل تقسيم يتعامل مع تصويته الخاص ويبلغ الأصوات المتلقاة من كل مرشح الى لجنة الانتخابات. يتم كتابة التصويت بالصيغة التالية:

الاسم الأول الاسم الأخير رقم المنطقة عدد الأصوات
لجنة الانتخابات تريد المخرجات بالصيغة المجدولة التالية:

----- نتائج الانتخابات -----

الأصوات

| اسم المرشح | المنطقة 1 | المنطقة 2 | المنطقة 3 | المنطقة 4 | الاجمالي |
|-------------|-----------|-----------|-----------|-----------|----------|
| بادي بالتو | 0 | 0 | 0 | 272 | 272 |
| دكتور دوك | 25 | 71 | 156 | 97 | 349 |
| داكي دونالد | 110 | 158 | 0 | 0 | 268 |

الفائز: ؟؟؟، الأصوات المتلقاة: ؟؟؟

اجمال الأصوات المقترعة: ؟؟؟

يجب أن تكون أسماء المرشحين مرتبة أبجدياً في المخرجات.

من أجل هذا البرنامج نفترض أن هناك ستة مرشحين يسعون الى وظيفة رئيس مجلس الطلاب. هذا البرنامج يمكن تعزيه للتعامل مع أي عدد من المرشحين.

يتم تقديم البيانات في ملفين: الأول هو candData.txt الذي يتكون من أسماء المرشحين الساعين الى وظيفة الرئيس. أسماء المرشحين في هذا الملف لا تكون متية ترتيبياً معيناً. في الملف الثاني voteData.txt يتكون كل صف من نتائج التصويت بالصيغة التالية:

الاسم الأول الاسم الأخير رقم المنطقة عدد الأصوات

هذا يعني أن كل صف في الملف voteData.txt يتكون من اسم المرشح ورقم المنطقة وعدد الأصوات المتلقاة من المرشح في هذه المنطقة. يوجد مدخل واحد في كل صف. على سبيل المثال يكون ملف المدخلات المحتوي على بيانات التصويت كما يلي:

| | | |
|-------------|---|----|
| داكي دونالد | 1 | 23 |
| بيتر بلوتو | 2 | 56 |
| دكتور دوك | 1 | 25 |
| بيتر بلوتو | 4 | 23 |

الصف الأول يشير الى أن داكي دونالد تلقى 23 صوت من المنطقة 1.

المدخلات: ملفان: ملف يحتوي على أسماء المرشحين والآخر يحتوي على بيانات التصويت كما تم وصفها من قبل.

المخرجات: نتائج الانتخابات في صيغة جدول كما تم وصفها من قبل والفائز.

تحليل المشكلة وتصميم الخوارزمية:

من المخرجات يتضح أن البرنامج يجب ان يقوم بتنظيم بيانات التصويت بالمنطقة وحساب الأصوات الكلية التي تم تلقيها بواسطة كل مرشح وكذلك التي تم اقتراعها للانتخاب. فضلاً عن هذا يجب أن تظهر أسماء المرشحين بترتيب أبجدي.

المكون الأساسي من هذا البرنامج هو المرشح. لهذا نقوم أولاً بتصميم الفئة candidateType لتطبيق عنصر المرشح. كل مرشح له اسم ويتلقى أصوات. بما أن هناك أربعة مناطق اذن يمكننا استخدام مصفوفة من أربع مكونات. في المثال 1-6 (الفصل 1) قمنا بتصميم الفئة personType لتطبيق اسم الشخص. تذكر أن هدف من النوع personType يمكنه تخزين الاسم الأول والاسم الأخير. الآن بعد أن قمنا بمناقشة ائصال المعامل (انظر الفصل 2) يمكننا اعادة تصميم الفئة personType وتعريف المعاملات المرتبطة بها حتى يمكن المقارنة بين أسماء شخصين. كما يمكننا ائصال معامل الاسناد لسهولة الاسناد ونستخدم مهاملات ادخال واستخلاص التدفق من أجل المدخلات / المخرجات.

بما أن كل مرشح عبارة عن شخص فاننا نستمند الفئة candidateType من الفئة personType.

personType

تقوم الفئة personType بتطبيق الاسم الأول والاسم الأخير للفرد. لهذا يكون للفئة personType عنصري بيانات هما: عنصر البيانات firstName لتخزين الاسم الأول وعنصر البيانات lastName لتخزين الاسم الأخير. اننا نعلن هذه العناصر كعناصر محمية حتى يمكن مد تعريف الفئة

personType بسهولة اكون متلأمة مع متطلبات تطبيق محدد مطلوب لتطبيق اسم الشخص.
تعريف الفئة personType معطى فيما بعد (انظر شكل 10-65 أيضاً).

```
#include <iostream>
#include <string>

using namespace std;

class personType
{
    //Overload the stream insertion and extraction operators.
    friend istream& operator>>(istream&, personType&);
    friend ostream& operator<<(ostream&, const personType&);

public:
    const personType& operator=(const personType&);
    //Overload the assignment operator.

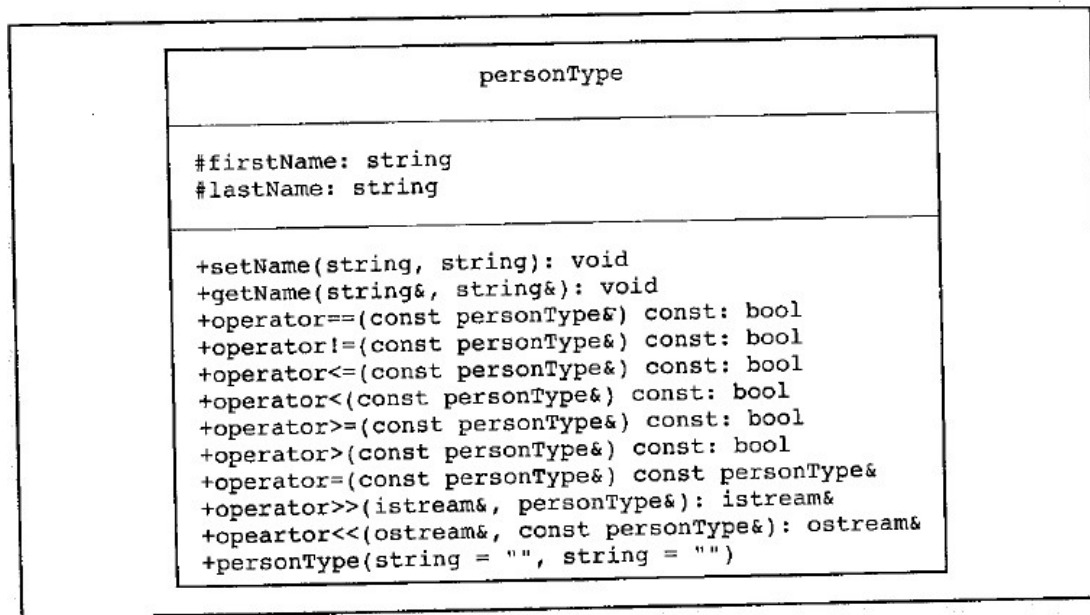
    void setName(string first, string last);
    //Function to set firstName and lastName according to
    //the parameters.
    //Postcondition: firstName = first; lastName = last

    void getName(string& first, string& last);
    //Function to return firstName and lastName via the
    //parameters.
    //Postcondition: first = firstName; last = lastName

    personType(string first = "", string last = "");
    //constructor with parameters
    //Set firstName and lastName according to the parameters.
    //Postcondition: firstName = first; lastName = last

    //Overload the relational operators.
    bool operator==(const personType& right) const;
    bool operator!=(const personType& right) const;
    bool operator<=(const personType& right) const;
    bool operator<(const personType& right) const;
    bool operator>=(const personType& right) const;
    bool operator>(const personType& right) const;

protected:
    string firstName; //variable to store the first name
    string lastName;  //variable to store the last name
};
```



شكل 10-65: رسم لغة التشكيل الموحدة للفئة personType
 نعطي تعريفات الدالات فقط لاثقال المعاملات = = و >> ونترك المعاملات الأخرى كتمرين
 لك. انظر تمرين البرمجة رقم 11 في نهاية هذا الفصل.

```
//Overload the operator ==
bool personType::operator==(const personType& right) const
{
    return(firstName == right.firstName
           && lastName == right.lastName);
}

//Overload the stream extraction operator.
istream& operator>>(istream& isObject, personType& pName)
{
    isObject>>pName.firstName>>pName.lastName;
    return isObject;
}
```

:candidateType

المكون الأساسي من هذا البرنامج هو المرشح الذي تم وصفه وتطبيقه في هذا القسم. كل مرشح له اسم أول واسم أخير ويتلقى أصوات. بما أن هناك أربعة مناطق فإننا نعلن مصفوفة من أربع مكونات لتتبع أصوات كل منطقة. كما نحتاج إلى عنصر بيانات لتخزين إجمالي عدد الأصوات التي تلقاها كل مرشح. بما أن كل مرشح عبارة عن شخص وبما أننا قمنا بتصميم فئة لتطبيق الاسم الأول والأخير فإننا نستخدم الفئة candidateType من الفئة personType.

بما أن عناصر بيانات الفئة personType عناصر محمية فإن عناصر البيانات هذه يمكن تناولها بشكل مباشر في الفئة candidateType.

هناك ستة مرشحين ولهذا نعلن قائمة من ستة مرشحين من النوع candidateType. هذا الفصل يقوم بتعريف وتطبيق الفئة orderedArrayListType لتطبيق قائمة مرتبة ونستخدم هذه الفئة للاحتفاظ بقائمة من المرشحين. سوف يتم ترتيب وبحث هذه القائمة من المرشحين. لهذا يجب أن نقوم بتعريف (أي انقال) عوامل الاسناد والعوامل المرتبطة للفئة candidateType لأن هذه العوامل يتم استخدامها بواسطة خوارزميات الترتيب والبحث.

البيانات الموجودة في الملف المحتوي على بيانات المرشحين تتكون من أسماء المرشحين فقط. لهذا بالإضافة الى انقال معامل الاسناد حتى يمكن تحديد قيمة هدف عند هدف آخر فاننا نقوم كذلك بانقال معامل الاسناد للفئة candidateType حتى يمكن فقط تحديد اسم المرشح لهدف مرشح. هذا يعني أننا ننقل معامل الاسناد مرتين: مرة من أجل الأهداف من النوع candidateType ومرة أخرى من أجل الأهداف من الأنواع candidateType و personType (انظر كذلك شكل 10-66).

```
#include <string>
#include "personType.h"
using namespace std;
const int noOfRegions = 4;
class candidateType: public personType
{
public:
    const candidateType& operator=(const candidateType&);
    //Overload the assignment operator for the objects of the
    //type candidateType.

    const candidateType& operator=(const personType&);
    //Overload the assignment operator for the objects so
    //that the value of an object of the type personType can
    //be assigned to an object of the type candidateType.

    void setVotes(int region, int votes);
    //Function to set the votes of a candidate for a
    //particular region.
    //Postcondition: The votes specified by the parameter votes
    //                are assigned to the region specified by the
    //                parameter region.

    void updateVotesByRegion(int region, int votes);
    //Function to update the votes of a candidate for a
    //particular region.
    //Postcondition: The votes specified by the parameter votes
    //                are added to the region specified by the
    //                parameter region.
```

```

void calculateTotalVotes();
//Function to calculate the total votes received by a
//candidate.
//Postcondition: The votes received in each region are added.

int getTotalVotes();
//Function to return the total votes received by a
//candidate.
//Postcondition: The total votes received by the candidate
// are returned.

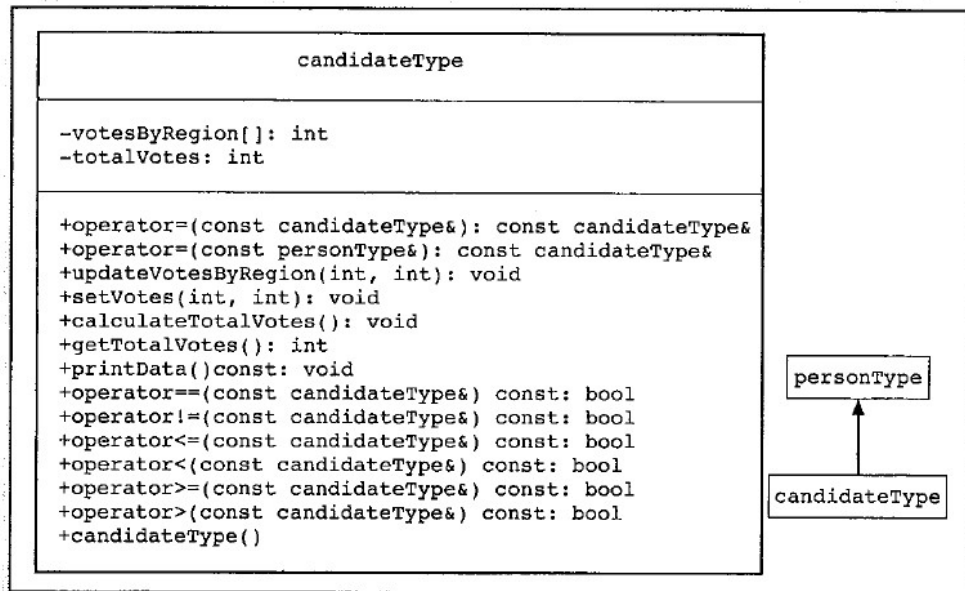
void printData() const;
//Function to output the candidate's name, the votes
//received in each region, and the total votes received.

candidateType();
//default constructor
//Postcondition: Initialize the votes received in each
// region, and the total votes received, to zero.

//Overload the relational operators.
bool operator==(const candidateType& right) const;
bool operator!=(const candidateType& right) const;
bool operator<=(const candidateType& right) const;
bool operator<(const candidateType& right) const;
bool operator>=(const candidateType& right) const;
bool operator>(const candidateType& right) const;

private:
int votesByRegion[noOfRegions];
int totalVotes;
};

```



شكل 10-66: رسم لغة التشكيل الموحدة للفئة candidateType

تعريفات دالات عنصر الفئة candidateType معطاة فيما بعد.

لتحديد اصوات منطقة معينة يتم تمرير رقم المنطقة وعدد الأصوات كمعاملات للدالة setVotes. بما أن مؤشر المصفوفة يبدأ عند صفر فإن المنطقة 1 تقابل مكون المصفوفة في الموضع صفر وهكذا. لهذا من أجل تحديد قيمة مكون المصفوفة الصحيح يتم طرح 1 من المنطقة. تعريف الدالة setVotes هو:

```
void candidateType::setVotes(int region, int votes)
{
    votesByRegion[region - 1] = votes;
}
```

لتحديث الأصوات لمنطقة معينة يتم تمرير رقم المنطقة وعدد الأصوات لهذه المنطقة كمعاملات. بعد هذا يتم اضافة الأصوات الى قيمة المنطقة السابقة. تعريف الدالة updateVotesByRegion هو:

```
void candidateType::updateVotesByRegion(int region, int votes)
{
    votesByRegion[region - 1] = votesByRegion[region - 1]
    + votes;
}
```

تعريفات الدالات calculateTotalVotes، getTotalVotes، و printData والمقوم الافتراضي، و getName معطاة فيما بعد.

```
void candidateType::calculateTotalVotes()
{
    int i;
    totalVotes = 0;
    for(i = 0; i < noOfRegions; i++)
        totalVotes += votesByRegion[i];
}

int candidateType::getTotalVotes()
{
    return totalVotes;
}

void candidateType::printData() const
{
    cout<<left
        <<setw(10)<<firstName<<" "
        <<setw(10)<<lastName<<" ";

    cout<<right;

    for(int i = 0; i < noOfRegions; i++)
        cout<<setw(7)<<votesByRegion[i]<<" ";
    cout<<setw(7)<<totalVotes<<endl;
}

candidateType::candidateType()
{
    for(int i = 0; i < noOfRegions; i++)
        votesByRegion[i] = 0;

    totalVotes = 0;
}
```

لاثقالة العوامل المترابطة للفة candidateType تتم مقارنة أسماء المرشحين. على سبيل المثال يكون المرشحان متماثلين اذا كان لهما نفس الاسم. تعريفات هذه الدالات مماثلة لتعريفات دالات اثقال العوامل المترابطة للفة personType. اننا نعطي تعريف الدالة لاثقال المعامل == ونترك لك كتمرين بقية العوامل. انظر تمرين البرمجة رقم 11.

```
bool candidateType::operator==(const candidateType& right) const
{
    return(firstName == right.firstName
           && lastName == right.lastName);
}
```

تعريفات الدالات لاثقال عوامل الاسناد للفة candidateType متروكة كتمرين لك. انظر تمرين البرمجة 11.

البرنامج الأساسي:

الآن بعد تصميم الفة candidateType نقوم بالتركيز على تصميم البرنامج الأساسي. بما أن هناك ستة مرشحين نقوم بعمل قائمة candidateList تحتوي على ست مكونات من النوع candidateType. الشئ الأول الذي يجب أن يقوم به البرنامج هو قراءة اسم كل مرشح من الملف candData.txt داخل القائمة candidateList. بعد هذا نقوم بترتيب candidateList. الخطوة التالية هي معالجة بيانات التصويت من الملف voteData.txt الذي يضم بيانات التصويت. بعد معالجة بيانات التصويت يجب أن يقوم البرنامج بحساب اجمالي الأصوات التي تلقها كل مرشح ثم طباعة البيانات كما هي موضحة من قبل. لهذا الخوارزمية العامة تكون:

1. اقرأ اسم كل مرشح موجود بداخل candidateList.

2. رتب candidateList.

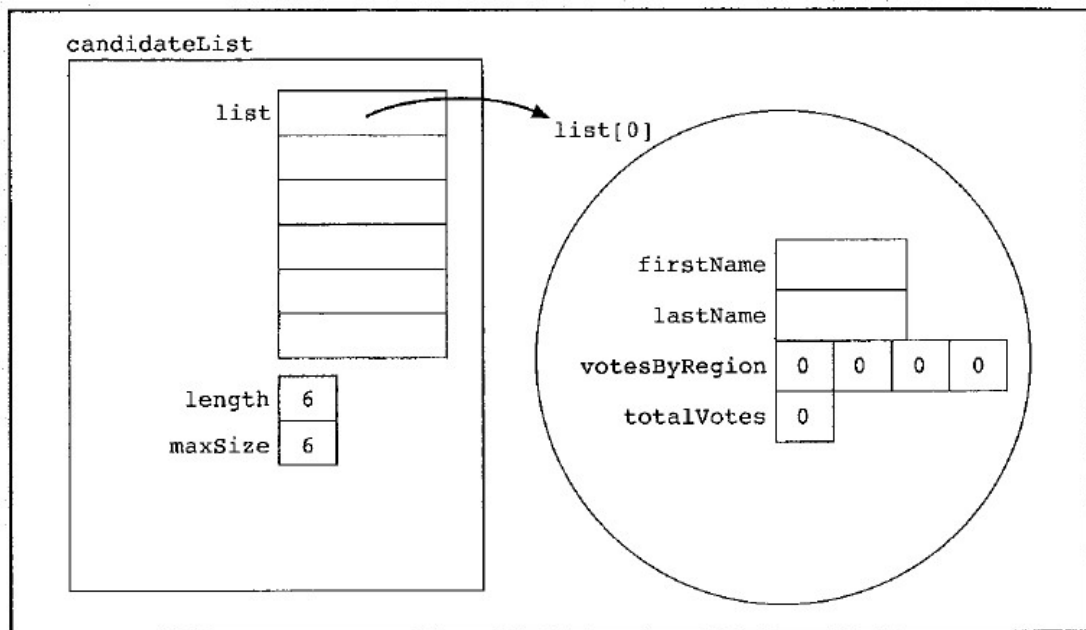
3. عالج بيانات التصويت.

4. اطبع النتائج.

البيان التالي يصنع الهدف candidateList من النوع orderedArrayListType:

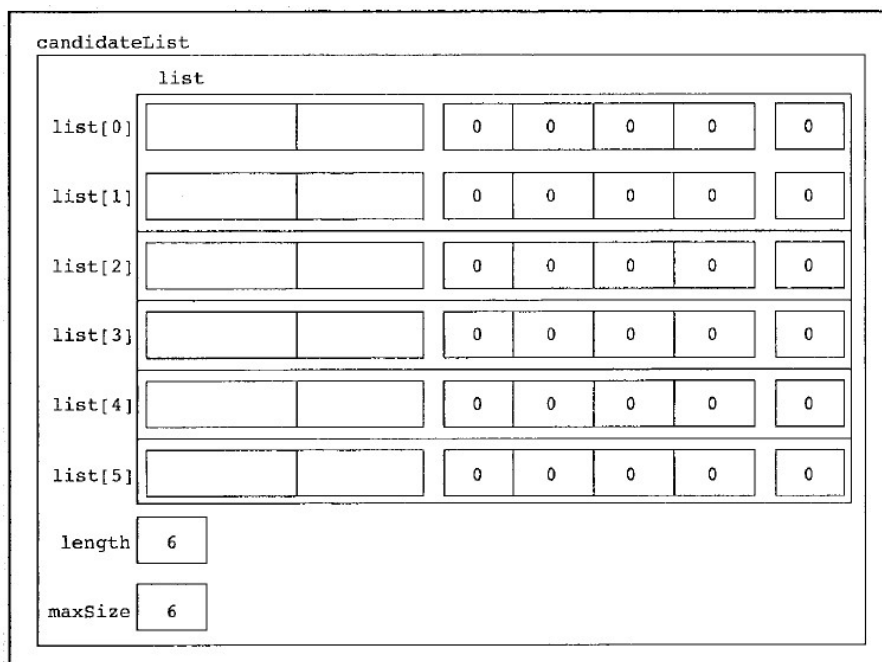
```
orderedArrayListType<candidateType> candidateList(noOfCandidates);
```

شكل 67-10 يوضح الهدف candidateList. كل مكون من المصفوفة list هدف من النوع candidateType:



شكل 67-10: candidateList

في شكل 67-10 تمت تهيئة المصفوفة `votesByRegion` والمتغير `totalVotes` عند صفر بواسطة المقوم الافتراضي للفئة `candidateType`. للاحتفاظ بمساحة عند الحاجة سوف نرسم الهدف `candidateList` كما هو موضح في الشكل 68-10.



شكل 68-10: الهدف candidateList

:fillNames

الشيء الأول الذي يجب أن يقوم به البرنامج هو قراءة أسماء المرشحين بداخل candidateList. لهذا نقوم بكتابة دالة لعمل هذه المهمة. الملف candData.txt يتم فتحه في الدالة main. اسم ملف المدخلات و candidateList يتم تمريرهما كمعاملات للدالة fillNames. بما أن عنصر بيانات الهدف candidateList عبارة عن عنصر بيانات محمي فلا يمكن تناوله مباشرةً. لهذا نقوم بعمل هدف temp من النوع candidateType لتخزين أسماء المرشحين ولإستخدام الدالة insertAt (من الفئة ArrayListType) لتخزين اسم كل مرشح في الهدف candidateList. تعريف الدالة fillNames يكون كما يلي:

```
void fillNames(ifstreams inFile,
               orderedArrayListType<candidateType>& cList)
{
    string firstN;
    string lastN;
    int i;
    candidateType temp;

    for(i = 0; i < noOfCandidates; i++)
    {
        inFile>>firstN>>lastN;
        temp.setName(firstN, lastN);
        cList.insertAt(i, temp);
    }
}
```

بعد استدعاء الدالة fillNames يوضح الشكل 10-69 الهدف candidateList.

| candidateList | | | | | | |
|---------------|--------|--------|---|---|---|---|
| list | | | | | | |
| list[0] | Goldy | Goofy | 0 | 0 | 0 | 0 |
| list[1] | Monty | Mickey | 0 | 0 | 0 | 0 |
| list[2] | Ducky | Donald | 0 | 0 | 0 | 0 |
| list[3] | Peter | Pluto | 0 | 0 | 0 | 0 |
| list[4] | Doctor | Doc | 0 | 0 | 0 | 0 |
| list[5] | Buddy | Balto | 0 | 0 | 0 | 0 |
| length | 6 | | | | | |
| maxSize | 6 | | | | | |

شكل 10-69: الهدف candidateList بعد استدعاء الدالة fillNames
ترتيب الأسماء:

بعد قراءة أسماء المرشحين نقوم بعد هذا بترتيب المصفوفة list من الهدف candidateList باستخدام أي من خوارزميات الترتيب (القائمة على المصفوفة) التي نوقشت في هذا الفصل. بما أن candidateList هدف من النوع orderedArrayListType فإن جميع خوارزميات الترتيب التي نوقشت في هذا الفصل متاحة لها.

لأهداف توضيحية نستخدم الترتيب بالاختيار. البيان التالي ينجز هذه المهمة:

candidateList.selectionSort ();

بعد تنفيذ هذا البيان تكون candidateList كما هي موضحة في شكل 10-70.

| | | | | | | | | | |
|---------------|--------|--------|---|---|---|---|---|--|--|
| candidateList | | | | | | | | | |
| list | | | | | | | | | |
| list[0] | Buddy | Balto | 0 | 0 | 0 | 0 | 0 | | |
| list[1] | Doctor | Doc | 0 | 0 | 0 | 0 | 0 | | |
| list[2] | Ducky | Donald | 0 | 0 | 0 | 0 | 0 | | |
| list[3] | Goldy | Goofy | 0 | 0 | 0 | 0 | 0 | | |
| list[4] | Monty | Mickey | 0 | 0 | 0 | 0 | 0 | | |
| list[5] | Peter | Pluto | 0 | 0 | 0 | 0 | 0 | | |
| length | 6 | | | | | | | | |
| maxSize | 6 | | | | | | | | |

شكل 10-70: الهدف candidateList بعد تنفيذ البيان (); candidateList.selectionSort معالجة بيانات التصويت:

نناقش الآن كيفية معالجة بيانات التصويت. كل مدخل في الملف voteData.txt يكون بالصيغة:
الاسم الأول الاسم الأخير رقم المنطقة عدد الأصوات
بعد قراءة أي مدخل من الملف voteData.txt نحدد الصف في المصفوفة list (من الهدف candidateList) الذي يتوافق مع المرشح المحدد ونقوم بتحديث المدخل المحدد بواسطة regionNumber.

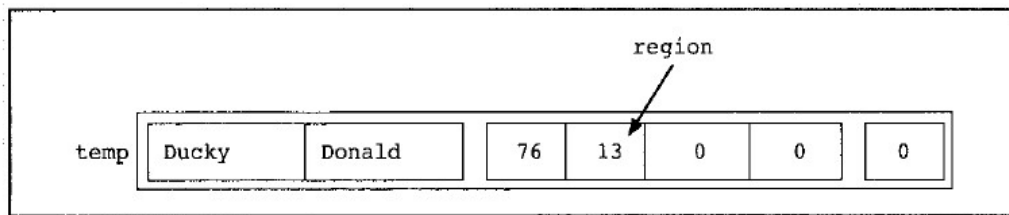
المكون votesByRegion عنصر بيانات خاص لكل مكون من المصفوفة list. فضلاً عن هذا تكون list عنصر بيانات خاص من candidateList. الطريقة الوحيدة التي يمكننا بها تحديث أصوات المرشحين هي عمل نسخة من سجل هذا المرشح داخل هدف مؤقت، وتحديث الهدف، ثم نسخ الهدف المؤقت مرة أخرى داخل list عن طريق استبدال القيمة القديمة بقيمة جديدة من الهدف المؤقت. اننا نستخدم دالة العنصر retrieveAt لعمل نسخة من المرشح التي تحتاج أصواته الى تحديث. بعد تحديث الهدف المؤقت نستخدم دالة العنصر replaceAt لنسخ الهدف المؤقت مرة أخرى داخل القائمة. افترض أن المدخل التالي المقروء هو:

داكي دونالد 2 35

يقول هذا المدخل أن داكي دونالد يتلقى 35 صوت من المنطقة 2. افترض أنه قبل معالجة هذا المدخل تكون candidateList كما هي موضحة في شكل 10-71.

| candidateList | | | | | | |
|---------------|--------|--------|-----|----|----|----|
| list | | | | | | |
| list[0] | Buddy | Balto | 0 | 0 | 50 | 0 |
| list[1] | Doctor | Doc | 10 | 0 | 56 | 0 |
| list[2] | Ducky | Donald | 76 | 13 | 0 | 0 |
| list[3] | Goldy | Goofy | 0 | 45 | 0 | 0 |
| list[4] | Monty | Mickey | 80 | 0 | 78 | 0 |
| list[5] | Peter | Pluto | 100 | 0 | 0 | 20 |
| length | 6 | | | | | |
| maxSize | 6 | | | | | |

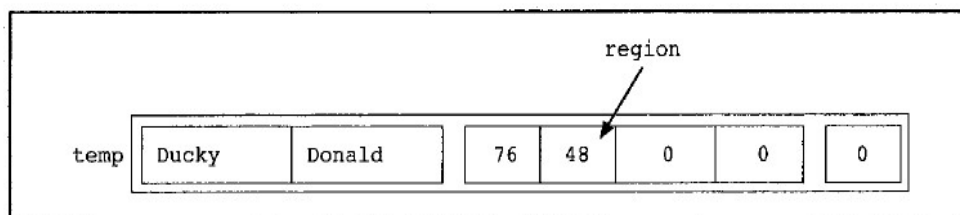
شكل 10-71: الهدف candidateList قبل معالجة المدخل داكي دونالد 2 35
نقوم بعمل نسخة من الصف المقابل لداكي دونالد (انظر شكل 10-72).



شكل 10-72: الهدف temp
بعد هذا يقوم البيان التالي بتحديث بيانات التصويت للمنطقة 2. (هنا المنطقة = 2 والأصوات = 35).

`temp.updateVotesByRegion(region, votes);`

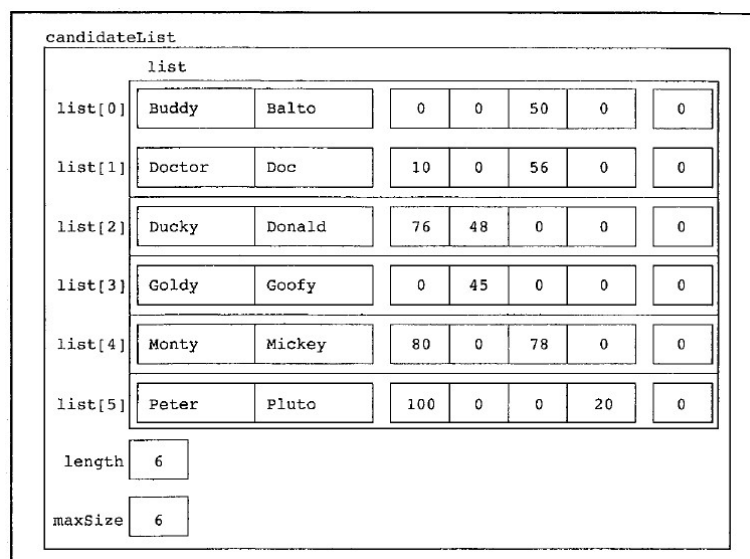
بعد تنفيذ هذا البيان يكون الهدف temp كما هو موضح في شكل 10-73.



شكل 10-73: الهدف temp بعد تنفيذ البيان

`temp.updateVotesByRegion(region, votes);`

الآن نقوم بنسخ الهدف temp بداخل list (انظر الشكل 10-74).



شكل 10-74: candidateList بعد نسخ temp.

بما أن العنصر list من candidateList مرتب اذن يمكننا استخدام خوارزمية البحث الثنائي لايجاد موضع الصف في القائمة المقابل للمرشح الذي تحتاج أصواته الى تحديث. كذلك تكون الدالة binarySearch عنصر من الفئة orderedArrayListType ولهذا يمكننا استخدام هذه الدالة لبحث المصفوفة list. نترك لك كتمرين تعريف الدالة processVotes لمعالجة بيانات التصويت (انظر تمرين البرمجة 11 في نهاية هذا الفصل).

جمع الأصوات:

بعد معالجة بيانات التصويت الخطوة التالية تكون ايجاد اجمالي الأصوات التي تلقاها كل مرشح. يتم القيام بهذا عن طريق جمع الأصوات التي تم تلقيها في كل منطقة. الآن يكون votesByRegion عبارة عن عنصر بيانات خاص من الفئة candidateType وتكون list عنصر بيانات محمي من candidateList. لهذا ومن أجل جمع الأصوات لكل مرشح نقوم باستخدام الدالة retrieveAt لعمل نسخة مؤقتة من بيانات كل مرشح، وجمع الأصوات في الهدف المؤقت ثم نسخ الهدف المؤقت مرة أخرى بداخل candidateList. الدالة التالية تقوم بهذا:

```
void addVotes(orderedArrayListType<candidateType>& cList)
{
    int i;
    candidateType temp;
    for(i = 0; i < noOfCandidates; i++)
    {
        cList.retrieveAt(i, temp);
        temp.calculateTotalVotes();
        cList.replaceAt(i, temp);
    }
}
```

الشكل 10-75 يوضح candidateList بعد جمع الأصوات لكل مرشح – أي بعد استدعاء الدالة .addVotes

| candidateList | | | | | | |
|---------------|--------|--------|-----|-----|-----|-----|
| list | | | | | | |
| list[0] | Buddy | Balto | 0 | 0 | 0 | 272 |
| list[1] | Doctor | Doc | 25 | 71 | 156 | 97 |
| list[2] | Ducky | Donald | 110 | 158 | 0 | 0 |
| list[3] | Goldy | Goofy | 75 | 34 | 134 | 0 |
| list[4] | Monty | Mickey | 112 | 141 | 156 | 89 |
| list[5] | Peter | Pluto | 285 | 56 | 0 | 46 |
| length | 6 | | | | | |
| maxSize | 6 | | | | | |

شكل 10-75: candidateList بعد استدعاء الدالة addVotes

طباعة العنوان وطباعة النتائج:

لاكمال البرنامج ندخل دالة لطباعة العنوان وهو أول أربعة سطور من المخرجات. الدالة التالية تحقق هذه المهمة:

```
void printHeading()
{
    cout<<"-----Election Results-----"
    <<"-----"<<endl<<endl;
    cout<<"                      Votes"<<endl;
    cout<<"      Candidate Name      Region1  Region2  Region3  "
    <<"Region4      Total"<<endl;
    cout<<"-----"
    <<"-----"
    <<"-----"
    <<"-----"
    <<endl;
}
```

الآن نقوم بوصف الدالة printResults التي تقوم بطباعة النتائج. افترض أن المتغير sumVotes يضم الأصوات الاجمالية التي تم اقتراعها من أجل الانتخابات والمتغير largestVotes يضم أكبر عدد من الأصوات التي يتلقاها مرشح، والمتغير winLoc يضم مؤشر المرشح الفائز في المصفوفة list. افترض كذلك أن temp عبارة عن هدف من النوع candidateType. خوارزمية هذه الدالة هي:

1. تهيئة sumVotes، largestVotes، و winLoc عند صفر.

2. بالنسبة الى كل مرشح:

- أ- استعادة بيانات المرشح بداخل temp.
- ب- طباعة اسم المرشح والبيانات المرتبطة به.

ت- استعادة الأصوات الاجمالية التي تلقاها المرشح وتحديث sumVotes.

```
if(largestVotes < temp.getTotalVotes())
{
    largestVotes = temp.getTotalVotes();
    winLoc = i;
}
```

3. اخراج السطور النهائية من المخرجات.

نترك لك كتمرين تعريف الدالة printResults لطباعة النتائج. انظر تمرين البرمجة 11 في نهاية هذا الفصل.

قائمة البرنامج (البرنامج الرئيسي):

```
#include <iostream>
#include <string>
#include <fstream>
#include "candidateType.h"
#include "orderedArrayListType.h"

using namespace std;

const int noOfCandidates = 6;
```

```

void fillNames(ifstream& inFile,
               orderedArrayListType<candidateType>& cList);
void processVotes(ifstream& inFile,
                  orderedArrayListType<candidateType>& cList);
void addVotes(orderedArrayListType<candidateType>& cList);
void printHeading();
void printResults(orderedArrayListType<candidateType>& cList);

int main()
{
    orderedArrayListType<candidateType> candidateList(noOfCandidates)
    candidateType temp;
    ifstream inFile;
    inFile.open("a:\\candData.txt");
    fillNames(inFile, candidateList);
    candidateList.selectionSort();
    inFile.close();
    inFile.open("a:\\voteData.txt");
    processVotes(inFile, candidateList);
    addVotes(candidateList);
    printHeading();
    printResults(candidateList);
    return 0;
}

```

// ضع تعريفات الدالات ،fillNames و،addVotes

// وprintHeading هنا. قم كذلك بكتابة ووضع تعريفات

// الدالات processVotes وprintResults هنا.

مخرجات العينة: (بعد أن قمت بكتابة تعريفات الدالات personType و candidateType والدالات

processVotes و printResults ثم قم بتشغيل برنامجك. يجب أن ينتج المخرجات التالية. (انظر

تمرين البرمجة 11).

----- نتائج الانتخابات -----

| اسم المرشح | المنطقة 1 | المنطقة 2 | المنطقة 3 | المنطقة 4 | الاجمالي |
|------------|-----------|-----------|-----------|-----------|----------|
| بادي | 0 | 0 | 0 | 272 | 272 |
| دكتور | 25 | 71 | 156 | 97 | 349 |

| | | | | | | |
|-----|----|-----|-----|-----|--------|-------|
| 268 | 0 | 0 | 158 | 110 | دونالد | داكي |
| 243 | 0 | 134 | 34 | 75 | جووفي | جولدي |
| 498 | 89 | 156 | 141 | 112 | ميكي | مونتي |
| 387 | 46 | 0 | 56 | 285 | بلوتو | بيتر |

الفائز: مونتي ميكي، الأصوات المتلقاة: 498

اجمالي الأصوات المقترعة: 2017

ملف المدخلات:

candData.txt

جولدي جووفي

مونتي ميكي

داكي دونالد

بيتر بلوتو

دكتور دوك

بادي بالتو

voteData.txt

جولدي جووفي 2 34

مونتي ميكي 1 56

داكي دونالد 2 56

بيتر بلوتو 1 78

دكتور دوك 4 29

بادي بالتو 4 78

مونتي ميكي 2 63

داكي دونالد 1 23

بيتر بلوتو 2 56

دكتور دوك 1 25

بيتر بلوتو 4 23

دكتور دوك 4 12

جولدي جووفي 3 134

بادي بالتو 4 82

مونتي ميكي 3 67

داكي دونالد 2 67

دكتور دوك 3 67

بادي بالتو 4 23

مونتي ميكي 1 56

داكي دونالد 2 35

بيتر بلوتو 1 27

دكتور دوك 2 34

جولدي جووفي 1 75

بيتر بلوتو 4 23

مونتي ميكي 4 89

بيتر بلوتو 1 23

دكتور دوك 3 89

مونتي ميكي 3 89

| | | |
|-------------|---|----|
| دكتور دوک | 2 | 37 |
| بادي بالتو | 4 | 89 |
| مونتي ميكي | 2 | 78 |
| داكي دونالد | 1 | 87 |
| بيتر بلوتو | 1 | 90 |
| دكتور دوک | 4 | 56 |

مراجعة سريعة

1. خوارزمية الترتيب بالاختيار تقوم بترتيب القائمة عن طريق ايجاد العنصر الأصغر (أو الأكبر) في القائمة ثم نقله الى بداية (أو نهاية) القائمة.
2. بالنسبة الى قائمة طولها n حيث $n > 0$ تقوم خوارزمية الترتيب بالاختيار بعمل $\frac{1}{2}n^2 - \frac{1}{2}n$ مقارنة و $3(n-1)$ اسنادات عنصر.
3. بالنسبة الى قائمة طولها n حيث $n > 0$ تقوم خوارزمية الترتيب بالادخال في المتوسط بعمل $\frac{1}{4}n^2 + O(n)$ و $\frac{1}{4}n^2 + O(n)$ عنصر.
4. لتكن L قائمة بها عدد n من العناصر المستقلة. أي خوارزمية ترتيب تقوم بترتيب القائمة L عن طريق مقارنة المفاتيح فقط في أسوأ حالاتها بعمل $O(n \log_2 n)$ فاتيح على الأقل.
5. تقوم كل من خوارزميات الترتيب السريع والترتيب بالدمج بترتيب قائمة عن طريق تقسيم القائمة.
6. لتقسيم قائمة تقوم خوارزمية الترتيب السريع أولاً باختيار عنصر من القائمة يسمى محور ثم تقوم الخوارزمية بعد هذا باعادة ترتيب العناصر حتى تكون العناصر الموجودة في واحدة من القوائم الفرعية أصغر من المحور والعناصر الموجودة في القائمة الفرعية الأخرى أكبر من أو تساوي المحور.
7. في الترتيب السريع يتم القيام بعمل الترتيب في تقسيم القائمة.
8. عدد مقارنات المفتاح في المتوسط في الترتيب السريع هو $O(n \log_2 n)$ أسوأ الحالات يكون عدد مقارنات المفتاح في الترتيب السريع هو $O(n^2)$.
9. خوارزمية الترتيب بالدمج تقوم بتقسيم القائمة عن طريق قسمتها في المنتصف.
10. في الترتيب بالدمج يتم القيام بعمل الترتيب في دمج القائمة.
11. عدد مقارنات المفتاح في الترتيب بالدمج يكون $O(n \log_2 n)$.
12. الكومة عبارة عن قائمة يحتوب كل عنصر بها على مفتاح حيث أن المفتاح في العنصر عند الموضع k في القائمة يكون على الأقل كبير مثل المفتاح في العنصر عند الموضع $2k + 1$ (إذا وجد) و $2k + 2$ (إذا وجد).

13. الخطوة الأولى في خوارزمية الترتيب بالتكدس هي تحويل القائمة الى كومة تسمى buildHeap. بعد قيامنا بتحويل المصفوفة الى كومة تبدأ مرحلة الترتيب.
14. افترض أن L قائمة بها n عنصر حيث $n > 0$. في أسوأ حالة يكون عدد مقارنات المفتاح في خوارزمية الترتيب بالتكدس لترتيب L هو $2n \cdot \log_2 n + O(n)$ أيضاً يكون عدد اسنادات العنصر في خوارزمية الترتيب بالتكدس لترتيب L هو $n \cdot \log_2 n + O(n)$

تمارين

- افترض القائمة التالية من المفاتيح:
5، 18، 21، 10، 55، 20
أول ثلاثة مفاتيح مرتبة. لنقل 10 الى مكانها الصحيح باستخدام خوارزمية الترتيب بالادخال كما تم وصفه في هذا الفصل، كم عدد مقارنات المفتاح التي يتم تنفيذها؟
- افترض القائمة التالية من المفاتيح:
7، 28، 31، 40، 5، 20
أول أربعة مفاتيح مرتبة. لنقل 5 الى مكانها الصحيح باستخدام خوارزمية الترتيب بالادخال كما تم وصفه في هذا الفصل، كم عدد مقارنات المفتاح التي يتم تنفيذها؟
- افترض القائمة التالية من المفاتيح:
28، 18، 21، 10، 25، 30، 12، 71، 32، 58، 15
سوف يتم ترتيب هذه القائمة باستخدام خوارزمية الترتيب بالادخال كما تم وصفها في هذا الفصل من أجل القوائم المبنية على مصفوفة، وضح القائمة الناتجة بعد 6 حالات مرور بمرحلة الترتيب – أي بعد 6 مكررات من الحلقة for.
- تذكر خوارزمية الترتيب بالادخال (النسخة المتجاورة) كما نوقشت في هذا الفصل. افترض القائمة التالية من المفاتيح:
18، 8، 11، 9، 15، 20، 32، 61، 22، 48، 75، 83، 35، 3
كم عدد مقارنات المفتاح التي يتم تنفيذها لترتيب هذه القائمة باستخدام خوارزمية الترتيب بالادخال؟
- كل من خوارزميات الترتيب بالدمج والترتيب السريع تقوم بترتيب القائمة عن طريق تقسيمها. وضح كيف تختلف خوارزمية الترتيب بالدمج عن خوارزمية الترتيب السريع في تقسيم القائمة.
- افترض القائمة التالية من المفاتيح:
16، 38، 54، 80، 22، 65، 55، 48، 64، 95، 5، 100، 58، 25، 36
هذه القائمة سوف يتم ترتيبها باستخدام خوارزمية الترتيب السريع كما نوقشت في هذا الفصل. استخدم المحور كعنصر أوسط في القائمة.
- أ- اعطي القائمة الناتجة بعد استدعاء واحد لاجراء التقسيم.

ب- اعطي القائمة الناتجة بعد استدعاءين لاجراء التقسيم.

7. افترض القائمة التالية من المفاتيح:

18، 40، 16، 82، 64، 57، 50، 37، 47، 72، 14، 17، 27، 35

هذه القائمة سوف يتم ترتيبها باستخدام خوارزمية الترتيب السريع كما نوقشت في هذا الفصل. استخدم المحور كمتوسط بين العنصر الأول والأخير والأوسط بالقائمة.

أ- ما هو المحور؟

ب- اعطي القائمة الناتجة بعد استدعاء واحد لاجراء التقسيم.

8. استخدم الدالة buildHeap كما تم اعطائها في هذا الفصل لتحويل المصفوفة التالية الى كومة وأوضح الشكل النهائي للمصفوفة.

47، 78، 81، 52، 50، 82، 58، 42، 65، 80، 92، 53، 63، 87، 95، 59، 34، 37، 7، 20

9. افترض أن القائمة التالية تم عملها بالدالة buildHeap أثناء عملية عمل الكومة من خوارزمية الترتيب بالتكدس.

100، 85، 94، 47، 72، 82، 76، 30، 20، 60، 65، 50، 45، 17، 35، 14، 28، 5

وضح المصفوفة الناتجة بعد مرورين بخوارزمية الترتيب بالتكدس. (استخدم الدالة heapify كما هي معطاة في هذا الفصل). كم عدد مقارنات المفتاح التي يتم تنفيذها أثناء المرور الأول.

10. اكتب تعريف الفئة orderedArrayListType التي تقوم بتطبيق خوارزميات البحث (التي تم وصفها في الفصل 9) وخوارزميات الترتيب للقوائم المبنية على المصفوفة كما نوقشت في هذا الفصل.

11. اكتب تعريف الفئة orderedArrayListType التي تقوم بتطبيق خوارزميات البحث (التي تم وصفها في الفصل 5) وخوارزميات الترتيب للقوائم المتصلة كما نوقشت في هذا الفصل.

تمارين برمجة:

1. اكتب واختبر صورة من خوارزمية الترتيب بالاختيار للقوائم المتصلة.

2. اكتب برنامج لاختبار خوارزمية الترتيب بالادخال للقوائم المبنية على مصفوفة كما هي معطاة في هذا الفصل.

3. اكتب برنامج لاختبار خوارزمية الترتيب بالادخال للقوائم المتصلة كما هي معطاة في هذا الفصل.

4. اكتب برنامج لاختبار خوارزمية الترتيب السريع للقوائم المبنية على مصفوفة كما هي معطاة في هذا الفصل.

5. اكتب واختبر صورة من خوارزمية البحث السريع للقوائم المتصلة.

6. (C.A.R.Hoare) لتكن L قائمة حجمها n. يمكن استخدام خوارزمية الترتيب السريع لايجاد أصغر عنصر رقم k في القائمة L حيث $0 < k < n - 1$ دون ترتيب L كلياً. قم بكتابة وتطبيق دالة C++ المسماة kTheSmallestItem التي تستخدم نسخة من خوارزمية الترتيب السريع لتحديد أصغر عنصر k في L بدون ترتيب L تماماً.

7. قم بترتيب مصفوفة من 10.000 عنصر باستخدام خوارزمية الترتيب السريع كما يلي:

- أ- رتب المصفوفة باستخدام المحور كعنصر أوسط من المصفوفة.
- ب- رتب المصفوفة باستخدام المحور كمتوسط بين العنصر الأول والأخير والأوسط بالمصفوفة.
- ت- رتب المصفوفة باستخدام المحور كعنصر أوسط من المصفوفة. بالرغم من هذا عندما يقل حجم أي قائمة فرعية عن 20 رتب القائمة الفرعية باستخدام الترتيب بالادخال.
- ث- رتب المصفوفة باستخدام المحور كمتوسط بين العنصر كمتوسط بين العنصر الأول والأخير والأوسط بالمصفوفة. عندما يقل حجم أي قائمة فرعية عن 20 رتب القائمة الفرعية باستخدام الترتيب بالادخال.

ج- قم بحساب وطباعة زمن الحاسوب لكل من الأربع خطوات التالية،

لايجاد زمن الحاسوب الحالي قم باعلان متغير مثل x من النوع `clock_t`. البيان $x = \text{clock}()$ يقوم بتخزين زمن الحاسوب الحالي في x . يمكنك التحقق من زمن الحاسوب الحالي قبل وبعد مرحلة معينة من البرنامج. بعد هذا لايجاد زمن الحاسوب لهذه المرحلة المعينة من البرنامج قم بطرح زمن ما قبل من زمن ما بعد. كما يجب عليك ادخال الملف الرئيسي `ctime` لاستخدام نوع البيانات `clock_t` والدالة `clock`. استخدم مولد عشوائي لملا المصفوفة مبدئياً.

8. اكتب برنامج لاختبار خوارزمية الترتيب بالدمج للقوائم المتصلة كما هي معطاة في هذا الفصل.

9. اكتب برنامج لاختبار خوارزمية الترتيب بالتكدس للقوائم المتصلة كما هي معطاة في هذا الفصل.

10. أ. اكتب تعريف قالب الفئة لتعريف طوابير الأسبقية كما نوقشت في هذا الفصل كنوع بيانات مجرد.

ب. اكتب تعريفات قوالب الدالة لتطبيق عمليات طوابير الأسبقية كما تم تعريفها في (أ).

ج. اكتب برنامج لاختبار عمليات متنوعة على طوابير الأسبقية.

11. أ. اكتب تعريفات الدالات للفئة `personType` من مثال البرمجة الخاص بنتائج البرمجة الغير معطاة في مثال البرمجة.

ب. اكتب تعريفات دالات الفئة `candidateType` من مثال البرمجة الخاص بنتائج البرمجة الغير معطاة في مثال البرمجة.

ج. اكتب تعريفات الدالات `processVotes` و `printResults` من مثال البرمجة الخاص بنتائج البرمجة.

د. بعد الانتهاء من أ و ب و ج اكتب برنامج لانتاج المخرجات الموضحة في تنفيذ العينة لمثال برمجة نتائج الانتخابات.

12. في مثال البرمجة الخاص بنتائج الانتخابات تحتوي الفئة `candidateType` على الدالة `calculateTotalVotes`. بعد معالجة بيانات التصويت تقوم هذه الدالة بحساب اجمالي عدد الأصوات

التي يتلقاها المرشح. الدالة updateVotesByRegion (من الفئة candidateType) تقوم بتحديث عدد الأصوات لمنطقة معينة فقط. قم بتعديل تعريف هذه الدالة حتى تقوم كذلك بتحديث اجمالي عدد الأصوات التي يتلقاها المرشح. بالقيام بهذا لا يعد هناك حاجة الى الدالة addVotes في البرنامج الرئيسي. قم بتعديل وتشغيل برنامجك بالتعريف المعدل للدالة updateVotesByRegion.

13. في مثال البرمجة الخاص بنتائج الانتخابات يتم اعلان الهدف candidateList من النوع orderedArrayListType لمعالجة بيانات التصويت. عمليات ادخال بيانات المرشح وتحديث واستعادة الأصوات كانت معقدة نوعاً ما. لتحديث أصوات المرشح قمنا بنسخ بيانات هذا المرشح من candidateList داخل هدف مؤقت من النوع candidateList وقمنا بتحديث الهدف المؤقت ثم استبدلنا بيانات المرشح بالهدف المؤقت. هذا يرجع الى حقيقة أن عنصر البيانات list عنصر محمي من CandidateList وكل مكون من list عنصر بيانات خاص. في هذا التمرين عليك تعديل مثال البرمجة الخاص بنتائج الانتخابات لتبسيط تناول بيانات المشح كما يلي: استمد الفئة candidateListType من الفئة orderedArrayListType.

```
class candidateListType: public orderedArrayListType<candidateType>
{
public:
    candidateListType();
        //default constructor
    candidateListType(int size);
        //constructor
    void processVotes(string fName, string lName, int region,
        int votes);
        //Function to update the number of votes for a
        //particular candidate for a particular region.
        //Postcondition: The name of the candidate, the region
        //                    number, and the number of votes are
        //                    passed as parameters.
    void addVotes();
        //Function to find the total number of votes
        //received by each candidate.
    void printResult();
        //Function to output the voting data.
};
```

بما أن الفئة candidateListType مستمدة من الفئة orderedArrayListType والقائمة list عبارة عن عنصر بيانات محمي للفئة orderedArrayListType (موروث من الفئة arrayListType) اذن يمكن تناول القائمة مباشرةً بواسطة أي عنصر من الفئة candidateListType.

اكتب تعريفات دالات عنصر الفئة candidateListType وأعد كتابة وتشغيل برنامجك باستخدام الفئة candidateListType.

الفصل

11

أشجار ثنائية

في هذا الفصل سوف:

- تتعلم ما هي الشجرة الثنائية.
- تستكشف العديد من خوارزميات اجتياز الشجرة الثنائية.
- تتعلم كيفية تنظيم البيانات في شجرة بحث ثنائية.
- تكتشف كيفية ادخال وحذف عنصر في شجرة بحث ثنائية.
- تستكشف خوارزميات اجتياز الشجرة الثنائية الغير تكرارية.
- تتعلم ما هي الشجرات المتوازنة الارتفاع.

عند القيام بترتيب البيانات تكون أولوية البرنامج الأعلى هي ترتيبها بطريقة ما تجعل ادخال وحذف والبحث عن العناصر سريع. لقد رأيت بالفعل كيفية تخزين ومعالجة البيانات في مصفوفة. بما أن المصفوفة هي بنية بيانات تناول عشوائي فان البيانات اذا تم ترتيبها بشكل صحيح يمكننا استخدام خوارزمية بحث مثل الشجرة الثنائية لايجاد واستعادة عنصر من القائمة بشكل فعال. بالرغم من هذا فنحن نعلم أن تخزين البيانات في مصفوفة له حدوده. على سبيل المثال قد يكون ادخال العنصر (خاصةً اذا كانت المصفوفة مرتبة) وحذف عنصر مستهلكاً للوقت بشكل كبير خاصةً اذا كان حجم القائمة كبير للغاية لأن كل من هذه العمليات تتطلب نقل البيانات. للاسراع من ادخال وحذف العنصر قمنا باستخدام قوائم متصلة. ان ادخال وحذف عنصر من قائمة متصلة لا يحتاج الى نقل أية بيانات ونقوم ببساطة بتعديل بعضاً من المؤشرات في القائمة بالرغم من هذا فان واحداً من عيوب القوائم المتصلة أنها يجب أن تتم معالجتها بالتتابع. هذا يعني أننا عند ادخال أو حذف عنصر أو بحث القائمة من أجل عنصر معين يجب أن نبدأ بحثنا عند أول عقدة في القائمة. كما تعلم البحث التتابعي يكون جيد فقط بالنسبة الى القوائم الشديدة الصغر لأن متوسط الطول البحثي للبحث التتابعي هو نصف حجم القائمة.

أشجار ثنائية:

هذا الفصل يناقش كيفية ترتيب البيانات حركياً حتى تكون عمليات ادخال وحذف العناصر والبحث عنها أكثر كفاءة.

نقوم أولاً بتقديم بعض التعريفات لتسهيل مناقشتنا.

تعريف: الشجرة الثنائية T اما أنها فارغة أو أن:

(1) الشجرة بها عقدة تسمى **العقدة الجذرية**.

(2) الشجرة بها مجموعتان من العقد L1 و R1 تسميان الشجرة الفرعية اليسرى والشجرة الفرعية

اليمنى من الشجرة T على التوالي.

(3) L1 و R1 عبارة عن أشجار ثنائية.

يمكن توضيح الشجرة الثنائية تصويرياً. افترض أن T عبارة عن شجرة ثنائية ذات عقدة جذرية A.

لتكن L_A الشجرة الفرعية اليسرى من A وتعتبر R_A الفرعية اليمنى من A.

الآن و $L_A \subset R_A$. افترض أن B هي العقدة الجذرية للشجرة الفرعية اليسرى و أن C هي

العقدة الجذرية للشجرة الفرعية اليمنى. يطلق على B الثمرة اليسرى من A ويطلق على C الثمرة

اليمنى من A. فضلاً عن هذا يطلق على A أصل B و C.

في رسم الشجرة الثنائية يتم تمثيل كل عقدة من الشجرة الثنائية بدائرة ويتم تسمية الدائرة بالعقدة. يتم

رسم العقدة الجذرية للشجرة الثنائية على القمة ويتم رسم الثمرة اليسرى للعقدة الجذرية (إذا وجدت)

أسفل وعلى يسار العقدة الجذرية. بالمثل يتم رسم الثمرة اليمنى للعقدة الجذرية (إذا وجدت) أسفل وعلى

يمين العقدة الجذرية. تتصل الثمار بالأصل عن طريق سهم من الأصل الى الثمرة. عادةً يطلق على

السهم **حافة موجهة** أو **فرع موجه** (أو ببساطة فرع). بما أن العقدة الجذرية B من الشجرة الفرعية

اليسرى مرسومة بالفعل اذن نقوم بتطبيق نفس الاجراء لرسم الأجزاء المتبقية من الشجرة الفرعية

اليسرى ويتم رسم الشجرة الفرعية اليمنى بالمثل. اذا لم يكن للعقدة ثمرة يسرى فاننا على سبيل المثال

عندما نرسم سهم من العقدة الى اليسار ننتهي بالسهم عند ثلاثة سطور. هذا يعني أن وجود ثلاثة

سطور في نهاية السهم يشير الى أن الشجرة الفرعية فارغة.

الرسم في شكل 1-11 عبارة عن مثال للشجرة الثنائية. العقدة الجذرية لهذه الشجرة الثنائية هي A.

الشجرة الفرعية اليسرى من العقدة الجذرية التي نعبر عنها بواسطة L_A المجموعة

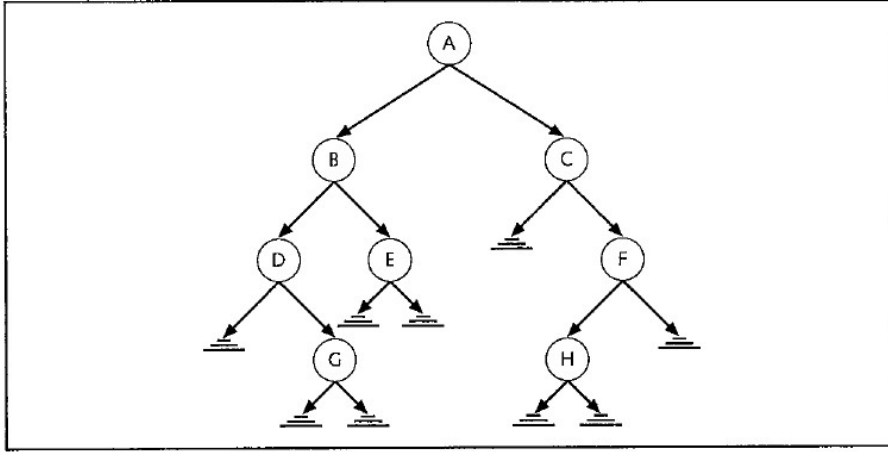
$L_A = \{B, D, E, G\}$ شجرة الفرعية اليمنى من العقدة الجذرية التي نعبر عنها بواسطة

$R_A = \{C, F, H\}$ العقدة الجذرية للشجرة الفرعية اليسرى من A

– أي العقدة الجذرية من هي العقدة الجذرية من الشجرة الفرعية اليمنى هي C وهكذا.

تضح أن كل من و أشجار L_A و R_A أن وجود ثلاثة سطور في نهاية السهم تعني أن الشجرة

الفرعية فارغة اذن ينتج أن الشجرة الفرعية اليسرى من D فارغة.

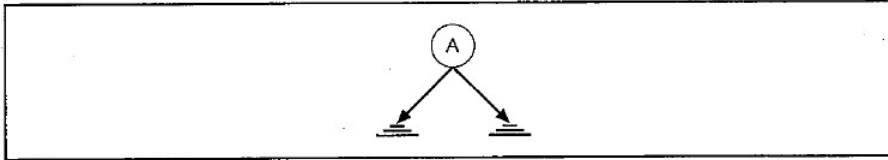


شكل 1-11: شجرة ثنائية

في شكل 1-11 الثمرة اليسرى من A هي B والثمرة اليمنى من A هي C. بالمثل بالنسبة الى العقدة F تكون الثمرة اليسرى هي H وليس لها ثمرة يمنى. الأمثلة من 1-11 الى 5-11 توضح أشجار ثنائية غير فارغة.

مثال 1-11:

هذا المثال يوضح شجرة ثنائية ذات عقدة واحدة. انظر شكل 2-11.



شكل 2-11: شجرة ثنائية ذات عقدة واحدة

في الشجرة الثنائية بالشكل 2-11:

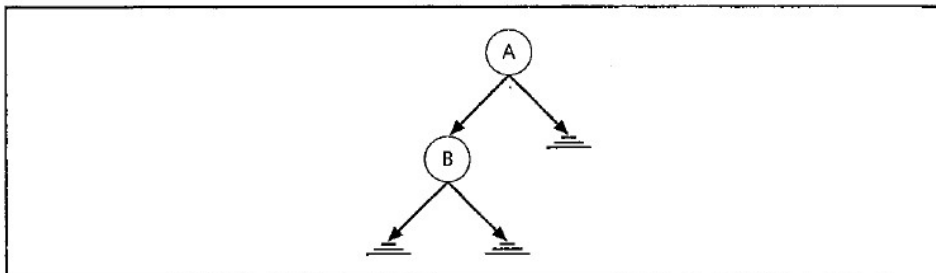
العقدة الجذرية للشجرة الثنائية = A

L_A = فارغة

R_A = فارغة

مثال 2-11:

هذا المثال يوضح شجرة ثنائية ذات عقدتين. انظر شكل 3-11.



شكل 3-11: شجرة ثنائية ذات عقدتين، الشجرة الفرعية اليمنى من العقدة الجذرية فارغة

في الشجرة الثنائية في شكل 3-11:

العقدة الجذرية من الشجرة الثنائية A

الشجرة الفرعية اليسرى $\{B\}$

الشجرة الفرعية اليمنى = فارغة

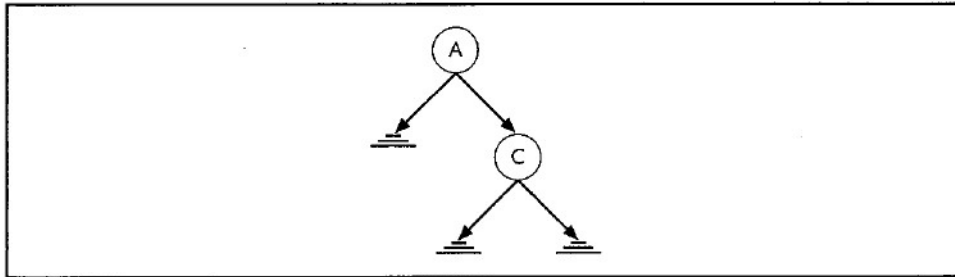
العقدة الجذرية للشجرة الفرعية اليسرى B

L_B :
غرة

R_B :
غرة

مثال 3-11:

هذا المثال يوضح شجرة ثنائية ذات عقدتين. انظر شكل 4-11.



شكل 4-11: شجرة ثنائية ذات عقدتين، الشجرة الفرعية اليسرى من العقدة الجذرية فارغة

في الشجرة الثنائية في شكل 4-11:

العقدة الجذرية من الشجرة الثنائية A

الشجرة الفرعية اليسرى = فارغة

الشجرة الفرعية اليمنى $\{C\}$

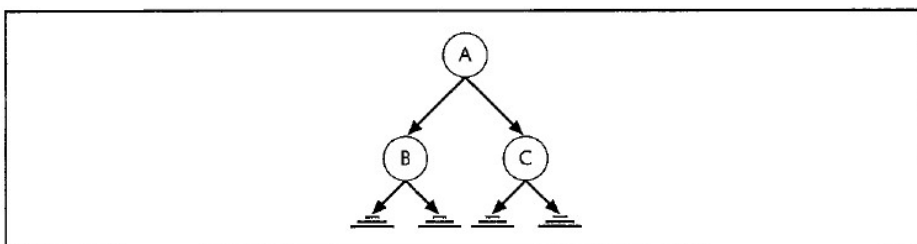
العقدة الجذرية للشجرة الفرعية اليمنى C

L_C :
غرة

R_C :
غرة

مثال 4-11:

هذا المثال يوضح شجرة ثنائية ذات ثلاثة عقد. انظر شكل 5-11.



شكل 11-5: شجرة ثنائية ذات ثلاثة عقد

في الشجرة الثنائية في شكل 11-5:

العقدة الجذرية من الشجرة الثنائية A

الشجرة الفرعية اليسرى $\{B\}$

الشجرة الفرعية اليمنى $\{C\}$

العقدة الجذرية للشجرة الفرعية اليسرى B

L_B : غة

R_B : غة

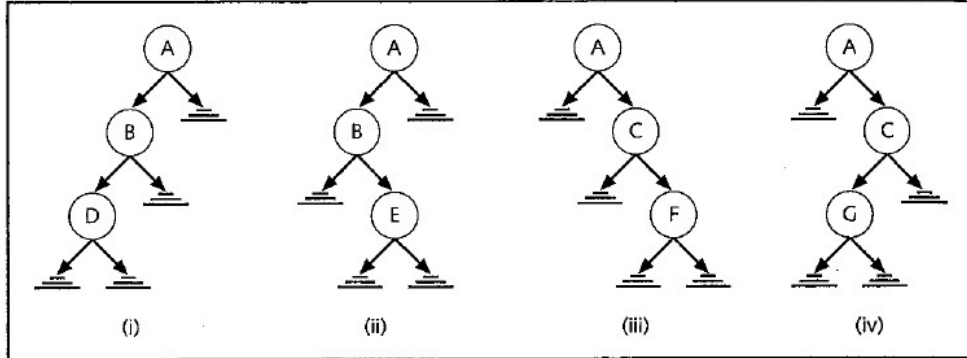
العقدة الجذرية للشجرة الفرعية اليمنى C

L_C : غة

R_C : غة

مثال 11-5:

هذا المثال يوضح حالات أخرى من شجرات فرعية غير فارغة وذات ثلاثة عقد. انظر شكل 11-6.



شكل 11-6: شجرات ثنائية متعددة ذات ثلاثة عقد

كما يمكنك أن ترى من الأمثلة السابقة تحتوي كل عقدة في الشجرة الثنائية على ثمرتين. لهذا يجب أن تقوم كل عقدة بخلاف تخزين المعلومات الخاصة بها بتتبع شجرتها الفرعية اليسرى وشجرتها الفرعية اليمنى. هذا يشير ضمناً إلى أن كل عقدة لها مؤشرين مثل $llink$ و $rlink$. المؤشر $llink$ يشير إلى العقدة الجذرية للشجرة الفرعية اليسرى ويشير المؤشر $rlink$ إلى العقدة الجذرية للشجرة الفرعية اليمنى.

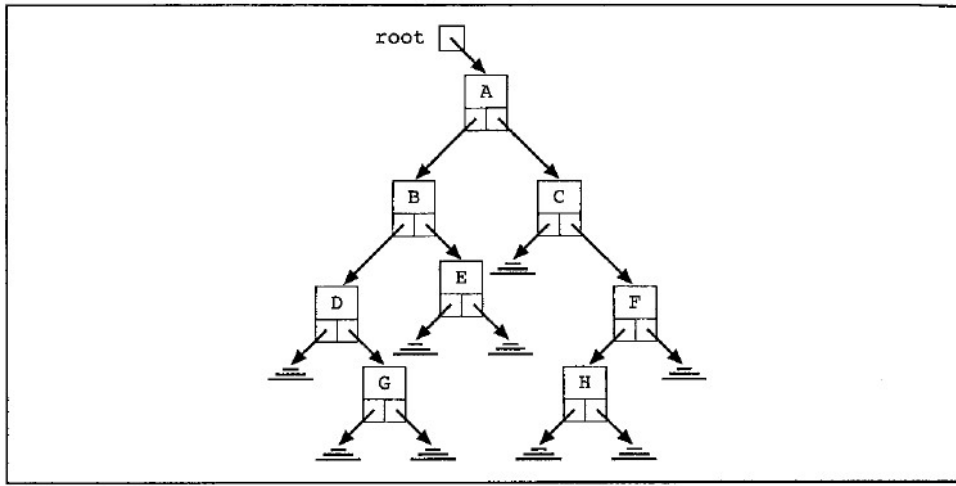
البنية التالية تقوم بتعريف عقدة شجرة ثنائية:

```
template<class elemType>
struct nodeType
{
    elemType info;
    nodeType<elemType> *llink;
    nodeType<elemType> *rlink;
};
```

من تعريف العقدة يتضح أن بالنسبة لكل عقدة:

- يتم تخزين البيانات في info.
- يتم تخزين المؤشر الى الثمرة اليسرى في llink.
- يتم تخزين المؤشر الى الثمرة اليمنى في rlink.

فضلاً عن هذا يتم تخزين المؤشر الى العقدة الجذرية للشجرة الثنائية خارج الشجرة الثنائية في متغير مؤشر يطلق عليه عادةً **الجذر** وهو من النوع nodeType. بهذا فان الشجرة الثنائية بوجه عام تبدو مثل الرسم الموجود في شكل 7-11.



للتبسيط سوف نستمر في رسم أشجار ثنائية كما سبق. هذا يعني أننا سوف نستخدم دوائر لكي تمثل العقد وأسهم لليمين واليسار لكي تمثل الوصلات. كما سبق تعبر ثلاثة سطور في نهاية السهم عن أن الشجرة الفرعية فارغة.

قبل أن نترك هذا القسم لنقم بتعريف القليل من المصطلحات الأخرى.

العقدة في الشجرة الثنائية تسمى **ورقة** اذا لم يكن لها ثمار يسرى ويمنى. لتكن U و V عقدتان في الشجرة الفرعية T. يطلق على U أصل V اذا كان هناك فرع من U الى V. المسار من العقدة x الى

العقدة y في شجرة فرعية عبارة عن تتابع من العقد

$$X_0, X_1, \dots, X_n$$

$$X = X_0, X_n = Y \quad \text{أ-}$$

ب- X_{i-1} أصل X_i لـ $i = 1, 2, \dots, n$. هذا يعني أن هناك فرع من X_0 الى

$$X_1, X_1 \text{ to } X_2, \dots, X_{i-1} \text{ to } X_i, \dots, X_{n-1} \text{ to } X_n.$$

بما أن الأفرع تذهب فقط من الأصل الى الثمار اذن يتضح من المناقشة السابقة أن هناك في الشجرة الثنائية مسار وحيد من الجذر الى كل عقدة في الشجرة الثنائية.

تعريف: مستوى العقدة في شجرة ثنائية هو عدد الأفرع على المسار من الجذر الى العقدة.

يتضح أن مستوى العقدة الجذرية لأي شجرة ثنائية يكون صفر ومستوى ثمار العقدة الجذرية هو 1.

تعريف: ارتفاع الشجرة الثنائية هو عدد العقد الموجودة على أطول مسار من الجذر الى ورقة.

افترض أن المؤشر p الى العقدة الجذرية للشجرة الفرعية معطى. نقوم بعد هذا نصف دالة C++ المسماة height لايجاد ارتفاع الشجرة الثنائية. يتم تمرير المؤشر الى العقدة الجذرية كعامل للدالة height.

إذا كانت الشجرة الثنائية فارغة يكون الارتفاع height هو صفر. افترض أن الشجرة الثنائية غير فارغة. لايجاد ارتفاع الشجرة الثنائية نقوم أولاً بايجاد ارتفاع الشجرة الفرعية اليسرى وارتفاع الشجرة الفرعية اليمنى. بعد هذا نأخذ الحد الأقصى من هذين الارتفاعين ونضيف 1 لايجاد ارتفاع الشجرة الثنائية. لايجاد ارتفاع الشجرة الفرعية اليسرى (اليمنى) نقوم بتطبيق نفس الاجراء لأن الشجرة الفرعية اليسرى (اليمنى) عبارة عن شجرة ثنائية. لهذا تكون الخوارزمية العامة لايجاد ارتفاع الشجرة الثنائية كما يلي (افترض أن الارتفاع p يعبر عن ارتفاع الشجرة الثنائية ذات الجذر p):

```
if(p is NULL)
    height(p) = 0
else
    height(p) = 1 + max(height(p->llink), height(p->rlink))
```

بوضوح تكون هذه خوارزمية تكرارية. الدالة التالية تقوم بتطبيق هذه الخوارزمية:

```
template<class elemType>
int height(nodeType<elemType> *p)
{
    if(p == NULL)
        return 0;
    else
        return 1 + max(height(p->llink), height(p->rlink));
}
```

تعريف الدالة height يستخدم الدالة max لتحديد الأكبر من بين عددين صحيحين. يمكن تطبيق الدالة max بسهولة.

بالمثل يمكننا تطبيق الخوارزميات لايجاد عدد العقد وعدد الأوراق في شجرة ثنائية.

نسخ شجرة:

من احدى العمليات المفيدة على الأشجار الثنائية عمل نسخة متطابقة من الشجرة الثنائية. الشجرة الثنائية عبارة عن بنية بيانات حركية أي أنه يتم تخصيص وإزالة تخصيص الذاكرة لعقداتها أثناء تنفيذ البرنامج. لهذا إذا استخدمنا فقط قيمة مؤشر العقدة الجذرية لعمل نسخة من شجرة جذرية فإننا نحصل على نسخة سطحية من البيانات. لعمل نسخة متطابقة من شجرة ثنائية نحتاج الى عمل أكبر عدد من العقد الموجودة في الشجرة الثنائية التي يتم نسخها. فضلاً عن هذا في الأشجار المنسوخة يجب أن تظهر هذه العقد بنفس الترتيب التي تكون عليه في الشجرة الفرعية الأصلية.

بالحصول على المؤشر الى العقدة الجذرية للشجرة الفرعية نقوم بعد هذا بوصف الدالة copyTree التي تقوم بعمل نسخة من شجرة ثنائية معطاة. هذه الدالة تكون مفيدة أيضاً في تطبيق مقوم النسخ واثقال معامل الاسناد كما تم وصفه لاحقاً في هذا الفصل (انظر قسم "تطبيق الأشجار الثنائية").

```
template<class elemType>
void copyTree(nodeType<elemType>* &copiedTreeRoot,
              nodeType<elemType>* otherTreeRoot)
{
    if(otherTreeRoot == NULL)
        copiedTreeRoot = NULL;
    else
    {
        copiedTreeRoot = new nodeType<elemType>;
        copiedTreeRoot->info = otherTreeRoot->info;
        copyTree(copiedTreeRoot->llink, otherTreeRoot->llink);
        copyTree(copiedTreeRoot->rlink, otherTreeRoot->rlink);
    }
} //end copyTree
```

اننا نستخدم الدالة copyTree عندما نقوم باثقال معامل الاسناد وتطبيق مقوم النسخ.

اجتياز الشجرة الثنائية:

عمليات ادخال العنصر وحذفه والبحث عنه تتطلب اجتياز الشجرة الثنائية. لهذا تكون العملية المؤداة بشكل أكثر شيوعاً على الشجرة الثنائية هي اجتياز الشجرة الثنائية أو زيارة كل عقدة من الشجرة الثنائية. كما ترى من رسم الشجرة الثنائية (على سبيل المثال شكل 7-11) يجب أن يبدأ الاجتياز من العقدة الجذرية لأن هناك مؤشر الى العقدة الجذرية. لدينا اختياران بالنسبة الى كل عقدة:

- زيارة العقدة أولاً .

- زيارة الشجرة الثنائية أولاً .

هذه الاختيارات تؤدي الى ثلاث اجتيازات مختلفة للشجرة الثنائية كما يتم وصفها في الأقسام الثلاثة التالية:

- اجتياز في الترتيب.

- اجتياز قبل الترتيب.

- اجتياز بعد الترتيب.

الاجتياز في الترتيب:

في الاجتياز أثناء الترتيب يتم اجتياز الشجرة الثنائية كما يلي:

- اجتياز الشجرة الفرعية اليسرى.

- زيارة العقدة.

- اجتياز الشجرة الفرعية اليمنى.

الاجتياز قبل الترتيب:

في الاجتياز قبل الترتيب يتم اجتياز الشجرة الثنائية كما يلي:

1. زيارة العقدة.
2. اجتياز الشجرة الفرعية اليسرى.
3. اجتياز الشجرة الفرعية اليمنى.

الاجتياز بعد الترتيب:

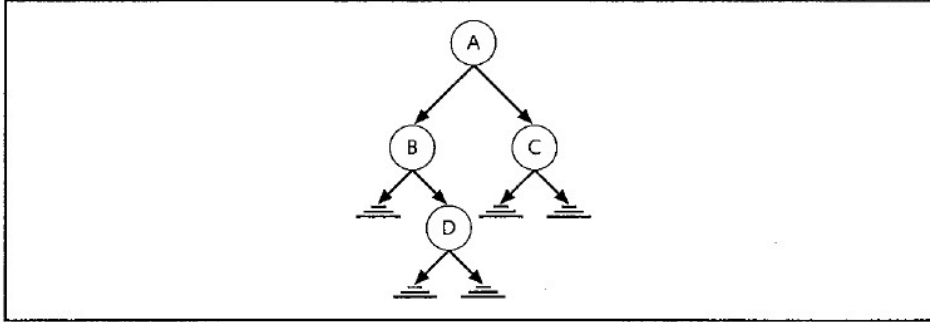
في الاجتياز بعد الترتيب يتم اجتياز الشجرة الثنائية كما يلي:

1. اجتياز الشجرة الفرعية اليسرى.
2. اجتياز الشجرة الفرعية اليمنى.
3. زيارة العقدة.

يتضح أن كل من خوارزميات الاجتياز هذه تكرارية.

تسجيل العقد الناتجة من اجتياز الشجرة الثنائية أثناء الترتيب يسمى **التتابع في الترتيب**. تسجيل العقد الناتجة من اجتياز الشجرة الثنائية قبل الترتيب يسمى **التتابع قبل الترتيب**. تسجيل العقد الناتجة من اجتياز الشجرة الثنائية بعد الترتيب يسمى **التتابع بعد الترتيب**.

قبل اعطاء كود C++ لكل من هذه الاجتيازات لنقم بتوضيح اجتياز الشجرة الثنائية أثناء الترتيب في شكل 8-11. للتبسيط نفترض أن زيارة العقدة تعني اخراج البيانات المخزنة في العقدة. قسم "اجتياز الشجرة الثنائية والدالات كمعاملات" الموجود لاحقاً في هذا الفصل توضح كيفية تعديل خوارزميات اجتياز الشجرة الثنائية حتى يمكن للمستخدم باستخدام الدالة أن يحدد الاجراء الذي يتم القيام به على العقدة عندما تتم زيارة العقدة.



شكل 8-11: الشجرة الثنائية للاجتياز أثناء الترتيب

يتم تخزين مؤشر الى الشجرة الثنائية في شكل 8-11 في متغير المؤشر root (الذي يشير الى العقدة ذات البيان A). لهذا نبدأ الاجتياز عند A.

1. اجتياز القائمة الفرعية اليسرى من A أي أن اجتياز $L_A = \{B, D\}$

2. زيارة A.

3. اجتياز القائمة الفرعية اليمنى من A أي أن اجتياز $R_A = \{C\}$

الآن لا يمكننا القيام بالخطوة 2 حتى ننتهي من الخطوة 1.

1. اجتياز القائمة الفرعية اليسرى من A أي أن اجتياز $L_A = \{B, D\}$ عبارة عن شجرة ثنائية ذات العقدة الجذرية B. بما أن L_A شجرة ثنائية إذن نطبق معيار الاجتياز أثناء الترتيب على L_A .

1-1 اجتياز الشجرة الفرعية اليسرى من B أي أن اجتياز $L_B = \text{فارغ}$

2-1 زيارة B.

3-1 اجتياز الشجرة الفرعية اليمنى من B أي أن اجتياز $R_B = \{D\}$

كما سبق نقوم أولاً باكمال الخطوة 1-1 قبل الانتقال الى الخطوة 2-1.

1-1 بما أن الشجرة الفرعية اليسرى من B فارغة فإنه لا يوجد شئ يحتاج الى الاجتياز.

تم الانتهاء من الخطوة 1-1 ولهذا نتقدم الى الخطوة 2-1.

2-1 زيارة B. هذا يعني اخراج B على جهاز اخراج. العقدة الأولى التي يتم طبعها هي

B. هذا يكمل الخطوة 2-1 ولهذا نتقدم الى الخطوة 3-1.

3-1 اجتياز القائمة الفرعية اليمنى من B أي أن اجتياز $R_B = \{D\}$ عن R_B

شجرة ثنائية ذات عقدة جذرية D. بما أن R_B ثنائية نقوم بتطبيق معيار الاجتياز أثناء الترتيب عند R_B .

1-3-1 اجتياز الشجرة الفرعية اليسرى من D أي أن اجتياز $L_D = \text{غ.}$

2-3-1 زيارة D.

3-3-1 اجتياز الشجرة الفرعية اليمنى من D أي أن $R_D = \text{غ.}$

1-3-1 بما أن الشجرة الفرعية اليسرى من D فارغة إذن لا يوجد شئ يحتاج الى

الاجتياز. الخطوة 1-3-1 اكتملت ولهذا نتقدم الى الخطوة 2-3-1

2-3-1 زيارة D أي اخراج D على جهاز اخراج. هذا يكمل الخطوة 2-3-1 ولهذا نتقدم الى

الخطوة 3-3-1.

3-3-1 بما أن الشجرة الفرعية اليمنى من D فارغة إذن لا يوجد شئ يحتاج الى الاجتياز.

الخطوة 3-3-1 اكتملت.

هذا يكمل الخطوة 3-1 وبما أن الخطوات 1-1 و 2-1 و 3-1 اكتملت وبهذا اكتملت الخطوة 1 ولهذا

ننتقل الى الخطوة 2.

2. زيار A أي اخراج A على جهاز اخراج. هذا يكمل الخطوة 2 ولهذا نتقدم الى الخطوة 3.

3. اجتياز الشجرة الفرعية اليمنى من A أي أن اجتياز $R_A = \{C\}$. الآن R_A شجرة ثنائية ذات

عقدة جذرية C. بما أن R_A شجرة ثنائية نقوم بتطبيق معيار الاجتياز أثناء الترتيب على

1-3 اجتياز الشجرة الفرعية اليسرى من C أي أن اجتياز $L_C = \text{فارغ}$.
2-3 زيارة C.

3-3 اجتياز الشجرة الفرعية اليمنى من C أي أن اجتياز $R_C = \text{فارغ}$

1-3 بما أن الشجرة الفرعية اليسرى من C فارغة إذن لا يوجد شيء يحتاج إلى الاجتياز. الخطوة
1-3 اكتملت ولهذا نتقدم إلى الخطوة 2-3
2-3 زيارة C أي اخراج C على جهاز اخراج. هذا يكمل الخطوة 2-3 ولهذا نتقدم إلى الخطوة 3-3.
3.

3-3 بما أن الشجرة الفرعية اليمنى من C فارغة إذن لا يوجد شيء يحتاج إلى الاجتياز. الخطوة 3-3.
3 اكتملت.

هذا يكمل الخطوة 3 وهذا بدوره يكمل اجتياز الشجرة الثنائية.

من الواضح أن الاجتياز أثنائي الترتيب للشجرة الثنائية السابقة يخرج العقدة بالترتيب التالي.

النتابع في الترتيب: B D A C

بالمثل نقوم الاختيارات قبل وبعد الترتيب باخراج العقد بالترتيب التالي:

النتابع قبل الترتيب: A B C D

النتابع بعد الترتيب: D B C A

كما يمكنك أن ترى من اجراء الاجتياز أثناء الترتيب فاننا بعد زيارة الشجرة الفرعية اليسرى من العقدة يجب أن نعود إلى العقدة ذاتها. الوصلات في اتجاه واحد فقط أي أن العقدة الأصلية تشير إلى الثمار اليسرى واليمنى ولكن لا يوجد مؤشر من كل ثمرة إلى الأصل. لهذا قبل الذهاب إلى ثمرة يجب أن نقوم نوعاً ما بحفظ مؤشر إلى العقدة الأصلية. من الطرق المناسبة للقيام بهذا كتابة دالة تكرارية وهذا لأن بعد الانتهاء من استدعاء معين في الاستدعاء التكراري يعود التحكم إلى الداعي. (لاحقاً نناقش كيفية كتابة دالات اجتياز غير تكرارية).

التعريف التكراري للدالة لتطبيق خوارزمية الاجتياز في الترتيب يكون:

```
template<class elemType>
void inorder(nodeType<elemType> *p)
{
    if(p != NULL)
    {
        inorder(p->llink);
        cout<<p->info<<" ";
        inorder(p->rlink);
    }
}
```

للقيام بالاجتياز أثناء الترتيب على شجرة ثنائية يتم تمرير العقدة الجذرية للشجرة الثنائية كعامل للدالة inorder. على سبيل المثال اذا كان الجذر يشير الى العقدة الجذرية للشجرة الثنائية يكون استدعاء الدالة inorder هو:

inorder (root) ;

بالمثل يمكننا كتابة الدالات لتطبيق الاجتيازات قبل وبعد الترتيب. تعريفات هذه الدالات معطاة فيما بعد.

```
template<class elemType>
void preorder(nodeType<elemType> *p)
{
    if(p != NULL)
    {
        cout<<p->info<<" ";
        preorder(p->llink);
        preorder(p->rlink);
    }
}

template<class elemType>
void postorder(nodeType<elemType> *p)
{
    if(p != NULL)
    {
        postorder(p->llink);
        postorder(p->rlink);
        cout<<p->info<<" ";
    }
}
```

تطبيق أشجار ثنائية:

قامت الأقسام السابقة بوصف عمليات متنوعة يمكن أدائها على الشجرة الثنائية وكذلك دالات تطبيق هذه العمليات. يقوم هذا القسم بوصف الأشجار الثنائية كنوع بيانات مجرد. قبل تصميم الفئة لتطبيق شجرة ثنائية كنوع بيانات مجرد لنقم بتحديد عمليات متنوعة يتم أدائها عادةً على الشجرة الثنائية.

1. تحديد ما اذا كانت الشجرة الثنائية فارغة.

2. البحث في الشجرة الثنائية عن عنصر محدد.

3. ادخال عنصر في الشجرة الثنائية.

4. حذف عنصر من الشجرة الثنائية.

5. ايجاد ارتفاع الشجرة الثنائية.

6. ايجاد عدد العقدات في الشجرة الثنائية.

7. ايجاد عدد الأوراق في الشجرة الثنائية.

8. اجتياز الشجرة الثنائية.

9. نسخ الشجرة الثنائية.

جميع عمليات ادخال واخراج والبحث عن عنصر تحتاج الى اجتياز الشجرة. بالرغم من هذا وبما أن عقد الشجرة الثنائية لا تكون في ترتيب معين فان هذه الخوارزميات ليست شديدة الكفاءة على الأشجار الثنائية الكيفية. هذا يعني أنه لا يوجد معيار لتوجيه البحث على هذه الأشجار الثنائية كما سوف نرى في القسم التالي. لهذا سوف نناقش تلك الخوارزميات عندما نناقش أنواع خاصة من الأشجار الثنائية.

بخلاف عمليات البحث والادخال والحذف تقوم الفئة التالية بتعريف الأشجار الثنائية كنوع بيانات مجرد. تعريف العقدة هو نفس التعريف السابق. بالرغم من هذا ومن أجل الكمال والمرجعية السهلة نقوم باعطاء تعريف العقدة وبعده تعريف الفئة:

```
//Definition of the node
template<class elemType>
struct nodeType
{
    elemType info;
    nodeType<elemType> *llink;
    nodeType<elemType> *rlink;
};

//Definition of the class
template<class elemType>
class binaryTreeType
{
public:
    const binaryTreeType<elemType>& operator=
        (const binaryTreeType<elemType>&);
    //Overload the assignment operator.
    bool isEmpty();
    //Function to determine whether the binary tree is empty.
    //Postcondition: Returns true if the binary tree is empty;
    //                otherwise, returns false.
    void inorderTraversal();
    //Function to do an inorder traversal of the binary tree.
    //Postcondition: The nodes of the binary tree are output
    //                in the inorder sequence.
    void preorderTraversal();
    //Function to do a preorder traversal of the binary tree.
    //Postcondition: The nodes of the binary tree are output
    //                in the preorder sequence.
    void postorderTraversal();
    //Function to do a postorder traversal of the binary tree.
    //Postcondition: The nodes of the binary tree are output
    //                in the postorder sequence.
```

```

int treeHeight();
    //Function to determine the height of the binary tree.
    //Postcondition: The height of the binary tree is returned.
int treeNodeCount();
    //Function to determine the number of nodes in the
    //binary tree.
    //Postcondition: The number of nodes in the binary tree
    //                is returned.
int treeLeavesCount();
    //Function to determine the number of leaves in the
    //binary tree.
    //Postcondition: The number of leaves in the binary tree
    //                is returned.
void destroyTree();
    //Deallocates the memory space occupied by the binary tree.
    //Postcondition: root = NULL

binaryTreeType(const binaryTreeType<elemType>& otherTree);
    //copy constructor

binaryTreeType();
    //default constructor

~binaryTreeType();
    //destructor

protected:
    nodeType<elemType> *root;

private:
    void copyTree(nodeType<elemType>* &copiedTreeRoot,
                  nodeType<elemType>* otherTreeRoot);
        //Function to make a copy of the binary tree to
        //which otherTreeRoot points.
        //Postcondition: The pointer copiedTreeRoot points to
        //                the root of the copied binary tree.

    void destroy(nodeType<elemType>* &p);
        //Function to destroy the binary tree to which p points.
        //Postcondition: The nodes of the binary tree to which
        //                p points are deallocated; p = NULL.

    void inorder(nodeType<elemType> *p);
        //Function to do an inorder traversal of the binary
        //tree to which p points.
        //Postcondition: The nodes of the binary tree to which p
        //                points are output in the inorder sequence.

```

```

void preorder(nodeType<elemType> *p);
//Function to do a preorder traversal of the binary
//tree to which p points.
//Postcondition: The nodes of the binary tree to which p
//                points are output in the preorder sequence.
void postorder(nodeType<elemType> *p);
//Function to do a postorder traversal of the binary
//tree to which p points.
//Postcondition: The nodes of the binary tree to which p
//                points are output in the postorder sequence

int height(nodeType<elemType> *p);
//Function to determine the height of the binary tree
//to which p points.
//Postcondition: The height of the binary tree to which p
//                points is returned.

int max(int x, int y);
//Function to determine the larger of x and y.
//Postcondition: The larger of x and y is returned.

int nodeCount(nodeType<elemType> *p);
//Function to determine the number of nodes in the binary
//tree to which p points.
//Postcondition: The number of nodes in the binary tree
//                to which p points is returned.

int leavesCount(nodeType<elemType> *p);
//Function to determine the number of leaves in the binary
//tree to which p points.
//Postcondition: The number of leaves in the binary tree
//                to which p points is returned.

};

```

لاحظ أن تعريف الفئة `binaryTreeType` يحتوي على البيان لاثقال معامل الاسناد ومقوم النسخ والمدمر. هذا لأن الفئة `binaryTreeType` تحتوي على عناصر بيانات مؤشر. تذكر أنه بالنسبة الى الفئات ذات عناصر بيانات المؤشر هناك ثلاثة أشياء يجب علينا القيام بها وهي اثقال معامل الاسناد، وتضمين مقوم النسخ، وتضمين المدمر.

تعريف الفئة `binaryTreeType` يحتوي على العديد من دالات العنصر التي تكون عبارة عن عناصر خاصة من الفئة. يتم استخدام هذه الفئات لتطبيق عناصر البيانات العامة للفئة ولا يحتاج المستخدم الى معرفة وجودها. على سبيل المثال وللقيام بعمل اجتياز أثناء الترتيب تقوم الدالة `inorderTraversal` باستدعاء الدالة `inorder` وتمرير المؤشر `root` كمعامل لهذه الدالة. افترض أن لديك البيان التالي:

```
binaryTreeType<int> myTree;
```

البيان التالي يقوم بعمل اجتياز في الترتيب لـ myTree:

```
myTree.inorderTraversal();
```

لاحظ كذلك أن في تعريف الفئة binaryTreeType تم اعلان المؤشر root كعنصر محمي حتى يمكننا لاحقاً استمداد أشجار ثنائية خاصة.

بعد هذا نعطي تعريفات دالات عنصر الفئة binaryTreeType.

تكون الشجرة الثنائية فارغة اذا كانت root تساوي NULL ولهذا فان تعريف الدالة isEmpty يكون:

```
template<class elemType>
bool binaryTreeType<elemType>::isEmpty()
{
    return (root == NULL);
}
```

يقوم المقوم الافتراضي بتهيئة الشجرة الثنائية في وضع فارغ أي أنه يحدد المؤشر root عند NULL. لهذا فان تعريف المقوم الافتراضي هو:

```
template<class elemType>
binaryTreeType<elemType>::binaryTreeType()
{
    root = NULL;
}
```

تعريفات الدالات الأخرى تكون:

```
template<class elemType>
void binaryTreeType<elemType>::inorderTraversal()
{
    inorder(root);
}

template<class elemType>
void binaryTreeType<elemType>::preorderTraversal()
{
    preorder(root);
}

template<class elemType>
void binaryTreeType<elemType>::postorderTraversal()
{
    postorder(root);
}

template<class elemType>
int binaryTreeType<elemType>::treeHeight()
{
    return height(root);
}
```

```

template<class elemType>
int binaryTreeType<elemType>::treeNodeCount()
{
    return nodeCount(root);
}

template<class elemType>
int binaryTreeType<elemType>::treeLeavesCount()
{
    return leavesCount(root);
}

template<class elemType>
void binaryTreeType<elemType>::inorder(nodeType<elemType> *p)
{
    if(p != NULL)
    {
        inorder(p->llink);
        cout<<p->info<<" ";
        inorder(p->rlink);
    }
}

template<class elemType>
void binaryTreeType<elemType>::preorder(nodeType<elemType> *p)
{
    if(p != NULL)
    {
        cout<<p->info<<" ";
        preorder(p->llink);
        preorder(p->rlink);
    }
}

template<class elemType>
void binaryTreeType<elemType>::postorder(nodeType<elemType> *p)
{
    if(p != NULL)
    {
        postorder(p->llink);
        postorder(p->rlink);
        cout<<p->info<<" ";
    }
}

template<class elemType>
int binaryTreeType<elemType>::height(nodeType<elemType> *p)
{
    if(p == NULL)
        return 0;
    else
        return 1 + max(height(p->llink), height(p->rlink));
}

template<class elemType>
int binaryTreeType<elemType>::max(int x, int y)
{
    if(x >= y)
        return x;
    else
        return y;
}

```

يتم ترك تعريفات الدالات nodeCount و leavesCount كتمرين لك. (انظر تمرين البرمجة رقم 1 و2 في نهاية هذا الفصل).

بعد هذا نعطي تعريفات الدالات copyTree و destroy و destroyTree ومقوم النسخ والمدمر وكذلك ائقال معامل الاسناد.

تعريف الدالة copyTree هو نفسه كما سبق وهنا تكون هذه الدالة عنصر من الفئة .binaryTreeType

```
template<class elemType>
void binaryTreeType<elemType>::copyTree
    (nodeType<elemType>* &copiedTreeRoot,
     nodeType<elemType>* otherTreeRoot)
{
    if(otherTreeRoot == NULL)
        copiedTreeRoot = NULL;
    else
    {
        copiedTreeRoot = new nodeType<elemType>;
        copiedTreeRoot->info = otherTreeRoot->info;
        copyTree(copiedTreeRoot->llink, otherTreeRoot->llink);
        copyTree(copiedTreeRoot->rlink, otherTreeRoot->rlink);
    }
}
//end copyTree
```

لتدمير شجرة ثنائية نقوم أولاً بتدمير الشجرة الفرعية اليسرى لكل عقدة ثم شجرتها الفرعية اليمنى ثم العقدة ذاتها. يجب أن نقوم باستخدام المعامل delete لازالة تخصيص الذاكرة التي تحتلها كل عقدة. تعريف الدالة destroy هو:

```
template<class elemType>
void binaryTreeType<elemType>::destroy(nodeType<elemType>* &p)
{
    if(p != NULL)
    {
        destroy(p->llink);
        destroy(p->rlink);
        delete p;
        p = NULL;
    }
}
```

لتطبيق الدالة destroyTree نستخدم الدالة destroy ونقوم بتمرير المؤشر root للعقدة الجذرية من الشجرة الثنائية الى الدالة destroy. تعريف الدالة destroyTree هو:

```
template<class elemType>
void binaryTreeType<elemType>::destroyTree()
{
    destroy(root);
}
```

تذكر أنه عندما يتم تمرير هدف فئة بالقيمة يقوم مقوم النسخ بنسخ قيم العوامل الأساسية الى داخل المعاملات الرسمية. بما أن الفئة `binaryTreeType` لها عناصر بيانات مؤشر تقوم بعمل ذاكرة حركية اذن يجب أن نقدم تعريف مقوم النسخ لتجنب النسخ السطحي للبيانات. تعريف مقوم النسخ المعطى فيما بعد يستخدم الدالة `copyTree` لعمل نسخة متطابقة من الشجرة الثنائية التي يتم تمريرها كمعامل.

```
//copy constructor
template<class elemType>
binaryTreeType<elemType>::binaryTreeType
    (const binaryTreeType<elemType>& otherTree)
{
    if(otherTree.root == NULL) //otherTree is empty
        root = NULL;
    else
        copyTree(root, otherTree.root);
}
```

تعريف المدمر معطى فيما بعد. عندما يخرج هدف من النوع `binaryTreeType` من المجال يقوم المدمر بإزالة تخصيص الذاكرة التي تحتلها عقد الشجرة الثنائية. تعريف المدمر يستخدم الدالة `destroy` لانجاز هذه المهمة.

```
//destructor
template<class elemType>
binaryTreeType<elemType>::~~binaryTreeType()
{
    destroy(root);
}
```

بعد هذا نناقش الدالة لانتقال معامل الاسناد. لاسناد قيمة شجرة ثنائية واحدة لشجرة ثنائية أخرى نقوم بعمل نسخة متطابقة من الشجرة الثنائية حتى يتم اسنادها باستخدام الدالة `copyTree`. تعريف الدالة لانتقال معامل الاسناد هو:

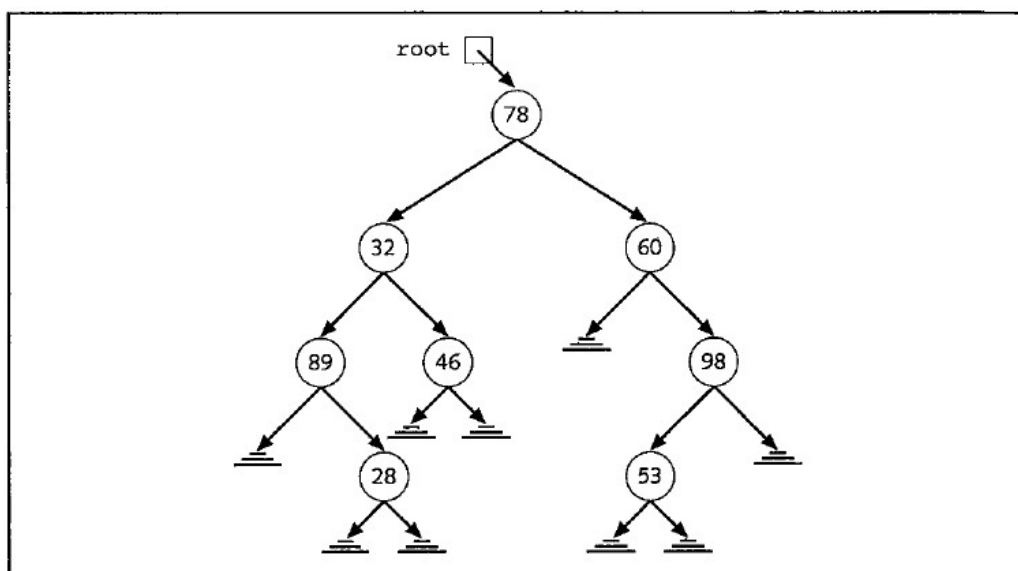
```
//Overload the assignment operator.
template<class elemType>
const binaryTreeType<elemType>& binaryTreeType<elemType>::operator=
    (const binaryTreeType<elemType>& otherTree)
{
    if(this != &otherTree) //avoid self-copy
    {
        if(root != NULL) //if the binary tree is not empty,
            //destroy the binary tree
            destroy(root);

        if(otherTree.root == NULL) //otherTree is empty
            root = NULL;
        else
            copyTree(root, otherTree.root);
    } //end else

    return *this;
}
```

أشجار البحث الثنائية:

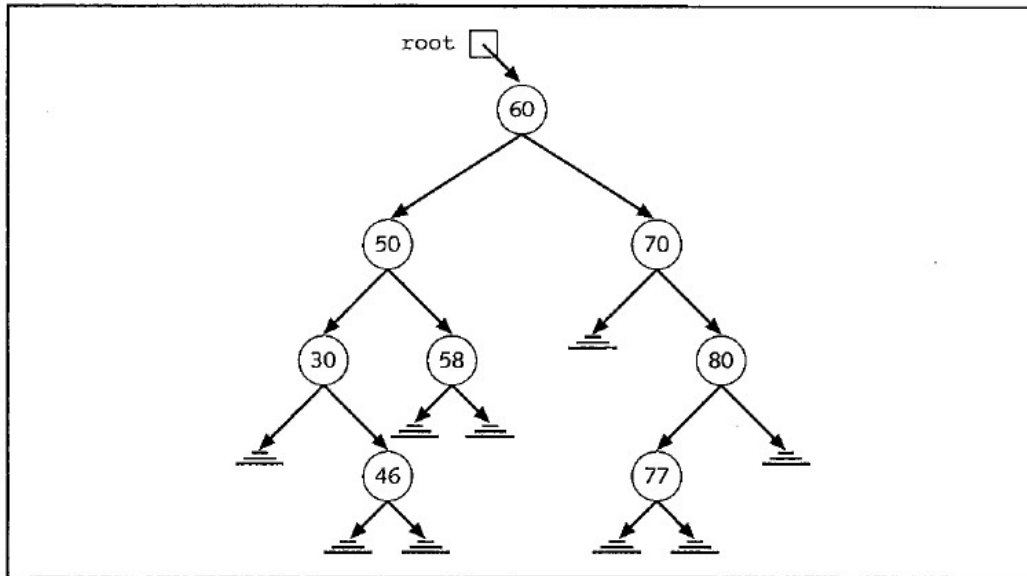
الآن بعدما علمت العمليات الأساسية على الشجرة الثنائية يقوم هذا القسم بمناقشة نوع خاص من الأشجار الثنائية يسمى شجرة البحث الثنائية. انظر الى الشجرة الثنائية في شكل 9-11.



شكل 9-11: شجرة ثنائية وهمية

افترض أننا نريد تحديد ما اذا كان العدد 50 موجود في الشجرة الثنائية. للقيام بهذا يمكننا استخدام أيًا من خوارزميات الاجتياز السابقة لزيارة كل عقدة ومقارنة عنصر البحث بالبيانات المخزنة في العقدة. بالرغم من هذا فقد يقتضي هذا منا اجتياز جزء كبير من الشجرة الثنائية ولهذا سوف يكون البحث بطيئاً. اننا نحتاج الى زيارة كل عقدة في الشجرة الثنائية حتى يتم العثور على العنصر أو حتى نقوم باجتياز الشجرة الثنائية بأكملها لأنه لا يوجد معيار يرشد بحثنا. هذه الحالة مثل قائمة متصلة وهمية حيث يجب أن نبدأ بحثنا عند العقدة الأولى ونستمر في النظر الى كل عقدة حتى يتم العثور على العنصر أو حتى يتم بحث كامل القائمة.

على الجهة الأخرى انظر الى الشجرة الثنائية في شكل 10-11.



شكل 10-11: شجرة بحث ثنائية

في الشجرة الثنائية في شكل 10-11 تكون البيانات في كل عقدة:

- أكبر من البيانات في ثمرتها اليسرى.
- أصغر من البيانات في ثمرتها اليمنى.

الشجرة الثنائية في شكل 10-11 لها بعض البنية. افترض أننا نريد تحديد ما إذا كان العدد 58 موجود في هذه الشجرة الثنائية. كما سبق يجب أن نبدأ بحثنا عند العقدة الجذرية. اننا نقارن 58 مع البيانات الموجودة في العقدة الجذرية أي أننا نقارن 58 مع 60. بما أن $58 \neq 60$ وأن $58 > 60$ إذن من المضمون أن 58 ليس موجود في الشجرة الفرعية اليمنى من العقدة الجذرية. لهذا إذا كانت 58 موجودة في الشجرة الثنائية إذن يجب أن تكون في الشجرة الفرعية اليسرى من العقدة الجذرية. اننا نتبع المؤشر الأيسر للعقدة الجذرية ونذهب الى العقدة ذات البيان 50. الآن نقوم بتطبيق نفس المعيار على هذه العقدة. بما أن $58 > 50$ إذن يجب أن نتبع المؤشر الأيسر لهذه العقدة ونذهب الى العقدة ذات البيان 58. عند هذه العقدة نجد العنصر 58.

هذا المثال يوضح أن كل مرة ننتقل فيها الى أسفل الى ثمرة ما نزيل واحدة من الأشجار الفرعية للعقدة من بحثنا. إذا تم بناء الشجرة الثنائية جيداً إذن يكون البحث مشابه للغاية بالبحث الثنائي على المصفوفات.

الشجرة الثنائية المعطاة في شكل 10-11 عبارة عن نوع خاص من الشجرة الثنائية يسمى شجرة البحث الثنائية (في التعريف التالي نقصد بمصطلح مفتاح العقدة مفتاح عنصر البيانات الذي يحدد العنصر بتفرد).

تعريف: شجرة بحث ثنائية T اما تكون فارغة أو:

1. T بها عقدة خاصة تسمى العقدة الجذرية.
 2. T بها مجموعتان من العقد وهما L_T و R_T ميان الشجرة الفرعية اليسرى والشجرة الفرعية اليمنى من T على التوالي.
 3. المفتاح في العقدة الجذرية أكبر من كل مفتاح في الشجرة الفرعية اليسرى وأصغر من كل مفتاح في الشجرة الفرعية اليمنى.
 4. L_T و R_T شجري بحث ثنائية.
- في هذا التعريف بما أن الشجرتين الفرعيتين اليسرى واليمنى من العقدة الجذرية عبارة عن أشجار بحث ثنائية (iii) متساوي مع (iii). المفتاح في العقدة الجذرية أكبر من المفتاح في ثمرتها اليسرى (إذا وجدت) وأصغر من المفتاح في ثمرتها اليمنى (إذا وجدت).

العمليات التالية من المعتاد أدائها على شجرة بحث ثنائية:

1. تحديد ما إذا كانت شجرة البحث الثنائية فارغة أم لا.
 2. البحث في شجرة البحث الثنائية عن عنصر معين.
 3. ادخال عنصر في شجرة البحث الثنائية.
 4. حذف عنصر من شجرة البحث الثنائية.
 5. ايجاد ارتفاع شجرة البحث الثنائية.
 6. ايجاد عدد العقدات في شجرة البحث الثنائية.
 7. ايجاد عدد الأوراق في شجرة البحث الثنائية.
 8. اجتياز شجرة البحث الثنائية.
 9. نسخ شجرة البحث الثنائية.
- من الواضح أن كل شجرة بحث ثنائية تكون عبارة عن شجرة ثنائية. ارتفاع شجرة البحث الثنائية يتم تحديده بنفس طريقة تحديد ارتفاع الشجرة الثنائية. بالمثل العمليات المؤداة لاجاد عدد العقد و ايجاد عدد الأوراق ولعمل الاجتازات أثناء وقبل وبعد الترتيب على أشجار البحث الثنائية هي نفس العمليات المؤداة على الأشجار الثنائية. لهذا يمكننا توريث جميع تلك العمليات من الشجرة الثنائية. هذا يعني أنه يمكننا مد تعريف الشجرة الثنائية باستخدام مبدأ التوريث ومن هنا تعريف شجرة البحث الثنائية.

الفئة التالية تقوم بتعريف شجرة بحث ثنائية كنوع بيانات مجرد عن طريق مد تعريف الشجرة الثنائية:

```
template<class elemType>
class bSearchTreeType: public binaryTreeType<elemType>
{
public:
    bool search(const elemType& searchItem);
    //Function to determine whether searchItem is in the binary
    //search tree.
```

```

//Postcondition: Returns true if searchItem is found in the
//                binary search tree; otherwise, returns false.

void insert(const elemType& insertItem);
//Function to insert insertItem in the binary search tree.
//Postcondition: If no node in the binary search tree has
//                the same info as insertItem, a node with
//                the info insertItem is created and inserted
//                in the binary search tree.

void deleteNode(const elemType& deleteItem);
//Function to delete deleteItem from the binary search tree.
//Postcondition: If a node with the same info as deleteItem
//                is found, it is deleted from the binary
//                search tree.

private:
void deleteFromTree(nodeType<elemType>* &p);
//Function to delete the node, to which p points, from the
//binary search tree.
//Postcondition: The node to which p points is deleted
//                from the binary search tree.
};

```

بعد هذا نصف كل من هذه العمليات.

البحث:

الدالة search تقوم ببحث شجرة البحث الثنائية من أجل عنصر محدد. إذا تم العثور على العنصر في شجرة البحث الثنائية تنتج الدالة true وبخلاف هذا تنتج false. بما أن المؤشر root يشير إلى العقدة الجذرية لشجرة البحث الثنائية إذن يجب أن نبدأ بحثنا عند العقدة الجذرية. فضلاً عن هذا وبما أن root يجب أن يشير دائماً إلى العقدة الجذرية إذن فنحن نحتاج إلى مؤشر مثل current لاجتياز شجرة البحث الثنائية. تتم تهيئة المؤشر current عند root.

إذا لم تكن شجرة البحث الثنائية فارغة نقوم أولاً بمقارنة عنصر البحث مع البيان الموجود في العقدة الجذرية. إذا كانوا متماثلين نوقف البحث وننتج true. بخلاف هذا أي إذا كان عنصر البحث أصغر من البيان الموجود في العقدة فإننا نتبع llink (الوصلة اليسرى) للذهاب إلى الشجرة الفرعية اليسرى وبخلاف هذا نتبع rlink (الوصلة اليمنى) للذهاب إلى الشجرة الفرعية اليمنى. نكرر هذه العملية بالنسبة إلى العقدة التالية. إذا كان عنصر البحث موجود في شجرة البحث الثنائية ينتهي بحثنا عند العقدة المحتوية على عنصر البحث وبخلاف هذا ينتهي البحث عند شجرة فرعية فارغة. بهذا تكون الخوارزمية العامة هي:

```

if root is NULL
    Cannot search an empty tree, returns false.
else
{
    current = root;
    while(current is not NULL and not found)
        if(current->info is the same as the search item)

```

```

        set found to true;
    else
        if(current->info is greater than the search item)
            follow the llink of current
        else
            follow the rlink of current
    }

```

تتم ترجمة هذه الخوارزمية الى دالة C++ التالية:

```

template<class elemType>
bool bSearchTreeType<elemType>::search(const elemType& searchItem)
{
    nodeType<elemType> *current;
    bool found = false;

    if(root == NULL)
        cerr<<"Cannot search an empty tree."<<endl;
    else
    {
        current = root;

        while(current != NULL && !found)
        {
            if(current->info == searchItem)
                found = true;
            else
            {
                if(current->info > searchItem)
                    current = current->llink;
                else
                    current = current->rlink;
            }
        }
    }

    return found;
}

```

الادخال:

تقوم الدالة insert بادخال عنصر جديد في شجرة البحث الثنائية. بعد ادخال عنصر في شجرة البحث الثنائية يجب أن تكون الشجرة الثنائية الناتجة شجرة بحث ثنائية كذلك. لادخال عنصر جديد نقوم أولاً ببحث شجرة البحث الثنائية وايجاد المكان الذي سوف يتم ادخال العنصر الجديد فيه. خوارزمية البحث مماثلة لخوارزمية بحث الدالة search. هنا نجتاز شجرة البحث الثنائية بمؤشرين – مؤشر مثل current للتحقق من العقدة الحالية ومؤشر مثل trailCurrent الذي يشير الى أصل current. بما أن العناصر المكررة غير مسموح بها فان بحثنا يجب أن ينتهي عند شجرة فرعية فارغة. يمكننا عند هذا استخدام المؤشر trailCurrent لادخال العنصر الجديد في المكان الصحيح.

العنصر الذي يتم ادخاله insertItem سوف يتم تمريره كمعامل للدالة insert. الخوارزمية العامة هي:

أ. عمل عقدة جديدة ونسخ insertItem (عنصر الإدخال) داخل العقدة الجديدة. كذلك تحديد الوصلة اليسرى والوصلة اليمنى للعقدة الجديدة عند NULL.

ب. إذا كانت root عند NULL اذن الشجرة فارغة لهذا اجعل root تشير الى العقدة الجديدة.

```

else
{
    current = root;
    while(current is not NULL)    //search the binary tree
    {
        trailCurrent = current;
        if(current->info is the same as insertItem)
            Error: Cannot insert duplicate items.
            exit
        else
            if(current->info > insertItem)
                Follow llink of current
            else
                Follow rlink of current
    }

    //insert the new node in the binary tree

    if(trailCurrent->info > insertItem)
        make the new node the left child of trailCurrent
    else
        make the new node the right child of trailCurrent
}

```

تتم ترجمة هذه الخوارزمية الى دالة C++ التالية:

```

template<class elemType>
void bSearchTreeType<elemType>::insert(const elemType& insertItem)
{
    nodeType<elemType> *current; //pointer to traverse the tree
    nodeType<elemType> *trailCurrent; //pointer behind current
    nodeType<elemType> *newNode; //pointer to create the node

    newNode = new nodeType<elemType>;
    assert(newNode != NULL);
    newNode->info = insertItem;
    newNode->llink = NULL;
    newNode->rlink = NULL;
}

```


شكل 11-11: شجرة بحث ثنائية قبل حذف عقدة

بعد حذف العنصر المطلوب (إذا وجد في شجرة البحث الثنائية) يجب أن تكون الشجرة الناتجة شجرة بحث ثنائية. عملية الحذف لها أربع حالات:

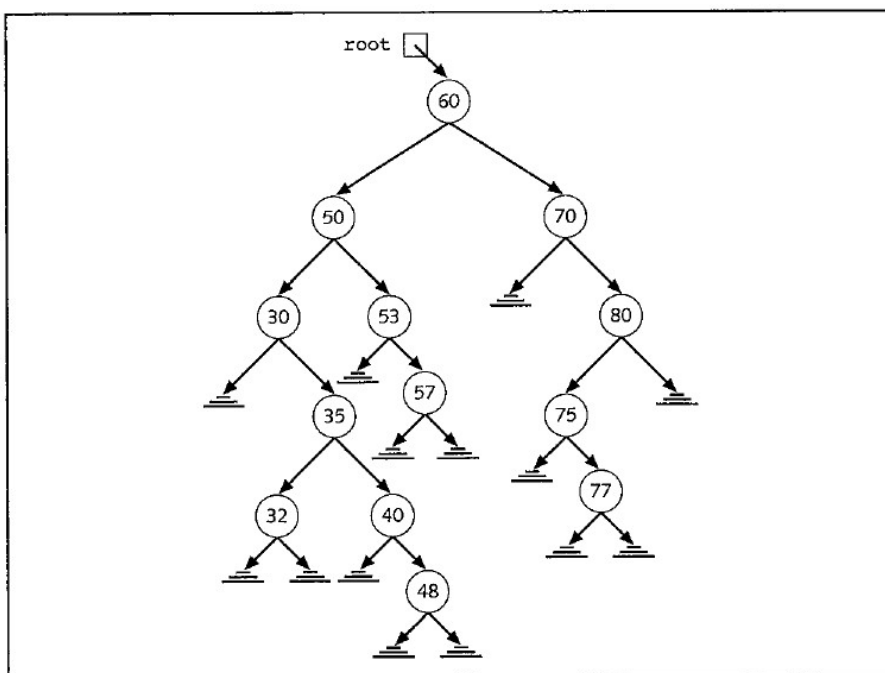
الحالة 1: العقدة التي يتم حذفها ليس لها شجرة فرعية يسرى ويمنى أي أن العقدة التي يتم حذفها عبارة عن ورقة. على سبيل المثال العقدة ذات البيان 45 عبارة عن ورقة.

الحالة 2: العقدة التي يتم حذفها ليس لها شجرة فرعية يسرى أي أن الشجرة الفرعية اليسرى خالية ولكن لديها شجرة فرعية يمنى غير فارغة. على سبيل المثال الشجرة الفرعية اليسرى للعقدة ذات البيان 40 فارغة والشجرة الفرعية اليمنى غير فارغة.

الحالة 3: العقدة التي يتم حذفها ليس لها شجرة فرعية يمنى أي أن الشجرة الفرعية اليمنى خالية ولكن لديها شجرة فرعية يسرى غير فارغة. على سبيل المثال الشجرة الفرعية اليمنى للعقدة ذات البيان 80 فارغة والشجرة الفرعية اليسرى غير فارغة.

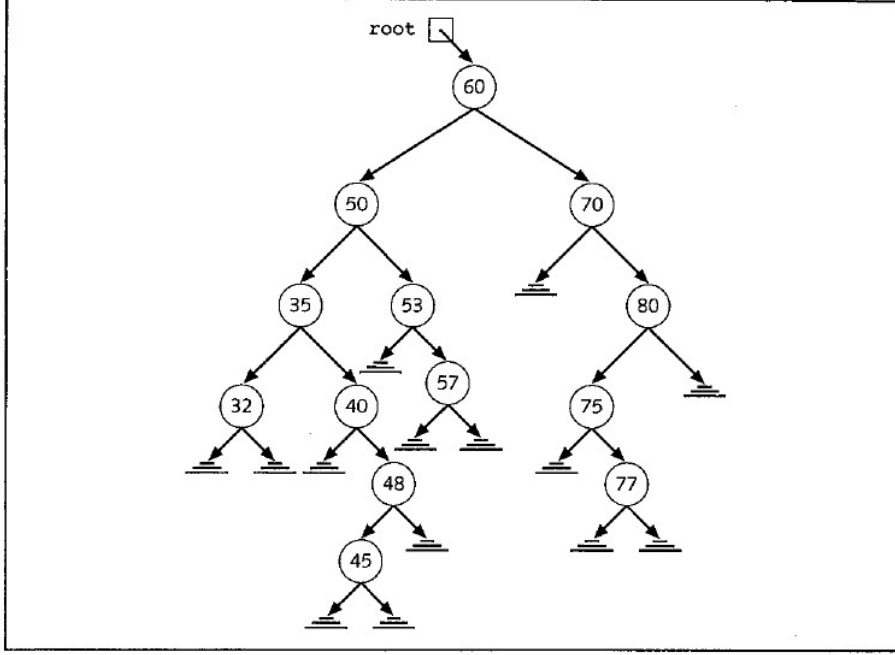
الحالة 4: العقدة التي يتم حذفها لها شجرتان فرعيتان يمنى ويسرى غير فارغتين. على سبيل المثال الشجرتين الفرعيتين اليمنى واليسرى للعقدة ذات البيان 50 غير فارغتان.

الحالة 1: افترض أننا نريد حذف 45 من شجرة البحث الثنائية في شكل 11-11. اننا نقوم ببحث شجرة البحث الثنائية ونصل إلى العقدة المحتوية على 45. بما أن هذه العقدة عبارة عن ورقة وهي الثمرة اليسرى للأصل إذن يمكننا ببساطة تحديد الوصلة اليسرى للعقدة الأصل 45 عند NULL ونزيل تخصيص الذاكرة التي تحتلها هذه العقدة. بعد حذف هذه العقدة الشكل 12-11 يوضح شجرة البحث الثنائية الناتجة.



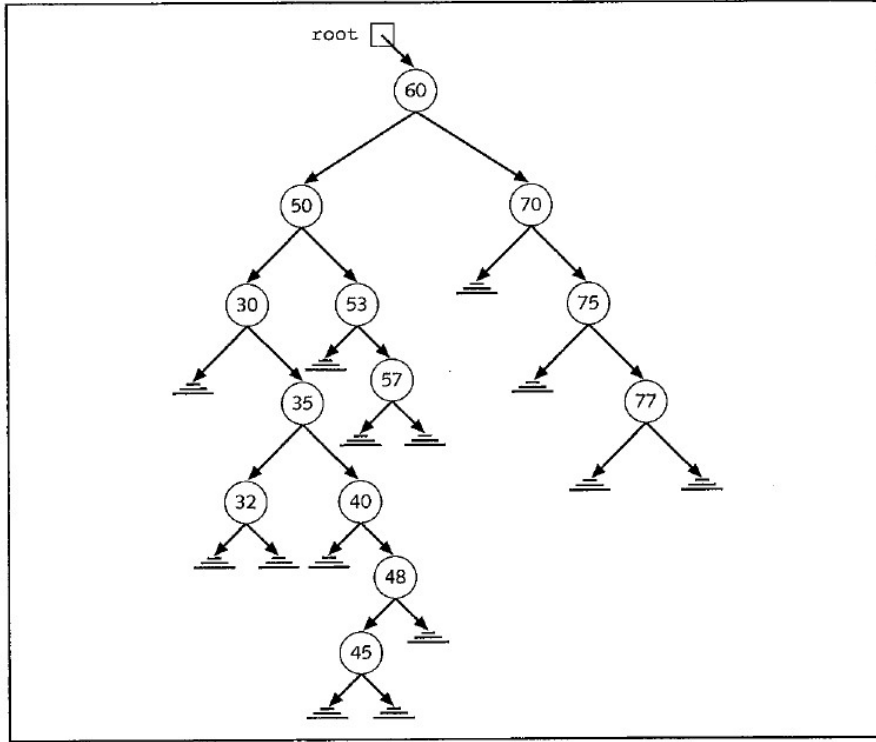
شكل 11-12: شجرة بحث ثنائية بعد حذف 45

الحالة 2: افترض أننا نريد حذف 30 من شجرة البحث الثنائية في شكل 11-11. في هذه الحالة لا يكون للعقدة المراد حذفها شجرة فرعية يسرى. بما أن 30 هي الثمرة اليسرى للعقدة الأصلية اذن نجعل الوصلة اليسرى للعقدة الأصلية 30 تشير الى الثمرة اليمنى لـ 30 – أي 35 – ثم نزيل تخصيص الذاكرة التي تحتلها 30. الشكل 11-13 يوضح شجرة البحث الثنائية الناتجة.



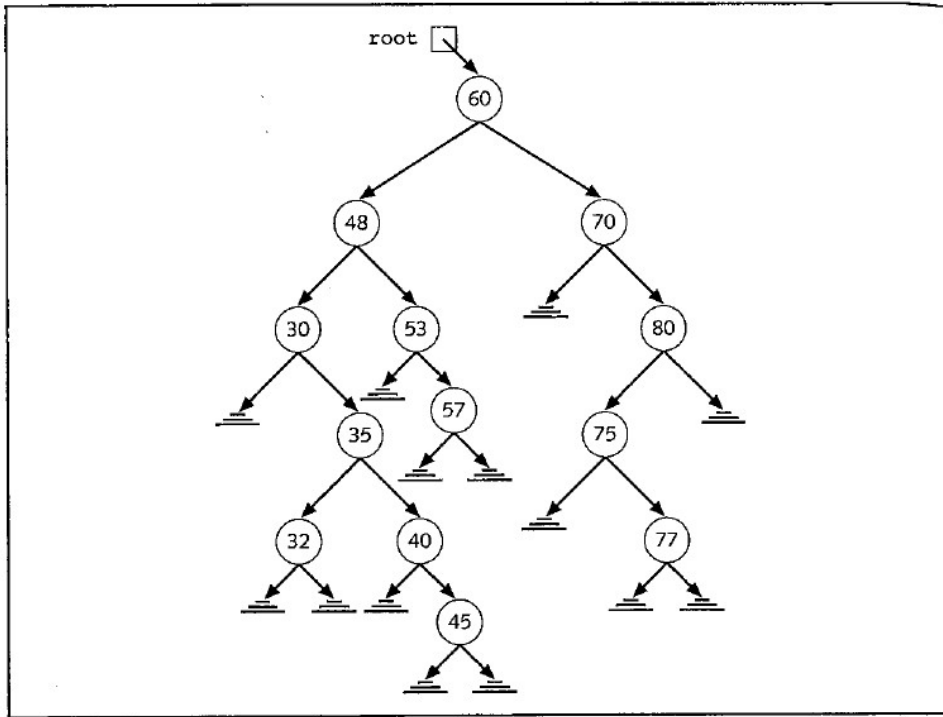
شكل 11-13: شجرة بحث ثنائية بعد حذف 30

الحالة 3: افترض أننا نريد حذف 80 من شجرة البحث الثنائية في شكل 11-11. العقدة المحتوية على 80 ليس لها ثمرة يمنى وهي الثمرة اليمنى للأصل. لهذا نجعل الوصلة اليمنى للأصل الخاص بـ 80 – أي 70 – يشير الى الثمرة اليسرى لـ 80. الشكل 11-14 يوضح شجرة البحث الثنائية الناتجة.



شكل 11-14: شجرة بحث ثنائية بعد حذف 80

الحالة 4: افترض أننا نريد حذف 50 من شجرة البحث الثنائية في شكل 11-11. العقدة ذات البيان 50 لها شجرة فرعية يسرى غير فارغة وشجرة فرعية اليمنى غير فارغة. هنا نقوم أولاً بتقليل هذه الحالة اما الى الحالة 2 أو الحالة 3 كما يلي. لكي نكون محددين افترض أننا نقللها الى الحالة 3 – أي أن العقدة التي يتم حذفها ليس لها شجرة فرعية اليمنى. بالنسبة الى هذه الحالة نوجد ما يسبق 50 مباشرة في هذه الشجرة الثنائية وهو 48. يتم القيام بهذا عن طريق الذهاب أولاً الى الثوراة اليسرى لـ 50 ثم وضع العقدة اليمنى للشجرة الفرعية اليسرى من 50. للقيام بهذا نتبع الوصلة اليمنى للعقد. بما أن شجرة البحث الثنائية محدودة اذن نصل في النهاية الى عقدة ليس لها شجرة فرعية اليمنى. بعد هذا نستبدل 48 مع 50. هذا يختزل الأمر الى الحالة التي تكون فيها العقدة المختزلة بدون شجرة فرعية اليمنى. نقوم الآن بتطبيق الحالة 3 لحذف العقدة. (لاحظ أنه بما أننا نحذف العنصر السابق مباشرة من شجرة البحث الثنائية اذن فاننا في الحقيقة ننسخ فقط بيان العنصر السابق بداخل العقدة التي يتم حذفها.) بعد حذف 50 من شجرة البحث الثنائية في شكل 11-11 تكون الشجرة الثنائية الناتجة كما هي موضحة في شكل 11-15.



شكل 11-15: شجرة بحث ثنائية بعد حذف 50

في كل حالة نرى أن الشجرة الثنائية الناتجة تكون مرة أخرى شجرة بحث ثنائية. من هذه المناقشة ينتج أنه لحذف عنصر من شجرة بحث ثنائية يجب أن نقوم بما يلي:

1. إيجاد العقدة المحتوية على العنصر (إذا وجد) المراد حذفه.

2. حذف العقدة.

نقوم بانجاز الخطوة الثانية عن طريق دالة منفصلة نطلق عليها deleteFromTree. باعطاء مؤشر الى العقدة المراد حذفها تقوم هذه الدالة بحذف العقدة عن طريق أخذ الأربع حالات السابقة في الاعتبار.

الأمثلة السابقة توضح أن في أي وقت نقوم فيه بحذف عقدة من شجرة ثنائية نقوم بتعديل واحد من مؤشرات العقدة الأصلية. بما أن التعديل يجب أن يتم اجرائها على العقدة الأصلية اذن يجب أن نستدعي الدالة deleteFromTree باستخدام مؤشر مناسب للعقدة الأصلية. على سبيل المثال افترض أن العقدة المراد حذفها هي 35 وهي الثمرة اليمنى لعقدتها الأصلية.

فضلاً عن هذا افترض أن المؤشر trailCurrent يشير الى العقدة المحتوية على 30 وهي العقدة الأصلية ل35. استدعاء الدالة deleteFromTree يكون:

```
deleteFromTree(trailCurrent->rlink);
```

بالطبع اذا كانت العقدة المراد حذفها هي العقدة الجذرية اذن يكون استدعاء الدالة deleteFromTree هو:

`deleteFromTree(root);`

الآن نقوم بتعريف دالة C++ المسماة deleteFromTree.

```
template<class elemType>
void bSearchTreeType<elemType>::deleteFromTree
    (nodeType<elemType>* &p)
{
    nodeType<elemType> *current;    //pointer to traverse
                                   //the tree

    while(current->rlink != NULL)
    {
        trailCurrent = current;
        current = current->rlink;
    }//end while

    p->info = current->info;

    if(trailCurrent == NULL) //current did not move;
                           //current == p->llink; adjust p
        p->llink = current->llink;
    else
        trailCurrent->rlink = current->llink;

    delete current;
} //end else
} //end deleteFromTree
else if(p->rlink == NULL)
{
    temp = p;
    p = temp->llink;
    delete temp;
}
else
{
    current = p->llink;
    trailCurrent = NULL;
```

بعد هذا نقوم بوصف الدالة deleteNode. الدالة deleteNode تقوم أولاً ببحث شجرة البحث الثنائية لإيجاد العقدة المحتوية على العنصر المراد حذفه. العنصر المراد حذفه deleteItem يتم تمريره كمعامل للدالة. إذا تم العثور على العقدة المحتوية على deleteItem في شجرة البحث الثنائية تقوم الدالة deleteNode باستدعاء الدالة deleteFromTree لحذف العقدة. تعريف الدالة deleteNode معطى فيما بعد.

```
template<class elemType>
void bSearchTreeType<elemType>::deleteNode
    (const elemType& deleteItem)
{
    nodeType<elemType> *current; //pointer to traverse the tree
    nodeType<elemType> *trailCurrent; //pointer behind current
    bool found = false;

    if(root == NULL)
        cerr<<"Cannot delete from the empty tree."<<endl;
    else
    {
        current = root;
        trailCurrent = root;

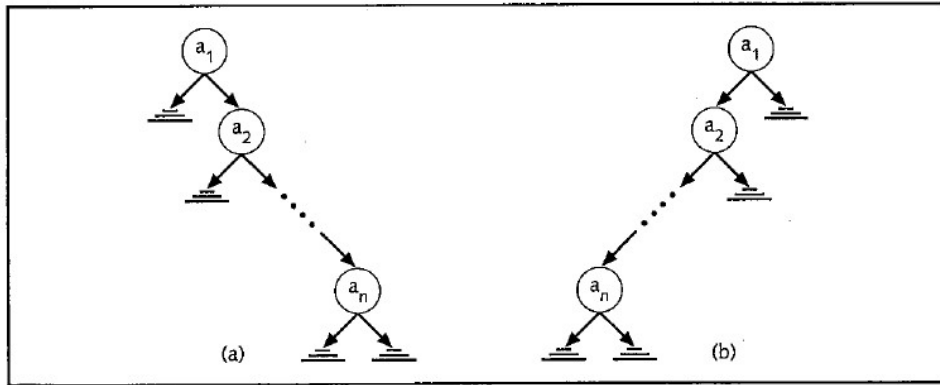
        while(current != NULL && !found)
        {
            if(current->info == deleteItem)
                found = true;
            else
            {
                trailCurrent = current;

                if(current->info > deleteItem)
                    current = current->llink;
                else
                    current = current->rlink;
            }
        } //end while

        if(current == NULL)
            cout<<"The delete item is not in the list."<<endl;
        else
            if(found)
            {
                if(current == root)
                    deleteFromTree(root);
                else
                {
                    if(trailCurrent->info > deleteItem)
                        deleteFromTree(trailCurrent->llink);
                    else
                        deleteFromTree(trailCurrent->rlink);
                } //end if
            }
    } //end deleteNode
}
```

شجرة البحث الثنائية: تحليل

هذا القسم يقدم تحليلاً لأداء أشجار البحث الثنائية. لتكتمل شجرة بحث ثنائية ذات عدد n من العقد حيث $n > 0$. افترض أننا نريد تحديد ما إذا كان العنصر x موجود في الشجرة T . يعتمد أداء خوارزمية البحث على شكل الشجرة T . لنقم أولاً بالنظر إلى أسوأ حالة. في أسوأ حالة تكون T طولية أي تكون T في شكل من الأشكال الموضحة في شكل 11-16.



شكل 11-16: أشجار بحث ثنائية طولية

بما أن الشجرة T طولية إذن يكون أداء خوارزمية البحث على T هو نفس أداءها على قائمة طولية. لهذا في الحالة الناجحة تقوم خوارزمية البحث في المتوسط بعمل عدد $\frac{n+1}{2}$ مقارنات. تقوم الخوارزمية في الحالة الغير ناجحة بعمل n مقارنة.

لنقم الآن بالنظر إلى سلوك الحالة العادية. في الحالة الناجحة ينتهي البحث عند عقدة. بما أن هناك عدد n من العناصر إذن فهناك عدد $n!$ من الترتيبات الممكنة للمفاتيح. نفترض أن جميع الترتيبات البالغ عددها $n!$ للمفاتيح محتملة. لتكن $S(n)$ معبرة عن عدد المقارنات في الحالة العادية الناجحة ولتكن $U(n)$ معبرة عن عدد المقارنات في الحالة العادية الغير ناجحة.

عدد المقارنات اللازمة لتحديد ما إذا كان العنصر x موجود في الشجرة T يزيد عن عدد المقارنات اللازمة لإدخال x في T بمقدار واحد. فضلاً عن هذا فإن عدد المقارنات اللازمة لإدخال x في T هو نفس عدد المقارنات التي يتم عملها في البحث الغير ناجح وهذا يعكس أن x ليست في T . من هذا ينتج أن:

$$S(n) = 1 + \frac{U(0) + U(1) + \dots + U(n-1)}{n} \quad (11-1)$$

من المعلوم كذلك أن:

$$S(n) = \left(1 + \frac{1}{n}\right)U(n) - 3 \quad (11-2)$$

بحل المعادلتين رقم (1-11) و(2-11) يمكن توضيح أن:

$$U(n) \approx 2.77 \log_2 n$$

وأن:

$$S(n) \approx 1.39 \log_2 n$$

يمكننا الآن تكوين النتيجة التالية.

النظرية: لتكن T شجرة بحث ثنائية ذات عدد n من العقد حيث $n > 0$. متوسط عدد العقد التي تتم زيارتها في بحث الشجرة T يكون بالتقريب $1.39 \log_2 n$.

خوارزميات اجتياز الشجرة الثنائية الغير تكرارية:

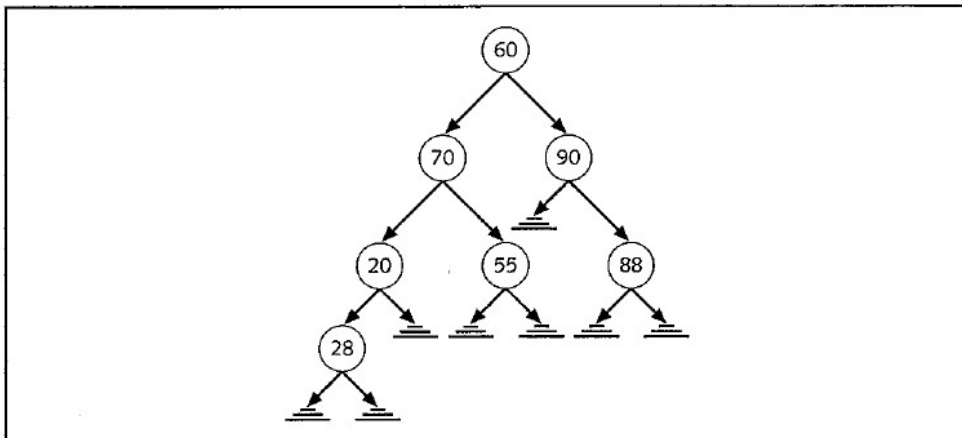
الأقسام السابقة قامت بتوضيح كيفية القيام بما يلي:

- اجتياز شجرة ثنائية باستخدام أساليب ما أثناء وقبل وبعد الترتيب.
- بناء شجرة بحث ثنائية.
- ادخال عنصر في شجرة بحث ثنائية.
- حذف عنصر من شجرة بحث ثنائية.

خوارزميات الاجتياز – أثناء وقبل وبعد الترتيب – التي تمت مناقشتها سابقاً خوارزميات تكرارية. بما أن اجتياز شجرة ثنائية يتبر عملية ضرورية وبما أن هناك دالات تكرارية أقل كفاءة نوعاً ما من صورها المكررة فإن هذا القسم يناقش الخوارزميات الغير تكرارية للاجتياز أثناء وقبل وبعد الترتيب.

الاجتياز أثناء الترتيب الغير تكراري:

في اجتياز شجرة ثنائية أثناء الترتيب يتم زيارة الشجرة الفرعية اليسرى لكل عقدة ثم زيارة العقدة ثم زيارة الشجرة الفرعية اليمنى. ينتج من هذا أنه في الاجتياز أثناء الترتيب تكون العقدة الأولى التي تتم زيارتها هي العقدة اليسرى من الشجرة الثنائية. على سبيل المثال في الشجرة الثنائية في شكل 11-17 تكون العقدة اليسرى هي العقدة ذات البيان 28.



شكل 11-17: شجرة ثنائية والعقدة اليسرى هي 28

للذهاب الى العقدة اليسرى للشجرة الثنائية نبدأ باجتياز الشجرة الثنائية عند العقدة الجذرية ثم نتبع الوصلة اليسرى لكل عقدة حتى تصبح الوصلة اليسرى لاحدى العقد NULL. بعد هذا نعود الى العقدة الأصلية ونقوم بزيارة العقدة ثم ننتقل الى العقدة اليمنى. بما أن الوصلات تذهب في اتجاه واحد فقط اذن من أجل الرجوع الى عقدة ما يجب أن نحتفظ بمؤشر الى العقدة قبل الانتقال الى العقدة الثمرة. فضلاً عن هذا يجب أن يتم تتبع العقد من الخلف بالترتيب الذي تم اجتيازها به. ينتج من هذا أنه أثناء التتبع من الخلف يجب أن تتم زيارة العقد بأسلوب الداخل أخيراً يخرج أولاً. يمكن القيام بهذا عن طريق استخدام مرصوصة. لهذا نقوم بالاحتفاظ بمؤشر الى العقدة في مرصوصة. الخوارزمية العامة تكون كما يلي:

```
1. current = root; //start traversing the binary tree at the
//root node
2. while(current is not NULL or stack is nonempty)
    if(current is not NULL)
    {
        push current onto stack;
        current = current->llink;
    }
    else
    {
        pop stack into current;
        visit current; //visit the node
        current = current->rlink; //move to the
//right child
    }
```

الدالة التالية تقوم بتطبيق الاجتياز الغير تكراري أثناء الترتيب لشجرة ثنائية:

```
template<class elemType>
void binaryTreeType<elemType>::nonRecursiveInTraversal()
{
    stackType<nodeType<elemType>* > stack;
    nodeType<elemType> *current;
    current = root;

    while((current != NULL) || (!stack.isEmptyStack()))
        if(current != NULL)
        {
            stack.push(current);
            current = current->llink;
        }
        else
        {
            current = stack.top();
            stack.pop();
            cout<<current->info<<" ";
            current = current->rlink;
        }

    cout<<endl;
}
```

الاجتياز الغير تكرار قبل الترتيب:

في اجتياز شجرة ثنائية قبل الترتيب يتم أولاً زيارة العقدة ثم زيارة الشجرة الفرعية اليسرى ثم الشجرة الفرعية اليمنى. كما في حالة الاجتياز أثناء الترتيب يجب علينا بعد زيارة عقدة وقبل الانتقال الى الشجرة الفرعية اليسرى أن نحتفظ بمؤشر الى العقدة حتى يمكننا بعد زيارة الشجرة الفرعية اليسرى أن نزور الشجرة الفرعية اليمنى. الخوارزمية العامة تكون كما يلي:

```
1. current = root; //start the traversal at the root node
2. while(current is not NULL or stack is nonempty)
    if(current is not NULL)
    {
        visit current;
        push current onto stack;
        current = current->llink;
    }
    else
    {
        pop stack into current;
        current = current->rlink; //prepare to visit
                                //the right subtree
    }
```

الدالة التالية تقوم بتطبيق خوارزمية الاجتياز الغير تكراري قبل الترتيب:

```
template<class elemType>
void binaryTreeType<elemType>::nonRecursivePreTraversal()
{
    stackType<nodeType<elemType>* > stack;
    nodeType<elemType> *current;

    current = root;

    while((current != NULL) || (!stack.isEmptyStack()))
        if(current != NULL)
        {
            cout<<current->info<<" ";
            stack.push(current);
            current = current->llink;
        }
        else
        {
            current = stack.top();
            stack.pop();
            current = current->rlink;
        }

    cout<<endl;
}
```

الاجتياز الغير تكرار بعد الترتيب:

في اجتياز شجرة ثنائية بعد الترتيب يتم أولاً زيارة الشجرة الفرعية اليسرى ثم زيارة الشجرة الفرعية اليمنى ثم زيارة العقدة. كما في حالة الاجتياز أثناء الترتيب والاجتياز بعد الترتيب تكون العقدة الأولى التي تتم زيارتها هي العقدة اليسرى من الشجرة الثنائية. بما أن – بالنسبة الى كل عقدة – تتم زيارة الشجرتين الفرعيتين اليمنى واليسرى قبل زيارة العقدة اذن يجب علينا الاشارة الى العقدة ما اذا تمت زيارة الشجرتين الفرعيتين اليسرى واليمنى. بعد زيارة الشجرة الفرعية اليسرى لعقدة وقبل زيارة العقدة يجب أن نخبر العقدة أن الشجرة الفرعية اليمنى تحتاج الى زيارتها وبعد زيارة الشجرة الفرعية اليمنى يجب علينا اخبار العقدة أنه يمكن الآن زيارتها. للقيام بهذا فاننا بدلاً من الاحتفاظ بمؤشر للعقدة (للرجوع الى الشجرة الفرعية اليمنى والى العقدة نفسها) نقوم كذلك بالاحتفاظ بقيمة عدد صحيح تبلغ 1 قبل الانتقال الى الشجرة الفرعية اليسرى و بقيمة عدد صحيح تبلغ 2 قبل الانتقال الى الشجرة الفرعية اليمنى. عند ازالة المرصوصة تتم ازالة قيمة العدد الصحيح المرتبطة بهذا المؤشر أيضاً. قيمة هذا العدد الصحيح تقول ما اذا كانت تمت زيارة الشجرتان الفرعيتان اليمنى واليسرى من العقدة.

الخوارزمية العامة هي:

```
1. current = root;    //start the traversal at the root node
2. v = 0;
3. if(current is NULL)
    the binary tree is empty
4. if(current is not NULL)
    a. push current into stack;
    b. push 1 onto stack;
    c. current = current->llink;
    d. while(stack is not empty)
        if(current is not NULL and v is 0)
        {
            push current and 1 onto stack;
            current = current->llink;
        }
        else
        {
            pop stack into current and v;
            if(v == 1)
            {
                push current and 2 onto stack;
                current = current->rlink;
                v = 0;
            }
            else
                visit current;
        }
    }
```

اننا نقوم باستخدام مرصوصتان (متوازيتان): مرصوصة للاحتفاظ بمؤشر الى العقدة وأخرى للاحتفاظ بقيمة العدد الصحيح (1 أو 2) المرتبط بهذا المؤشر. نترك لك كتمرين كتابة تعريف دالة C++ لتطبيق خوارزمية الاجتياز بعد الترتيب السابقة. (انظر تمرين البرمجة 6 في نهاية هذا الفصل).

اجتياز الشجرة الثنائية والدالات كمعاملات:

افترض أنك قمت بتخزين بيانات موظف في شجرة بحث ثنائية وفي نهاية العام تم منح كل موظف زيادة في الراتب أو مكافأة. هذه المهمة تتطلب أن تتم زيارة كل عقدة في شجرة البحث الثنائية وأن يتم تحديث راتب كل موظف. الأقسام السابقة ناقشت العديد من الطرق لاجتياز شجرة ثنائية. بالرغم من هذا فان في خوارزميات الاجتياز هذه – أثناء وقبل وبعد الترتيب – عندما تتم زيارة عقدة فاننا من أجل البساطة ولأهداف توضيحية نقوم فقط باخراج البيانات الموجودة في كل عقدة. كيف نقوم باستخدام خوارزمية اجتياز لزيارة كل عقدة وتحديث البيانات الموجودة في كل عقدة؟ من احدى طرق القيام بهذا أولاً عمل شجرة ثنائية أخرى تكون فيها البيانات في كل عقدة هي البيانات التي يتم تحديثها من الشجرة الثنائية الأصلية ثم نقوم بتدمير الشجرة الثنائية القديمة. هذا قد يتطلب زمن حاسوبي اضافي وربما مال اضافي ولهذا فانه ليس ذو كفاءة. هناك حل آخر وهو كتابة خوارزميات اجتياز منفصلة لتحديث البيانات. هذا الحل يتطلب منك تعديل تعريف الفئة التي تقوم بتطبيق شجرة البحث الثنائية. بالرغم من هذا اذا استطاع المستخدم كتابة دالة مناسبة لتحديث بيانات كل موظف ثم تمرير هذه الدالة كمعامل لخوارزميات الاجتياز فيمكننا تعزيز مرونة البرنامج بقدر كبير. هذا الفصل يصف كيفية تمرير الدالات كمعاملات لدالات أخرى.

في C++ يتم اعتبار اسم الدالة بدون أي أقواس مؤشر الى الدالة. لتحديد دالة كمعامل رسمي لدالة أخرى نقوم بتحديد نوع الدالة يليها اسم الدالة كمؤشر ويليه ، أنواع معاملات الدالة. على سبيل المثال انظر الى البيانات التالية:

```
void fParamFunc1(void (*visit) (int));           //Line 1
void fParamFunc2(void (*visit) (elemType&));     //Line 2
```

البيان في الصف 1 يعلن أن fParamFunc1 دالة تتخذ من أي دالة void لها معامل قيمة واحد من النوع int كمعامل لها. البيان في الصف 2 يعلن أن fParamFunc2 دالة تتخذ من أي دالة void لها معامل اشارة واحد من النوع elemType كمعامل لها.

يمكننا الآن اعادة كتابة دالة الاجتياز أثناء الترتيب للفئة binaryTreeType. كما يمكننا اثقال دالات الاجتياز المتواجدة أثناء الترتيب. لتوضيح اثقال الدالة سوف نقوم باثقال دالات الاجتياز في الترتيب.

لهذا ندخل البيانات التالية في تعريف الفئة binaryTreeType:

```
void inorderTraversal(void (*visit) (elemType&));
//Function to do an inorder traversal of the binary tree.
//The parameter visit, which is a function, specifies the
//action to be taken at each node.
```

```
void inorderTraversal (void (*visit) (elemType& ) );
// دالة لعمل اجتياز أثناء الترتيب لشجرة ثنائية.
// زيارة المعامل وهو الدالة يحدد
// الاجراء الذي يتم اتخاذه عند كل عقدة.
```

```
void inorder(nodeType<elemType> *p, void (*visit) (elemType&));
//Function to do an inorder traversal of the binary tree,
//starting at the node specified by the parameter p.
//The parameter visit, which is a function, specifies the
//action to be taken at each node.
```

```
void inorder (nodeType<elemType> *p, void (*visit) (elemType& ) );
// دالة لعمل اجتياز أثناء الترتيب لشجرة ثنائية.
// بدءاً من العقدة المحددة بواسطة المعامل p
// زيارة المعامل وهو الدالة يحدد
// الاجراء الذي يتم اتخاذه عند كل عقدة.
```

تعريفات هذه الدالات تكون كما يلي:

```
template<class elemType>
void binaryTreeType<elemType>::inorderTraversal
    (void (*visit) (elemType& item))
{
    inorder(root, *visit);
}

template<class elemType>
void binaryTreeType<elemType>::inorder(nodeType<elemType>* p,
    void (*visit) (elemType& item))
{
    if(p != NULL)
    {
        inorder(p->llink, *visit);
        (*visit)(p->info);
        inorder(p->rlink, *visit);
    }
}
```

البيان:

```
(*visit) (p -> info);
```

في تعريف الدالة inorder يقوم بعمل استدعاء للدالة ذات معامل اشارة واحد من النوع elemType المشار اليه بواسطة المؤشر visit.

المثال 6-11 يوضح كيفية تمرير الدالات كمعاملات لدالات أخرى.

مثال 6-11:

هذا المثال يوضح كيفية تمرير دالة محددة بواسطة المستخدم كمعامل لخوارزميات اجتياز الشجرة الثنائية. لأهداف توضيحية نوضح كيفية استخدام دالة الاجتياز أثناء الترتيب فقط.

البرنامج التالي يستخدم الفئة bSearchTreeType المستمدة من الفئة binaryTreeType لبناء الشجرة الثنائية. دالات الاجتياز موجودة في الفئة binaryTreeType التي يتم توريثها بعد هذا بواسطة الفئة bSearchTreeType.

```
#include <iostream>
#include "binarySearchTree.h"

using namespace std;

void print(int& x);
void update(int& x);

int main()
{
    bSearchTreeType<int> treeRoot; //Line 1

    int num; //Line 2

    cout<<"Line 3: Enter numbers ending with -999"
        <<endl; //Line 3
    cin>>num; //Line 4

    while(num != -999) //Line 5
    {
        treeRoot.insert(num); //Line 6
        cin>>num; //Line 7
    }

    cout<<endl<<"Line 8: Tree nodes in inorder: "; //Line 8
    treeRoot.inorderTraversal(print); //Line 9
    cout<<endl<<"Line 10: Tree Height: "
        <<treeRoot.treeHeight()
        <<endl<<endl; //Line 10

    cout<<"Line 11: ***** Update Nodes *****"
        <<endl; //Line 11
    treeRoot.inorderTraversal(update); //Line 12

    cout<<"Line 13: Tree nodes in inorder after "
        <<"the update: " <<endl<<" "; //Line 13
    treeRoot.inorderTraversal(print); //Line 14
    cout<<endl<<"Line 15: Tree Height: "
        <<treeRoot.treeHeight()
        <<endl; //Line 15

    return 0; //Line 16
}

void print(int& x) //Line 17
{
    cout<<x<<" "; //Line 18
}

void update(int& x) //Line 19
{
    x = 2 * x; //Line 20
}
```

تنفيذ العينة: في تنفيذ العينة هذا يتم تظليل مدخلات المستخدم.

الصف 3: ادخل أعداد تنتهي ب-999

56 87 23 65 34 45 12 90 66 999-

الصف 8: عقد الشجرة أثناء الترتيب: 12 23 34 45 56 65 66 87 90

الصف 10: ارتفاع الشجرة: 4

الصف 11: ***** تحديث العقد *****

الصف 13: عقد الشجرة أثناء الترتيب بعد التحديث:

24 46 68 90 112 130 132 174 180

الصف 15: ارتفاع الشجرة: 4

هذا البرنامج يعمل كما يلي. البيان في الصف 1 يعلن أن treeRoot هدف شجرة بحث ثنائية تكون فيه البيانات في كل عقدة من النوع int. البيانات في الصفوف من 4 وحتى 7 تبني شجرة البحث الثنائية. البيان في الصف 9 يستخدم دالة العنصر inorderTraversal من treeRoot لاجتياز شجرة البحث الثنائية treeRoot. معامل الدالة inorderTraversal في الصف 9 هو الدالة print (المعرفة في الصف 17). بما أن الدالة print تخرج قيمة معطياتها فإن البيان في الصف 9 يخرج بيانات عقد شجرة البحث الثنائية treeNode. البيان في الصف 10 يخرج ارتفاع شجرة البحث الثنائية. البيان في الصف 12 يستخدم دالة العنصر inorderTraversal لاجتياز شجرة البحث الثنائية treeRoot. في الصف 12 يكون المعامل الحقيقي للدالة inorderTraversal هو الدالة update (المعرف في الصف 11). تقوم الدالة update بمضاعفة قيمة معطياتها. لهذا يقوم البيان في الصف 12 بتحديث بيانات كل عقدة من شجرة البحث الثنائية عن طريق مضاعفة القيمة. البيانات في الصف 14 و15 تخرج العقد وارتفاع شجرة البحث الثنائية.

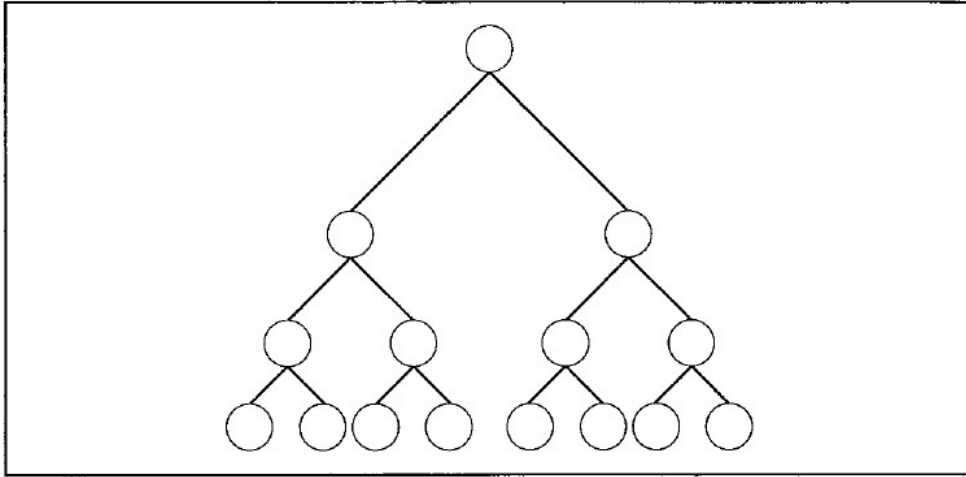
أشجار AVL (متزنة الارتفاع):

قمت في الأقسام السابقة بتعلم كيفية بناء وسيطرة شجرة بحث ثنائية. يعتمد أداء خوارزمية البحث في شجرة البحث الثنائية على مجموعة البيانات. إذا كانت مجموعة البيانات مرتبة اذن تكون شجرة البحث الثنائية طويلة وهكذا لن تكون خوارزمية البحث ذات كفاءة. من الجهة الأخرى اذا تم بناء الشجرة جيداً اذن سوف يكون البحث سريعاً. في الحقيقة كلما قل ارتفاع الشجرة كلما كان البحث أسرع. لهذا نريد أن يكون ارتفاع شجرة البحث الثنائية صغيراً بقدر الامكان. يقوم هذا القسم بوصف نوع خاص من شجرة البحث الثنائية ويسمى شجرة AVL (تسمى أيضاً شجرة متوازنة الارتفاع) تكون فيها شجرة البحث الثنائية الناتجة متوازنة تقريباً. تم تقديم أشجار AVL عن طريق عالم الرياضيات جي ام اديلسون – فيليسكي واي ام لاندس في عام 1962 وتم تسميتها هكذا نسبة اليهما. نبدأ بتعريف المصطلحات التالية:

تعريف: الشجرة الثنائية المتوازنة بشكل كامل تكون عبارة عن شجرة ثنائية بحيث أن:

1. ارتفاع الشجرتين الفرعيتين اليمنى واليسرى من الجذر متساوي.
2. الأشجار الفرعية اليمنى واليسرى للجذر عبارة عن أشجار ثنائية كاملة الاتزان.

الشكل 11- 18 يوضح شجرة ثنائية كاملة الاتزان.



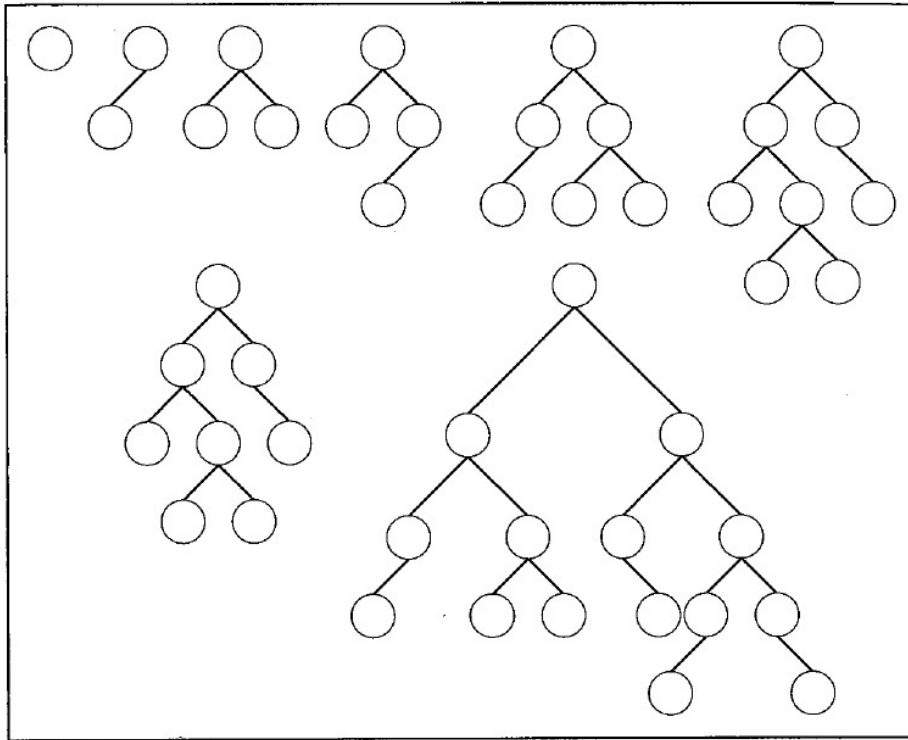
الشكل 11- 18: شجرة ثنائية كاملة الاتزان

لتكن T شجرة ثنائية ولتكن x عقدة في الشجرة T . إذا كانت T متزنة تماماً اذن من تعريف الشجرة الكاملة الاتزان ينتج أن ارتفاع الشجرة الفرعية اليسرى من x هو نفسه ارتفاع الشجرة الفرعية اليمنى من x .

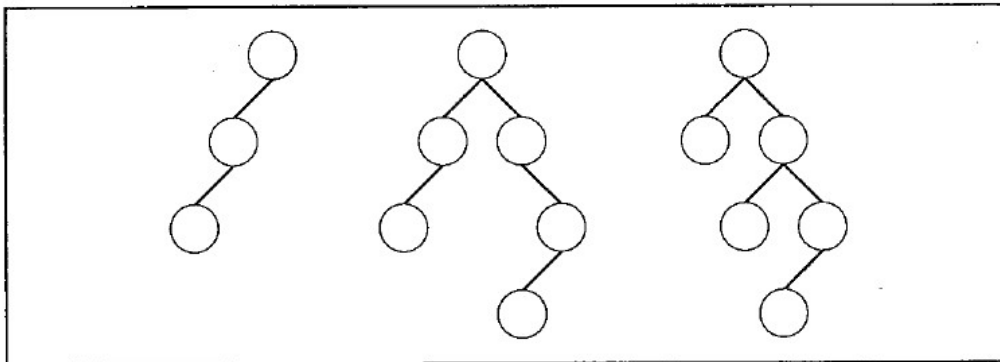
يمكن اثبات أنه إذا كانت T شجرة ثنائية كاملة الاتزان وارتفاعها h فان عدد العقد في الشجرة T يكون . ينتج من 2^{h-1} إذا لم يكن عدد العناصر في مجموعة البيانات ذو قوة 2 فاذن لا يمكننا بناء شجرة ثنائية كاملة الاتزان. فضلاً عن هذا تكون الأشجار الثنائية الكاملة الاتزان عبارة عن تحسن شديد الصرامة.

تعريف: شجرة AVL (أو الشجرة المتوازنة الارتفاع) عبارة عن شجرة بحث ثنائية بحيث أن:

1. ارتفاع الأشجار الفرعية اليمنى واليسرى للجذر تختلف بمقدار 1 على الأكثر.
 2. الأشجار الفرعية اليمنى واليسرى من الجذر عبارة عن أشجار AVL.
- الأشكال 11-19 و 11-20 تعطي أمثلة عن الأشجار AVL والأشجار التي ليست من النوع AVL.



شكل 11-19: أشجار AVL.



شكل 11-20: أشجار ليسن من النوع AVL.

لتكن x عقدة في شجرة ثنائية. لتكن x_l معبرة عن ارتفاع الشجرة الفرعية اليسرى لـ x وكانت x_r معبرة عن ارتفاع الشجرة الفرعية اليمنى لـ x .

مسألة: لتكن T شجرة AVL ولتكن x عقدة في الشجرة T . اذن $|x_r - x_l| \leq 1$ تعبر عن القيمة المجردة لـ $x_r - x_l$

لتكن x عقدة في شجرة AVL. اذن:

1. اذا كانت $x_l > x_r$ أن x مرتفع أيسر. في هذه الحالة $x_l = x_r + 1$

2. اذا كانت $x_l = x_r$ أن x مرتفع معادل.

3. اذا كانت $x_r > x_l$ أن x مرتفع أيمن. في هذه الحالة $x_r = x_l + 1$

تعريف: عامل التوازن لـ x وتكتب $bf(x)$ وتعرف هكذا $bf(x) = x_r - x_l$

لتكن x عقدة في الشجرة T التي من النوع AVL. اذن:

1. اذا كان x مرتفع أيسر فاذن $bf(x) = -1$

2. اذا كان x مرتفع متساوي فاذن $bf(x) = 0$

3. اذا كان x مرتفع أيمن فاذن $bf(x) = 1$

تعريف: لتكن x عقدة في شجرة ثنائية. نقول أن العقدة x تتعدى على معيار الاتزان اذا كانت

$|x_r - x_l| > 1$ الفرق بين ارتفاع الشجرتين الفرعيتين اليسرى واليمنى أكبر من 1.

من المناقشة السابقة ينتج أنه بالاضافة الى البيانات والمؤشرات الى الشجرتين الفرعيتين اليسرى واليمنى هناك شئ آخر مرتبط بكل عقدة x في الشجرة AVL وهو عامل اتزان x . بهذا يجب أن تقوم كل عقدة بتتبع عامل اتزانها. لجعل الخوارزميات ذات كفاءة نقوم بتخزين عامل اتزان كل عقدة في العقدة ذاتها. من هنا يكون تعريف العقدة في شجرة AVL هو:

```
template<class elemType>
struct AVLNode
{
    elemType info;
    int bfactor; //balance factor
    AVLNode<elemType> *llink;
    AVLNode<elemType> *rlink;
};
```

بما أن الشجرة AVL شجرة بحث ثنائية تكون خوارزمية البحث للشجرة AVL هي نفس خوارزمية بحث شجرة البحث الثنائية. يمكن تطبيق عمليات أخرى مثل ايجاد الارتفاع وتحديد عدد العقد والتحقق مما اذا كانت الشجرة فارغة واجتياز الشجرة وغيرها على الأشجار AVL بنفس الطريقة التي يتم بها تطبيقها على الأشجار الثنائية. بالرغم من هذا فان عمليات الادخال والحذف على أشجار AVL تكون مختلفة نوعاً ما عن العمليات التي نوقشت على أشجار البحث الثنائية. هذا لأنه بعد

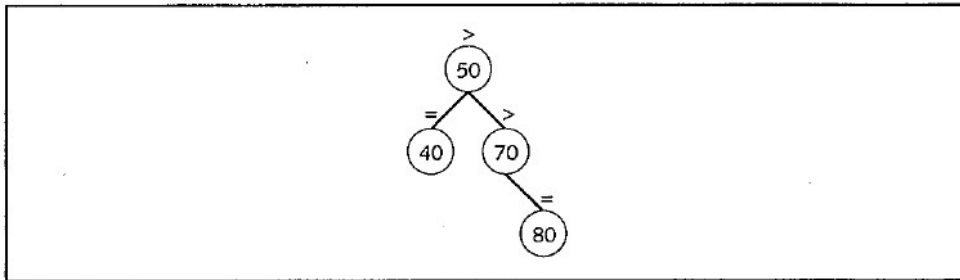
ادخال (أو حذف) عقدة في شجرة AVL فان الشجرة الثنائية الناتجة يجب أن تكون شجرة AVL. بعد هذا نصف هذه العمليات.

الادخال في أشجار AVL:

لادخال عنصر في شجرة AVL نقوم أولاً ببحث الشجرة وإيجاد المكان الذي يتم ادخال العنصر الجديد فيه. بما أن شجرة AVL عبارة عن شجرة بحث ثنائية فانه يمكننا لإيجاد مكان من أجل العنصر الجديد أن نبحث شجرة AVL باستخدام خوارزمية بحث مماثلة لخوارزمية البحث التي تم تصميمها من أجل أشجار البحث الثنائية. اذا كان العنصر المراد ادخاله موجود بالفعل في الشجرة اذن ينتهي البحث عند شجرة فرعية فارغة. بما أن المكررات غير مسموح بها فاننا في هذه الحالة يمكننا اخراج رسالة خطأ مناسبة. افترض أن العنصر المراد ادخاله غير موجود في الشجرة AVL. اذن ينتهي البحث عند شجرة فرعية فارغة ونقوم بادخال العنصر في هذه الشجرة الفرعية. بعد ادخال العنصر الجديد في الشجرة قد لا تكون الشجرة الناتجة شجرة AVL. لهذا يجب أن نستعيد معيار اتزان الشجرة. يتحقق هذا عن طريق اجتياز نفس المسار عودةً الى العقدة الجذرية الذي يتم اتباعه عندما تم ادخال العنصر الجديد في الشجرة AVL. تتم زيارة العقد الموجودة على هذا المسار (عودةً الى العقدة الجذرية) واما أن عوامل اتزانهم تتغير أو قد نضطر الى اعادة بناء جزء من الشجرة. نقوم بتوضيح هذه الحالات بمساعدة الأمثلة التالية.

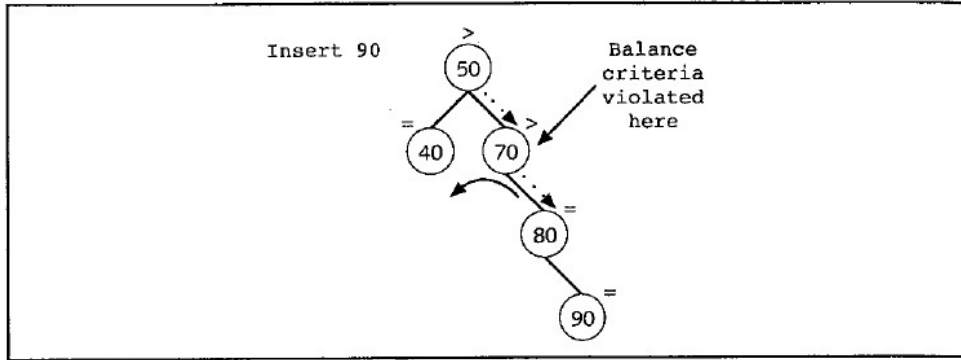
في الأشكال من 11-21 وحتى 11-32 نقوم لكل خطوة بتوضيح البيانات المخزنة فقط في العقدة وفضلاً عن هذا تشير علامة يساوي (=) فوق العقدة الى أن عامل اتزان هذه العقدة يساوي صفر وتشير علامة أصغر من (<) الى أن عامل اتزان هذه العقدة يساوي -1 وتشير علامة أكبر من (>) الى أن عامل اتزان هذه العقدة يساوي 1.

انظر الى شجرة AVL في شكل 11-21.



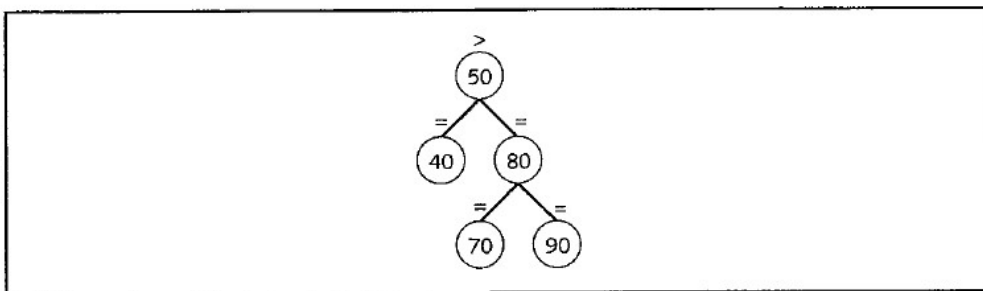
شكل 11-21: شجرة AVL قبل ادخال 90

لنقم بادخال 90 داخل شجرة AVL. نقوم ببحث الشجرة بدءاً من العقدة الجذرية لإيجاد مكان من أجل 90. السهم المنقط يوضح المسار الذي تم اجتيازه. نقوم بادخال العقدة ذات البيان 90 ونحصل على شجرة البحث الثنائية في شكل 11-22.



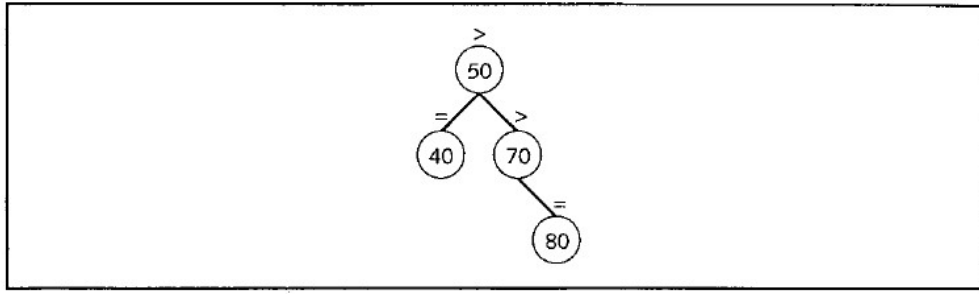
شكل 22-11: الشجرة الثنائية من شكل 21-11 قبل ادخال 90. العقد بخلاف 90 توضح عوامل اتزانها قبل الادخال

شجرة البحث الثنائية الموضحة في شكل 22-11 ليست شجرة AVL. لهذا نتتبع من الخلف ونذهب الى العقدة 80. قبل الادخال كان عامل اتزان 80 يساوي صفر. بما أن العقدة الجديدة تم ادخالها داخل الشجرة الفرعية اليمنى (الفارغة) من 80 فاننا نقوم بتغيير عامل اتزانها الى 1 (غير موضح فس الشكل). الآن نذهب مرة أخرى الى العقدة 70. قبل الادخال كان عامل اتزان 70 يساوي 1. بعد الادخال يزداد ارتفاع الشجرة الفرعية اليمنى للعقدة 70 وبهذا نرى أن الشجرة الفرعية ذات العقدة الجذرية 70 ليست شجرة AVL. في هذه الحالة نعيد اقامة هذه الشجرة الفرعية (يطلق على هذا دوران الشجرة عند العقدة الجذرية 70). بهذا نحصل على الشجرة AVL كما هي موضحة في شكل 23-11.



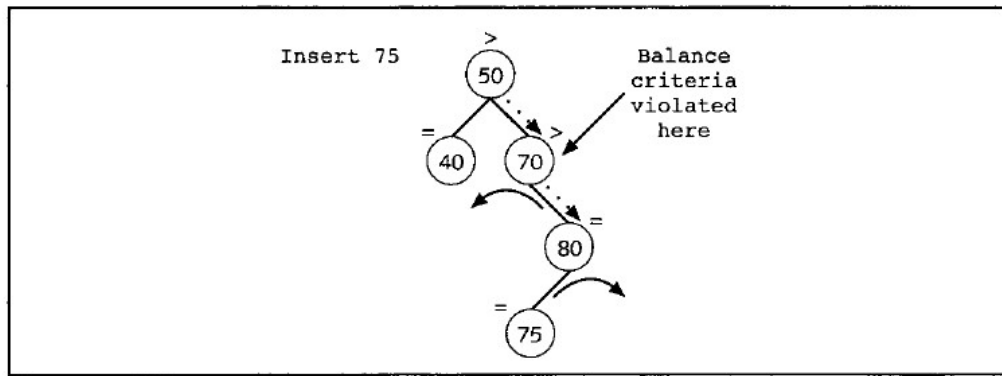
شكل 23-11: الشجرة الثنائية من شكل 21-11 قبل ادخال 90 وتعديل عوامل الاتزان

شجرة البحث الثنائية الموجودة في شكل 23-11 ليست شجرة AVL. الآن انظر الى شجرة AVL في شكل 24-11.



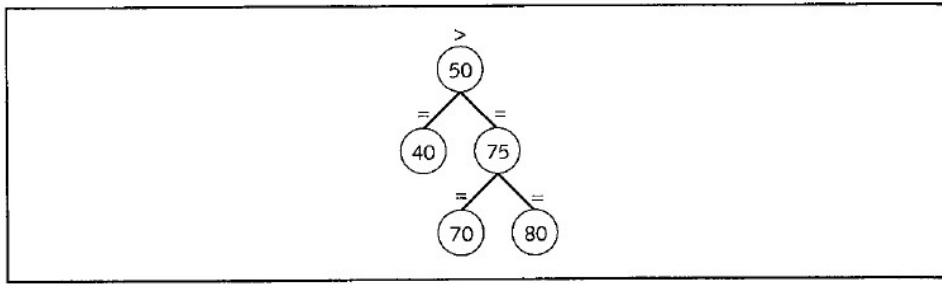
شكل 24-11: شجرة AVL قبل ادخال 75

لنقم بادخال 75 داخل الشجرة AVL الموضحة في شكل 24-11. كما سبق نقوم ببحث الشجرة بدءاً من العقدة الجذرية. الأسهم المنقطة توضح المسار الذي يتم اجتيازه. بعد ادخال 75 تكون شجرة البحث الثنائية كما هي موضحة في شكل 25-11.



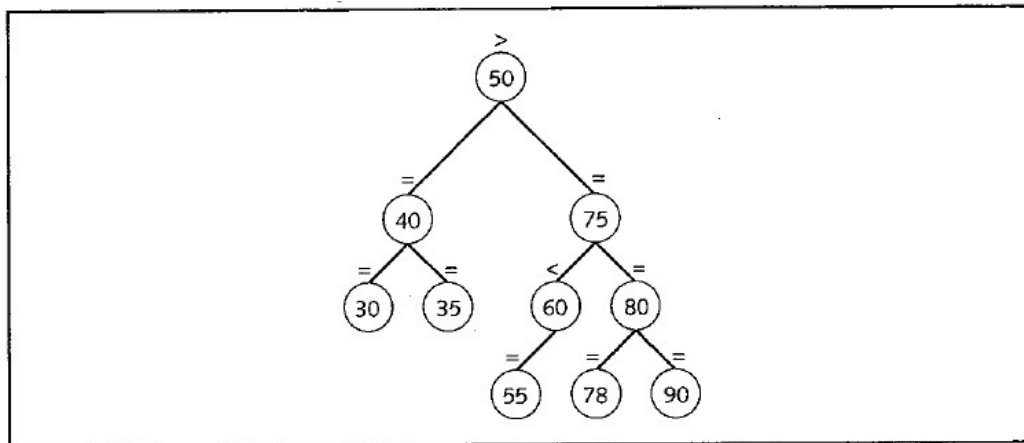
شكل 25-11: الشجرة الثنائية من شكل 24-11 قبل ادخال 75، العقد بخلاف 75 توضح عوامل اتزانها قبل الادخال

بعد ادخال 75 نقوم بالتتبع من الخلف. أولاً نذهب الى العقدة 80 ونقوم بتغيير عامل اتزانها الى -1. الشجرة الفرعية ذات العقدة الجذرية 80 عبارة عن شجرة AVL. الآن نعود الى 70. يتضح أن الشجرة الفرعية ذات العقدة الجذرية 70 ليست شجرة AVL. لهذا نعيد اقامة هذه الشجرة الفرعية وفي هذه الحالة نعيد أولاً اقامة الشجرة الفرعية عند العقدة الجذرية 80 ثم نعيد اقامة الشجرة الفرعية عند العقدة الجذرية 70 للحصول على شجرة البحث الثنائية كما هي موضحة في شكل 26-11. (هذه الاقامات أي الدورانات يتم توضيحها في القسم التالي "دورانات الشجرة AVL").



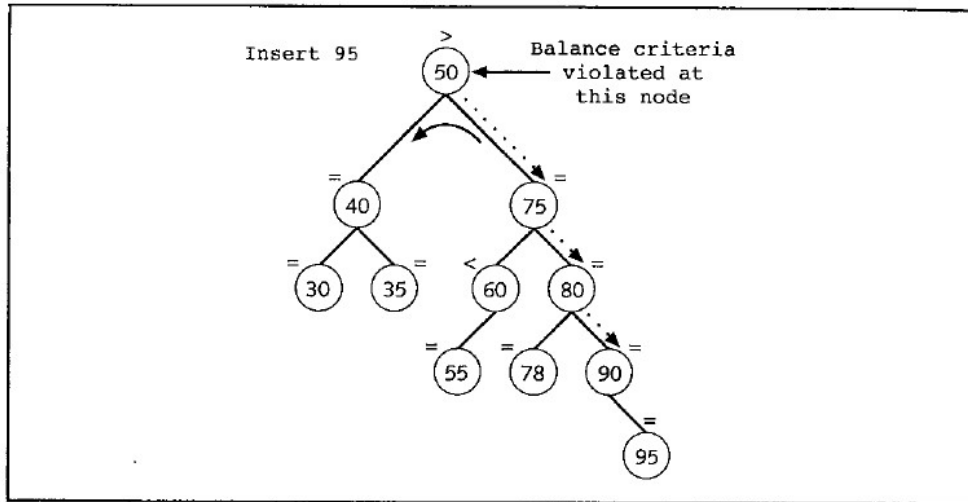
شكل 26-11: شجرة AVL من الشكل 24-11 بعد ادخال 75 وتعديل عوامل الاتزان

بعد اعادة البناء تكون العقدة الجذرية للشجرة الفرعية المقامة هي 75. لاحظ أن في الأشكال 23-11 و 26-11 بعد اعادة بناء الأشجار الفرعية عند العقد لا تعد الأشجار الفرعية تنمو في الارتفاع. عند هذه النقطة نقوم عادةً بإرسال رسالة تذكر أن الشجرة بأكملها لم تكتسب أي ارتفاع إلى العقد المتبقية على المسار العائد إلى العقدة الجذرية للشجرة. بهذا لا تحتاج العقد المتبقية على المسار إلى القيام بأي شيء. بعد هذا انظر إلى الشجرة AVL في شكل 27-11.



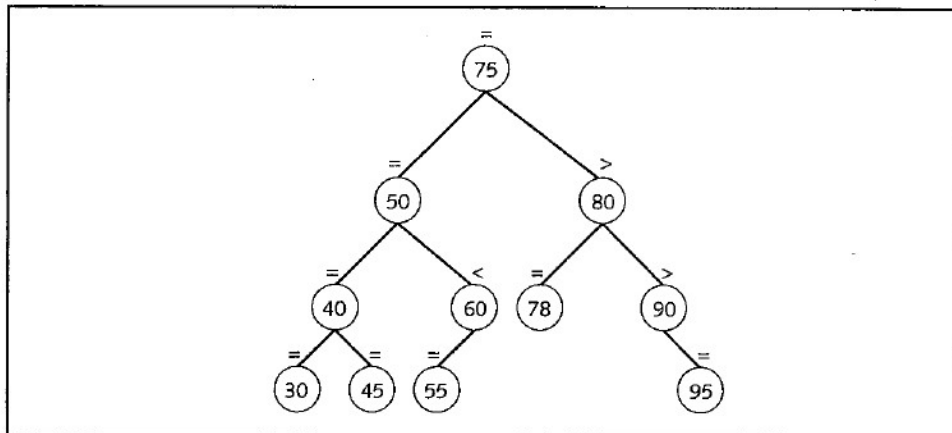
شكل 27-11: الشجرة AVL قبل ادخال 95

لنقم بادخال 95 داخل هذه الشجرة AVL. نقوم ببحث الشجرة وادخال 95 كما هو موضح في شكل 28-11.



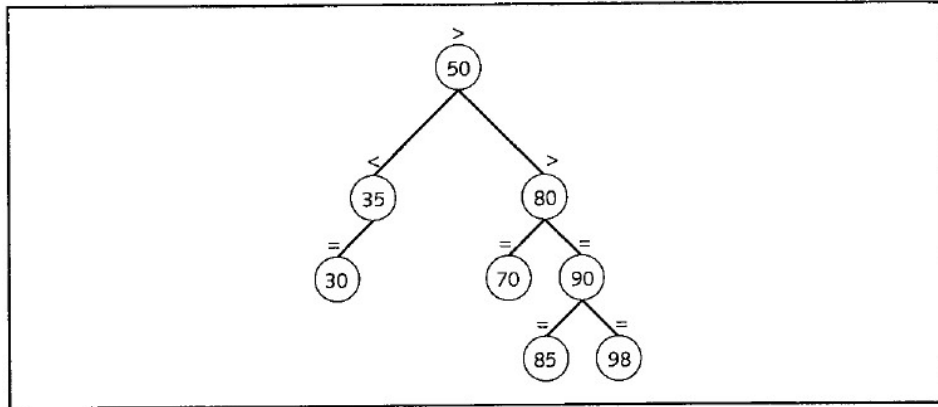
شكل 11-28: شجرة ثنائية من شكل 11-27 بعد ادخال 95، العقد بخلاف 95 توضح عوامل اتزانها قبل الادخال

بعد ادخال 95 نرى أن الأشجار الفرعية ذات العقد الجذرية 90 و 80 و 75 لاتزال أشجار AVL. عند تتبع المسار مرة أخرى نقوم ببساطة بتعديل عوامل اتزان هذه العقد (إذا لزم الأمر). بالرغم من هذا فإننا عند التتبع مرة أخرى الى العقدة الجذرية نكتشف أن الشجرة عند هذه العقدة لم تعد شجرة AVL. قبل الادخال كان عامل اتزان 50 يساوي 1 أي أن شجرتها الفرعية اليمنى كانت أعلى من شجرتها الفرعية اليسرى. بعد الادخال ازدادت الشجرة الفرعية في الارتفاع وبهذا تعدت على معيار الاتزان عند 50. لهذا نعيد بناء شجرة البحث الثنائية عند العقدة 50 وفي هذه الحالة يتم اعادة بناء الشجرة كما هو موضح في شكل 11-29.



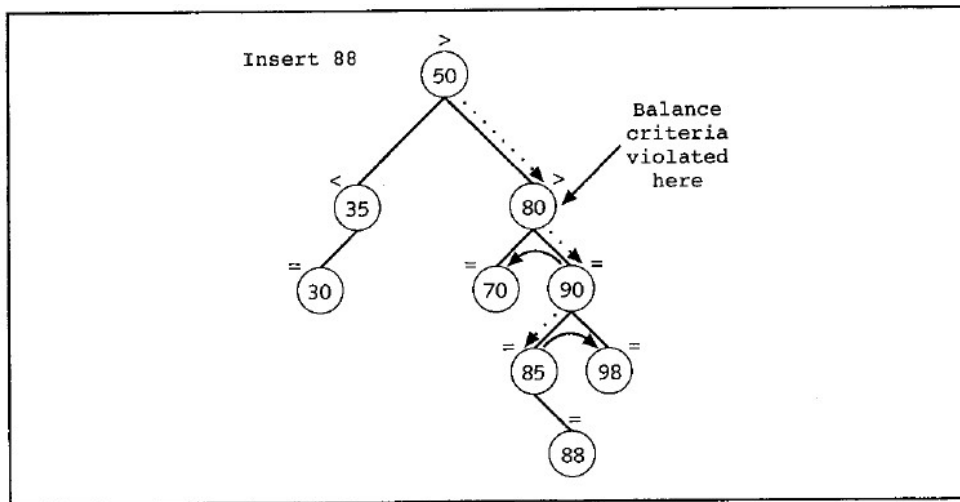
شكل 11-29: شجرة ثنائية من شكل 11-27 بعد ادخال 95 وتعديل عوامل الاتزان

قبل مناقشة الخوارزميات العامة لإعادة بناء (دوران) شجرة فرعية لنقم بالنظر الى حالة أخرى. انظر الى الشجرة AVL الموضحة في شكل 30-11.



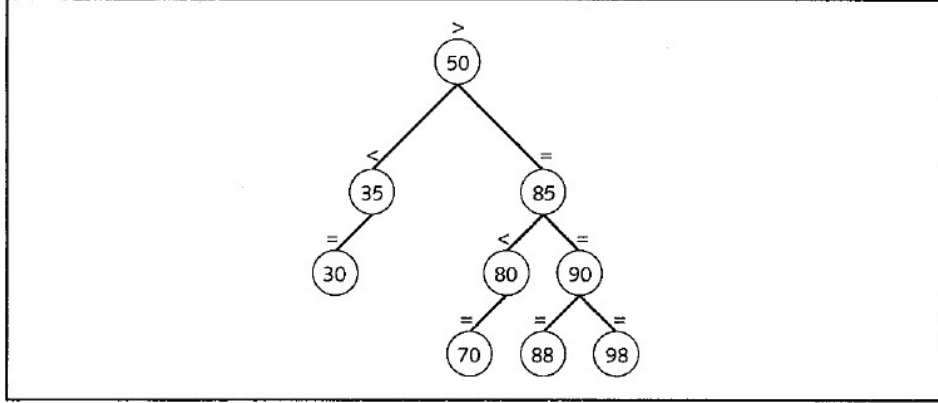
شكل 30-11: الشجرة AVL بعد ادخال 88

لنقم بادخال 88 داخل الشجرة في شكل 30-11. باتباع اجراءات الادخال التي تم توضيحها من قبل نحصل على شجرة البحث الثنائية الموضحة في شكل 31-11.



شكل 30-11: الشجرة الثنائية في شكل 30-11 بعد ادخال 88، العقد بخلاف 88 توضح عوامل اتزانها قبل الادخال

كما سبق نقوم الآن بالتتبع مرة أخرى إلى العقدة الجذرية. نقوم بتعديل عوامل اتزان العقد 85 و 90. عندما نزور العقدة 80 نكتشف أننا نحتاج عند هذه العقدة إلى إعادة بناء الشجرة الفرعية. في هذه الحالة يتم إعادة بناء الشجرة الفرعية كما هو موضح في شكل 11-32.



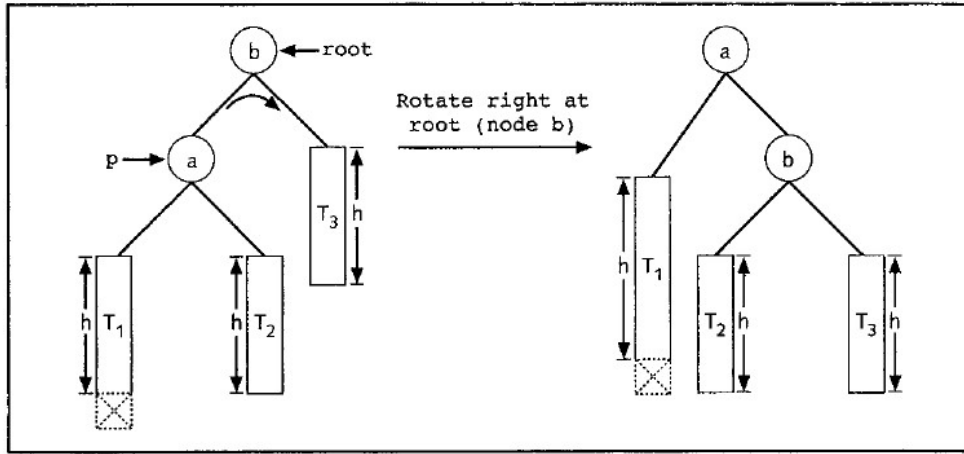
شكل 11-32: شجرة AVL من شكل 11-30 بعد ادخال 88 وتعديل عوامل الاتزان

كما سبق بعد إعادة بناء الشجرة الفرعية تكون الشجرة بأكملها متوازنة. لهذا لا نقوم بعمل أي شيء بالنسبة إلى العقد المتبقية على المسار العائد إلى العقدة الجذرية. الأمثلة السابقة تشير إلى أنه إذا احتاج جزء من شجرة البحث الثنائية إلى إعادة البناء إذن يمكننا بعد إعادة بناء هذا الجزء من شجرة البحث الثنائية أن نهمل العقد المتبقية على المسار العائد إلى العقدة الجذرية. (هذا هو الحال). كذلك بعد ادخال العقدة يمكن أن تحدث إعادة البناء عند أي عقدة على المسار العائد إلى العقدة الجذرية.

دورانات شجرة AVL:

نقوم الآن بتوضيح إجراءات إعادة البناء المسماة **تدوير** الشجرة. هناك نوعان من الدورانات هما **الدوران لليسار والدوران لليمين**. افترض أن الدوران يحدث عند العقدة x. إذا كان الدوران لليسار إذن هناك عقد معينة من الشجرة الفرعية اليمنى من x تنتقل إلى شجرتها الفرعية اليسرى ويصبح جذر الشجرة الفرعية اليمنى من x الجذر الجديد للشجرة الفرعية التي أعيد بنائها. بالمثل إذا كان الدوران عند x لليمين إذن هناك عقد معينة من الشجرة الفرعية اليسرى من x تنتقل إلى شجرتها الفرعية اليمنى ويصبح جذر الشجرة الفرعية اليسرى من x الجذر الجديد للشجرة الفرعية التي أعيد بنائها.

الحالة 1: انظر إلى الشكل 11-33.



شكل 11-33: دوران لليمين عند b

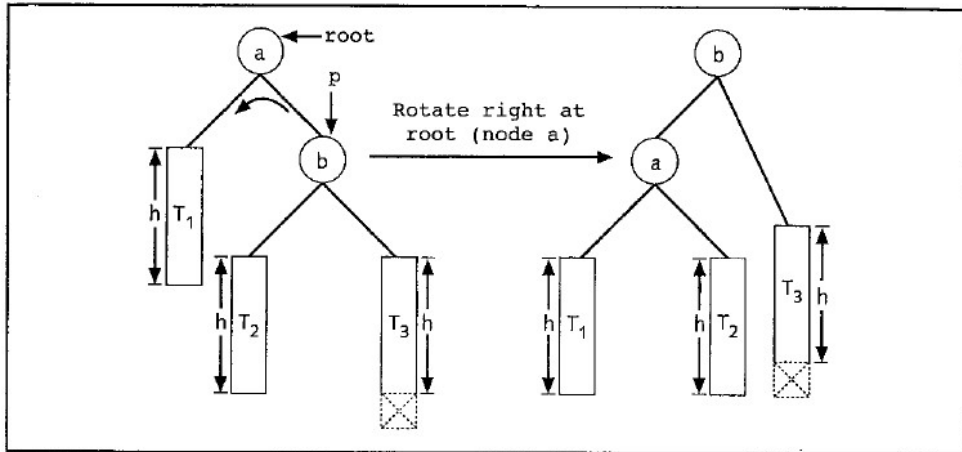
في شكل 11-33 يكون ارتفاع الأشجار الفرعية T_1 و T_2 و T_3 متساوي وليكن h . المستطيل المنقط يوضح ادخال عنصر في T_1 مسبباً زيادة ارتفاع الشجرة الفرعية T_1 بمقدار 1. الشجرة الفرعية عند العقدة a لاتزال شجرة AVL ولكن تم التعدي على معيار الاتزان عند العقدة الجذرية. نلاحظ ما يلي في هذه الشجرة. بما أن الشجرة عبارة عن شجرة بحث ثنائية اذن:

- كل مفتاح في T_1 أصغر من المفتاح في العقدة a.
- كل مفتاح في T_2 أكبر من المفتاح في العقدة a.
- كل مفتاح في T_2 أصغر من المفتاح في العقدة b.

لهذا:

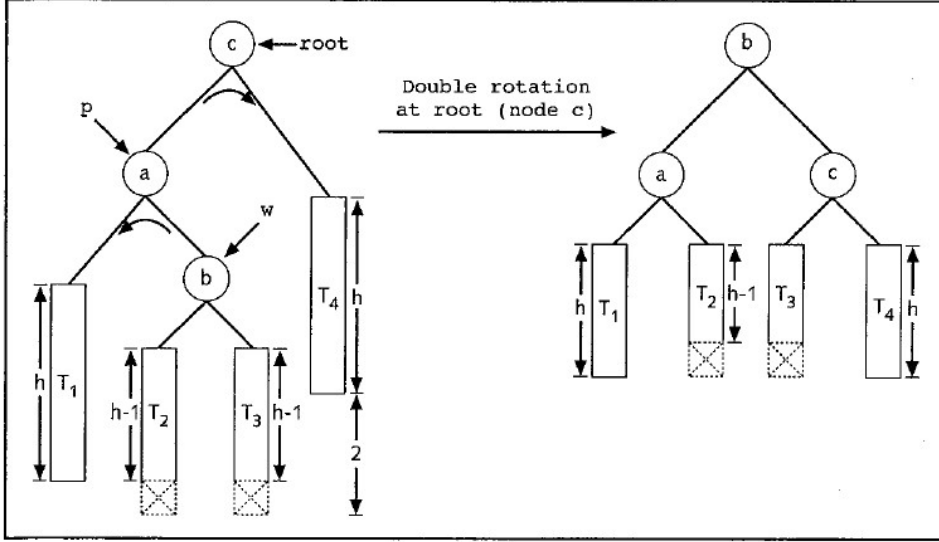
1. نجعل T_2 (الشجرة الفرعية اليمنى للعقدة a) الشجرة الفرعية اليسرى للعقدة b.
2. نجعل العقدة b الثمرة اليمنى للعقدة a.
3. العقدة a تصبح العقدة الجذرية للشجرة التي أعيد بنائها كما هو موضح في شكل 11-33.

الحالة 2: هذه الحالة تعتبر صورة طبق الأصل من الحالة 1. انظر شكل 11-34.



شكل 11-34: دوران اليسار عند a.

الحالة 3: انظر الى الشكل 11-35.

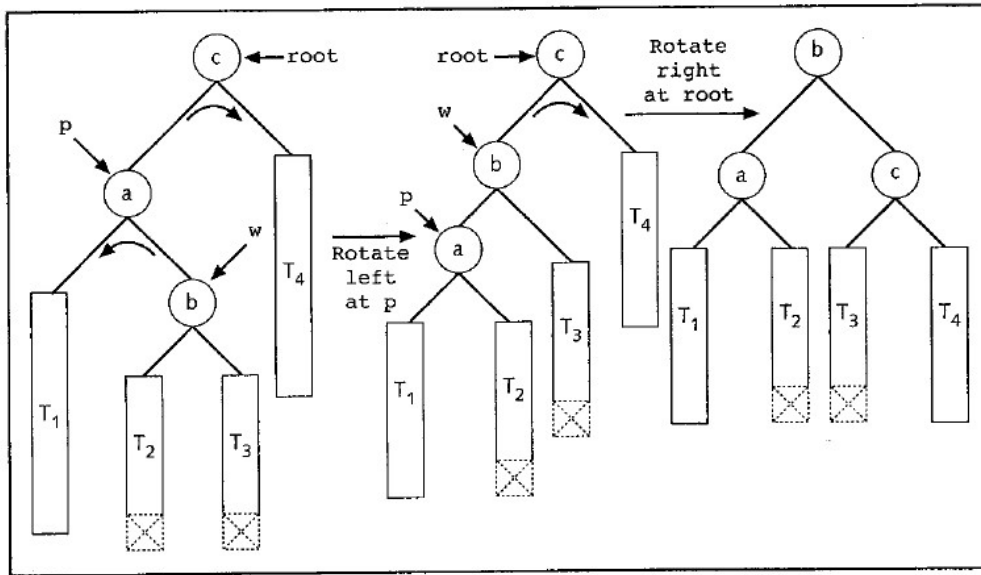


شكل 11-35: دوران مزدوج: أولاً دوران اليسار عند a ثم دوران اليمين عند c.

في شكل 11-35 تكون الشجرة الواقعة على اليسار هي الشجرة الموجود قبل اعادة البناء. أطوال الأشجار الفرعية موضحة في الشكل. المستطيل المنقط يوضح أنه يتم ادخال عنصر جديد في الشجرة الفرعية T2 أو T3 مسبباً في نمو الشجرة الفرعية في الارتفاع. نلاحظ ما يلي (في الشجرة السابقة لاعادة البناء):

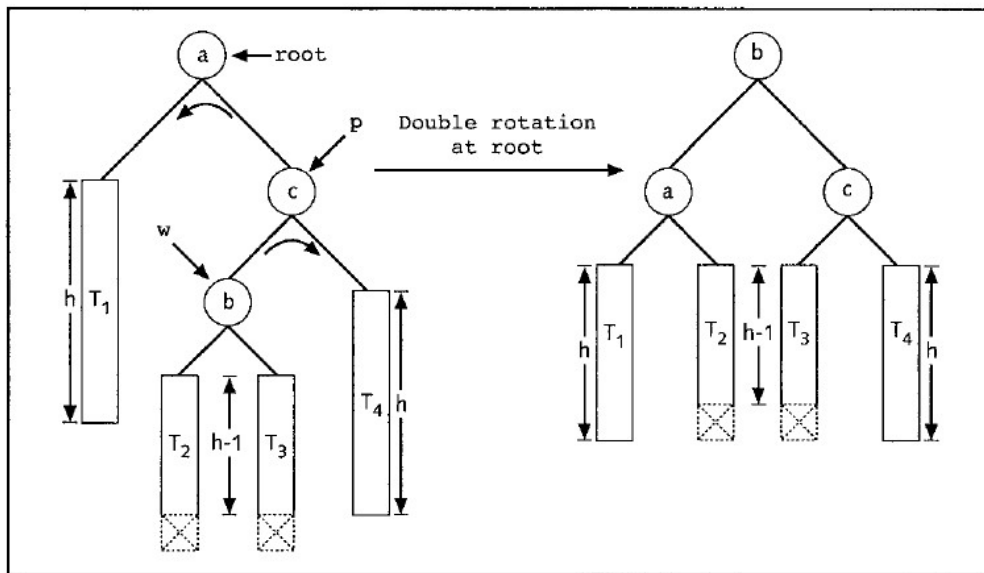
- جميع المفاتيح في T3 أصغر من المفتاح في العقدة c.
 - جميع المفاتيح في T3 أكبر من المفتاح في العقدة b.
 - جميع المفاتيح في T2 أصغر من المفتاح في العقدة b.
 - جميع المفاتيح في T2 أكبر من المفتاح في العقدة a.
 - بعد الادخال لا تزال الأشجار الفرعية ذات العقد الجذرية a و b أشجار AVL.
 - يتم انتهاك معيار الاتزان عند العقدة الجذرية c للشجرة.
 - عامل اتزان العقدة c يساوي -1 وعامل اتزان العقدة a يساوي 1 أي أنهما متضادين.
- هذا مثال على الدوران المزدوج. هناك حاجة الى دوران واحد عند العقدة a ودوران آخر لازم عند العقدة c. اذا كان عامل اتزان العقدة التي يتم عندها اعادة بناء الشجرة وعامل اتزان الشجرة الفرعية الأعلى متضادين فان هذه العقدة تحتاج الى دوران مزدوج. أولاً نقوم بلف الشجرة عند العقدة a وبعد هذا عند العقدة c. الآن الشجرة عند العقدة a مرتفعة لليمين ولهذا نقوم بعمل دوران اليسار عند a. بعد هذا وبما أن الشجرة عند العقدة c مرتفعة لليساار فاننا نقوم بعمل دوران اليمين عند c. الشكل 11-35

يوضح الشجرة الناتجة (الواقعة على يمين الشجرة بعد الإدخال). الشكل 11-36 بالرغم من هذا يوضح كلا من الدورانين بالتتابع.



شكل 11-36: دوران لليساار عند العقدة a يليه دوران لليمين عند c

الحالة 4: هذه صورة طبق الأصل من الحالة 3. نقوم بتوضيح هذا بمساعدة الشكل 11-37.



شكل 11-37: دوران مزدوج: أولاً دوران لليمين عند c ثم دوران لليساار عند a

باستخدام تلك الحالات الأربع نقوم الآن بوصف نوع الدوران اللازم عند العقدة.

افترض أن الشجرة التي يتم إعادة بنائها بالدوران عند العقدة x . اذن الشجرة الفرعية ذات العقدة الجذرية x تحتاج الى اما دوران واحد أو الى دوران مزدوج.

1. افترض أن عامل اتزان العقدة x وعامل اتزان العقدة الجذرية للشجرة الفرعية الأعلى لـ x لهما نفس الإشارة أي أن كليهما موجب أم سالب.

أ- اذا كانت عوامل الاتزان هذه موجبة قم بعمل دوران واحد الى اليسار عند x . (قبل الادخال كانت الشجرة الفرعية اليمنى لـ x أعلى من شجرتها الفرعية اليسرى. تم ادخال العنصر الجديد في الشجرة الفرعية اليمنى لـ x متسببة في زيادة ارتفاع الشجرة الفرعية اليمنى وهذا بدوره تعدى على معيار الاتزان عند x).

ب- اذا كانت عوامل الاتزان هذه سالبة قم بعمل دوران واحد الى اليمين عند x . (قبل الادخال كانت الشجرة الفرعية اليسرى لـ x أعلى من شجرتها الفرعية اليمنى. تم ادخال العنصر الجديد في الشجرة الفرعية اليسرى لـ x متسببة في زيادة ارتفاع الشجرة الفرعية اليسرى وهذا بدوره تعدى على معيار الاتزان عند x).

2. افترض أن عامل اتزان العقدة x وعامل اتزان الشجرة الفرعية الأعلى لـ x لهما اشارات عكسية. لكي نكون محددين افترض أن عامل اتزان العقدة x قبل الادخال كان -1 وافترض أن y هي العقدة الجذرية للشجرة الفرعية اليسرى من x . بعد الادخال يكون عامل اتزان العقدة y يساوي 1. هذا يعني ان بعد الادخال ازدادت الشجرة الفرعية اليمنى للعقدة y في الارتفاع. في هذه الحالة نحتاج الى دوران مزدوج عند x . أولاً نقوم بعمل دوران الى اليسار عند y (لأن y مرتفعة عند اليمين) ثم نقوم بعمل دوران الى اليمين عند x . الحالة الأخرى التي هي صورة طبق الأصل من هذه الحالة يتم التعامل معها بالمثل.

دالات C++ التالية تقوم بتطبيق دورانات العقدة الى اليسار والى اليمين. مؤشر العقدة الذي يحتاج الدوران يتم تمريره كمعامل للدالة.

```

template<class elemType>
void rotateToLeft(AVLNode<elemType>* &root)
{
    AVLNode<elemType> *p; //pointer to the root of the
                          //right subtree of root

    if(root == NULL)
        cerr<<"Error in the tree."<<endl;
    else
        if(root->rlink == NULL)
            cerr<<"Error in the tree:"
                <<" No right subtree to rotate."<<endl;
        else
        {
            p = root->rlink;
            root->rlink = p->llink; //the left subtree of p
                                  //becomes the right subtree of root
            p->llink = root;
            root = p; //make p the new root node
        }
} //end rotateToLeft

template<class elemType>
void rotateToRight(AVLNode<elemType>* &root)
{
    AVLNode<elemType> *p; //pointer to the root of the
                          //left subtree of root

    if(root == NULL)
        cerr<<"Error in the tree."<<endl;
    else
        if(root->llink == NULL)
            cerr<<"Error in the tree:"
                <<" No left subtree to rotate."<<endl;

        else
        {
            p = root->llink;
            root->llink = p->rlink; //the right subtree of p
                                  //becomes the left subtree of root
            p->rlink = root;
            root = p; //make p the new root node
        }
} //end rotateToRight

```

الآن بعدما عرفنا كيفية تطبيق كلا الدورانين نقوم بعد هذا بكتابة دالات C++ وهي
 balanceFromLeft (اتزان من اليسار) و balanceFromRight (اتزان من اليمين) اللتين يتم
 استخدامهما لاعادة بناء الشجرة عند عقدة محددة. يتم تمرير مؤشر العقدة حيث تحدث اعادة البناء
 كمعامل لهذه الدالة. تقوم هذه الدالات باستخدام الدالات rotateToLeft (دوران لليسار) و
 rotateToRight (دوران لليمين) لاعادة بناء الشجرة وكذلك تعديل عوامل اتزان العقد التي تأثرت
 باعادة البناء. الدالة balanceFromLeft يتم استدعاءها عندما تكون الشجرة الفرعية مزدوجة

الارتفاع من اليسار وعند الحاجة الى نقل عقدة معينة الى الشجرة الفرعية اليمنى. الدالة
balanceFromRight لها تقاليد مماثلة.

```
template<class elemType>
void balanceFromLeft(AVLNode<elemType>* &root)
{
    AVLNode<elemType> *p;
    AVLNode<elemType> *w;

    p = root->llink;    //p points to the left subtree of root

    switch(p->bfactor)
    {
    case -1: root->bfactor = 0;
             p->bfactor = 0;
             rotateToRight(root);
             break;
    case 0:  cerr<<"Error: Cannot balance from the left."<<endl;
             break;
    case 1:  w = p->rlink;
             switch(w->bfactor)    //adjust the balance factors
             {
             case -1: root->bfactor = 1;
                      p->bfactor = 0;
                      break;
             case 0:  root->bfactor = 0;
                      p->bfactor = 0;
                      break;
             case 1:  root->bfactor = 0;
                      p->bfactor = -1;
             }
             //end switch

             w->bfactor = 0;
             rotateToLeft(p);
             root->llink = p;
             rotateToRight(root);
    }
    //end switch;
}
//end balanceFromLeft
```

من أجل الكمال نقوم كذلك باعطاء تعريف الدالة balanceFromRight.

```

template<class elemType>
void balanceFromRight(AVLNode<elemType>* &root)
{
    AVLNode<elemType> *p;
    AVLNode<elemType> *w;

    p = root->rlink;    //p points to the right subtree of root

    switch(p->bfactor)
    {
    case -1: w = p->llink;
        switch(w->bfactor)    //adjust the balance factors
        {
        case -1: root->bfactor = 0;
            p->bfactor = 1;
            break;
        case 0: root->bfactor = 0;
            p->bfactor = 0;
            break;
        case 1: root->bfactor = -1;
            p->bfactor = 0;
        }//end switch

        w->bfactor = 0;
        rotateToRight(p);
        root->rlink = p;
        rotateToLeft(root);
        break;
    case 0: cerr<<"Error: Cannot balance from the right."<<endl;
        break;
    case 1: root->bfactor = 0;
        p->bfactor = 0;
        rotateToLeft(root);
    }//end switch;
} //end balanceFromRight

```

نقوم الآن بتركيز انتباهنا على الدالة insertIntoAVL. الدالة insertIntoAVL تقوم بادخال عنصر جديد في شجرة AVL. العنصر الذي يتم ادخاله ومؤشر العقدة الجذرية للشجرة AVL يتم تمريرهما كمعاملات لهذه الدالة.

الخطوات التالية تصف الدالة insertIntoAVL:

1. عمل عقدة ونسخ العنصر المراد ادخاله في العقدة التي تم عملها حديثاً .
 2. بحث الشجرة والعثور على مكان من أجل العقدة الجديدة في الشجرة.
 3. ادخال العنصر الجديد في الشجرة.
 4. تتبع المسار من الخلف الذي تم بنائه لايجاد مكان من أجل العقدة الجديدة في الشجرة الى العقدة الجذرية. اذا لزم الأمر تعديل عوامل اتزان العقد أو اعادة بناء الشجرة عند عقدة على المسار.
- بما أن الخطوة رقم 4 تقتضي منا تتبع المسار من الخلف الى العقدة الجذرية وبما اننا في الشجرة الثنائية يكون لدينا وصلات فقط من الأصل الى الثمار فان الطريقة الأسهل لتطبيق الدالة

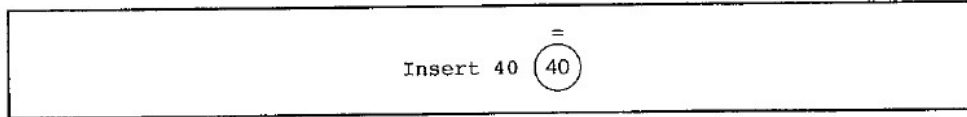
insertIntoAVL هي استخدام التكرار. (تذكر أن التكرار يقوم تلقائياً بالاعتناء بالتتابع الخلفي.) هذا هو بالضبط ما نقوم به. كما أن الدالة insertIntoAVL تستخدم معامل إشارة bool وهو isTaller للإشارة إلى الأصل سواء كانت الشجرة الفرعية نمت في الارتفاع أم لا.

```
template<class elemType>
void insertIntoAVL(AVLNode<elemType>* &root,
                  AVLNode<elemType> *newNode,
                  bool& isTaller)
{
    if(root == NULL)
    {
        root = newNode;
        isTaller = true;
    }
    else
    {
        if(root->info == newNode->info)
            cerr<<"No duplicates are allowed."<<endl;
        else
        {
            if(root->info > newNode->info) //newItem goes in
                                           //the left subtree
            {
                insertIntoAVL(root->llink, newNode, isTaller);

                if(isTaller)                //after insertion, the
                                           //subtree grew in height
                {
                    switch(root->bfactor)
                    {
                        case -1: balanceFromLeft(root);
                                isTaller = false;
                                break;
                        case 0:  root->bfactor = -1;
                                isTaller = true;
                                break;
                        case 1:  root->bfactor = 0;
                                isTaller = false;
                    }
                }
            }
            else
            {
                insertIntoAVL(root->rlink, newNode, isTaller);

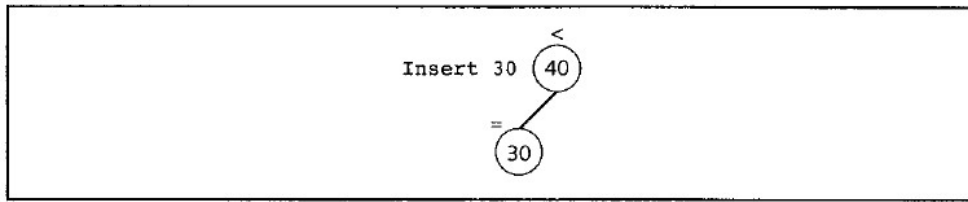
                if(isTaller)                //after insertion, the
                                           //subtree grew in height
                {
                    switch(root->bfactor)
                    {
                        case -1: root->bfactor = 0;
                                isTaller = false;
                                break;
                        case 0:  root->bfactor = 1;
                                isTaller = true;
                                break;
                        case 1:  balanceFromRight(root);
                                isTaller = false;
                    }
                }
            }
        }
    }
}
//end insertIntoAVL
```

بعد هذا نوضح كيفية عمل الدالة insertIntoAVL وكيفية بناء شجرة AVL من مجرد رسم مبدئياً تكون الشجرة فارغة. كل شكل (الأشكال من 11-37 وحتى 11-45) يوضح العنصر المراد حذفه وكذلك عامل اتزان كل عقدة. تشير علامة يساوي (=) فوق العقدة الى أن عامل اتزان هذه العقدة يساوي صفر ورمز أصغر من (<) يشير الى أن عامل اتزان هذه العقدة يساوي -1 ويشير رمز أكبر من (>) الى أن عامل اتزان هذه العقدة يساوي 1. مبدئياً تكون شجرة AVL فارغة. لنقم بادخال 40 في شجرة AVL الفارغة. انظر شكل 11-38.



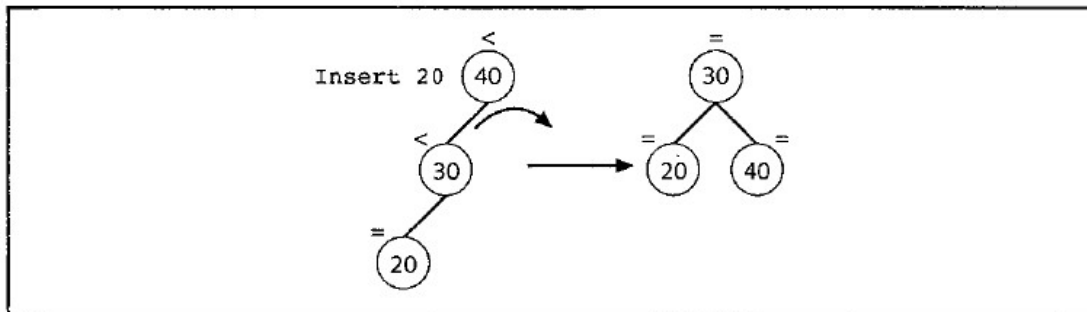
شكل 11-38: شجرة AVL بعد ادخال 40

بعد هذا نقوم بادخال 30 في شجرة AVL. انظر شكل 11-39.



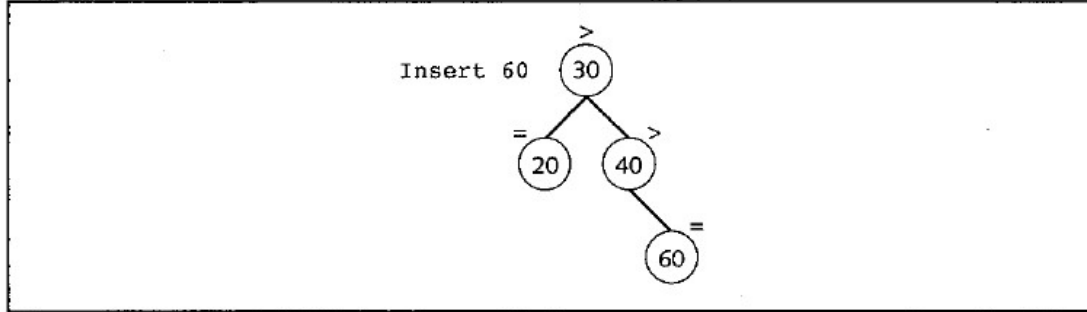
شكل 11-39: الشجرة AVL بعد ادخال 30

يتم ادخال العنصر 30 في الشجرة الفرعية اليسرى للعقدة 40 متسببة في نمو الشجرة الفرعية اليسرى للعقدة 40 في الارتفاع. بعد الادخال يكون عامل اتزان العقدة 40 هو -1. بعد هذا نقوم بادخال 20 الى شجرة AVL. انظر شكل 11-40.



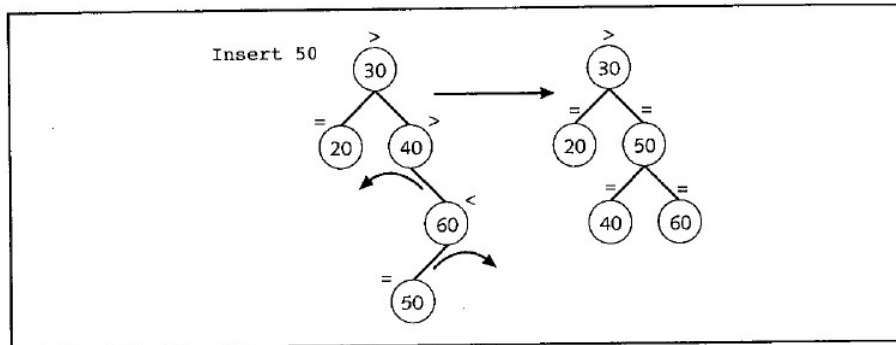
شكل 11-40: شجرة AVL بعد ادخال 20

ادخال 20 يتعدى على معيار الاتزان عند العقدة 40. تتم اعادة بناء الشجرة عند العقدة 40 عن طريق عمل دوران واحد الى اليمين. بعد هذا نقوم بادخال 60 الى شجرة AVL. انظر شكل 41-11.



شكل 41-11: شجرة AVL بعد ادخال 60

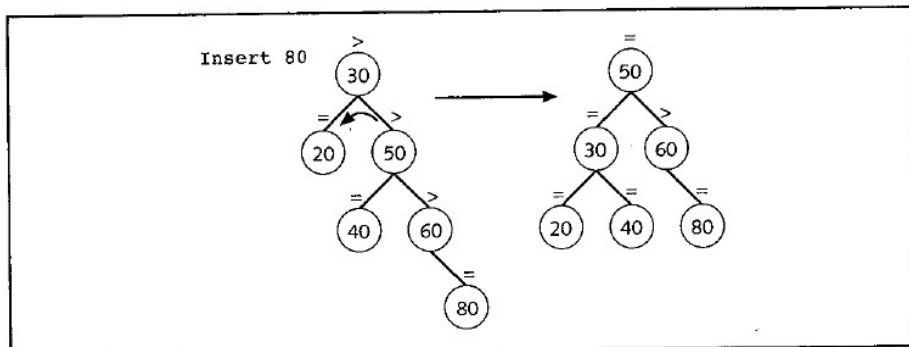
ادخال 60 لا يتطلب اعادة البناء فقط يتم تعديل عامل الاتزان عند العقدتين 40 و 30. بعد هذا نقوم بادخال 50. انظر شكل 42-11.



شكل 42-11: شجرة AVL بعد ادخال 50

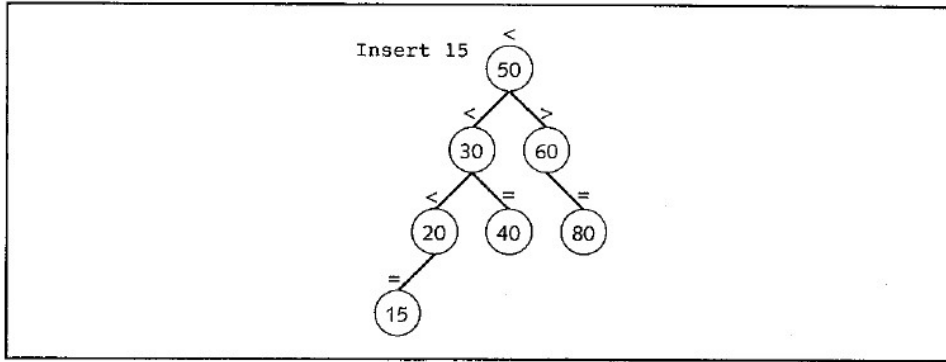
ادخال 50 يتطلب أن تتم اعادة بناء الشجرة عند 40. لاحظ أن الدوران المزدوج تم عمله عند العقدة 40.

بعد هذا نقوم بادخال 80. انظر شكل 43-11.



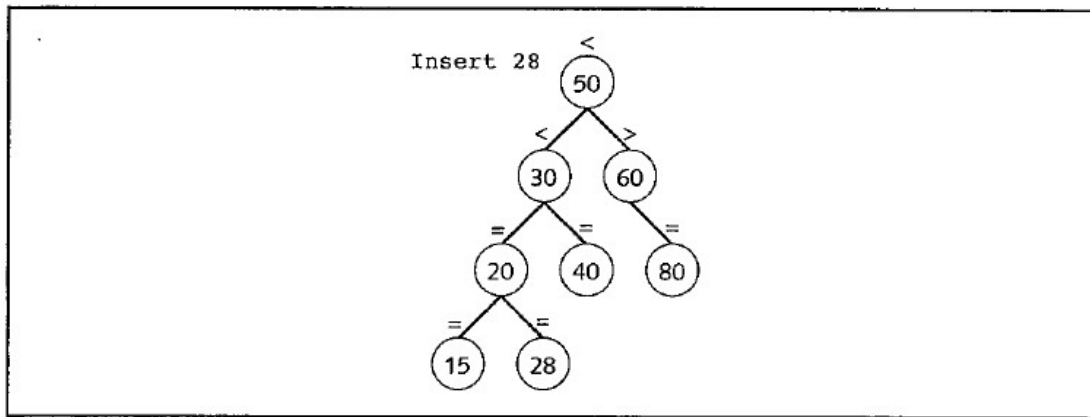
شكل 11-43: شجرة AVL بعد ادخال 80

ادخال 80 يتطلب أن تتم اعادة بناء الشجرة عند العقدة 30. بعد هذا نقوم بادخال 15. انظر شكل 11-44.



شكل 11-44: شجرة AVL بعد ادخال 15

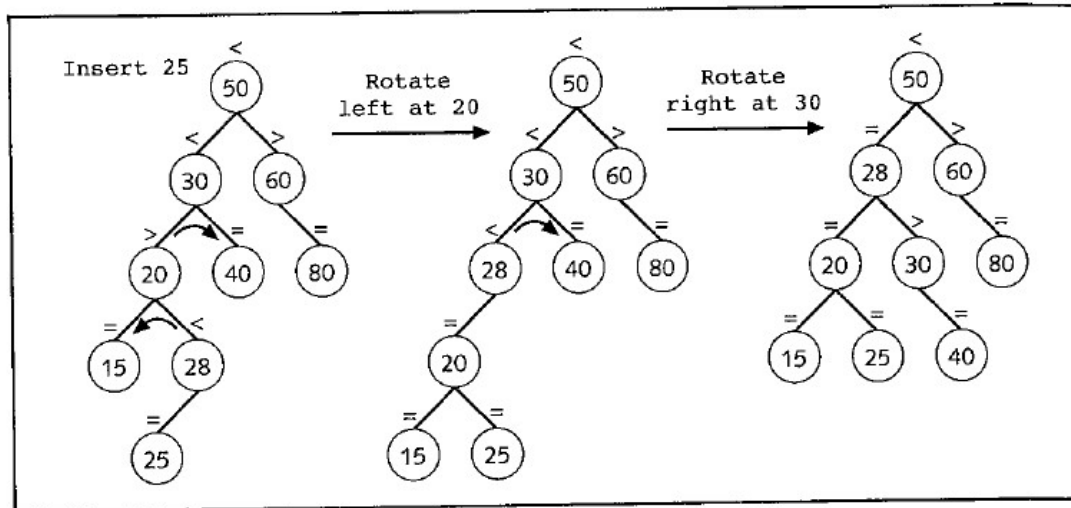
ادخال العقدة 15 لا يتطلب اعادة بناء أي جزء من الشجرة بل نحتاج فقط الى تعديل عوامل اتزان العقد 20 و 30 و 50. بعد هذا نقوم بادخال 28. انظر شكل 11-45.



شكل 11-45: شجرة AVL بعد ادخال 28

ادخال العقدة 28 لا يتطلب اعادة بناء أي جزء من الشجرة بل نحتاج فقط الى تعديل عوامل اتزان العقد 20 و 30 و 50.

بعد هذا نقوم بادخال 25. ادخال 25 يحتاج الى دوران مزدوج عند العقدة 30. الشكل 11-46 يوضح كلا الدورانين بالتتابع.



شكل 11-46: شجرة AVL بعد ادخال 25

في شكل 11-46 تم دوران الشجرة أولاً الى اليسار عند العقدة 20 ثم دورانها الى اليمين عند العقدة 30.

الدالة التالية تقوم بعمل عقدة وتخزين البيان في العقدة وتستدعي الدالة insertIntoAVL لادخال العقدة الجديدة في الشجرة AVL:

```
template<class elemType>
void insert(const elemType &newItem)
{
    bool isTaller = false;
    AVLNode<elemType> *newNode;

    newNode = new AVLNode<elemType>;
    newNode->info = newItem;
    newNode->bfactor = 0;
    newNode->llink = NULL;
    newNode->rlink = NULL;

    insertIntoAVL(root, newNode, isTaller);
}
```

نترك لك كتمرين تصميم الفئة لتطبيق الشجرات AVL كنوع بيانات مجرد. انزر تمرين البرمجة رقم 9 في نهاية هذا الفصل. (لاحظ أنه بما أن تركيب عقدة الشجرة AVL مختلف عن تركيب عقدة الشجرة الثنائية التي نوقشت في بداية هذا الفصل فانه لا يمكنك استخدام التوريث لاستمداد الفئة لتطبيق الشجرات AVL من الفئة binaryTreeType).

الحذف من أشجار AVL:

لحذف عنصر من شجرة AVL نقوم أولاً بإيجاد العقدة المحتوية على العنصر المراد حذفه. الحالات الأربع التالية تحدث:

الحالة 1: العقدة التي يتم حذفها عبارة عن ورقة.

الحالة 2: العقدة التي يتم حذفها ليس لها ثمرة اليمنى أي أن شجرتها الفرعية اليمنى فارغة.

الحالة 3: العقدة التي يتم حذفها ليس لها ثمرة يسرى أي أن شجرتها الفرعية اليسرى فارغة.

الحالة 4: العقدة التي يتم حذفها لها ثمرة يسرى وثمرتها اليمنى.

من الأسهل التعامل مع الحالات من رقم 1 إلى رقم 3 من الحالة رقم 4. لنقم أولاً بمناقشة الحالة رقم 4.

افترض أن العقدة التي يتم حذفها ولتكن x لها ثمرة يسرى وثمرتها اليمنى. كما في حالة الحذف من شجرة بحث ثنائية نقوم باختزال الحالة 4 إلى الحالة 2 أي أننا نوجد ما يسبقه مباشرةً وليكن y من x. بعد هذا يتم نسخ بيانات y بداخل x وبهذا تكون y هي العقدة المراد حذفها. بشكل واضح y ليس لها ثمرة اليمنى.

لحذف العقدة نقوم بتعديل واحد من مؤشرات العقدة الأصلية. بعد حذف العقدة قد لا تعد الشجرة الناتجة شجرة AVL. كما في حالة الإدخال في شجرة AVL نقوم باجتياز المسار (من العقدة الأصلية) إلى العقدة الجذرية. بالنسبة إلى كل عقدة على هذا المسار نحتاج في بعض الأحيان إلى تغيير عامل الاتزان فقط بينما في أوقات أخرى تتم إعادة بناء الشجرة عند عقدة محددة. الخطوات التالية تصف ما يجب فعله عند العقدة على المسار العائد إلى العقدة الجذرية. (كما في حالة الإدخال نقوم باستخدام المتغير shorter للإشارة إلى ما إذا تم تقليل ارتفاع الشجرة الفرعية). لتكن p عقدة على المسار العائد إلى العقدة الجذرية. ننظر إلى عامل الاتزان الحالي لـ p.

1. إذا كان عامل الاتزان p الحالي يساوي مرتفع اذن يتم تغيير عامل اتزان p الحالي وفقاً لما إذا

كان تم تقصير الشجرة الفرعية اليسرى لـ p أو ما إذا كان تم تقصير الشجرة الفرعية اليمنى

لـ p. يتم ضبط المتغير shorter عند false.

2. افترض أن عامل اتزان p غير متساوي ويتم تقصير الشجرة الفرعية الأطول من p. يتم تغيير

عامل اتزان p إلى يساوي مرتفع ويتم ترك المتغير shorter عند true.

3. افترض أن عامل اتزان p غير متساوي مرتفع ويتم تقصير الشجرة الفرعية الأقصر لـ p

وافترض أن q يشير إلى جذر شجرة p الفرعية الأطول.

أ- إذا كان عامل اتزان q متساوي مرتفع يلزم دوران واحد عند p ويتم ضبط shorter

عند false.

ب- إذا كان عامل اتزان q هو نفسه عامل اتزان p يلزم دوران واحد عند p ويتم ضبط

shorter عند true.

ت- افترض أن عوامل اتزان p و q متضادة يلزم دوران مزدوج عند p (دوران واحد عند q ثم دوران واحد عند p). نقوم بتعديل عوامل الاتزان ونضبط shorter عند true.

تحليل: أشجار AVL:

قم بالتفكير في جميع أشجار AVL الممكنة ذات الارتفاع h. لتكن T_h AVL طولها h حيث بها أقل عدد T_h العقد. لتكن T_{hl} جرة الفرعية اليسرى و T_{hr} جرة الفرعية اليمنى. اذن:

$$|T_h| = |T_{hl}| + |T_{hr}| + 1$$

بما أن T_h عن شجرة AVL طولها h حيث T_{hl} نل عدد من العقد اذن ينتج أن واحدة من شجرتي T_h الفرعيتين ارتفاعها h-1 والأخرى ارتفاعها h-2. لكي نكون محددين افترض أن ارتفاعها h-1 T_{hl} ارتفاعها h-2 T_{hr} تعريف T_{hr} ينتج أن شجرة T_{hl} AVL ارتفاعها h-1 T_{hr} حيث T_{hl} تمتلك أقل عدد من العقد ضمن جميع أشجار AVL التي ارتفاعها h-1. بالمثل T_{hr} عبارة عن شجرة ارتفاعها h-2 وبها أقل عدد من العقد ضمن جميع أشجار AVL التي ارتفاعها h-2. بهذا تكون T_{hl} بالصيغة وتكون بالصيغة T_{hr} T_{h-1} هنا:

$$|T_h| = |T_{h-1}| + |T_{h-2}| + 1$$

بوضوح:

$$|T_0| = 1$$

$$|T_1| = 2$$

لتكن

$$F_{h+2} = |T_h| + 1$$

اذن:

$$F_{h+2} = F_{h+1} + F_h$$

$$F_2 = 2$$

$$F_3 = 3.$$

هذا يسمى تتابع فيبوناكسي. حل F_h معطى بواسطة:

$$\phi = \frac{1+\sqrt{5}}{2}, \quad F_h \approx \frac{\phi^h}{\sqrt{5}}$$

من هنا:

$$|T_h| \approx \frac{\phi^{h+2}}{\sqrt{5}} = \frac{1}{\sqrt{5}} \left[\frac{1+\sqrt{5}}{2} \right]^{h+2}$$

من هذا يمكن استنتاج أن:

$$h \approx (1.44) \log_2 |T_h|$$

هذا يشير ضمناً الى أن ارتفاع الشجرة AVL ذات العدد n من العقد يكون في أسوأ الحالات $(1.44) \log_2 n$. بما أن ارتفاع الشجرة الثنائية الكاملة الاتزان وذات العدد n من العقد يبلغ $\log_2 n$ اذن ينتج أنه في أسوأ حالة لايزيد زمن التحكم في شجرة AVL عن 44% من الزمن الكلي. بالرغم من هذا فان أشجار AVL بوجه عام ليست متناثرة كما هي في أسوأ حالة. يمكن ايضاح أن متوسط زمن البحث في شجرة AVL يكون حوالي 4% أكثر مما هو أفضل.

مثال برمجة: متجر الفيديو (مرة أخرى):

قمنا في الفصل 5 بتصميم برنامج لمساعدة متجر فيديو على جعل عملية تأجير الشرائط آلية. قام هذا البرنامج باستخدام قائمة متصلة (غير مرتبة) لتتبع الشرائط في المتجر. بما أن خوارزمية البحث على قائمة متصلة تكون تتابعية وبما أن القائمة كبيرة فان البحث قد يكون مضيعاً للوقت. في هذا الفصل تعلمت كيفية تنظيم البيانات داخل شجرة ثنائية. اذا تم بناء الشجرة الثنائية جيداً (أي لم تكن طولية) اذن يمكن تحسين خوارزمية البحث بقدر معقول. فضلاً عن هذا وبوجه عام يكون ادخال عنصر وحذف عنصر من شجرة بحث ثنائية أسرع منه في قائمة متصلة. لهذا سوف نعيد تصميم برنامج متجر الفيديو حتى يمكن الاحتفاظ بمخزن الفيديو في شجرة ثنائية. كما حدث في الفصل 5 نترك لك كتمرين تصميم قائمة العملاء في شجرة ثنائية.

هدف الفيديو:

في الفصل 5 تم استخدام قائمة متصلة للاحتفاظ بقائمة الشرائط الموجودة في المتجر. بما أن القائمة المتصلة كانت غير مرتبة ومن أجل رؤية ما اذا كان شريط معين موجود في المخزن فقد قامت خوارزمية البحث التتبعي باستخدام عامل التساوي من أجل المقارنات. بالرغم من هذا ففي حالة الشجرة الثنائية نحتاج الى عوامل مترابطة أخرى من أجل عمليات البحث والادخال والحذف. لهذا نقوم باثقال جميع العوامل المترابطة. بخلاف هذا الاختلاف تكون الفئة videoType هي نفسها كما سبق. بالرغم من هذا فاننا نعطي تعريفها دون التوثيق هنا من أجل سهولة الاشارة ومن أجل الكمال.

```

#include <iostream>
#include <string>

using namespace std;

class videoType
{
    friend ostream& operator<<(ostream&, const videoType&);
public:
    void setVideoInfo(string title, string star1,
                      string star2, string producer,
                      string director, string productionCo,
                      int setInStock);
    int getNoOfCopiesInStock() const;
    void checkOut();
    void checkIn();
    void printTitle() const;
    void printInfo() const;
    bool checkTitle(string title);
    void updateInStock(int num);

    void setCopiesInStock(int num);
    string getTitle();
    videoType(string title = "", string star1 = "",
              string star2 = "", string producer = "",
              string director = "", string productionCo = "",
              int setInStock = 0);

    bool operator==(const videoType&) const;
    bool operator!=(const videoType&) const;
    bool operator<(const videoType&) const;
    bool operator<=(const videoType&) const;
    bool operator>(const videoType&) const;
    bool operator>=(const videoType&) const;

private:
    string videoTitle;
    string movieStar1;
    string movieStar2;
    string movieProducer;
    string movieDirector;
    string movieProductionCo;
    int copiesInStock;
};

```

تعريفات دالات عنصر الفئة videoType هي نفسها الموجودة في الفصل 5. بما أننا هنا نقوم بإثقال جميع العوامل المترابطة فإننا نعطي فقط تعريفات دالات العنصر هذه:

```

        //Overload the relational operators.
bool videoType::operator==(const videoType& right) const
{
    return (videoTitle == right.videoTitle);
}

bool videoType::operator!=(const videoType& right) const
{
    return (videoTitle != right.videoTitle);
}

bool videoType::operator<(const videoType& right) const
{
    return (videoTitle < right.videoTitle);
}

bool videoType::operator<=(const videoType& right) const
{
    return (videoTitle <= right.videoTitle);
}

bool videoType::operator>(const videoType& right) const
{
    return (videoTitle > right.videoTitle);
}

bool videoType::operator>=(const videoType& right) const
{
    return (videoTitle >= right.videoTitle);
}

```

قائمة الشروط:

يتم الاحتفاظ بقائمة الشروط في شجرة بحث ثنائية ولهذا نستمد الفئة videoBinaryTree من الفئة bSearchTreeType. تعريف الفئة videoBinaryTree يكون كما يلي:

```

#include <iostream>
#include <string>
#include "binarySearchTree.h"
#include "videoType.h"

using namespace std;

class videoBinaryTree: public bSearchTreeType<videoType>
{
public:
    bool videoSearch(string title);
        //Function to search the list to see whether a
        //particular title, specified by the parameter
        //title, is in stock.
        //Postcondition: Returns true if the title is found,
        //                  false otherwise.
    bool isVideoAvailable(string title);
        //Function to determine whether at least one copy of
        //a particular video is in stock.
        //Postcondition: Returns true if at least one copy is
        //                  in stock, false otherwise.
    void videoCheckOut(string title);
        //Function to check out a video, that is, rent a video.
        //Postcondition: copiesInStock is decremented by 1.
    void videoCheckIn(string title);
        //Function to check in a video returned by a customer.
        //Postcondition: copiesInStock is incremented by 1.
    bool videoCheckTitle(string title);
        //Function to determine whether a particular video is
        //in stock.
        //Postcondition: Returns true if the video is in stock,
        //                  false otherwise.

```

```

        bool videoSearch (string title);
            // دالة لبحث القائمة لرؤية ما اذا
            // كان شريط معين محدد بواسطة المعامل
            // title موجود في المخزن.
            // شرط تالي: تنتج true اذا تم العثور على العنوان
            // وبخلاف هذا تنتج false.

```

```

        bool isVideoAvailable (string title);
            // دالة لتحديد ما اذا كان هناك على الأقل نسخة واحدة
            // من شريط محدد في المخزن.
            // شرط تالي: تنتج true اذا كان هناك نسخة واحدة على الأقل
            // في المتجر وبخلاف هذا تنتج false.

```

```

        void videoChechOut (string title);
            // دالة لاجراج الشريط أي تأجير.

```

// شرط تالي: يتم تقليل عدد النسخ في المخزن بمقدار 1.

void videoCheckIn (string title);

// دالة لادخال الشريط المعاد بواسطة العميل.

// شرط تالي: يتم زيادة عدد النسخ في المخزن بمقدار 1.

bool videoCheckTitle (string title);

// دالة لتحديد ما اذا كان شريط معين

// موجود في المخزن.

// شرط تالي: تنتج true اذا كان الشريط في المخزن

// وبخلاف هذا تنتج false.

```
void videoUpdateInStock(string title, int num);
//Function to update the number of copies of a video
//by adding the value of the parameter num. The
//parameter title specifies the name of the video
//for which the number of copies is to be updated.
//Postcondition: copiesInStock = copiesInStock + num
```

```
void videoSetCopiesInStock(string title, int num);
//Function to reset the number of copies of a video.
//The parameter title specifies the name of the video
//for which the number of copies is to be reset; the
//parameter num specifies the number of copies.
//Postcondition: copiesInStock = num
```

```
void videoPrintTitle();
//Function to print the titles of all the videos in
//stock.
```

private:

```
void searchVideoList(string title, bool& found,
                     nodeType<videoType>* &current);
//Function to search the video list for a
//particular video, specified by the parameter title.
//Postcondition: If the video is found, the parameter
//                found is set to true, false otherwise.
//                The parameter current points to the
//                node containing the video.
```

```
void inorderTitle(nodeType<videoType> *p);
//Function to print the titles of all the videos in stock.
```

};

void videoUpdateInStock (string title, int num);

```

// دالة لتحديث عدد نسخ الشريط
// عن طريق جمع قيمة المعامل num. المعامل title
// يحدد اسم الشريط الذي يتم تحديث عدد نسخه.
// شرط تالي: النسخ في المخزن = النسخ في المخزن + num

void videoSetCopiesInStock (string title, int num);
// دالة لاعادة ضبط عدد نسخ الشريط.
// المعامل title يحدد اسم الشريط الذي يتم
// اعادة ضبط عدد نسخه والمعامل num يحدد
// عدد النسخ.
// شرط تالي: النسخ في المخزن = num.

void videoPrintTitle ( );
// دالة لطباعة عناوين جميع الشرائط في المخزن.
// خاصة:

void searchVideoList (string title, bool& found,
nodeType<videoType>* &current);
// دالة لبحث قائمة الشرائط من أجل شريط معين
// اذا تم العثور على الشريط يتم ضبط المعامل found
// عند true وبخلاف هذا تنتج false.
// المعامل current يشير الى
// العقدة المحتوية على الشريط.

void inorderTitle (nodeType<videoType> *p);
// دالة لطباعة عناوين جميع الشرائط في المخزن.

```

تعريفات دالات عنصر الفئة videoBinaryTree مماثلة للتعريفات المعطاة في الفصل 5. اننا نعطي تعريفات فقط للدالات searchVideoList و inorderTitle و videoPrintTitle. (انظر تمرين البرمجة رقم 10 في نهاية الفصل).

تقوم الدالة searchVideoList باستخدام خوارزمية بحث مشابهة لخوارزمية بحث شجرة البحث الثنائية المعطاة من قبل في هذا الفصل. انها تنتج true اذا تم العثور على عنصر البحث في القائمة وبخلاف هذا تنتج false كما أنها تنتج مؤشر الى العقدة المحتوية على عنصر البحث. لاحظ أن الدالة searchVideoList عنصر خاص للفئة videoBinaryTree. لهذا لا يمكن للمستخدم استخدام هذه الدالة مباشرة في برنامج. لهذا وبالرغم من أن هذه الدالة تنتج مؤشر الى العقدة في الشجرة الا أن

المستخدم لا يمكنه تناول العقدة بشكل مباشر. يتم استخدام الدالة searchVideoList فقط لتطبيق الدالات الأخرى من الفئة videoBinaryTree.

تعريف هذه الدالة يكون كما يلي:

```
void videoBinaryTree::searchVideoList(string title,
                                     bool& found,
                                     nodeType<videoType>* &current)
{
    found = false;

    videoType temp;

    temp.setVideoInfo(title, "", "", "", "", "", 0);

    if(root == NULL) //the tree is empty
        cout<<"Cannot search an empty list. "<<endl;
    else
    {
        current = root; //set current to point to the root node
                        //of the binary tree
        found = false; //set found to false

        while(current != NULL && !found) //search the tree
        {
            if(current->info == temp) //the item is found
                found = true;
            else
            {
                if(current->info > temp)
                    current = current->llink;
                else
                    current = current->rlink;
            }
        }
    }
}
```

باعطاء مؤشر الى العقدة الجذرية للشجرة الثنائية المحتوية على الشروط تقوم الدالة inorderTitle باستخدام خوارزمية الاجتياز أثناء الترتيب لطباعة عناوين الشروط. لاحظ أن هذه الدالة تنتج فقط عناوين الشروط. تعريف هذه الدالة يكون كما يلي:

```
void videoBinaryTree::inorderTitle(nodeType<videoType> *p)
{
    if(p != NULL)
    {
        inorderTitle(p->llink);
        p->info.printTitle();
        inorderTitle(p->rlink);
    }
}
```

تقوم الدالة videoprintTitle باستخدام الدالة inorderTitle لطباعة عناوين جميع الشرائط الموجودة في المتجر. تعريف هذه الدالة هو:

```
void videoBinaryTree::videoPrintTitle()
{
    inorderTitle(root);
}
```

البرنامج الرئيسي:

البرنامج الرئيسي هو نفسه كما سبق. هنا نقوم باعطاء قائمة هذا البرنامج فقط. نفترض أن اسم الملف الرئيسي المحتوي على تعريف الفئة videoBinaryTree هو videoBinaryTree.h وهكذا.

```
#include <iostream>
#include <fstream>
#include <string>
#include "binarySearchTree.h"
#include "videoType.h"
#include "videoBinaryTree.h"

using namespace std;

void createVideoList(ifstream& infile,
                    videoBinaryTree& videoList);
void displayMenu();

int main()
{
    videoBinaryTree videoList;
    int choice;
    char ch;
    string title;

    ifstream infile;

    infile.open("a:\\videoDat.txt");
    if(!infile)
    {
        cout<<"The input file does not exist."<<endl;
        return 1;
    }

    createVideoList(infile, videoList);
    infile.close();
}
```

```

displayMenu(); //show the menu
cout<<"Enter your choice: ";
cin>>choice; //get the request
cin.get(ch);
cout<<endl;

//process the request
while(choice != 9)
{
    switch(choice)
    {
        case 1: cout<<"Enter the title: ";
                getline(cin, title);
                cout<<endl;
                if(videoList.videoSearch(title))
                    cout<<"Title found."<<endl;
                else
                    cout<<"The store does not carry "
                     <<"this title."<<endl;
                break;
        case 2: cout<<"Enter the title: ";
                getline(cin, title);
                cout<<endl;
                if(videoList.videoSearch(title))
                {
                    if(videoList.isVideoAvailable(title))
                    {
                        videoList.videoCheckOut(title);
                        cout<<"Enjoy your movie: "<<title<<endl;
                    }
                    else
                        cout<<"The video is currently "
                         <<"out of stock."<<endl;
                }
                else
                    cout<<"The video is not in the store."<<endl;
                break;
        case 3: cout<<"Enter the title: ";
                getline(cin, title);
                cout<<endl;
                if(videoList.videoSearch(title))
                {
                    videoList.videoCheckIn(title);
                    cout<<"Thanks for returning "<<title<<endl;
                }
    }
}

```

```

        else
            cout<<"This video is not from our store."
                <<endl;

            break;
        case 4: cout<<"Enter the title: ";
                getline(cin, title);
                cout<<endl;
                if(videoList.videoSearch(title))
                {
                    if(videoList.isVideoAvailable(title))
                        cout<<"The video is currently in stock."
                            <<endl;
                    else
                        cout<<"The video is out of stock."<<endl;
                }
                else
                    cout<<"The video is not in the store."<<endl;

                break;
        case 5: videoList.videoPrintTitle();
                break;
        case 6: videoList.inorderTraversal();
                break;
        default: cout<<"Bad Selection."<<endl;
    } //end switch

    displayMenu(); //display the menu
    cout<<"Enter your choice: ";
    cin>>choice; //get the next request
    cin.get(ch);
    cout<<endl;
} //end while

return 0;
}

void createVideoList(ifstream& infile,
                    videoBinaryTree& videoList)
{
    string Title;
    string Star1;
    string Star2;
    string Producer;
    string Director;
    string ProductionCo;
    char ch;
    int InStock;

```

```

videoType newVideo;

getline(infile, Title);
while(infile)
{
    getline(infile, Star1);
    getline(infile, Star2);
    getline(infile, Producer);
    getline(infile, Director);
    getline(infile, ProductionCo);
    infile>>InStock;
    infile.get(ch);
    newVideo.setVideoInfo(Title, Star1, Star2, Producer,
                          Director, ProductionCo, InStock);
    videoList.insert(newVideo);

    getline(infile, Title);
} //end while

} //end createVideoList

void displayMenu()
{
    cout<<"Select one of the following "<<endl;
    cout<<"1: To check whether a particular video is in "
        <<"the store"<<endl;
    cout<<"2: To check out a video"<<endl;
    cout<<"3: To check in a video"<<endl;
    cout<<"4: To check whether a particular video is in stock"
        <<endl;
    cout<<"5: To print the titles of all the videos"<<endl;
    cout<<"6: To print a list of all the videos"<<endl;
    cout<<"9: To exit"<<endl;
}

```

مراجعة سريعة

1. الشجرة الثنائية اما أن تكون فارغة واما أن بها عقدة خاصة تسمى العقدة الجذرية. اذا لم تكن الشجرة فارغة يكون للعقدة الجذرية مجموعتان من العقد تسمى الشجرتين الفرعيتين اليمنى واليسرى بحيث أن الشجرتين الفرعيتين اليمنى واليسرى تكون عبارة عن أشجار ثنائية.
2. عقدة الشجرة الثنائية بها وصلتان.
3. العقدة في الشجرة الثنائية تسمى ورقة اذا لم يكن لها ثمار يمنى ويسرى.
4. يطلق على العقدة U أصل العقدة V اذا كان هناك فرع من U الى V.
5. المسار من العقدة x الى العقدة y في الشجرة الثنائية عبارة عن تتابع من العقد X_0, X_1, \dots, X_n حيث أن (أ) $X_n = Y, X = X_0$ (X_i, X_{i-1}) لكل $i = 1, 2, \dots, n$. هذا يعني أن هناك فرع من X_0 to X_1, X_1 to X_2, \dots, X_{i-1} X_i, \dots, X_{n-1} to X_n
6. مستوى العقدة في شجرة ثنائية هو عدد الأفرع الموجودة على المسار من الجذر الى العقدة.
7. مستوى العقدة الجذرية للشجرة الثنائيو هو صفر ومستوى ثمار العقدة الجذرية هو 1.
8. ارتفاع الشجرة الثنائية هو عدد العقد على أطول مسار من الجذر الى الورقة.
9. في الاجتياز أثناء الترتيب يتم اجتياز الشجرة الثنائية كما يلي:
 - أ- اجتياز الشجرة الفرعية اليسرى.
 - ب- زيارة العقدة.
 - ت- اجتياز الشجرة الفرعية اليمنى.
10. في الاجتياز قبل الترتيب يتم اجتياز الشجرة الثنائية كما يلي:
 - أ- زيارة العقدة.
 - ب- اجتياز الشجرة الفرعية اليسرى.
 - ت- اجتياز الشجرة الفرعية اليمنى.
11. في الاجتياز بعد الترتيب يتم اجتياز الشجرة الثنائية كما يلي:
 - أ- اجتياز الشجرة الفرعية اليسرى.
 - ب- اجتياز الشجرة الفرعية اليمنى.
 - ت- زيارة العقدة.
12. شجرة البحث الثنائية T اما تكون فارغة أو:
 - أ- T لها عقدة خاصة تسمى العقدة الجذرية.
 - ب- T لها مجموعتان من العقد L_T R_T ميان الشجرة الفرعية اليسرى والشجرة الفرعية اليمنى للشجرة T على التوالي.

- ت- المفتاح في العقدة الجذرية أكبر من أي مفتاح في الشجرة الفرعية اليسرى وأصغر من أي مفتاح في الشجرة الفرعية اليمنى.
- ث- $L_T = R_T$; عن شجرتين فرعيتين ثنائيتين.

13. لحذف عقدة من شجرة بحث ثنائية لها شجرتين فرعيتين اليمنى ويسرى وغير فارغتين يتم أولاً تحديد ما يسبقها مباشرة ثم يتم نسخ ما قبلها في العقدة وفي النهاية يتم حذف ما يسبقها.

14. الشجرة الثنائية الكاملة الاتزان عبار عن شجرة بحث ثنائية حيث:

- أ- ارتفاع الشجرتين الفرعيتين اليمنى واليسرى للجذر متساوي.
- ب- الشجرتان الفرعيتان اليسرى واليمنى من الجذر يكونا أشجار ثنائية كاملة الاتزان.
15. شجرة AVL (أو المتوازنة الارتفاع) عبارة عن شجرة بحث ثنائية بحيث:
- أ- هناك فرق بين ارتفاع الشجرتين الفرعيتين اليمنى واليسرى للجذر بمقدار واحد.
- ب- الشجرتان الفرعيتان اليسرى واليمنى من الجذر يكونا أشجار AVL.

16. لتكن x عقدة في شجرة ثنائية. اذن x_l تعبر عن x_l فاع الشجرة الفرعية اليسرى ل x وتعبر عن ارتفاع الشجرة x_r الفرعية اليمنى ل x .

17. لتكن T شجرة AVL ولتكن x عقدة في الشجرة T . اذن $|x_l - x_r| \leq 1$ تعبر عن القيمة $|x_l - x_r|$ ان $x_l - x_r$.

18. لتكن x عقدة في شجرة AVL مسماة T :

- أ- اذا كان $x_l > x_r$ أن x مرتفعة من اليسار. في هذه الحالة $x_l = x_r + 1$
- ب- اذا كان $x_l = x_r$ أن x مترفعة بالتساوي.
- ت- اذا كان $x_r > x_l$ أن x مرتفعة من اليمين. في هذه الحالة $x_r = x_l + 1$

19. عامل اتزان x يكتب $bf(x)$ ويعرف بأن $bf(x) = x_r - x_l$.

20. لتكن x عقدة في شجرة AVL اسمها T . اذن:

- أ- اذا كانت x مرتفعة من اليسار اذن عامل اتزان $x = -1$.
- ب- اذا كانت x مرتفعة بالتساوي اذن عامل اتزان $x = 0$.
- ت- اذا كانت x مرتفعة من اليمين اذن عامل اتزان $x = 1$.

21. لتكن x عقدة في شجرة ثنائية. نقول أن العقدة x تتعدى على معيار الاتزان اذا كانت

أي اذا كان $|x_l - x_r| > 1$ ع الشجرتين الفرعيتين اليمنى واليسرى من x أكثر من 1.

22. كل عقدة x في شجرة AVL بالاضافة الى البيانات والمؤشرات الى الشجرتين الفرعيتين اليمنى واليسرى يجب أن تتبع عامل اتزانها.

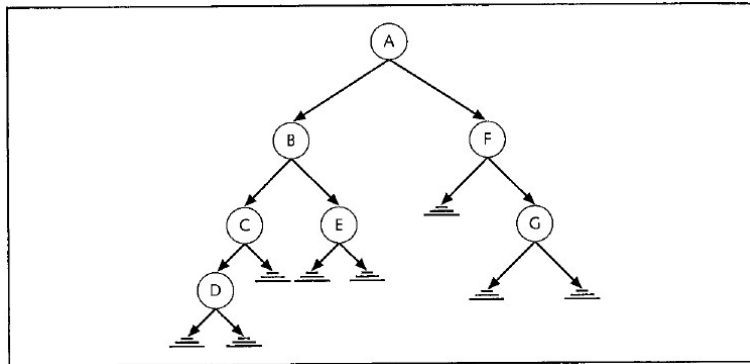
23. في شجرة AVL هناك نوعان من الدورانات هما الدوران الى اليسار والدوران الى اليمين.

افترض أن الدوران يحدث عند العقدة x . اذا كان الدوران لليسار يكون هناك عقد محددة من

الشجرة الفرعية اليمنى من x تنتقل الى شجرتها الفرعية اليسرى وجذر الشجرة الفرعية اليمنى يصبح الجذر الجديد للشجرة الفرعية التي أعيد بناءها. بالمثل اذا كان الدوران لليمين يكون هناك عقد محددة من الشجرة الفرعية اليسرى من x تنتقل الى شجرتها الفرعية اليمنى وجذر الشجرة الفرعية اليسرى يصبح الجذر الجديد للشجرة الفرعية التي أعيد بناءها.

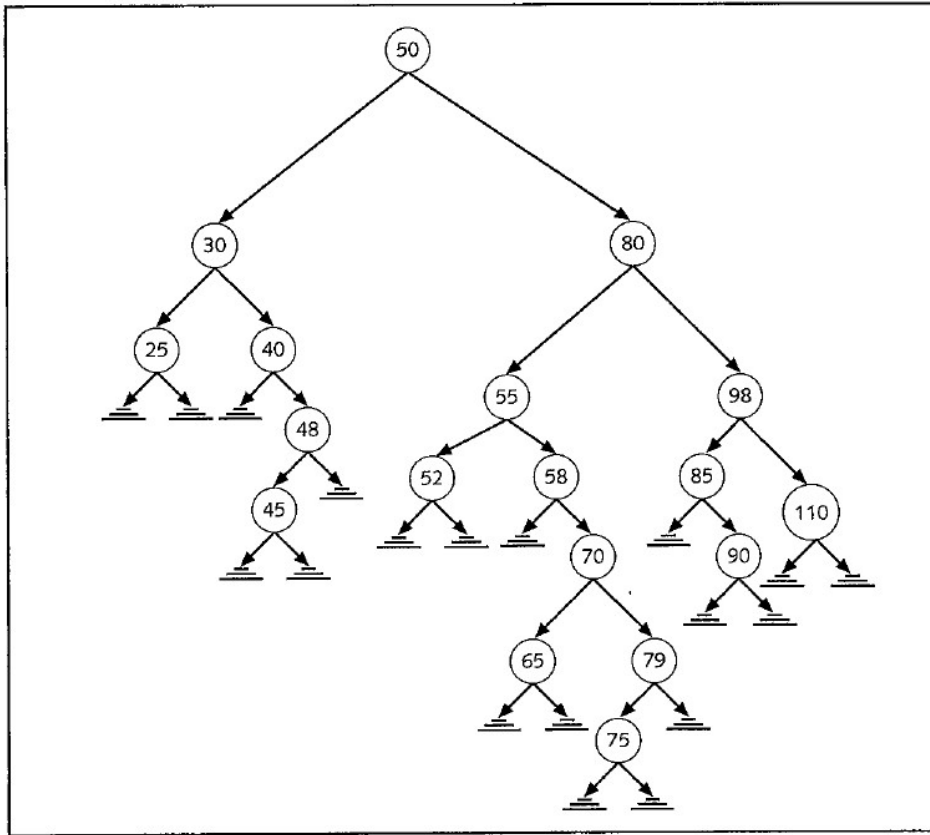
تمارين

1. ضع علامة صح أم خطأ أمام العبارات التالية:
 - أ- يجب ألا تكون الشجرة الثنائية فارغة.
 - ب- مستوى العقدة الجذرية يساوي صفر.
 - ت- اذا كان للشجرة عقدة واحدة فقط فان ارتفاع الشجرة يكون صفر أن عدد المستويات يكون صفر.
 - ث- اجتياز الشجرة الثنائية أثناء الترتيب يقوم دائماً باخراج البيانات بترتيب تصاعدي.
2. هناك 14 شجرة ثنائية مختلفة ذات أربعة عقد. ارسم جميعها.
- الشجرة الثنائية في شكل 11-47 يتم استخدامها من أجل التمارين من 3 الى 8.



- شكل 11-47: شكل من أجل التمارين من 3 الى 8
3. اوجد L_A العقدة في الشجرة الفرعية اليسرى من A.
 4. اوجد R_A العقدة في الشجرة الفرعية اليمنى من A.
 5. اوجد R_B العقدة في الشجرة الفرعية اليمنى من B.
 6. اذكر عقد هذه الشجرة الثنائية في تتابع أثناء الترتيب.
 7. اذكر عقد هذه الشجرة الثنائية في تتابع قبل الترتيب.
 8. اذكر عقد هذه الشجرة الثنائية في تتابع بعد الترتيب.

سوف يتم استخدام الشجرة الثنائية الموجودة في شكل 11-48 من أجل التمارين من 9 الى 13.



شكل 11-48: شكل من أجل التمارين من 9 الى 13

9. اذكر المسار من العقدة ذات البيان 80 الى العقدة ذات البيان 79.
10. سوف يتم ادخال عقدة ذات بيان 35 في الشجرة. اذكر العقدة التي تتم زيارتها بواسطة الدالة insert لادخال 35. أعد رسم الشجرة بعد ادخال 35.
11. احذف العقدة 52 وأعد رسم الشجرة الثنائية.
12. احذف العقدة 40 وأعد رسم الشجرة الثنائية.
13. احذف العقد 80 و58 بهذا الترتيب وأعد رسم الشجرة الثنائية بعد كل حذف.
14. افترض أنه تم اعطائك تتابعين من العناصر المقابلة للتتابع أثناء الترتيب والتتابع بعد الترتيب. اثبت أنه من الممكن اعادة بناء شجرة ثنائية فريدة.
15. الكود التالي يذكر العقد الموجودة في شجرة ثنائية بترتيبين مختلفين:

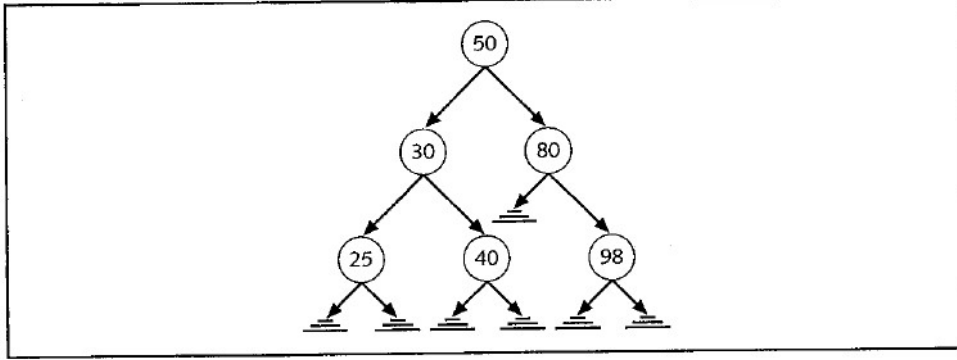
قبل الترتيب: ABCDEFGHIJKLM

أثناء الترتيب: CEDFBAHJIKGML

ارسم الشجرة الثنائية.

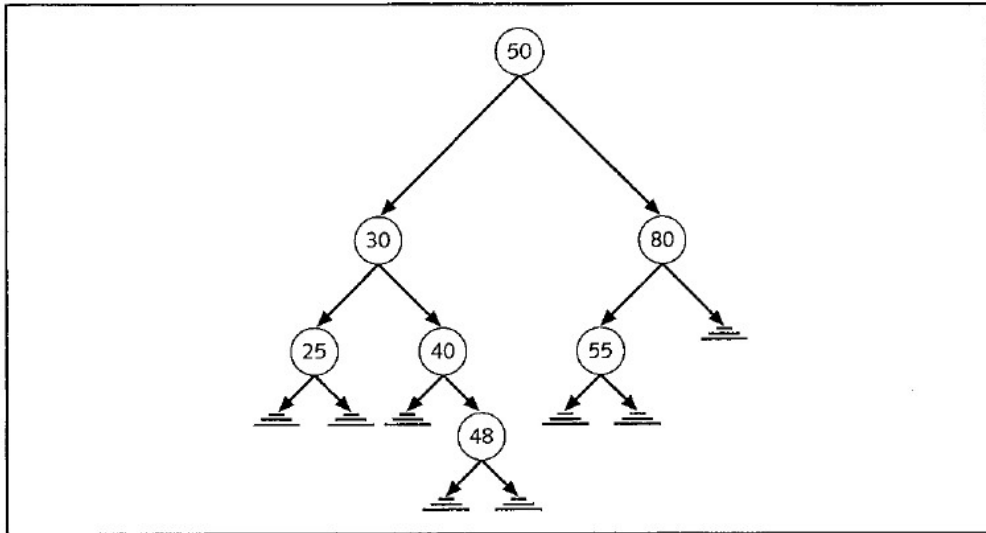
16. باعطاء التتابع قبل الترتيب والتتابع بعد الترتيب أوضح أنه قد يكون من غير الممكن إعادة بناء الشجرة الثنائية.

17. ادخل 100 في الشجرة AVL الموجودة في شكل 11-49. الشجرة الناتجة يجب أن تكون شجرة AVL. ما هو عامل الاتزان عند العقدة الجذرية بعد الإدخال؟



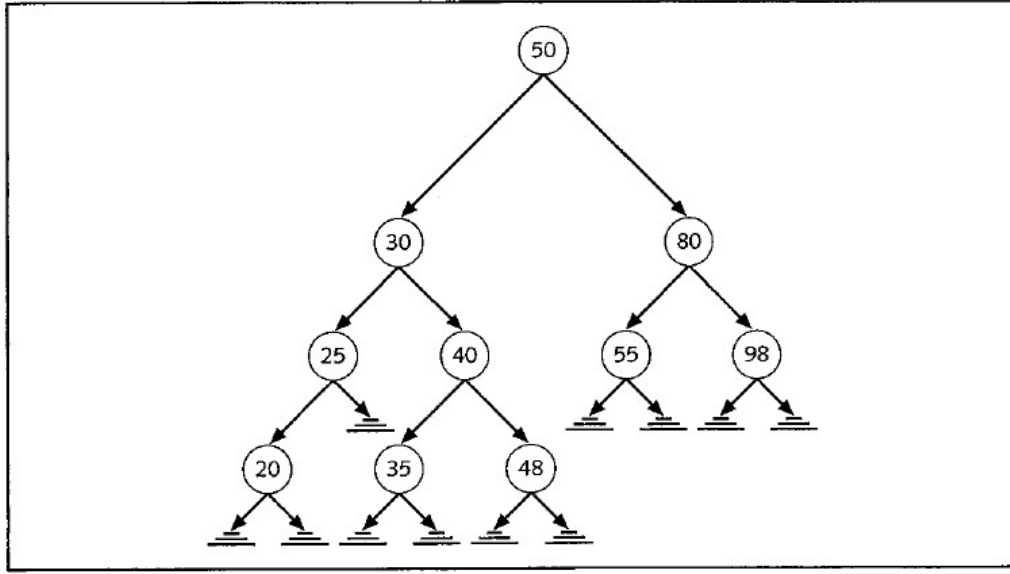
شكل 11-49: شجرة AVL من أجل تمرين 17

18. ادخل 45 في الشجرة AVL الموجودة في شكل 11-50. يجب أن تكون الشجرة الناتجة شجرة AVL. ما هو عامل الاتزان عند العقدة الجذرية بعد الإدخال؟



شكل 11-50: شجرة AVL من أجل تمرين رقم 18

19. ادخل 42 في الشجرة AVL الموجودة في شكل 11-51. يجب أن تكون الشجرة الناتجة شجرة AVL. ما هو عامل الاتزان عند العقدة الجذرية بعد الإدخال؟



شكل 11-51: شجرة AVL من أجل تمرين رقم 19

20. المفاتيح التالية يتم ادخالها (بالترتيب المعطى) داخل شجرة AVL فارغة مبدئياً . وضح الشجرة AVL بعد كل ادخال.

29، 5، 19، 34، 48، 46، 31، 39، 24

تمارين برمجة

1. اكتب تعريف الدالة nodeCount التي تنتج عدد العقد في شجرة ثنائية. أضف هذه الدالة الى الفئة binaryTreeType واصنع برنامج لاختبار هذه الدالة.
2. اكتب تعريف الدالة leavesCount التي تتخذ من الكؤشر الى العقدة الجذرية لشجرة ثنائية كمعامل لها وتنتج عدد الأوراق في شجرة ثنائية. أضف هذه الدالة الى الفئة binaryTreeType واصنع برنامج لاختبار هذه الدالة.
3. اكتب الدالة swapSubtree التي تستبدل جميع الأشجار الفرعية اليمنى واليسرى لشجرة ثنائية. أضف هذه الدالة الى الفئة binaryTreeType واصنع برنامج لاختبار هذه الدالة.
4. اكتب الدالة singleParent التي تنتج عدد العقد في شجرة ثنائية لها ثمرة واحدة فقط. أضف هذه الدالة الى الفئة binaryTreeType واصنع برنامج لاختبار هذه الدالة. (لاحظ: قم أولاً بعمل شجرة بحث ثنائية).
5. اكتب برنامج لاختبار عمليات متنوعة على شجرة البحث الثنائية.
6. أ. اكتب تعريف الدالة لتطبيق خوارزمية الاجتاز الغير تكرارية بعد الترتيب.

- ب. اكتب برنامج لاختبار خوارزميات الاجتياز الغير تكرارية أثناء وقبل وبعد الترتيب.
7. اكتب نسخة من خوارزمية الاجتياز قبل الترتيب التي يمكن فيها تمرير الدالة المحددة بواسطة المستخدم كمعامل لتحديد معيار الزيارة عند عقدة.
8. اكتب نسخة من خوارزمية الاجتياز بعد الترتيب التي يمكن فيها تمرير الدالة المحددة بواسطة المستخدم كمعامل لتحديد معيار الزيارة عند عقدة.
9. أ. اكتب تعريف قالب الفئة التي تطبق الشجرة AVL كنوع بيانات مجرد. (لا تحتاج الى تطبيق عملية الحذف).
- ب. اكتب تعريفات دالات عنصر الفئة التي قمت بتعريفها في (أ).
- ج. اكتب برنامج لاختبار عمليات متنوعة على شجرة AVL.
10. اكتب تعريفات دالات الفئة videoBinaryTree الغير معطاة في تمرين البرمجة الخاص بمتجر الفيديو.

11. (برنامج متجر الفيديو) في تمرين البرمجة رقم 14 في الفصل 5 تم سؤالك أن تقوم بتصميم وتطبيق فئة للاحتفاظ ببيانات العملاء في قائمة متصلة. بما أن البحث في القائمة المتصلة يكون تتابعياً ولهذا قد يكون مستهلكاً للوقت قم بتصميم وتطبيق الفئة customerTreeType حتى يمكن تخزين بيانات هذا العميل في شجرة ثنائية. يجب أن تكون الفئة customerTreeType مستمدة من الفئة bSearchTreeType كما تم تصميمها في هذا الفصل. اكتب برنامج لاختبار مكون العميل واكتب أيضاً تعريف الدالة nodeCount من الفئة binaryTreeType قبل تنفيذ البرنامج.

12. (برنامج متجر الفيديو) باستخدام فئات لتطبيق بيانات الفيديو، وبيانات قائمة الفيديو، وبيانات العملاء، وبيانات قائمة العملاء كما تم تصميمها في هذا الفصل وفي تمارين البرمجة رقم 10 و 11 قم بتصميم وتطبيق البرنامج لوضع متجر الفيديو داخل عملية. اكتب تعريف الدالة nodeCount من الفئة binaryTreeType قبل تنفيذ البرنامج.

الفصل 12 الرسوم البيانية

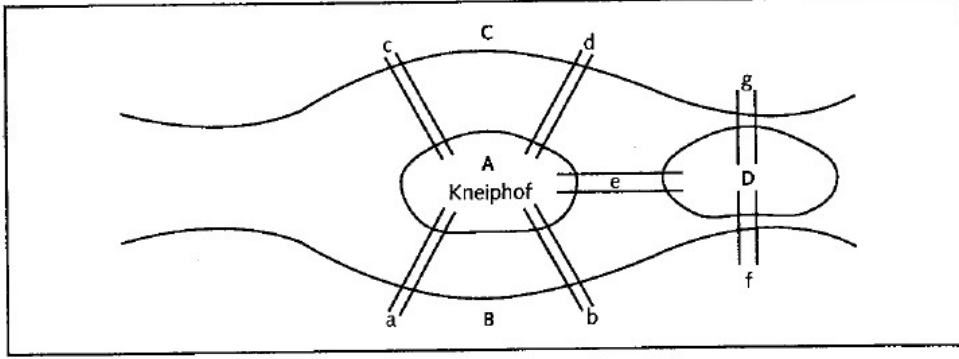
في هذا الفصل سوف:

- تتعلم ما هي الرسوم البيانية.
- تصبح على علم بالمصطلحات الأساسية لنظرية الرسم البياني.
- تكتشف كيفية تمثيل الرسوم البيانية في ذاكرة الحاسوب.
- تستكشف الرسوم البيانية كنوع بيانات مجرد.
- تدرس وتطبق خوارزميات متعددة لاجتياز الرسوم البيانية.
- تتعلم كيفية تطبيق خوارزمية المسار الأقصر.
- تدرس وتطبق خوارزمية ؟؟؟؟؟؟؟؟؟؟؟
- تستكشف الترتيب الطوبوغرافي.

في الفصول السابقة تعلمت عدة طرق لتمثيل البيانات والتحكم فيها. يقوم هذا الفصل بمناقشة كيفية تطبيق والسيطرة على الرسوم البيانية التي لها العديد من التطبيقات في علوم الحاسب الآلي.

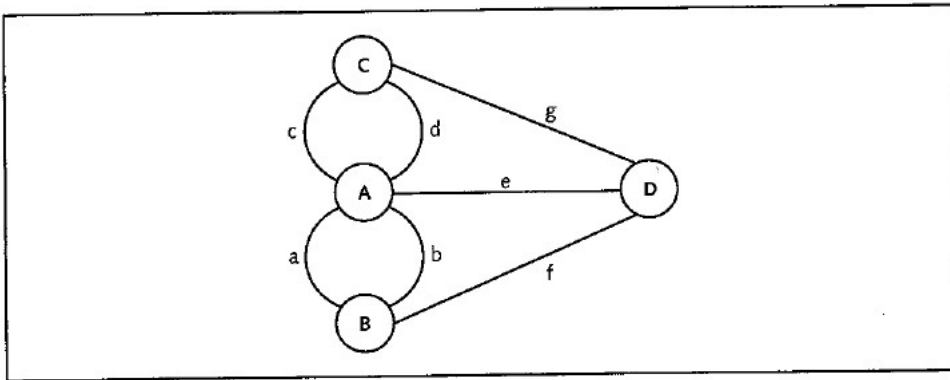
مقدمة:

في عام 1736 تم فرض المشكلة التالية. في بلدة كونيسبيرج (المسماة حالياً كالنجراد) يتدفق نهر بريجل (بريجوليا) حول جزيرة نايبهوف ثم انقسم الى اثنين. انظر شكل 1-12.



شكل 1-12: مشكلة كوبري كونيسبيرج

النهر له أربعة مناطق برية (A و B و C و D) كما هو موضح في الشكل. هذه المناطق البرية متصلة باستخدام سبعة كباري كما هو موضح في شكل 1-12. تمت تسمية الكباري a و b و c و d و e و f و g. مشكلة برج كونيسبيرج تكون كما يلي: بدءاً من إحدى المناطق الأرضية هل يمكن السير عبر جميع الكباري مرة واحدة فقط والعودة الى المنطقة البرية التي بدأت منها؟ في عام 1736 قام ايلور بتمثيل مشكلة كوبري كونيسبيرج كرسم بياني كما هو موضح في شكل 2-12 وأجاب السؤال بالنفي وهذا ما حدد مولد نظرية الرسوم البيانية (كما هو مسجل).



شكل 2-12: تمثيل الرسم البياني لمشكلة كوبري كونيسبيرج

على مر المائتين عام الماضية تم تطبيق نظرية الرسم البياني على مجموعة متنوعة من التطبيقات. يتم استخدام الرسوم البيانية لتشكيل الدوائر الكهربائية، والمركبات الكيميائية، وخرائط الطرق السريعة، وغيرها. كما يتم استخدامها في تحليل الدوائر الكهربائية، وإيجاد أقصر الطرق، وتخطيط المشروعات، واللغويات، وعلم الوراثة، والعلوم الاجتماعية، وغيرها. في هذا الفصل سوف نتعلم ما هي الرسوم البيانية وتطبيقاتها في علوم الحاسب الآلي.

تعريفات الرسم البياني و ترميزاته:

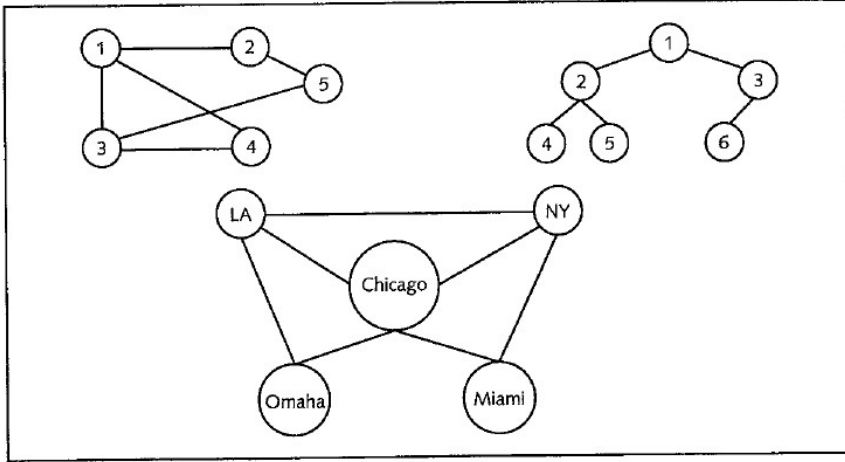
لتسهيل وتبسيط مناقشتنا نستعير القليل من التعريفات والمصطلحات من نظرية المجموعة. لتكن X مجموعة. إذا كانت a عنصر بالمجموعة X فإننا نكتب $a \in X$. (الرمز " \in " يعني "ينتمي الى") المجموعة y تسمى مجموعة فرعية من X إذا كان كل عنصر من y عنصر كذلك من X . إذا كانت Y مجموعة فرعية من X نقوم بكتابة $X \supseteq Y$. (الرمز " \supseteq " يعني "مجموعة فرعية من"). **تقاطع** المجموعتين A و B يكتب $A \cap B$ هو مجموعة جميع العناصر الموجودة في A و B أي أن $A \cap B$ $\{x \mid x \in A, x \in B\}$. (الرمز " \cap " يعني "تقاطع"). **اتحاد** المجموعتان A و B تكتب $A \cup B$ وهو مجموعة كل العناصر الموجودة في A أو الموجودة في B أي أن $A \cup B$ $\{x \mid x \in A \text{ or } x \in B\}$. (الرمز " \cup " يعني "اتحاد"). **مجموعتان** A و B **متساويتان** إذا كان $A \subseteq B$ و $B \subseteq A$ أي أن $A = B$. **مجموعة الأزواج المرتبة** من عناصر A و B أي أن $A \times B$ $\{(a, b) \mid a \in A, b \in B\}$. **الرسم البياني** G عبارة عن زوج $G = (V, E)$ حيث V مجموعة محدودة غير فارغة تسمى مجموعة قمم من G وأن $E \subseteq V \times V$. هذا يعني أن عناصر E هي زوج عناصر V . يطلق على E مجموعة الحواف.

لتكن $V(G)$ معبرة عن مجموعة قمم و $E(G)$ معبرة عن مجموعة حواف G . إذا كانت عناصر $E(G)$ أزواج مرتبة إذن يطلق على G **رسم بياني مباشر** أو **رسم ثنائي** وبخلاف هذا يطلق على G **رسم بياني غير مباشر**. في الرسم البياني الغير مباشر تقوم الأزواج (u, v) و (v, u) بتمثيل نفس الحافة. لتكن G رسم بياني. الرسم البياني H يطلق عليه **رسم بياني فرعي** من G إذا $V(H) \subseteq V(G)$ و $E(H) \subseteq E(G)$ أي أن كل قمة من H عبارة عن قمة من G وكل حافة من H عبارة عن حافة من G .

يمكن توضيح الرسم البياني تصويرياً. يتم رسم القمم كدوائر وعلامة بداخل الدائرة تمثل القمة. في الرسم البياني الغير مباشر يتم رسم الحواف باستخدام خطوط. في الرسم البياني المباشر يتم رسم الحواف باستخدام أسهم.

مثال 1-12:

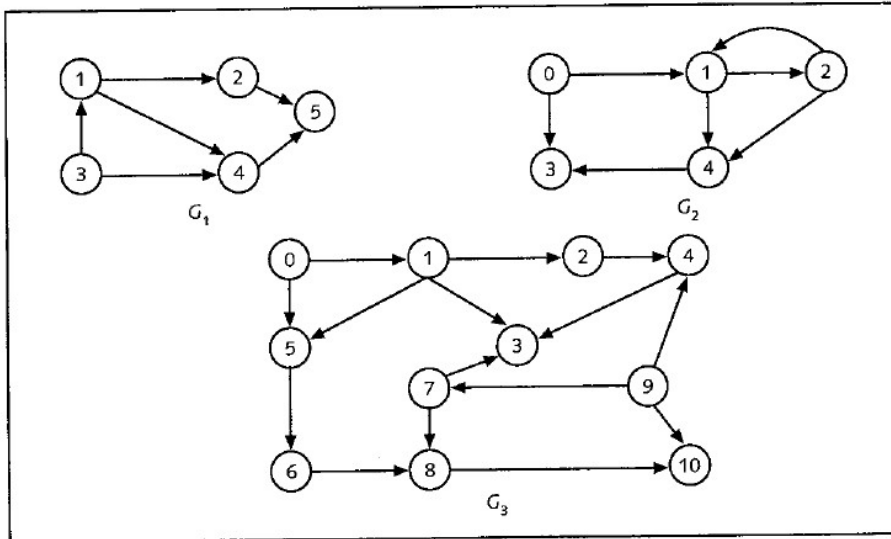
الشكل 3-12 يوضح بعض أمثلة على الرسوم البيانية الغير مباشرة.



شكل 3-12: رسوم بيانية متعددة غير مباشرة

مثال 2-12:

الشكل 4-12 يوضح بعض الأمثلة على رسوم بيانية مباشرة.



شكل 4-12: رسوم بيانية مباشرة متعددة

بالنسبة الى الرسوم البيانية في الشكل 4-12 لدينا:

$$V(G_1) = \{1, 2, 3, 4, 5\}$$

$$E(G_1) = \{(1, 2), (1, 4), (2, 5), (3, 1), (3, 4), (4, 5)\}$$

$$V(G_2) = \{0, 1, 2, 3, 4\}$$

$$E(G_2) = \{(0, 1), (0, 3), (1, 2), (1, 4), (2, 1), (2, 4), (4, 3)\}$$

$$V(G_3) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$E(G_3) = \{(0, 1), (0, 5), (1, 2), (1, 3), (1, 5), (2, 4), (4, 3), (5, 6), (6, 8), (7, 3), (7, 8), (8, 10), (9, 4), (9, 7), (9, 10)\}$$

لتكن G رسم بياني غير مباشر. لتكن u و v قمتين في G . اذن u و v تسميان **متجاوران** اذا كان هناك حافة من واحد الى الآخر أي أن $(u, v) \in E$. لتكن $e = (u, v)$ حافة في G . نقول اذن أن الحافة e حدث على القمم u و v . الحدث الحافة على قمة واحدة يسمى **حلقة**. اذا كانت حافتان e_1 و e_2 مرتبطتان بنفس زوج القمم اذن يطلق على e_1 و e_2 **حواف متوازية**. يطلق على الرسم البياني رسم بياني بسيط اذا لم يكن به حلقات أو حواف متوازية. هناك مسار من u الى v اذا كان هناك تتابع من القمم u_1, u_2, \dots, u_n حيث $u = u_1$ و $u_n = v$ وأن (u_i, u_{i+1}) حافة لكل $i = 1, 2, \dots, n-1$. يطلق على القمتين u و v **متصلتان** اذا كان هناك مسار من u الى v . **المسار البسيط** هو المسار التي تكون فيه جميع القمم مستقلة فيما عدا القمتين الأولى والأخيرة. **الدورة** في G عبارة عن مسار بسيط تكون فيه القمتان الأولى والأخيرة متماثلتين. يطلق على G **متصلة** اذا كان هناك مسار من أي قمة الى أي قمة أخرى. المجموعة الفرعية القصوى من القمم المتصلة يطلق عليها **مكون** من G .

لتكن G رسم بياني مباشر ولتكن u و v قمتين في G . اذا كان هناك قمة من u الى v أي أن $(u, v) \in E$ اذن نقول أن u قريبة من v وأن v قريبة من u . تعريفات المسارات والدورات في G مماثلة لهذه التعريفات بالنسبة الى الرسوم البيانية الغير مباشرة. يطلق على G **متصلة بقوة** اذا كان هناك قمتين متصلتين في G .

انظر الى الرسوم البيانية المباشرة في شكل 4-12. في G_1 كانت 5-4-1 مسار من القمة 1 الى القمة 5. لا توجد دورات في G_1 . في G_2 تكون 1-2-1 عبارة عن دورة. في G_3 تكون 3-4-2-1-0 مسار من القمة صفر الى القمة 3 وتكون 10-8-6-5-1 مسار من القمة 1 الى القمة 10. لا توجد دورات في G_3 .

تمثيل الرسوم البيانية:

لكتابة برامج تعالج وتتحكم في الرسوم البيانية يجب أن يتم تخزين الرسوم البيانية – أي تمثيلها – في ذاكرة الحاسوب. يمكن تمثيل الرسم البياني (في ذاكرة الحاسوب) بعدة طرق. الآن نقوم بمناقشة طريقتين يتم استخدامهم بشكل شائع وهما: مصفوفات التجاور وقوائم التجاور.

مصفوفة التجاور:

لتكن G رسم بياني ذو عدد n من القمم حيث $n > 0$. لتكن $V = \{v_1, v_2, \dots, v_n\}$ مصفوفة التجاور عبارة عن مصفوفة ثنائية الأبعاد $n \times n$ بحيث أن (i, j) مدخل A_G هو 1 اذا كان هناك حافة من v_i الى v_j وبخلاف هذا (i, j) مدخل يكون صفر. هذا يعني:

$$A_G(i, j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E(G) \\ 0 & \text{otherwise} \end{cases}$$

مثال 3-12:

انظر الى الرسوم البيانية المباشرة في شكل 4-12. مصفوفات تجاور الرسوم البيانية المباشرة G_1 و G_2 و G_3 كما يلي:

$$A_{G_1} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_{G_2} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

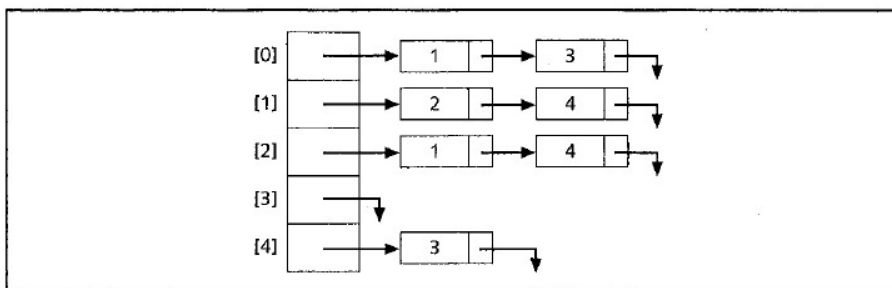
$$A_{G_3} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 9 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

قوائم التجاور:

لتكن G رسم بياني ذو عدد n من القمم حيث $n > 0$. لتكن $V(G) = \{v_1, v_2, \dots, v_n\}$ في تمثيل قائمة التجاور هناك قائمة متصلة متوافقة مع كل قمة v بحيث أن كل عقدة من القائمة المتصلة تحتوي على القمة u بحيث أن $(v, u) \in E(G)$. بما أن هناك عدد n من العقد فإننا نستخدم مصفوفة A حجمها n بحيث يكون $A[i]$ مؤشر الى القائمة المتصلة المحتوية على القمم التي يكون مجاور لها. يتضح أن كل عقدة لها مكونان v_i ونا vertex أي القمة والوصلة. المكون vertex يحتوي على مؤشر القمة المجاورة للقمة i .

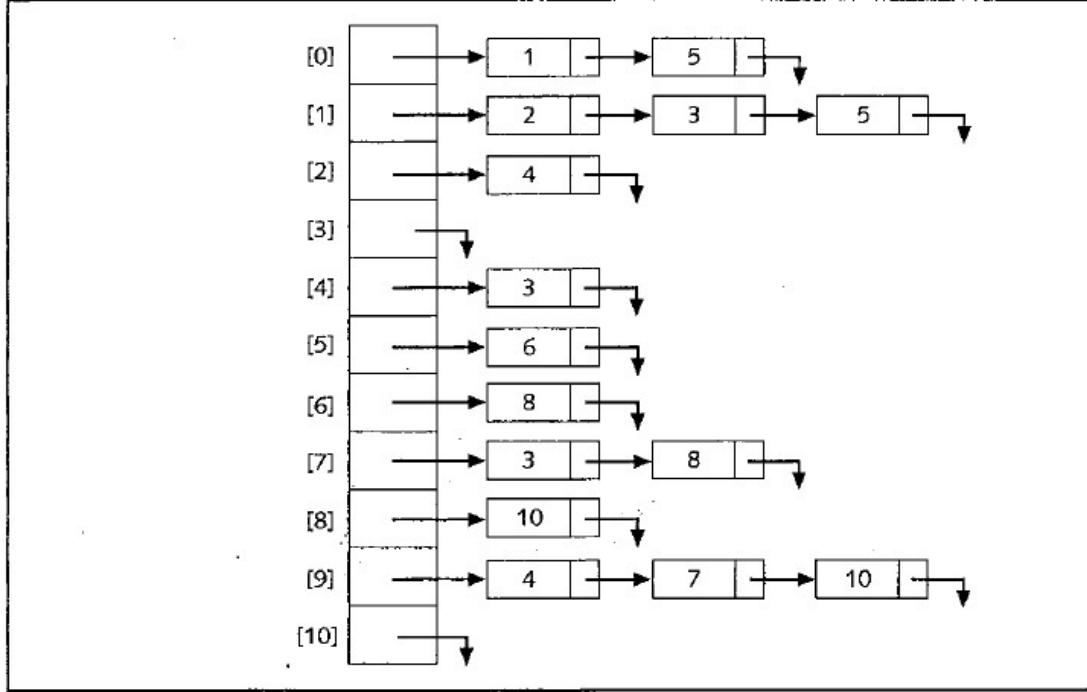
مثال 4-12:

انظر الى الرسوم البيانية المباشرة في شكل 4-12. الشكل 5-12 يوضح قائمة التجاور للرسم البياني المباشر G_2 .



شكل 5-12: قائمة تجاور للرسم البياني G2 من شكل 4-12

الشكل 6-12 يوضح قائمة التجاور للرسم البياني المباشر G_3 .



شكل 6-12: قائمة تجاور للرسم البياني G3 من شكل 4-12

العمليات على الرسوم البيانية:

الآن بعدما علمت كيفية تمثيل الرسوم البيانية في ذاكرة الحاسوب تكون الخطوة التالية الواضحة هي تعلم العمليات الأساسية على الرسم البياني. العمليات التي يتم أدائها بشكل شائع على الرسم البياني تكون كما يلي:

1. عمل رسم بياني وهذا يعني تخزين الرسم البياني في ذاكرة الحاسوب باستخدام تمثيل رسم بياني معين.

2. تفريغ الرسم البياني وهذه العملية تجعل الرسم البياني فارغاً .

3. تحديد ما اذا كان الرسم البياني فارغاً .

4. اجتياز الرسم البياني.

5. طباعة الرسم البياني.

سوف نقوم باضافة المزيد من العمليات على الرسم البياني عندما نناقش تطبيق محدد أو رسم بياني محدد لاحقاً في هذا الفصل.

ان كيفية تمثيل رسم بياني في ذاكرة الحاسوب تعتمد على التطبيق المحدد. لأهداف توضيحية نقوم باستخدام تمثيل قائمة التجاور (قائمة متصلة) للرسم البيانية. لهذا تكون القمم المجاورة لـ v لكل قمة v في الرسم البياني المباشر تسمى أيضاً **التالي المباشر** (مخزنة في القائمة المتصلة المرتبطة بـ v). لإدارة البيانات في قائمة متصلة نقوم باستخدام الفئة `linkedListType` التي نوقشت في الفصل 5. ان خوارزميات اجتياز الرسم البياني والخوارزميات التي سوف نقوم باضافتها ومناقشتها في أقسام لاحقة سوف تحتاج الى أن يتم اجتياز القائمة المتصلة بكل عقدة والى استعادة القمة الموجودة في كل عقدة. الفئة `linkedListType` لا تحتوي على أي دالة يمكنها اجتياز القائمة المتصلة واستعادة البيانات المخزنة في كل عقدة واحدة واحدة. (لاحظ أن دالة `print` تقوم باخراج البيانات الى جهاز الاخراج ببساطة.) لهذا نقوم أولاً بمد تعريف الفئة `linkedListType` (باستخدام آلية التوريث) حتى يمكن استعادة القمم المجاورة لقمة معطاة في مصفوفة ما. هذا يقوم بتبسيط معالجة القمم.

لنقم بتسمية هذه الدالة `linkedListGraph`. تعريف هذه الدالة هو:

```
template<class vType>
class linkedListGraph: public linkedListType<vType>
{
public:
    void getAdjacentVertices(vType adjacencyList[],
                           int& length);
    //Function to retrieve the vertices adjacent to a given
    //vertex.
    //Postcondition: The vertices adjacent to a given vertex
    //               are retrieved in the array adjacencyList.
    //               The parameter length specifies the number
    //               of vertices adjacent to a given vertex.
};
```

تعريف دالة العنصر `getAdjacentVertices` معطى فيما بعد.

```
template<class vType>
void linkedListGraph<vType>::getAdjacentVertices
    (vType adjacencyList[], int& length)
{
    nodeType<vType> *current;
    length = 0;
    current = first;
    while(current != NULL)
    {
        adjacencyList[length++] = current->info;
        current = current->link;
    }
}
```

بعد هذا نمد تعريف قوالب الفئة لكي تشمل تعبيرات ثابتية مثل العوامل.

القوالب (مرة أخرى):

حتى الآن قمنا بتمرير أنواع البيانات فقط كعوامل للقوالب. التعبيرات الثابتة مثلها مثل الأنواع (أنواع البيانات) يمكن تمريرها كعوامل للقوالب. على سبيل المثال انظر الى قالب الفئة التالي:

```
template<class elemType, int size>
class listType
{
public:
    .
    .
    .
private:
    int maxSize;
    int length;
    elemType listElem[size];
};
```

قالب الدالة هذا يحتوي على عنصر بيانات مصفوفة ويتم تمرير نوع عناصر المصفوفة وحجمها كعامل، لقالب الفئة لعمل قائمة من 100 مكان، من عناصر من نوع int نستخدم البيان التالي:

```
listType<int, 100> intList;
```

يتم تمرير كل من نوع العنصر وحجم المصفوفة الى قالب الدالة listType. لاحظ أن نوع العنصر (هو int) وحجم المصفوفة 100 موجودين في أقواس زاوية بعد اسم قالب الفئة.

بما أن تعريف الفئة السابقة لم يعد يحتوي على عناصر بيانات اذن فقد لا نحتاج الى ادخال المدمر. بالرغم من هذا نقوم بتضمين المقوم لتحديد قيم عناصر البيانات maxSize (الحجم الأقصى) وlength (الطول).

في هذا الفصل سوف تستخدم هذه الطريقة لاعلان عناصر بيانات المصفوفة. هذا يكون مفيد للغاية اذا احتجنا الى اعلان مصفوفات ذات بعدين أو مصفوفات أكبر.

الرسوم البيانية كنوع بيانات مجرد:

في هذا القسم نصف الفئة لتطبيق الرسوم البيانية كنوع بيانات مجرد ونقدم تعريفات الدالات لتطبيق العمليات على الرسم البياني.

تقوم الفئة التالية بتعريف الرسم البياني كنوع بيانات مجرد:

```

const int infinity = 10000000;    //This will be used in later
                                   //sections of this chapter, when
                                   //we discuss weighted graphs.

template<class vType, int size>
class graphType
{
public:
    bool isEmpty()
        //Function to determine whether the graph is empty.
        //Postcondition: Returns true if the graph is empty;
        //                otherwise, returns false.
    void createGraph();
        //Function to create the graph using the adjacency list
        //representation.
        //Postcondition: The graph is created in the form of
        //                adjacency lists.
    void clearGraph();
        //Function to deallocate the memory occupied by the linked
        //lists and the array of pointers pointing to the linked
        //lists.
    void printGraph() const;
        //Function to print the graph.

    graphType();
        //default constructor
        //Postcondition: The graph size is set to 0, that is,
        //                gSize = 0 and maxSize = size.

    ~graphType();
        //destructor
        //Postcondition: The storage occupied by the graph
        //                is deallocated.

```

const int infinity = 10000000

// سوف يتم استخدام هذا في أقسام لاحقة

// من هذا الفصل عندما نناقش

// الرسوم البيانية الموزونة.

bool isEmpty ()

// دالة لتحديد ما اذا ان الرسم البياني فارغاً .

// شرط تالي تنتج true اذا ان الرسم البياني فارغاً

// وبخلاف هذا تنتج false.

void createGraph ()

// دالة لعمل الرسم البياني باستخدام تمثيل قائمة التجاور.

// شرط تالي يتم عمل الرسم البياني في شكل قوائم تجاور.

void clearGraph ()

// دالة لازالة تخصيص الذاكرة التي تشغلها القوائم المتصلة
// ومصفوفة المؤشرات التي تشير الى القوائم المتصلة.

void printGraph () const;

// دالة لطباعة الرسم البياني.

graphType ();

// مقوم افتراضي

// شرط تالي: يتم ضبط حجم الرسم البياني عند صفر أي أن

gSize = 0 و maxSize = size.

-graphType ();

// مدمر

// شرط تالي: يتم ازالة تخصيص الذاكرة التي

// يحتلها الرسم البياني.

protected:

int maxSize; //maximum number of vertices

int gSize; //current number of vertices

linkedListGraph<vType> graph[size]; //array of pointers to
//create the adjacency
//lists (linked lists)

};

int maxSize;

// العدد الأقصى من القمم

int gSize;

// العدد الحالي من القمم

linkedListGraph<vType> graph [size];

// مصفوفة من مؤشرات لعمل قوائم التجاور (قوائم متصلة).

في الجزء المتبقي من هذا الفصل عندما نقوم بكتابة خوارزميات الرسم البياني في C++ نفترض أن القمم البالغة n للرسوم البيانية مرقمة 0، 1، ...، n - 1. لهذا يكون نوع القمة عدد صحيح. سوف نستمر في استخدام القوالب في حالة أن المستخدم يريد طريقة أخرى لتحديد نوع القمة ونترك للمستخدم عمل التعديلات اللازمة على الخوارزميات.

تعريفات دالات عنصر الفئة graphType تتم مناقشتها فيما بعد.
 يكون الرسم البياني فارغاً اذا كان عدد القمم بالغ صفر – أي اذا كان gSize يبلغ صفر. لهذا يكون تعريف الدالة isEmpty كما يلي:

```
template<class vType, int size>
bool graphType<vType, size>::isEmpty()
{
    return(gSize == 0);
}
```

تعريف الدالة creatGraph يعتمد على كيفية ادخال البيانات في البرنامج. لاهداف توضيحية نفترض أن البيانات للبرنامج يتم ادخالها من ملف ما.
 يتم حث المستخدم على ملف المدخلات. البيانات الموجودة في الملف تظهر بالصيغة التالية:

```
5
999- ... 4 2 0
999- ... 8 6 3 1
...
```

السطر الأول من المدخلات يحدد عدد القمم في الرسم البياني وأول مدخل في السطور المتبقية يحدد القمة وجميع المدخلات المتبقية في السطر (فيما عدا الأخير) تحدد القمم المجاورة للقمة وينتهي كل سطر بالعدد -999.

باستخدام هذه التحويلات يكون تعريف الدالة createGraph هو:

```

template<class vType, int size>
void graphType<vType, size>::createGraph()
{
    ifstream infile;
    char fileName[50];

    vType vertex;
    vType adjacentVertex;

    if(gSize != 0) //if the graph is not empty, make it empty
        clearGraph();

    cout<<"Enter the input file name: ";
    cin>>fileName;
    cout<<endl;

    infile.open(fileName);

    if(!infile)
    {
        cerr<<"Cannot open the input file."<<endl;
        return;
    }

    infile>>gSize; //get the number of vertices
    for(int index = 0; index < gSize; index++)
    {
        infile>>vertex;
        infile>>adjacentVertex;

        while(adjacentVertex != -999)
        {
            graph[vertex].insertLast(adjacentVertex);
            infile>>adjacentVertex;
        } //end while
    } //end for

    infile.close();
} //end createGraph

```

الدالة clearGraph تقوم بتفريغ الرسم البياني عن طريق ازالة تخصيص الذاكرة التي تحتلها كل قائمة متصلة ثم تحديد عدد القمم عند صفر.

```

template<class vType, int size>
void graphType<vType, size>::clearGraph()
{
    int index;

    for(index = 0; index < gSize; index++)
        graph[index].destroyList();

    gSize = 0;
}

```

تعريف الدالة printGraph معطى فيما بعد.

```

template<class vType, int size>
void graphType<vType, size>::printGraph() const
{
    int index;

    for(index = 0; index < gSize; index++)
        cout<<index<<" "<<graph[index]<<endl;

    cout<<endl;
} //end printGraph

```

تعريف المقوم الافتراضي والمدمر معطيان فيما يلي:

```
//default constructor
template<class vType, int size>
graphType<vType, size>::graphType()
{
    maxSize = size;
    gSize = 0;
}

//destructor
template<class vType, int size>
graphType<vType, size>::~~graphType()
{
    clearGraph();
}
```

اجتيازات الرسم البياني:

معالجة الرسم البياني يتطلب القدرة على اجتياز الرسم البياني. يقوم هذا القسم بمناقشة خوارزميات اجتياز الرسم البياني.

اجتياز الرسم البياني مماثل لاجتياز شجرة ثنائية الا أن اجتياز الرسم البياني أكثر تعقيداً قليلاً. تذكر أن الشجرة الثنائية ليس لها دورات بدءاً من العقدة الجذرية يمكننا اجتياز الشجرة بأكملها. من الجهة الأخرى قد يحتوي الرسم البياني على دورات وقد نكون غير قادرين على اجتياز الرسم البياني بأكمله من قمة واحدة (على سبيل المثال اذا لم يكن الرسم البياني متصلاً). لهذا يجب أن نتتبع القمم التي تمت زيارتها. يجب كذلك أن نجتاز الرسم البياني من كل قمة (لم يتم زيارتها) بالرسم البياني. هذا يؤكد أنه تم اجتياز الرسم البياني بأكمله.

خوارزميتان اجتياز الرسم البياني الأكثر شيوعاً هما **الاجتياز الأول بالعمق** و**الاجتياز الأول بالعرض** ويتم توضيحهما فيما بعد. للتبسيط نفترض أنه عندما تتم زيارة قمة ما يتم اخراج مؤشرها. فضلاً عن هذا تتم زيارة كل قمة مرة واحدة فقط. اننا نستخدم المصفوفة bool المسماة visited لتتبع القمم التي تتم زيارتها.

الاجتياز الأول بالعمق:

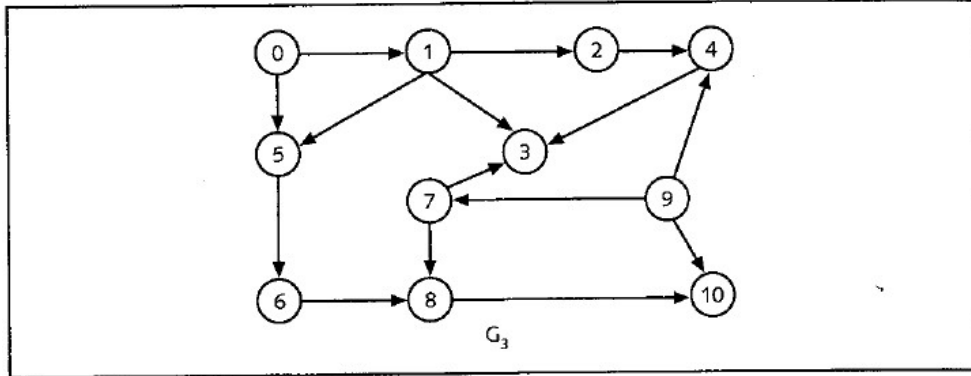
الاجتياز الأول بالعمق مماثل لاجتياز شجرة ثنائية قبل الترتيب. الخوارزمية العامة هي:

لكل قمة v في الرسم البياني

اذا لم تتم زيارة v

ابدأ الاجتياز الأول بالعمق من عند v

انظر الى الرسم البياني G3 في شكل 4-12. انه موضح هنا مرة أخرى كشكل 7-12 لسهولة المرجعية.



شكل 7-12: الرسم البياني المباشر G3

الترتيب الأول بعمق قمم الرسم البياني G3 في شكل 7-12 هو:

9 7 10 8 6 5 3 4 2 1 0

الخوارزمية العامة للقيام بالاجتياز الأول للعمق عند عقدة معطاة تكون:

1. وضع علامة عند العقدة v بأنه تمت زيارتها.

2. زيارة العقدة.

3. لكل قمة u مجاورة لـ v

إذا لم تتم زيارة v

ابدأ الاجتياز الأول للعمق عند u

يتضح أن هذه خوارزمية تكرارية. اننا نستخدم دالة تكرارية dft لتطبيق هذه الخوارزمية. القمة التي يتم من عندها بدء الاجتياز الأول للعمق والمصفوفة visited يتم تمريرهما كعوامل لهذه الدالة.

```
template<class vType, int size>
void graphType<vType, size>::dft(vType v, bool visited[])
{
    vType w;
    vType *adjacencyList;    //array to retrieve the
                           //adjacent vertices
    adjacencyList = new vType[gSize];
    int allLength = 0;    //the number of adjacent vertices
    visited[v] = true;
    cout<<" "<<v<<" ";    //visit the vertex
    graph[v].getAdjacentVertices(adjacencyList, allLength);
                           //retrieve the adjacent vertices into adjacencyList
    for(int index = 0; index < allLength; index++) //for each
    {                                              //vertex adjacent to v
        w = adjacencyList[index];
        if(!visited[w])
            dft(w, visited);
    } //end for
    delete [] adjacencyList;
} //end dft
```

بعد هذا نقوم باعطاء تعريف الدالة depthFirstTraversal لتطبيق الاجتياز الأول لعمق الرسم البياني.

```
template<class vType, int size>
void graphType<vType, size>::depthFirstTraversal()
{
    bool *visited;    //array to keep track of the visited
                      //vertices
    visited = new bool[gSize];

    int index;

    for(index = 0; index < gSize; index++)
        visited[index] = false;

    for(index = 0; index < gSize; index++) //for each vertex
        if(!visited[index])                //that is not visited,
            dft(index, visited);            //do a depth first
                                           //traversal

    delete [] visited;
} //end depthFirstTraversal
```

الدالة depthFirstTraversal تقوم بعمل اجتياز أول لعمق الرسم البياني بأكمله. تعريف الدالة dftAtVertex التي تقوم بعمل اجتياز أول بالعمق عند قمة معطاة يكون كما يلي:

```
template<class vType, int size>
void graphType<vType, size>::dftAtVertex(vType vertex)

{
    bool *visited;

    visited = new bool[gSize];

    for(int index = 0; index < gSize; index++)
        visited[index] = false;

    dft(vertex, visited);

    delete [] visited;
} //end dftAtVertex
```

الاجتياز الأول بالعرض:

الاجتياز الأول بالعرض للرسم البياني مماثل لاجتياز الشجرة الثنائية مستوى تلو مستوى (تتم زيارة العقد عند كل مستوى من اليسار الى اليمين). تتم زيارة جميع العقد عند أي مستوى i قبل زيارة العقدة الموجودة عند المستوى $i + 1$.

الترتيب الأول بعرض قمم الرسم البياني G_3 (شكل 7-12) هو:

9 7 10 8 4 6 3 2 5 1 0

بالنسبة الى الرسم البياني G_3 نبدأ الاجتياز بالعرض عند القمة صفر. بعد زيارة القمة صفر نقوم بعد هذا بزيارة القمم المتصلة مباشرةً بها والتي لم تتم زيارتها وهي 1 و 5. بعد هذا نقوم بزيارة القمم المتصلة مباشرةً ب 1 والتي لم تتم زيارتها وهي 2 و 3. بعد هذا نزور القمم التي تتصل مباشرةً ب 5

والتي لم تتم زيارتها وهي 6. بعد هذا نقوم بزيارة القمم التي تتصل مباشرة ب2 والتي لم تتم زيارتها وهكذا.

كما في حالة الاجتياز الأول بالعمق وبما أنه قد لا يكون من الممكن اجتياز الرسم البياني بأكمله من قمة واحدة فان الاجتياز الأول بالعرض أيضاً يجتاز الرسم البياني من كل عقدة لم تتم زيارتها بدءاً من العقدة الأولى يتم اجتياز الرسم البياني بقدر الامكان وعندها نذهب الى القمة التالية التي لم تتم زيارتها. لتطبيق خوارزمية البحث الأول بالعرض نستخدم طابور. الخوارزمية العامة هي:

أ. لكل قمة v في الرسم البياني

إذا لم تتم زيارة v

أضف v الى الطابور // ابدأ البحث الأول بالعرض عند v .

ب. ضع علامة أمام v بأنه تمت زيارتها.

ت. بينما الطابور غير فارغ

ت.1. أزل القمة u من الطابور.

ت.2. استعد القمم المجاور لـ u .

ت.3. لكل قمة w مجاورة لـ u

إذا لم تتم زيارة w

ت.1.3. أضف w الى الطابور

ت.2.3. ضع علامة أمام w أنه تمت زيارتها

دالة C++ التالية breadthFirstTraversal تطبق هذه الخوارزمية:

```
template<class vType, int size>
void graphType<vType, size>::breadthFirstTraversal()
```

```

{
    linkedQueueType<vType> queue;
    vType u;

    bool *visited;
    visited = new bool[gSize];

    for(int ind = 0; ind < gSize; ind++)
        visited[ind] = false;    //initialize the array
                                   //visited to false

    vType *adjacencyList;
    adjacencyList = new vType[gSize];

    int allLength = 0;

    for(int index = 0; index < gSize; index++)
        if(!visited[index])
        {
            queue.addQueue(index);
            visited[index] = true;
            cout<<" "<<index<<" ";

            while(!queue.isEmptyQueue())
            {
                u = queue.front();
                queue.deleteQueue();
                graph[u].getAdjacentVertices(adjacencyList, allLength);
                for(int w = 0; w < allLength; w++)
                    if(!visited[adjacencyList[w]])
                    {
                        queue.addQueue(adjacencyList[w]);
                        visited[adjacencyList[w]] = true;
                        cout<<" "<<adjacencyList[w]<<" ";
                    }
                //end if
            }
            //end while
        }
        //end if

    delete [] visited;
    delete [] adjacencyList;
} //end breadthFirstTraversal

```

بعد ادخال خوارزميات اجتياز الرسم البياني السابقة يكون تعريف الفئة graphType هو:

```

template<class vType, int size>
class graphType
{
public:
    bool isEmpty();
    //Function to determine whether the graph is empty.
    //Postcondition: Returns true if the graph is empty;
    //                otherwise, returns false.

```

bool isEmpty ()

//دالة لتحديد ما اذا ان الرسم البياني فارغاً .
 // شرط تالي تنتج true اذا ان الرسم البياني فارغاً
 // وبخلاف هذا تنتج false.

```
void createGraph();
//Function to create the graph using the adjacency list
//representation.
//Postcondition: The graph is created in the form of
//adjacency lists.
void clearGraph();
//Function to deallocate the memory occupied by the linked
//lists and the array of pointers pointing to the linked
//lists.
void printGraph() const;
//Function to print the graph.

void depthFirstTraversal();
//Function to perform the depth first traversal of
//the entire graph.
void dftAtVertex(vType vertex);
//Function to perform the depth first traversal of
//the graph at a node specified by the parameter vertex.

void breadthFirstTraversal();
//Function to perform the breadth first traversal of
//the entire graph.

graphType();
//default constructor
//Postcondition: The graph size is set to 0, that is,
//gSize = 0; maxSize = size.

~graphType();
//destructor
//Postcondition: The storage occupied by the graph
//is deallocated.

protected:
int maxSize;    //maximum number of vertices
int gSize;      //current number of vertices
linkedListGraph<vType> graph[size]; //array of pointers
//to create the adjacency lists (linked lists)

private:
void dft(vType v, bool visited[]);
//Function to perform the depth first traversal of
//the graph at a particular node.
};
```

void createGraph ()

// دالة لعمل الرسم البياني باستخدام تمثيل قائمة التجاور.
 // شرط تالي يتم عمل الرسم البياني في شكل قوائم تجاور.

```
void clearGraph ( )
// دالة لازالة تخصيص الذاكرة التي تشغلها القوائم المتصلة
// ومصفوفة المؤشرات التي تشير الى القوائم المتصلة.
```

```
void printGraph ( ) const;
// دالة لطباعة الرسم البياني.
```

```
void depthFirstTraversal ( );
// دالة لأداء الاجتياز الأول لعمق الرسم البياني بأكمله.
```

```
void dftAtVertex (vType vertex);
// دالة لأداء الاجتياز الأول لعمق الرسم البياني
// عند نقطة يحددها العامل vertex.
```

```
void breadthFirstTraversal ( );
// دالة لأداء الاجتياز الأول لعرض الرسم البياني بأكمله.
```

```
graphType ( );
// مقوم افتراضي
// شرط تالي: يتم ضبط حجم الرسم البياني عند صفر أي أن
// gSize = صفر و size = maxSize.
```

```
-graphType ( );
// مدمر
// شرط تالي: يتم ازالة تخصيص الذاكرة التي
// يحتلها الرسم البياني.
```

محمية:

```
int maxSize; // العدد الأقصى من القمم
int gSize; // العدد الحالي من القمم
```

linkedListGraph<vType> graph [size];
 // مصفوفة من مؤشرات لعمل قوائم التجاور
 (قوائم متصلة).

خاصة:

void dft (vType v, bool visited []);
 // دالة لأداء الاجتياز الأزل لعمق الرسم
 // البياني عند عقدة معينة.

مع استمرارنا في مناقشة خوارزميات الرسم البياني سوف نقوم بكتابة دالات C++ لتطبيق خوارزميات محددة وبهذا سوف نستمد (باستخدام التوريث) فئات جديدة من الفئة graphType.

خوارزمية المسار الأقصر:

نظرية الرسم البياني لها العديد من التطبيقات. على سبيل المثال يمكننا استخدام الرسوم البيانية لتوضيح كيفية ارتباط المواد الكيميائية المختلفة أو لتوضيح مسارات الخطوط الجوية. كما يمكن استخدامها لتوضيح تركيب الطرق السريعة لمدينة أو ولاية أو دولة. يمكن تحديد الحواف التي تصل بين قمتين عند عدد حقيقي غير سالب يسمى **وزن الحافة**. إذا كان الرسم البياني يمثل بنية الطرق السريعة يمكن للوزن أن يقوم بتمثيل المسافة بين مكانين أو زمن السفر من مكان الى آخر. هذه الرسوم البيانية يطلق عليها **رسوم بيانية موزونة**.

لتكن G رسم بياني موزون ولتكن u و v قمتين في G ولتكن p مسار في G من u الى v . **وزن المسار** P هو مجموع أوزان جميع الحواف على المسار P الذي يسمى أيضاً **وزن** v من u عبر P . لتكن G رسم بياني موزون يمثل بنية طرق سريعة. افترض أن وزن الحافة يمثل زمن السفر. على سبيل المثال فان مندوب المبيعات يريد من أجل تخطيط رحلات عمل شهرية أن يوجد **أقصر مسار** (أي المسار ذو أقصر وزن) من مدينته الى كل مدينة أخرى في الرسم البياني. العديد من هذه المشكلات تتواجد والتي نريد فيها ايجاد أقصر مسار من قمة معطاة تسمى **المصدر** الى كل قمة أخرى في الرسم البياني.

يقوم هذا القسم بوصف **خوارزمية أقصر مسار** التي تسمى كذلك **خوارزمية الطمع** التي قام بعملها ديجكسترا.

ليكن G رسم بياني به عدد n من القمم حيث $n > 0$. ولتكن $V(G) = \{v_1, v_2, \dots, v_n\}$. لتكن W مصفوفة ثنائية $n \times n$ بحيث:

$$W(i, j) = \begin{cases} w_{ij} & \text{if } (v_i, v_j) \text{ is an edge in } G \text{ and } w_{ij} \text{ is the weight of the edge } (v_i, v_j) \\ \infty & \text{if there is no edge from } v_i \text{ to } v_j \end{cases}$$

مدخلات البرنامج هي الرسم البياني ومصفوفة الوزن المرتبطة بالرسم البياني. لجعل ادخال البيانات أكثر سهولة نقوم بمد تعريف الفئة graphType (باستخدام التوريث) ونضيف الدالة creatWeightedGraph لعمل الرسم البياني ومصفوفة الوزن المرتبطة بالرسم البياني. لنطلق على هذه الدالة weightedGraphType. سوف يتم أيضاً اضافة الدالات لتطبيق خوارزمية أقصر مسار الى هذه الدالة.

تعريف الفئة weightedGraphType هو:

```
template<class vType, int size>
class weightedGraphType: public graphType<vType, size>
{
public:
    void createWeightedGraph();
        //Function to create the graph and the weight matrix.
    void shortestPath(vType vertex);
        //Function to determine the smallest weight from the
        //vertex, that is, the source, to every other vertex
        //in the graph.
    void printShortestDistance(vType vertex);
        //Function to print the smallest weight from the
        //source to all the other vertices in the graph.
```

عامة:

void createWeightedGraph ();
// دالة لعمل الرسم البياني ومصفوفة الوزن.

void shortestPath (vType vertex);
// دالة لتحديد أصغر وزن من القمة المصدر
// الى كل قمة أخرى في الرسم البياني.

void printShortestDistance (vType vertex);
// دالة لطباعة أصغر وزن من المصدر
// الى جميع القمم الأخرى في الرسم البياني.

```
protected:
    double weights[size][size]; //weight matrix
    double smallestWeight[size]; //smallest weight from the
        //source to the other
        //vertices
};
```

محمية:

double weights [size] [size]; // مصفوفة الوزن.

double smallestWeight [size]; // أصغر وزن من المصدر الى القيم الاخرى.

تعريف الدالة createWeightGraphs متروك كتمرين لك. بعد هذا نصف خوارزمية أقصر مسار.

أقصر مسار:

باعطاء قمة لتكن vertex (القمة المصدر) يقوم هذا القسم بوصف خوارزمية أقصر مسار.

الخوارزمية العامة هي:

1. تهيئة المصفوفة smallestWeight حتى يكون

`smallestWeight[u] = weights[vertex, u]`

2. تحديد `smallestWeight [vertex] = صفر`.

3. ايجاد القمة v الأكثر قرباً من القمة التي لم يتم تحديد أقصر طريق لها.

4. تحديد v كلقمة (التالية) التي يتم ايجاد أصغر وزن لها.

5. لكل قمة w في G بحيث لم يتم تحديد أقصر مسار من القمة الى w ومع وجود حافة (v,

w) اذا كان وزن المسار الى w من خلال v أصغر من وزنه الحالي تحديث وزن w الى وزن

v + وزن الحافة (v, w).

بما أن هناك عدد n من القيم أعد الخطوات من 3 الى 5 عدد n - 1 مرة.

المثال 5-12 يوضح خوارزمية أقصر طريق. (اننا نستخدم مصفوفة bool المسماة weightFound

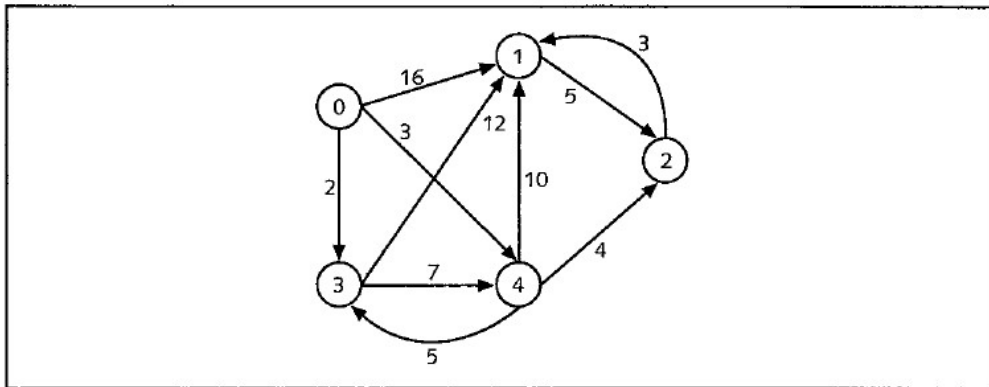
لنتبع القيم التي تم ايجاد أصغر وزن لها من القمة المصدر. اذا كان أصغر وزن لقمة ما من المصدر تم

ايجاده فاذن المدخل الموافق لهذه القمة في المصفوفة weightFound يتم ضبطه عند true وبخلاف

هذا يكون المدخل المتوافق هو false).

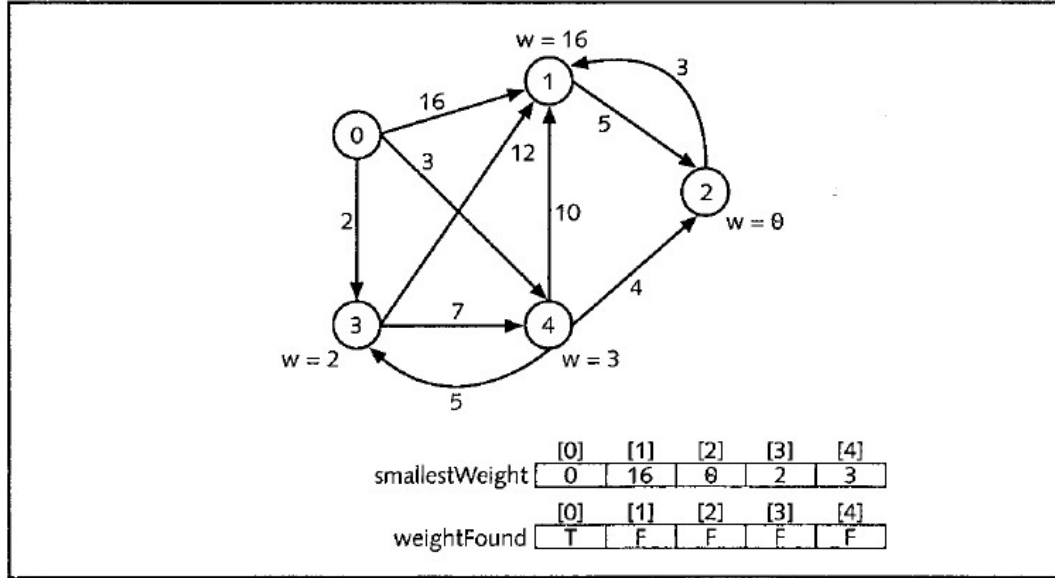
مثال 5-12:

لتكن G الرسم البياني الموضح في شكل 8-12.



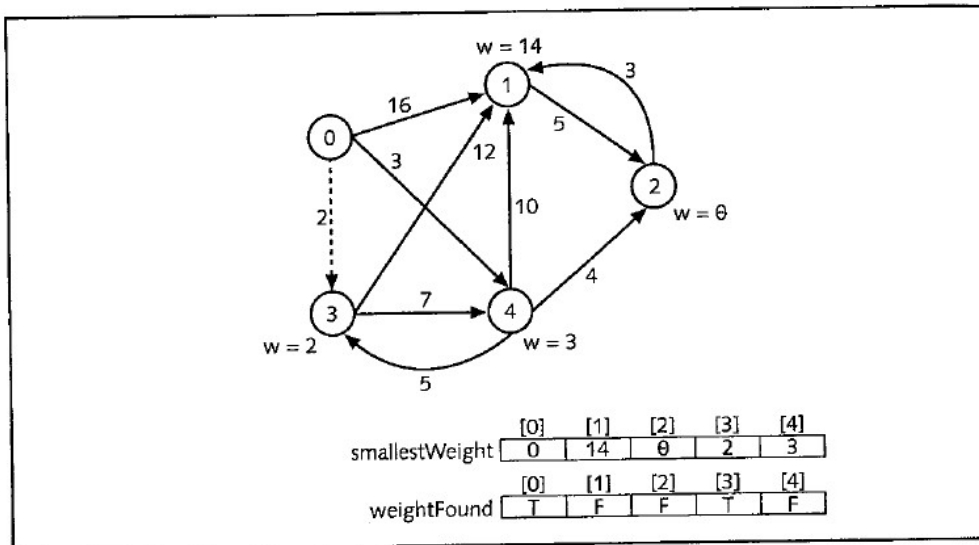
شكل 8-12: الرسم البياني الموزون G

افترض أن القمة المصدر لـ G هي صفر. الرسم البياني يوضح وزن كل حافة. بعد تنفيذ الخطوتين 1 و 2 يكون الرسم البياني الناتج كما هو موضح في شكل 9-12. (في شكل 9-12 الرمز θ يعني ∞).



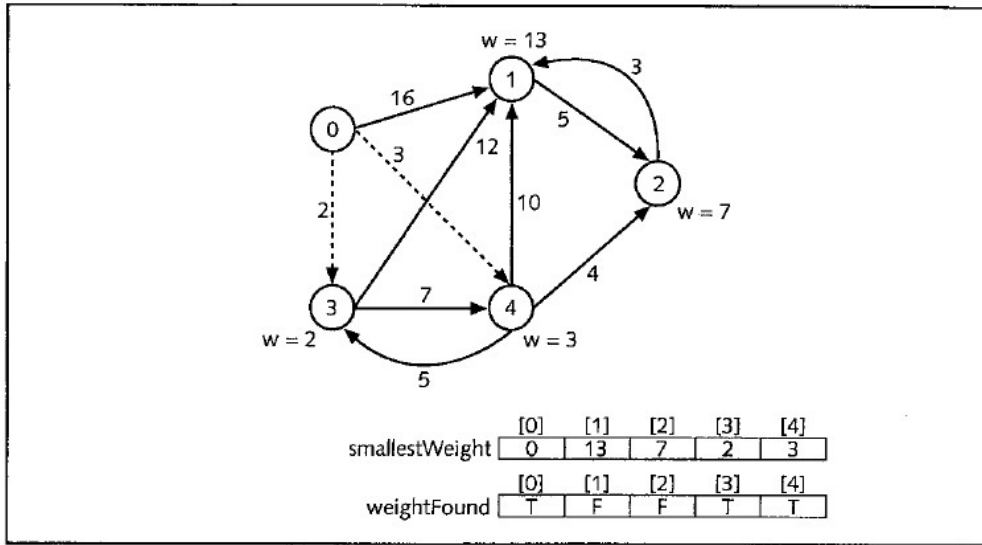
شكل 9-12: الرسم البياني بعد تنفيذ الخطوتين 1 و 2

أولاً نختار القمة 3 ونضع علامة true عند [3] weightFound. يتضح أن وزن المسار 3-1-0 وهو 14 من صفر إلى 1 يكون أصغر من وزن المسار 1-0. لهذا نقوم بتحديث smallestWeight [1] إلى 14. الشكل 10-12 يوضح الرسم البياني الناتج. (السهم المنقط يوضح أقصر مسار من المصدر وهو صفر إلى القمة).



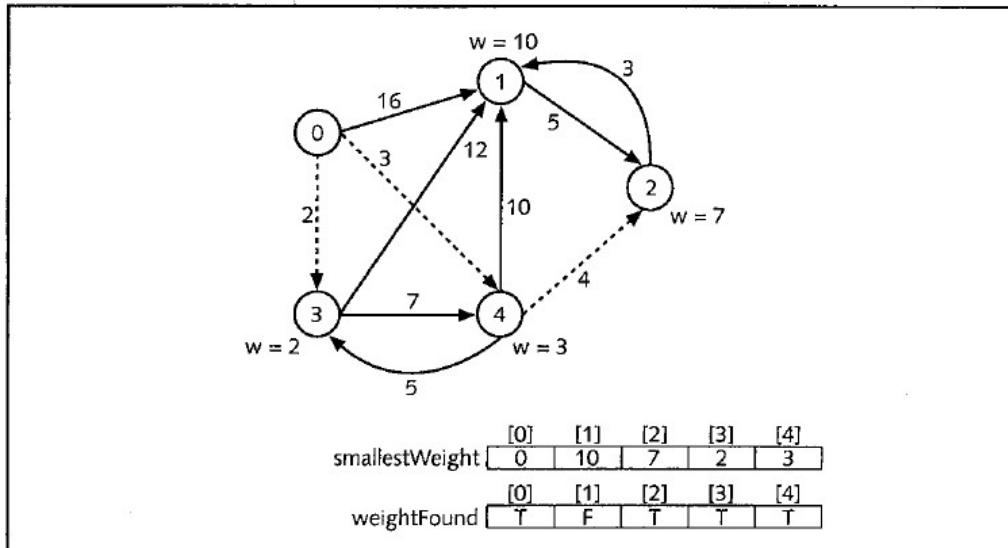
شكل 10-12: الرسم البياني بعد المكرر الأول من الخطوات 3 و 4 و 5

الآن نختار القمة 4 لأن هذه القمة الموجودة في المصفوفة `smallestWeight` وهي صاحبة أصغر وزن ومدخلها المقابل في المصفوفة `weightFound` يكون `false`. بعد هذا نكرر الخطوات السابقة ونضبط `weightFound [4]` عند `true`. يتضح أن وزن المسار 0-4-1 وهو 13 أصغر من الوزن الحالي للقمة 1 وهو 14. لهذا نقوم بتحديث `smallestWeight [1]` وبالمثل نقوم بتحديث `smallestWeight [2]`. الشكل 11-12 يوضح الرسم البياني الناتج.



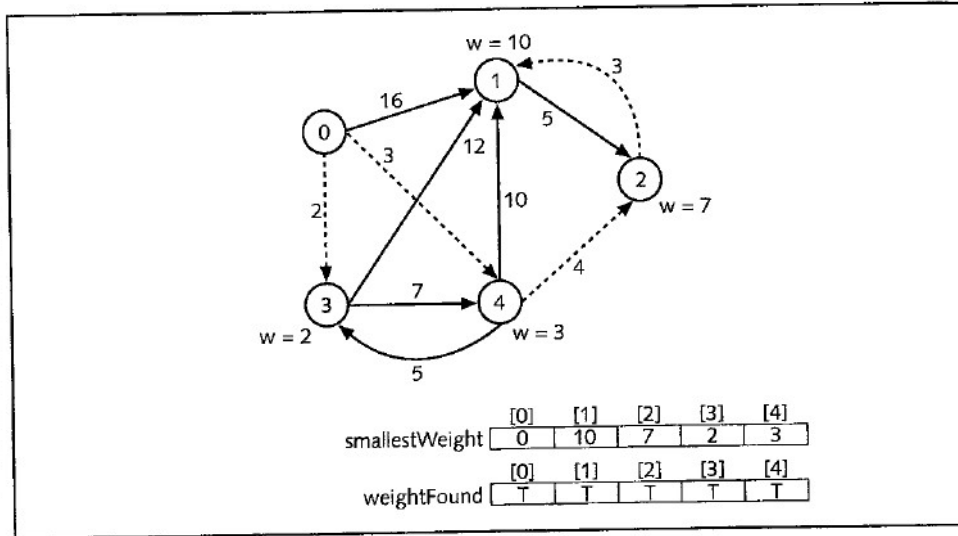
شكل 11-12: الرسم البياني بعد المكرر الثاني من الخطوات 3 و4 و5

القمة المختارة بعد هذا هي 2. نقوم بضبط `weightFound [2]` عند `true`. يتضح أن وزن المسار 0-2-4 هو 10 من صفر الى واحد أصغر من الوزن الحالي للقمة 1 (وهي 13). لهذا نقوم بتحديث `smallestWeight [1]`. الشكل 12-12 يوضح الرسم البياني الناتج.



شكل 12-12: الرسم البياني بعد المكرر الثالث من الخطوات 3 و4 و5

في النهاية يتم اختيار القمة 1 ويتم وضع علامة على المسار. الشكل 12-13 يوضح الرسم البياني الناتج.



شكل 12-13: الرسم البياني بعد المكرر الرابع من الخطوات 3 و4 و5

دالة C++ التالية وهي shortestPath تطبق الخوارزمية التالية:

```
template<class vType, int size>
void weightedGraphType<vType, size>::shortestPath(vType vertex)
{
    int i, j;
    int v;
    double minWeight;

    for(j = 0; j < gSize; j++)
        smallestWeight[j] = weights[vertex][j];

    bool weightFound[size];
    for(j = 0; j < gSize; j++)
        weightFound[j] = false;

    weightFound[vertex] = true;
    smallestWeight[vertex] = 0;
```

```

for(i = 0; i < gSize - 1; i++)
{
    minWeight = infinity;
    for(j = 0; j < gSize; j++)
        if(!weightFound[j])
            if(smallestWeight[j] < minWeight)
            {
                v = j;
                minWeight = smallestWeight[v];
            }

    weightFound[v] = true;
    for(j = 0; j < gSize; j++)
        if(!weightFound[j])
            if(minWeight + weights[v][j] < smallestWeight[j])
                smallestWeight[j] = minWeight + weights[v][j];
} //end for
} //end shortestPath

```

لاحظ أن الدالة shortestPath تقوم فقط بتسجيل وزن أقصر مسار من المصدر إلى القمة. اننا نترك لك كتمرين تعديل هذه الدالة حتى يتم تسجيل أقصر مسار من المصدر إلى القمة أيضاً.

تعريف الدالة printShortestDistance هو:

```

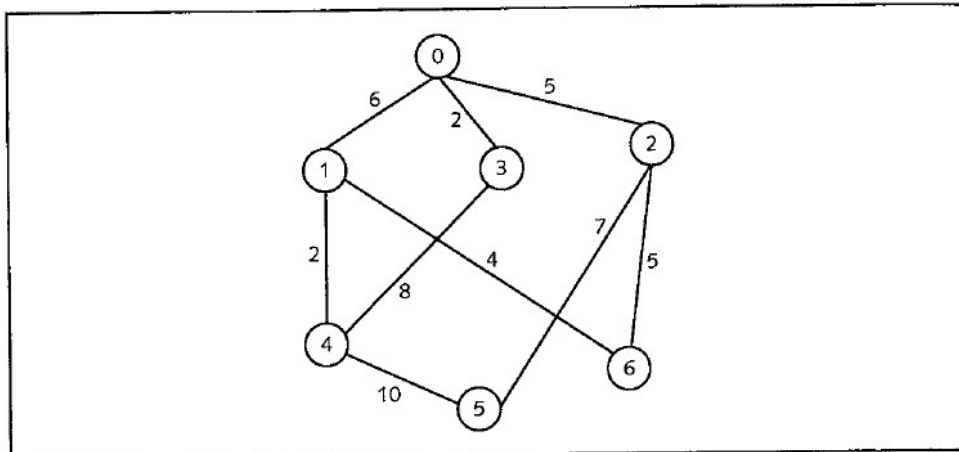
template<class vType, int size>
void weightedGraphType<vType, size>::printShortestDistance
    (vType vertex)
{
    cout<<fixed<<showpoint<<setprecision(2);
    cout<<"Source vertex: "<<vertex<<endl;
    cout<<"Shortest distance from the source to each vertex."
        <<endl;
    cout<<"Vertex Shortest_Distance"<<endl;

    for(int j = 0; j < gSize; j++)
        cout<<setw(4)<<j<<setw(12)<<smallestWeight[j]<<endl;
    cout<<endl;
}

```

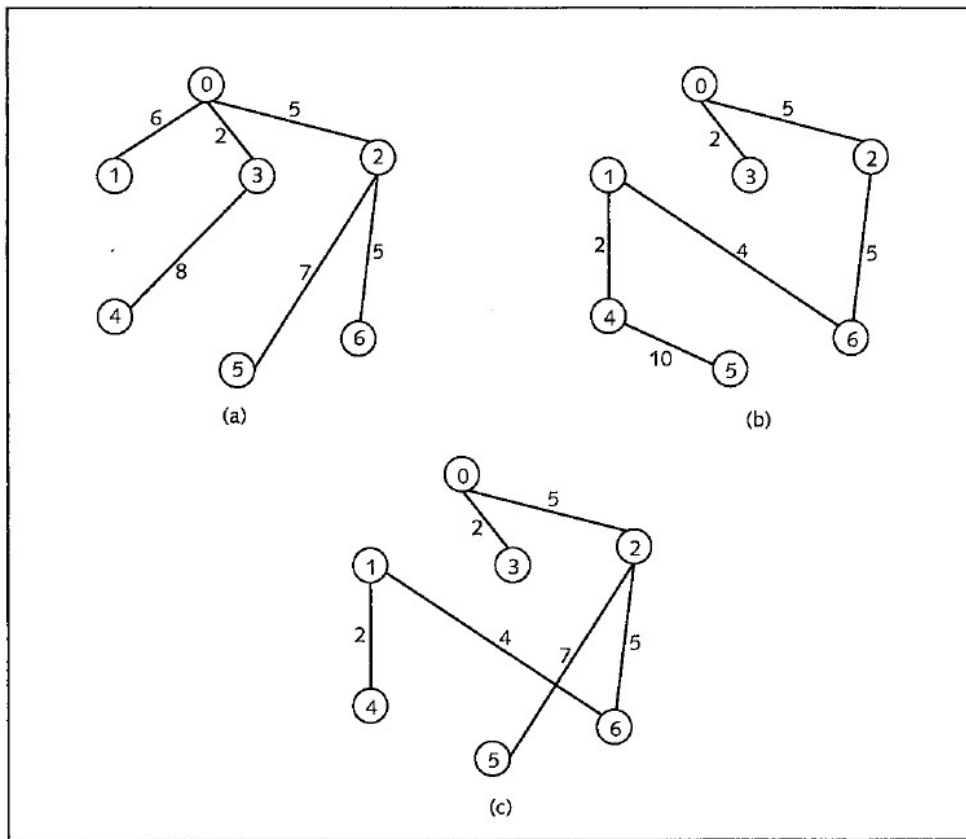
شجرة الامتداد الأدنى:

انظر الى الرسم البياني في شكل 12-14 الذي يمثل اصطصالات الخطوط الجوية لشركة ما بين سبعة مدن. الرقم على كل حافة يمثل بعض من عامل تكلفة الاحتفاظ بالاتصال بين المدن.



شكل 12-14: اتصالات الخطوط الجوية بين المدن وعامل تكلفة الاحتفاظ بالاتصالات

بسبب الصعوبات المالية تحتاج الشركة الى اغلاق العدد الأقصى من الاتصالات وأن تظل قادرة على الطيران من مدينة الى أخرى (قد لا تكون مباشرة). الرسوم البيانية في شكل 12-15 (أ) و(ب) و(ج) توضح ثلاثة حلول مختلفة.



شكل 12-15: الحلول الممكنة للرسم البياني للشكل 12-14

عامل التكلفة الكلية للاحتفاظ بالاتصالات المتبقية في شكل 12-15 (أ) هو 33 وفي شكل 12-15 (ب) يكون 28 وفي شكل 12-15 (ج) يكون 25. يتضح من هذه الحلول الثلاثة أن الحل المرجو هو الحل الموضح من الرسم البياني بشكل 12-15 (ج) لأنه يعطي عامل التكلفة الأقل. يطلق على الرسوم البيانية في شكل 12-15 أشجار امتداد للرسم البياني للشكل 12-14. لنقم بملاحظة ما يلي من الرسوم البيانية الموجودة في شكل 12-15. كل من الرسوم البيانية الموجودة في شكل 12-15 عبارة عن رسم بياني فرعي من الرسم البياني بشكل 12-14 وهناك مسار فريد من

عقدة الى أي عقدة أخرى. يطلق على مثل هذه الرسوم البيانية أشجار. هناك العديد من المواقف الأخرى التي يتم فيها تقديم رسم بياني موزون ونحتاج فيها الى تحديد رسم بياني كما في شكل 12-15 ذات أصغر وزن. في هذا القسم نعطي خوارزمية لتحديد هذه الرسوم البيانية. بالرغم من هذا فاننا نقدم أولاً بعض المصطلحات.

الشجرة (الفارغة) T عبارة عن رسم بياني بسيط بحيث أنه اذا كانت u و v قمتين في T اذن يكون هناك مسار وحيد من u الى v . الشجرة التي يتم فيها تخصيص قمة محددة كجذر تسمى **شجرة جذرية**. اذا تم تحديد وزن للحواف في T يطلق على T **شجرة موزونة**. اذا كانت T شجرة موزونة يكون وزن T الذي يتم التعبير عنه بـ $W(T)$ هو مجموع أوزان جميع الحواف في T .

الشجرة T تسمى **شجرة امتداد** للرسم البياني G اذا كانت T رسم بياني فرعي من G بحيث $V(T) = V(G)$ أي أن جميع قمم G موجودة في T .

افترض أن G تعبر عن الرسم البياني لشكل 12-14. اذن الرسوم البيانية من شكل 12-15 توضح ثلاثة أشجار امتداد لـ G . لنقم بملاحظة النظرية التالية:

النظرية: الرسم البياني G له شجرة امتداد اذا كانت G متصلة.

من هذه النظرية ينتج أنه من أجل تحديد شجرة امتداد لرسم بياني ما يجب أن يكون الرسم البياني متصل.

لنكن G رسم بياني موزون. **شجرة الامتداد الأدنى** لـ G عبارة عن شجرة امتداد تمتلك الحد الأدنى من الوزن.

هناك خوارزمتان معروفتان جيداً وهما خوارزمية بريم وخوارزمية كروسكال لايجاد شجرة الامتداد الأدنى لرسم بياني. يقوم هذا القسم بمناقشة خوارزمية بريم لايجاد شجرة الامتداد الأدنى.

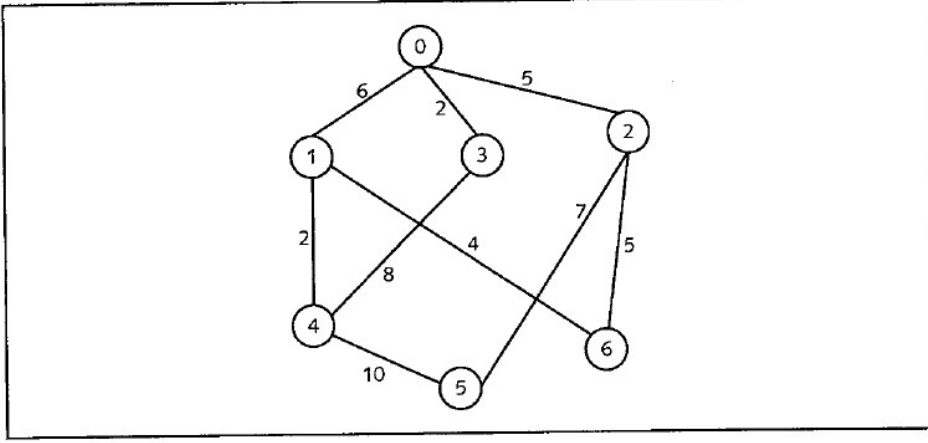
تقوم خوارزمية بريم ببناء الشجرة تكرارياً عن طريق اضافة حواف حتى يتم الحصول على شجرة امتداد أدنى. نبدأ بقمة مخصصة نطلق عليها القمة المصدر. عند كل تكرار تتم اضافة حافة جديدة لا تكمل دورة الى الشجرة.

لنكن G رسم بياني موزون بحيث $V(G) = n \{v_0, v_1, \dots, v_{n-1}\}$ هو عدد القمم موجب. لنكن v_0 القمة المصدر. لنكن T الشجرة المبنية جزئياً. مبدئياً تحتوي $V(T)$ على القمة المصدر وتكون $E(T)$ فارغة. عند التكرار التالي تتم اضافة قمة جديدة ليست في $V(T)$ الى $V(T)$ بحيث أنه توجد حافة من قمة في T الى القمة الجديدة حتى يكون للحافة المقابلة أصغر وزن. تتم اضافة الحافة المقابلة الى $E(T)$.

الصيغة العامة من خوارزمية بريم تكون كما يلي: (لتكن n عدد القمم في G):

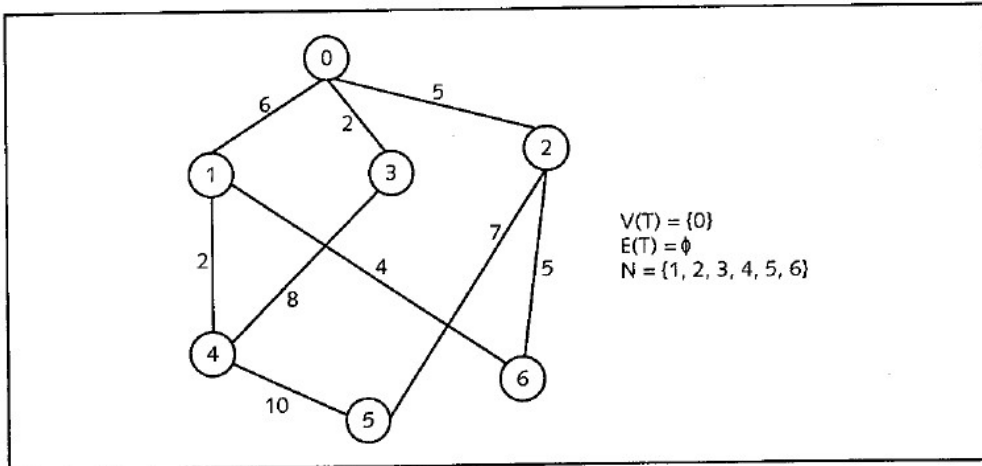
1. Set $V(T) = \{\text{source}\}$
2. Set $E(T) = \text{empty}$
3. for $i = 1$ to n
 - 3.1 minWeight = infinity;
 - 3.2 for $j = 1$ to n
 - if v_j is in $V(T)$
 - for $k = 1$ to n
 - if v_k is not in T and $\text{weight}[v_j][v_k] < \text{minWeight}$
 - {
 - endVertex = v_k ;
 - edge = (v_j, v_k) ;
 - minWeight = $\text{weight}[v_j][v_k]$;
 - }
 - 3.3 $V(T) = V(T) \cup \{\text{endVertex}\}$;
 - 3.4 $E(T) = E(T) \cup \{\text{edge}\}$;

لنقم بتوضيح خوارزمية بريم باستخدام الرسم البياني G من شكل 12-16 (وهو نفسه الرسم البياني للشكل 12-14).



شكل 12-16: رسم بياني موزون G

لتكن N معبرة عن قمم G الغير موجودة في T . افترض أن القمة المصدر هي 0. بعد تنفيذ الخطوات 1 و 2 تكون $V(T)$ و $E(T)$ و N كما هي موضحة في شكل 12-17.

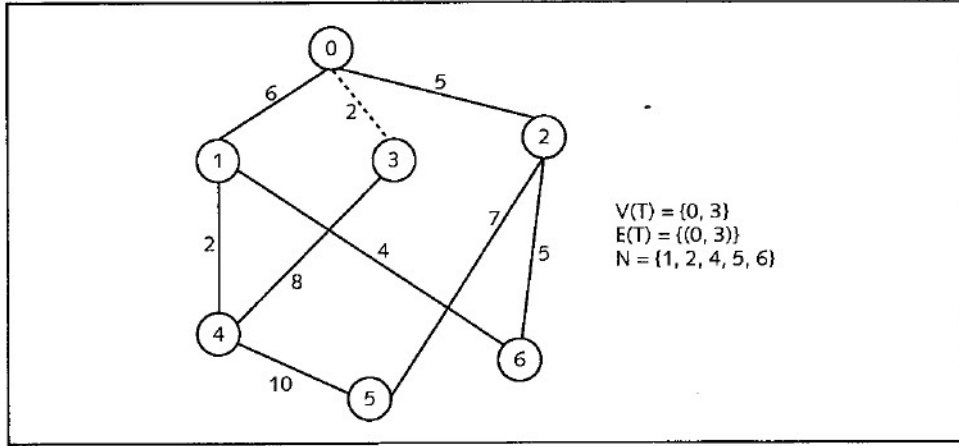


شكل 12-17: الرسم البياني G، وV(T)، وE(T)، وN بعد تنفيذ الخطوات 1 و2

الخطوة 2-3 تتحقق من الحواف التالية:

| الحافة | وزن الحافة |
|--------|------------|
| (1، 0) | 6 |
| (2، 0) | 5 |
| (3، 0) | 2 |

يتضح أن الحافة (3، 0) تمتلك أصغر وزن. لهذا تتم إضافة القمة 3 إلى V(T) وتتم إضافة الحافة (0، 3) إلى E(T). الشكل 12-18 يوضح الرسم البياني الناتج، وV(T)، وE(T)، وN. (السطور المنقطعة توضح الحواف في T)

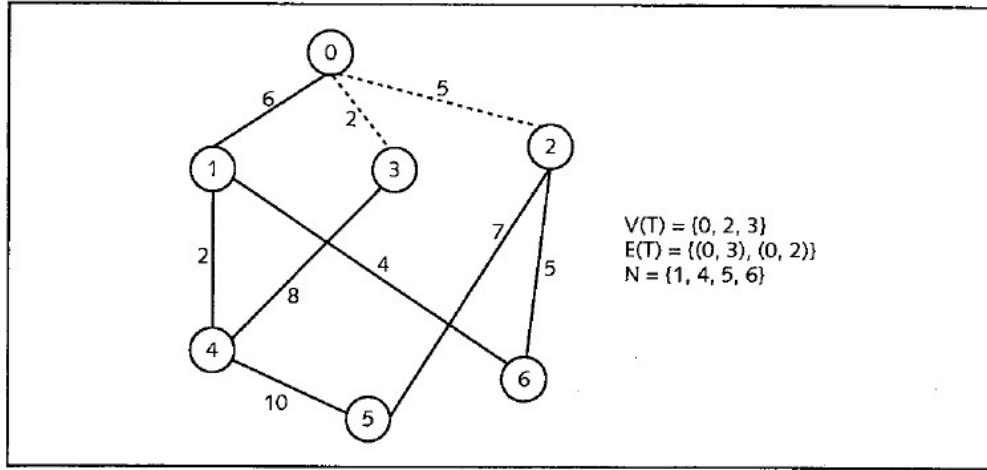


شكل 12-18: الرسم البياني G، وV(T)، وE(T)، وN بعد المكرر الأول من الخطوة 3

بعد هذا الخطوة 2-3 تتحقق من الحواف التالية:

| الحافة | وزن الحافة |
|--------|------------|
| (1، 0) | 6 |
| (2، 0) | 5 |
| (4، 3) | 8 |

يتضح أن الحافة (2، 0) تمتلك أصغر وزن. لهذا تتم إضافة القمة 2 إلى V(T) وتتم إضافة الحافة (0، 2) إلى E(T). الشكل 12-19 يوضح الرسم البياني الناتج، وV(T)، وE(T)، وN.

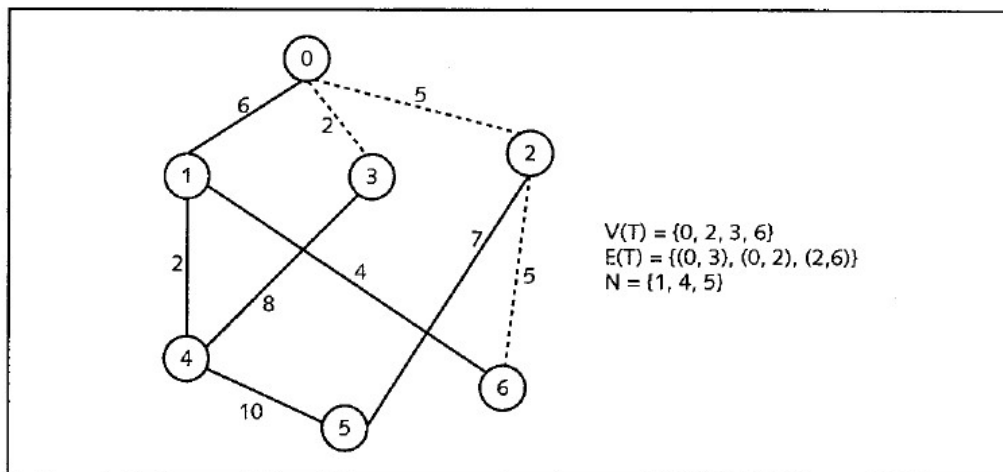


شكل 12-19: الرسم البياني G، و $V(T)$ ، و $E(T)$ ، و N بعد المكرر الثاني من الخطوة 3

عند المكرر التالي تحقق الخطوة 2-3 من الحواف التالية:

| الحافة | وزن الحافة |
|--------|------------|
| (1, 0) | 6 |
| (5, 2) | 7 |
| (6, 2) | 5 |
| (4, 3) | 8 |

يتضح أن الحافة (2, 6) تمتلك أصغر وزن. لهذا تتم إضافة القمة 6 إلى $V(T)$ وتتم إضافة الحافة (2, 6) إلى $E(T)$. الشكل 12-20 يوضح الرسم البياني الناتج، و $V(T)$ ، و $E(T)$ ، و N. (السطور المنقطعة توضح الحواف في T)

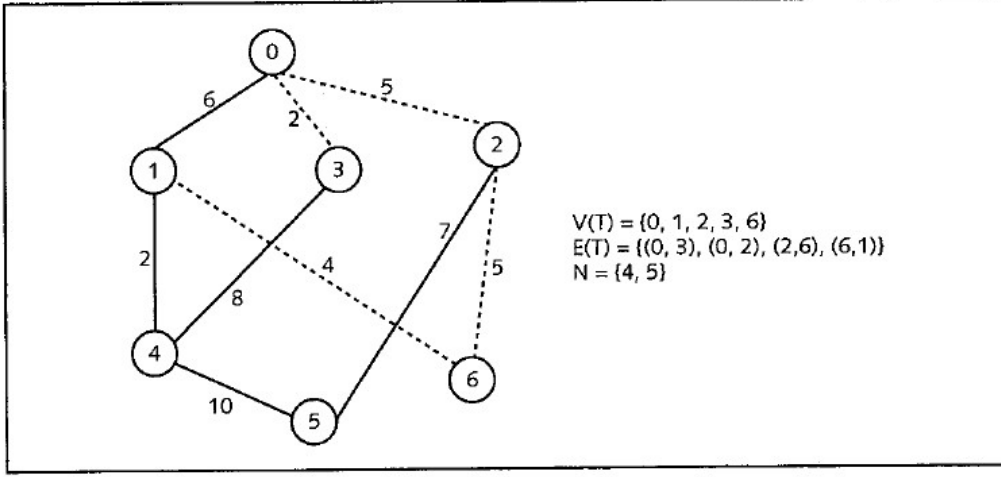


شكل 12-20: الرسم البياني G، و $V(T)$ ، و $E(T)$ ، و N بعد المكرر الثالث من الخطوة 3

عند الادخال التالي تتحقق الخطوة 2-3 من الحواف التالية:

| الحافة | وزن الحافة |
|--------|------------|
| (1, 0) | 6 |
| (5, 2) | 7 |
| (4, 3) | 8 |
| (1, 6) | 4 |

من الواضح أن الحافة (1, 6) تمتلك أصغر وزن. لهذا تتم اضافة القمة 1 الى $V(T)$ وتتم اضافة الحافة (1, 6) الى $E(T)$. الشكل 21-12 يوضح الرسم البياني الناتج، و $V(T)$ ، و $E(T)$ ، و N . (السطور المنقطه توضح الحواف في T)

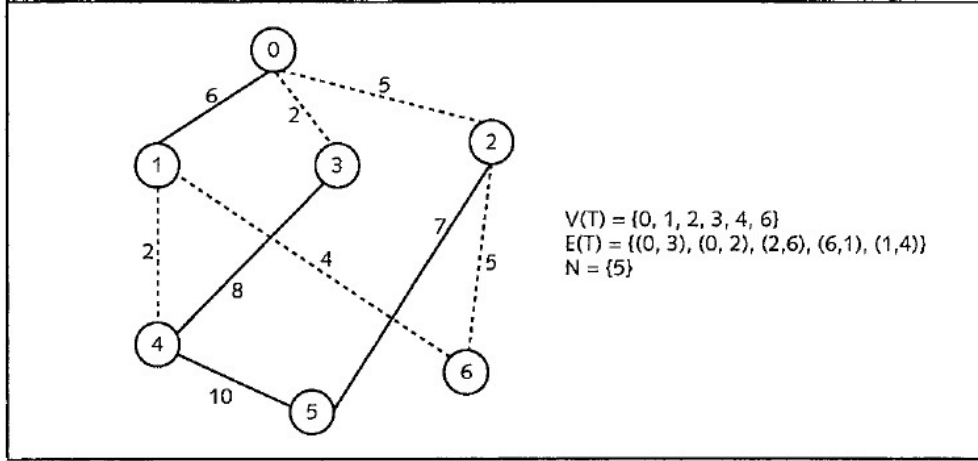


شكل 21-12: الرسم البياني G ، و $V(T)$ ، و $E(G)$ ، و N بعد المكرر الرابع من الخطوة 3

عند الادخال التالي تتحقق الخطوة 2-3 من الحواف التالية:

| الحافة | وزن الحافة |
|--------|------------|
| (4, 1) | 2 |
| (5, 2) | 7 |
| (4, 3) | 8 |

من الواضح أن الحافة (4, 1) تمتلك أصغر وزن. لهذا تتم اضافة القمة 4 الى $V(T)$ وتتم اضافة الحافة (4, 1) الى $E(T)$. الشكل 22-12 يوضح الرسم البياني الناتج، و $V(T)$ ، و $E(T)$ ، و N . (السطور المنقطه توضح الحواف في T)

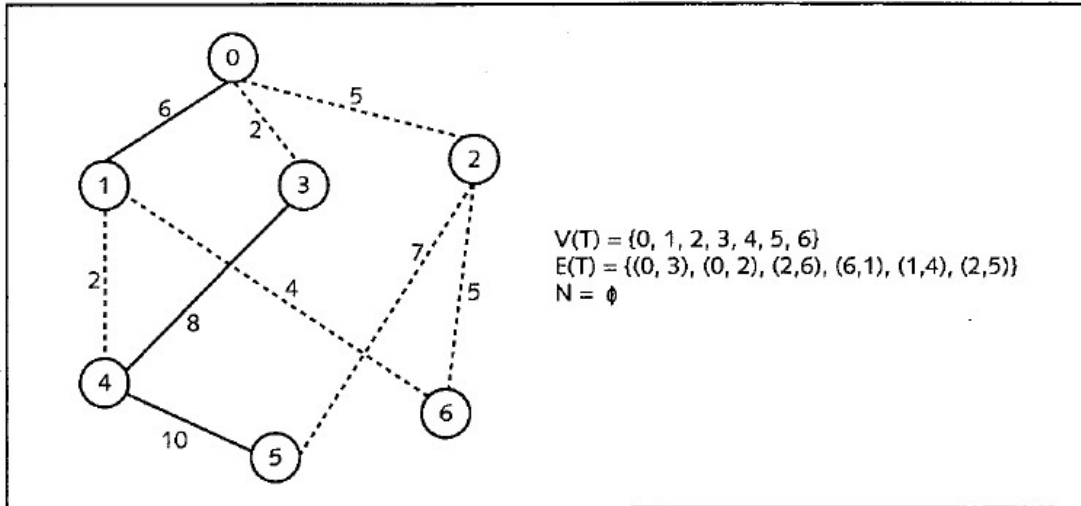


شكل 12-22: الرسم البياني G، و $V(T)$ ، و $E(G)$ ، و N بعد المكرر الخامس من الخطوة 3

عند الادخال التالي تتحقق الخطوة 2-3 من الحواف التالية:

| الحافة | وزن الحافة |
|--------|------------|
| (5, 2) | 7 |
| (5, 4) | 10 |

من الواضح أن الحافة (5, 2) تمتلك أصغر وزن. لهذا تتم إضافة القمة 5 إلى $V(T)$ وتتم إضافة الحافة (5, 2) إلى $E(T)$. الشكل 12-23 يوضح الرسم البياني الناتج، و $V(T)$ ، و $E(T)$ ، و N. (السطور المنقطه توضح الحواف في T)



شكل 12-23: الرسم البياني G، و $V(T)$ ، و $E(G)$ ، و N بعد المكرر السادس من الخطوة 3

السطور المنقطة توضح شجرة امتداد أدنى G ذات الوزن 25. قبل أن نعطي تعريف الدالة لتطبيق خوارزمية بريم انقم أولاً بتعريف شجرو الامتداد كنوع بيانات مجرد.

لتكن $msvt$ مصفوفة من النوع $bool$ بحيث أن $msvt[j]$ تكون $true$ اذا كانت القمة v_j موجودة في T وتكون $false$ بخلاف هذا. لتكن $edges$ (الحواف) مصفوفة بحيث أن $edges[j] = k$ اذا كان هناك حافة تصل القمتين v_k و v_j . افترض أن الحافة (v_k, v_j) موجودة في شجرة الامتداد الأدنى. لتكن $edgeWeights$ مصفوفة بحيث $edgeWeights[j]$ هي وزن الحافة (v_j, v_1) . باستخدام هذه التحويلات تقوم الفئة التالية بتعريف شجرة الامتداد كنوع بيانات مجرد:

```
template<class vType, int size>
class msTreeType: public graphType<vType, size>
{
public:
    void createSpanningGraph();
        //Function to create the graph and the weight matrix.
    void minimalSpanning(vType sVertex);
        //Function to create the edges of the minimal
        //spanning tree. The weight of the edges is also
        //saved in the array edgeWeights.
    void printTreeAndWeight();
        //Function to output the edges and the weight of the
        //minimal spanning tree.

protected:
    vType source;
    double weights[size][size];
    int edges[size];
    double edgeWeights[size];
};
```

`void createSpanningGraph ();`
 // دالة لعمل الرسم البياني ومصفوفة الوزن.

`void minimalSpanning (vType sVertex);`
 // دالة لعمل حواف شجرة الامتداد الأدنى
 // وزن الحواف محفوظ في المصفوفة `edgeWeights`.

`void printTreeAndWeight ();`
 // دالة لايخراج الحواف ووزن
 // شجرة الامتداد الأدنى.

تعريف الدالة `createSpanningGraph` متروك كتمرين لك. تقوم هذه الدالة بعمل الرسم البياني ومصفوفة الوزن المرتبطة بالرسم البياني.
 دالة `C++` التالية المسماة `minimalSpanning` تقوم بتطبيق خوارزمية بريم كما تم وصفها من قبل:

```

template<class vType, int size>
void msTreeType<vType, size>::minimalSpanning(vType sVertex)
{
    int i, j, k;
    vType startVertex, endVertex;
    double minWeight;

    source = sVertex;

    bool mstv[size];

    for(j = 0; j < gSize; j++)
    {
        mstv[j] = false;
        edges[j] = source;
        edgeWeights[j] = weights[source][j];
    }

    mstv[source] = true;
    edgeWeights[source] = 0;

    for(i = 0; i < gSize - 1; i++)
    {
        minWeight = infinity;

        for(j = 0; j < gSize; j++)
            if(mstv[j])
                for(k = 0; k < gSize; k++)
                    if(!mstv[k] && weights[j][k] < minWeight)
                    {
                        endVertex = k;
                        startVertex = j;
                        minWeight = weights[j][k];
                    }

        mstv[endVertex] = true;
        edges[endVertex] = startVertex;
        edgeWeights[endVertex] = minWeight;
    } //end for
} //end minimalSpanning

```

تعريف الدالة minimalSpanning يحتوي على ثلاث حلقات for لهذا فان خوارزمية برسيم المعطاة في هذا القسم تكون في أسوأ الحالة $O(n^2)$ ، النوع $O(n^3)$ من المصحح بصميم خوارزمية برسيم حتى تكون من النوع . تمرين البرمجة رقم 5 في نهاية هذا الفصل يطلب منك عمل ذلك.

تعريف الدالة printTreeAndWeight معطى فيما بعد.

```

template<class vType, int size>
void msTreeType<vType, size>::printTreeAndWeight()
{
    double treeWeight = 0;

    cout<<fixed<<showpoint<<setprecision(2);

    cout<<"Source Vertex: "<<source<<endl;
    cout<<"Edges      Weight"<<endl;

    for(int j = 0; j < gSize; j++)
    {
        if(edges[j] != j)
        {
            treeWeight = treeWeight + edgeWeights[j];
            cout<<"("<<edges[j]<<", "<<j<<")      "
                <<edgeWeights[j]<<endl;
        }
    }

    cout<<endl;
    cout<<"Tree Weight: "<<treeWeight<<endl;
} //end printTreeAndWeight

```

ترتيب طوبوغرافي:

ان الطلاب في الجامعة قبل أخذ برنامج تدريبي معين يجب عادةً أن يأخذوا جميع برامجها اللازمة اذا وجدت. على سبيل المثال يجب على الطلاب قبل أخذ البرنامج التدريبي رقم 2 من البرمجة أن يأخذوا البرنامج التدريبي للبرمجة رقم 1. بالرغم من هذا هناك برامج تدريبية معينة يمكن أخذها بشكل مستقل عن بعضها البعض. يمكن تمثيل البرامج التدريبية الموجودة في قسم ما كرسم بياني مباشر. الحافة المباشرة من مثلاً القمة u الى القمة v تعني أن البرنامج التدريبي الذي يتم تمثيلها بواسطة القمة u ضرورية من أجل البرنامج التدريبي الذي يتم تمثيله بواسطة القمة v . قد يكون من المفيد للطلاب أن يعلموا قبل البدء في الدراسات العليا التابع الذي يأخذوا به البرامج التدريبية حتى يمكنهم قبل أخذ برنامج تدريبي ما أن يأخذوا جميع البرامج الضرورية وتلبية مستلزمات تخرجهم في الوقت المناسب. يقوم هذا القسم بوصف خوارزمية يمكن استخدامها لاجراء قسم رسم بياني مباشر بهذا التابع. لنقم أولاً بتقديم بعض المصطلحات:

لتكن G رسم بياني مباشر و $v_{i1}, v_{i2}, \dots, v_{in} = V(G)$ حيث $n > 0$. الترتيب الطوبوغرافي لـ $V(G)$ عبارة عن ترتيب طولي $\{v_1, v_2, \dots, v_n\}$ بحيث أنه اذا كان واقع قبل v_{ij} ان v_{ik} و $k \neq j$ و $1 \leq j \leq n$ و $1 \leq k \leq n$ اذن v_{ij} سبق v_{ik} أي أن $j < k$ في هذا الترتيب الطولي.

هذا القسم يصف خوارزمية تخرج قمم الرسم البياني المباشر بترتيب طوبوغرافي. نفترض أن الرسم البياني ليس به دورات. نترك للقارئ كتمرين له تعديل الخوارزمية للرسم البيانية التي بها دورات. بما أن الشكل البياني ليس به دورات:

- توجد قمة u في G بحيث u ليس لها ما يسبقها.
- توجد قمة v في G بحيث v ليس لها ما يليها.

افترض أن المصفوفة `topologicalOrder` (ذات الحجم n عدد القمم) يتم استخدامها لتخزين قمم G بترتيب طوبوغرافي. لهذا إذا كانت القمة u تالية للقمة v و $v = \text{topologicalOrder}[j]$ و $u = \text{topologicalOrder}[k]$ إذن $j < k$.

يمكن تطبيق خوارزمية الترتيب الطوبوغرافي باستخدام اما الاجتياز الأول للعمق أو الاجتياز الأول للعرض. يقوم هذا القسم بمناقشة كيفية تطبيق الترتيب الطوبوغرافي باستخدام الاجتياز الأول للعرض. تمرين البرمجة رقم 7 في نهاية هذا الفصل يصف كيفية تطبيق الترتيب الطوبوغرافي باستخدام الاجتياز الأول للعمق.

نقوم بمد تعريف الفئة `graphType` (باستخدام التوريث) لتطبيق خوارزمية الترتيب الطوبوغرافي الأول بالعرض. لنقم بتسمية هذه الفئة `topologicalOrderT`. تعريف الفئة التبع تتضمن الدالات لتطبيق خوارزمية الترتيب الطوبوغرافي معطاة فيما يلي:

```
template<class vType, int size>
class topologicalOrderT: public graphType<vType, size>
{
public:
    void bfTopOrder();
    //Function to output the vertices in breadth first
    //topological order
};
```

بعد هذا نناقش كيفية تطبيق الدالة `bfTopOrder`.

الترتيب الطوبوغرافي الأول للعمق:

تذكر أن خوارزمية الاجتياز الأول للعرض مماثل لاجتياز مستوى شجرة ثنائية بالمستوى ولهذا تتم زيارة العقدة الجذرية (التي ليس لها ما يسبقها) أولاً. لهذا في الترتيب الطوبوغرافي الأول بالعرض يجب أن نقوم أولاً بايجاد قمة ليس لها قمم سابقة ووضعها الأولى في الترتيب الطوبوغرافي. بعد هذا نوجد القمة ولتكن y التي تم وضع جميع ما يسبقها بالترتيب الطوبوغرافي ونضع v تالية في الترتيب الطوبوغرافي. من أجل تتبع عدد القمم الخاصة بقمة ما نستخدم المصفوفة `predCount`. مبدئياً تكون `predCount[j]` عدد ما يسبق القمة v_j . تتم تهيئة الطابور المستخدم لتوجيه الاجتياز الأول بالعرض عند هذه القمم بحيث يكون `predCount[k]` تساوي صفر. جوهرياً الخوارزمية V_k عامة تكون:

1. عمل المصفوفة predCount وتهيئتها حتى تكون $\text{predCount}[i]$ عدد العناصر التي تسبق القمة v_i .

2. تهيئة الطابور وليكن queue لجميع تلك القمم v_k بحيث تكون $\text{predCount}[k]$ تساوي صفر. (بوضوح لا يكون الطابور فارغاً لأن الرسم البياني ليس به دورات).

3. بينما الطابور غير فارغ

3.1 ازالة العنصر الأمامي u من الطابور.

3.2 وضع u في الموضع المتوفر التالي وليكن $\text{topologicalOrder}[\text{topIndex}]$ وزيادة topIndex .

3.3 بالنسبة الى جميع العناصر w التي تلي u :

3.3.1 تقليل عدد العناصر السابقة من w بمقدار 1.

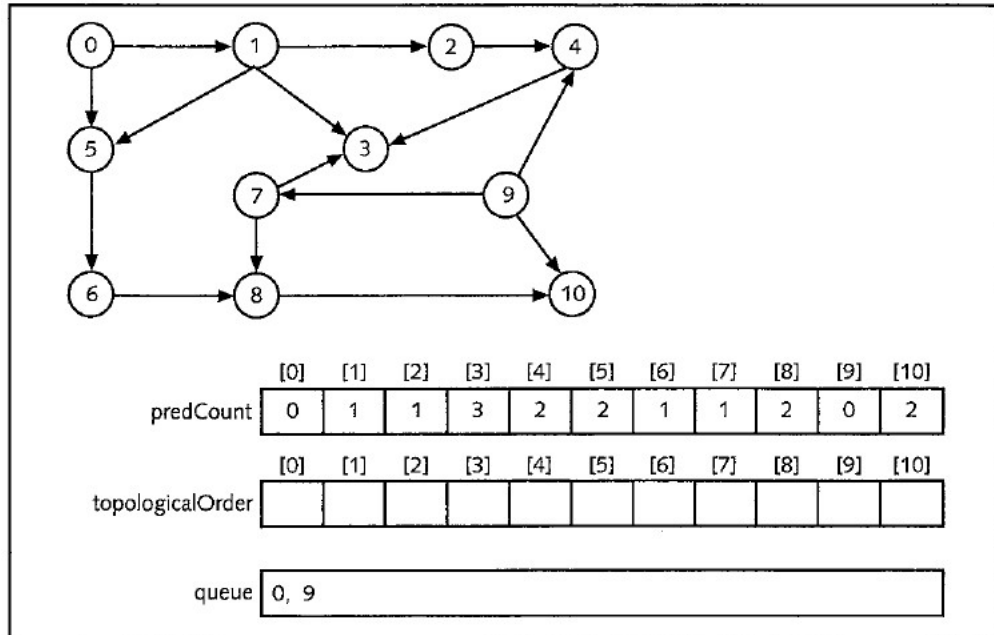
3.3.2 اذا كان عدد العناصر السابقة من w يبلغ صفر اضافة w الى الطابور.

الرسم البياني G_3 من الشكل 7-12 ليس له دورات. قم G_3 في الترتيب الطوبوغرافي الأول للعرض هي:

الترتيب الطوبوغرافي الأول للعرض: 10 8 3 6 4 5 2 7 1 9 0

بعد هذا نوضح الترتيب الطوبوغرافي الأول للعرض للرسم البياني G_3 .

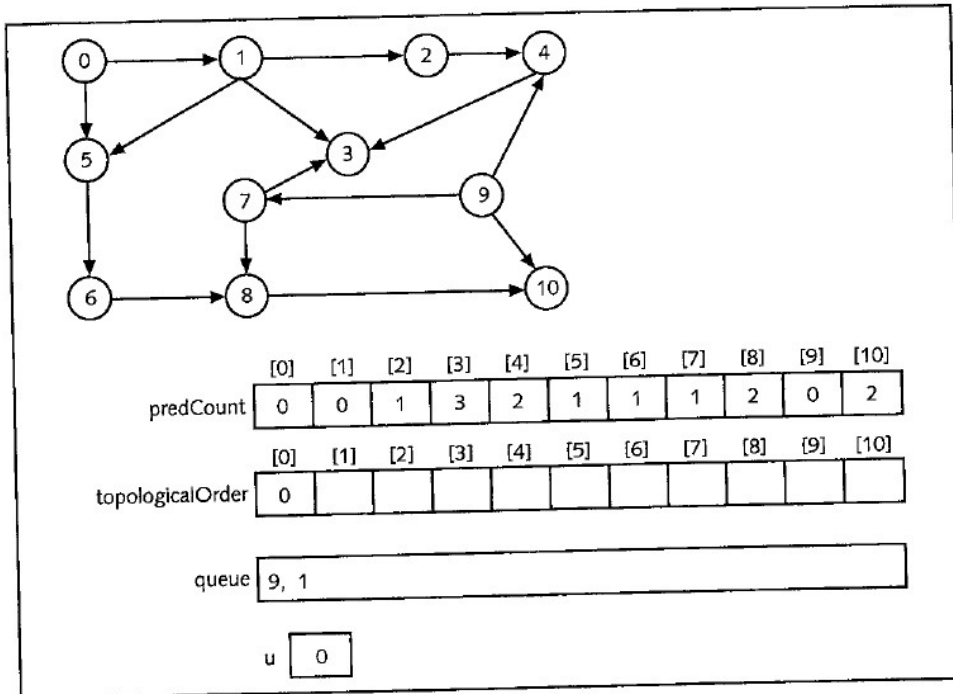
بعد تنفيذ الخطوات 1 و 2 تكون المصفوفات predCount و topologicalOrder و queue كما هي موضحة في شكل 24-12. (لاحظ أننا للتبسيط نوضح عناصر الطابور فقط).



شكل 24-12: المصفوفات predCount و topologicalOrder و queue بعد الخطوتين 1 و 2

يتم تنفيذ الخطوة 3 طالما أن الطابور غير فارغ.

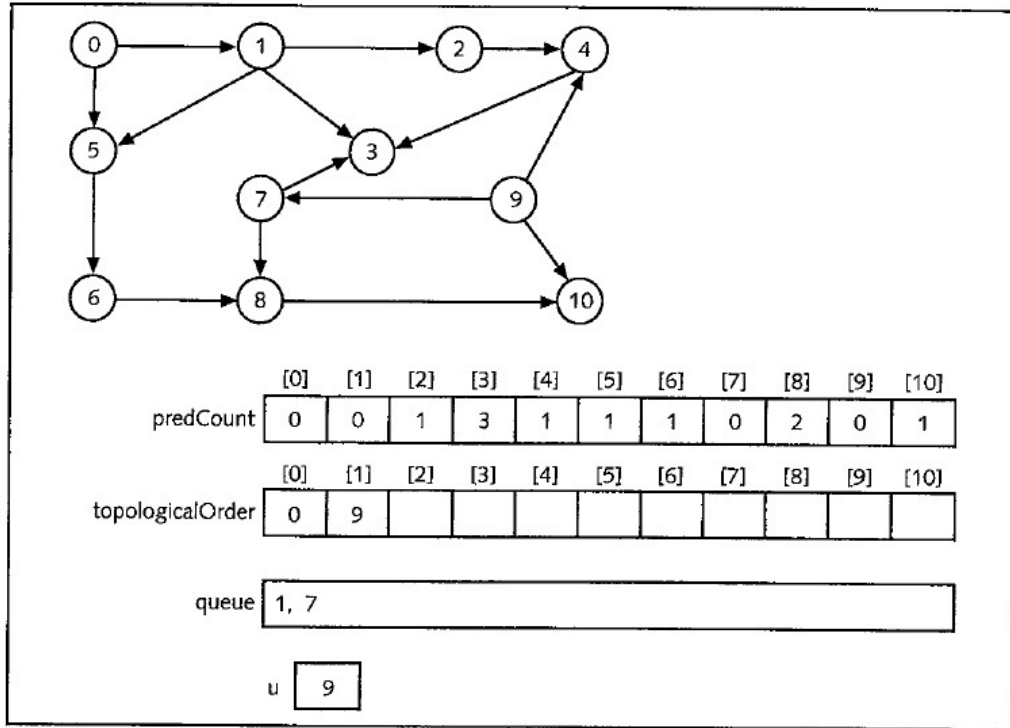
الخطوة 3: المكرر 1: بعد تنفيذ الخطوة 3-1 فإن قيمة u تكون صفر. تقوم الخطوة 3-2 بتخزين قيمة u البالغة صفر في الموضع التالي المتاح في المصفوفة `topologicalOrder`. لاحظ أن الصفر مخزن في الموضع صفر في هذه المصفوفة. تقوم الخطوة 3-3 بتقليل عدد العناصر السابقة لجميع العناصر التالية من صفر بمقدار 1 وإذا تم تقليل عدد العناصر السابقة لأي عقدة تلي صفر إلى صفر تتم إضافة هذه العقدة إلى الطابور. العقد التالية للعقدة 0 هي العقد 1 و5. عدد العناصر السابقة للعقدة 1 تقل إلى صفر وعدد العناصر السابقة للعقدة 5 تقل إلى 1. تتم إضافة العقدة إلى الطابور. بعد المكرر الأول من الخطوة 3 تكون المصفوفات `predCount` و `topologicalOrder` و `queue` كما هي موضحة في شكل 25-12.



شكل 25-12: المصفوفات `predCount` و `topologicalOrder` و `queue` بعد المكرر الأول من الخطوة 3

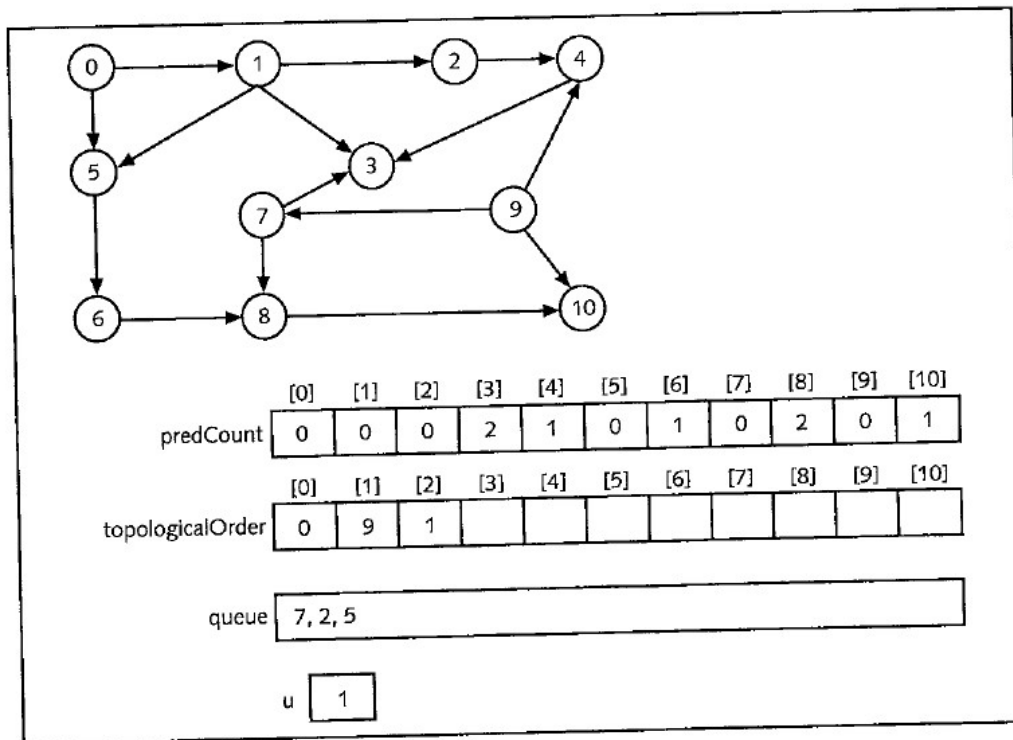
الخطوة 3: المكرر 2: الطابور غير فارغ. بعد تنفيذ الخطوة 3-1 فإن قيمة u تكون 9. تقوم الخطوة 3-2 بتخزين قيمة u البالغة تسعة في الموضع التالي المتاح في المصفوفة `topologicalOrder`. لاحظ أن التسعة مخزنة في الموضع 1 في هذه المصفوفة. تقوم الخطوة 3-3 بتقليل عدد العناصر السابقة لجميع العناصر التي تلي 1 بمقدار 1 وإذا تم تقليل عدد العناصر السابقة لأي عقدة تلي 9 إلى صفر تتم إضافة هذه العقدة إلى الطابور. العقد التالية للعقدة 9 هي العقد 4 و7 و10. عدد العناصر السابقة للعقدة

7 تقل الى صفر وعدد العناصر السابقة للعقد 4 و10 تقل الى 1. تتم اضافة العقدة 7 الى الطابور. بعد المكرر الثاني من الخطوة 3 تكون المصفوفات predCount و topologicalOrder و queue كما هي موضحة في شكل 26-12.



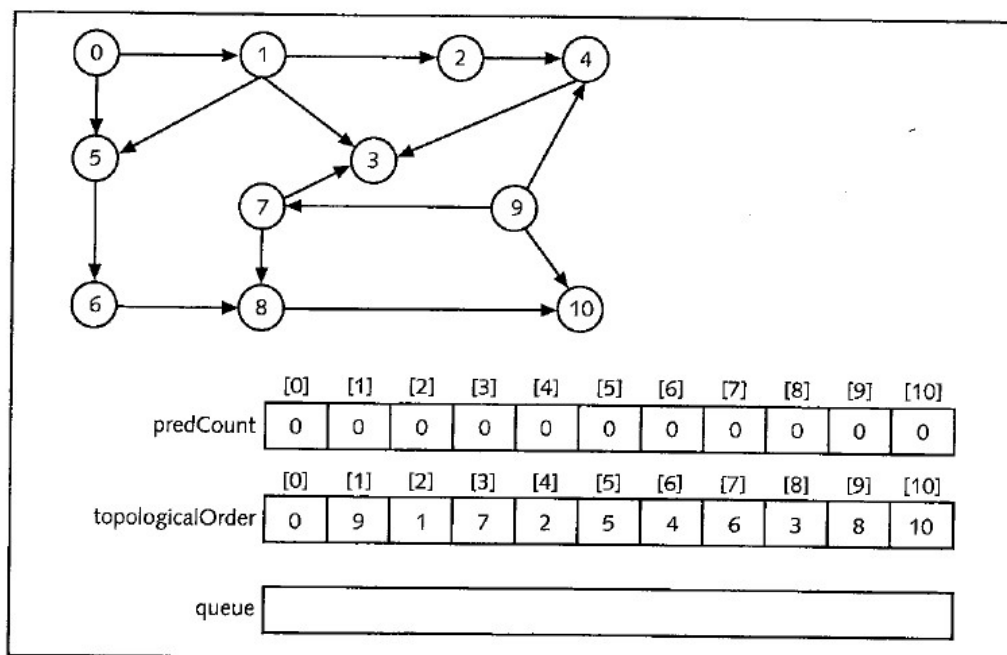
شكل 26-12: المصفوفات predCount و topologicalOrder و queue بعد المكرر الثاني من الخطوة 3

الخطوة 3: المكرر 3: الطابور غير فارغ. بعد تنفيذ الخطوة 3-1 فان قيمة u تكون 1. تقوم الخطوة 3-2 بتخزين قيمة u البالغة واحد في الموضع التالي المتاح في المصفوفة topologicalOrder. لاحظ أن عدد 1 المخزن في الموضع 2 في هذه المصفوفة. تقوم الخطوة 3-3 بتقليل عدد العناصر السابقة لجميع العناصر التي تلي 1 بمقدار 1 وإذا تم تقليل عدد العناصر السابقة لأي عقدة تلي 1 الى صفر تتم اضافة هذه العقدة الى الطابور. العقد التالية للعقدة 1 هي العقد 2 و3 و5. عدد العناصر السابقة للعقد 2 و5 تقل الى صفر وعدد العناصر السابقة للعقدة 3 تقل الى 2. تتم اضافة العقد 2 و5 الى الطابور بهذا الترتيب. بعد المكرر الثالث من الخطوة 3 تكون المصفوفات predCount و topologicalOrder و queue كما هي موضحة في شكل 27-12.



شكل 12-27: المصفوفات predCount و topologicalOrder و queue بعد المكرر الثالث من الخطوة 3

إذا كررت الخطوة 3 ثماني مرات أخرى تكون المصفوفات predCount و topologicalOrder و queue كما هي موضحة في شكل 12-28.



شكل 28-12: المصفوفات predCount و topologicalOrder و queue بعد تنفيذ الخطوة 3 ثماني مرات أخرى

في الشكل 28-12 توضح المصفوفة topologicalOrder الترتيب الطوبوغرافي الأخير بالعرض لعقد الرسم البياني G3.
دالة C++ التالية تقوم بتطبيق خوارزمية الترتيب الطوبوغرافي الأول بالعرض:

```
template<class vType, int size>
void topologicalOrderT<vType, size>::bfTopOrder()
{
    linkedQueueType<vType> queue;

    vType u;
    int ind, j;

    int *topologicalOrder;
    topologicalOrder = new int[gSize];

    for(ind = 0; ind < gSize; ind++)
        topologicalOrder[ind] = -1;

    int topIndex = 0;

    vType *adjacencyList; //array to store the adjacent vertices
    adjacencyList = new vType[gSize];
    int allLength = 0;
```

```

int *predCount;
predCount = new int[gSize];

for(ind = 0; ind < gSize; ind++)
    predCount[ind] = 0;

for(ind = 0; ind < gSize; ind++)
{
    graph[ind].getAdjacentVertices(adjacencyList, alLength);

    for(j = 0; j < alLength; j++)
        predCount[adjacencyList[j]]++;
}

for(ind = 0; ind < gSize; ind++)
    if(predCount[ind] == 0)
        queue.addQueue(ind);

while(!queue.isEmptyQueue())
{
    u = queue.front();
    queue.deleteQueue();
    topologicalOrder[topIndex++] = u;

    graph[u].getAdjacentVertices(adjacencyList, alLength);
    for(int w = 0; w < alLength; w++)
    {
        predCount[adjacencyList[w]]--;
        if(predCount[adjacencyList[w]] == 0)
            queue.addQueue(adjacencyList[w]);
    }
}

} //end while

//output the vertices in breadth first topological order
for(ind = 0; ind < gSize; ind++)
    cout<<topologicalOrder[ind]<<" ";
cout<<endl;

delete [] topologicalOrder;
delete [] predCount;
delete [] adjacencyList;

} //bfTopOrder

```

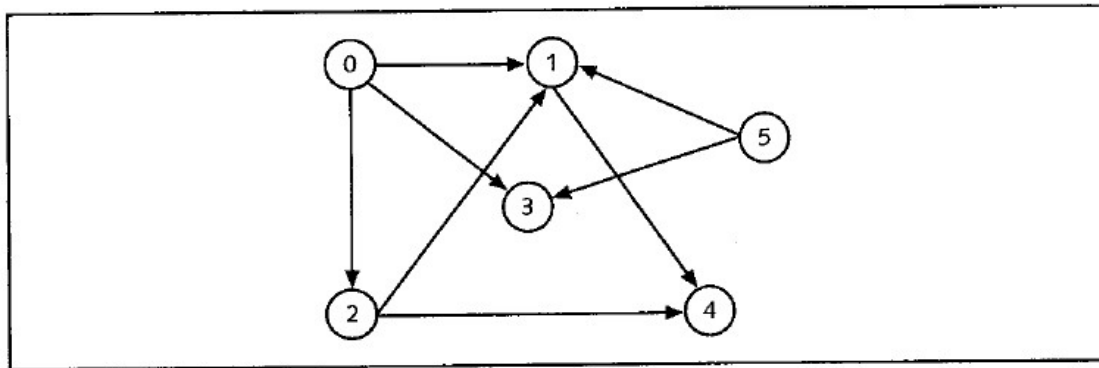
مراجعة سريعة

1. الرسم البياني G عبارة عن زوج $G = (V, E)$ حيث V مجموعة محدودة غير فارغة تسمى مجموعة قمم G و $E \subseteq V \times V$ تسمى مجموعة الحواف.
2. في الرسم البياني الغير مباشر $G = (V, E)$ تكون عناصر E أزواج غير مرتبة.
3. في الرسم البياني المباشر $G = (V, E)$ تكون عناصر E أزواج مرتبة.
4. لتكن G رسم بياني. يطلق على الرسم البياني H رسم بياني فرعي من G اذا كانت كل قمة من H قمة من G واذا كانت كل حافة من H حافة في G .
5. يطلق على القمتين u و v الموجودتين في رسم بياني غير مباشر متجاورتين اذا كان هناك حافة من واحدة الى أخرى.
6. لتكن $e = (u, v)$ حافة في رسم بياني غير مباشر G . يقال أن الحافة e حدث على القمتين u و v .
7. الحافة الطارئة على قمة واحدة تسمى حلقة.
8. في الرسم البياني الغير مباشر اذا كانت الحافتان e_1 و e_2 مرتبطتين بنفس الزوج من القمم اذن يطلق على e_1 و e_2 حواف متوازية.
9. يطلق على الرسم البياني رسم بياني بسيط اذا لم يكن به حلقات أو حواف متوازية.
10. المسار من قمة u الى قمة v عبارة عن تتابع من القمم u_1, u_2, \dots, u_n حيث $u = u_1$ و $u_n = v$ و (u_i, u_{i+1}) حافة لكل $i = 1, 2, \dots, n-1$.
11. القمتان u و v يطلق عليهما متصلتان اذا كان هناك مسار من u الى v .
12. المسار البسيط هو المسار التي تكون فيه جميع القمم مستقلة فيما عدا القمتين الأولى والأخيرة.
13. الدورة في G عبارة عن مسار بسيط تكون فيه القمتان الأولى والأخيرة متشابهتان.
14. يطلق على الرسم البياني الغير مباشر G متصل اذا كان هناك مسار من أي قمة الى أي قمة أخرى.
15. المجموعة الفرعية القصوى من القمم المتصلة يطلق عليها مكون من G .
16. افترض أن u و v قمتين في الرسم البياني المباشر G . اذا كان هناك حافة من u الى v أي أن $(u, v) \in E$ نقول أن u متجاورة مع v وأن v متجاورة مع u .
17. الرسم البياني المباشر G يطلق عليه متصل بقوة اذا كان هناك أي قمتين في G متصلتان.
18. لتكن G رسم بياني به عدد n من القمم حيث $n > 0$. لتكن $V(G) = \{v_1, v_2, \dots, v_n\}$ مصفوفة التجاور A_G عن مصفوفة ثنائية الأبعاد $n \times n$ بحيث أن المدخل (i, j) الخاص ب A_G اذا كان هناك حافة من v_i الى v_j وبخلاف هذا يكون المدخل (i, j) .

19. في تمثيل قائمة التجاور تقابل كل قمة v قائمة متصلة بحيث تحتوي كل عقدة من القائمة المتصلة على القمة u و $(v, u) \in E(G)$.
20. الاجتياز الأول لعمق الرسم البياني مماثل لاجتياز شجرة ثنائية قبل الترتيب.
21. الاجتياز الأول لعرض الرسم البياني مماثل لاجتياز شجرة ثنائية مستوى تلو الآخر.
22. خوارزمية أقصر مسار تعطي أقصر مسافة بالنسبة الى عقدة معينة عن كل عقدة أخرى في الرسم البياني.
23. في الرسم البياني الموزون يكون لكل حافة وزن غير سالب.
24. وزن المسار P هو مجموع أوزان جميع الحواف الواقعة على المسار P والذي يسمى أيضاً وزن v من u عبر P .
25. الشجرة (الفارغة) T عبارة عن رسم بياني بسيط بحيث أنه إذا كانت u و v قمتين في T يكون هناك مسار وحيد من u الى v .
26. الشجرة التي يتم فيها تصميم قمة معينة كجذر تسمى شجرة جذرية.
27. افترض أن T شجرة. إذا تم تحديد وزن للحواف في T اذن يطلق على T شجرة موزونة.
28. إذا كانت T شجرة موزونة فان وزن T الذي يتم التعبير عنه بـ $W(T)$ يكون مجموع أوزان جميع الحواف الموجودة في T .
29. الشجرة T تسمى شجرة امتداد للرسم البياني G إذا كانت T رسم بياني فرعي من G بحيث $V(T) = V(G)$ أب إذا كانت جميع قمم G موجودة في T .
30. لتكن G رسم بياني و $V(G) = \{v_1, v_2, \dots, v_n\}$ الترتيب الطوبوغرافي لـ $V(G)$ ترتيب طولي v_1, v_2, \dots, v_m للقمم بحيث أن إذا كان v_{ij} عنصر يسبق فان $j \neq k$ و $1 \leq j \leq n$ و $1 \leq k \leq m$ اذن تسبق أي أن $v_{ij} < v_{kj}$ هذا الترتيب الطولي.

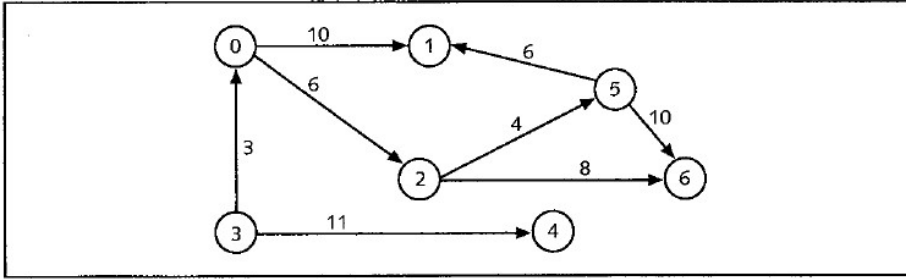
تمارين

استخدم الرسم البياني في شكل 12-29 للتمارين من 1 الى 4.



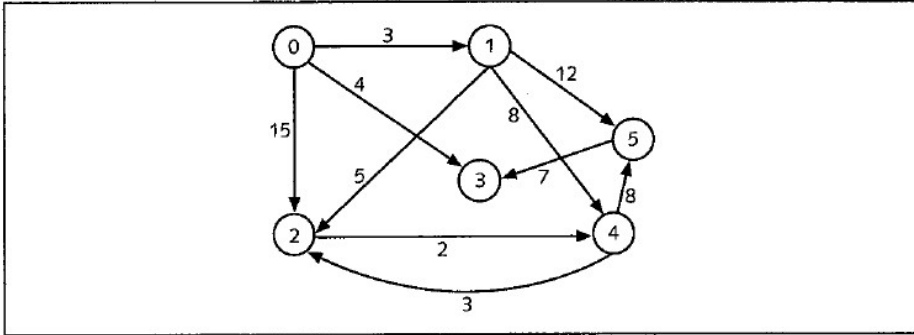
شكل 12-29: رسم بياني للتمرين من 1 حتى 4

1. اوجد مصفوفة التجاور للرسم البياني.
2. اريم قائمة التجاور للرسم البياني.
3. اذكر عقد الرسم البياني في الاجتياز الأول للعمق.
4. اذكر عقد الرسم البياني في الاجتياز الأول للعرض.
5. اوجد مصفوفة الوزن للرسم البياني في شكل 12-30.



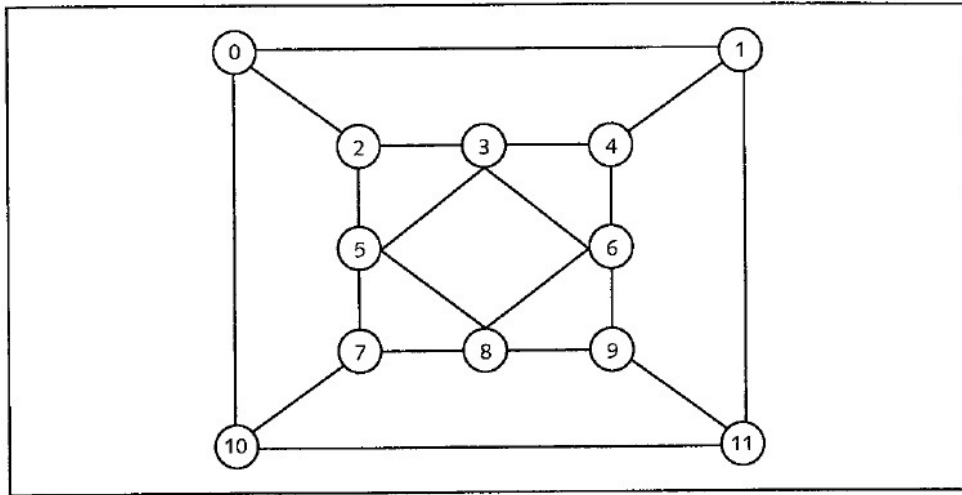
شكل 12-30: رسم بياني للتمرين رقم 5

6. انظر الى الرسم البياني في شكل 12-31 واوجد أقصر مسافة من العقدة صفر الى كل عقدة أخرى في الرسم البياني.



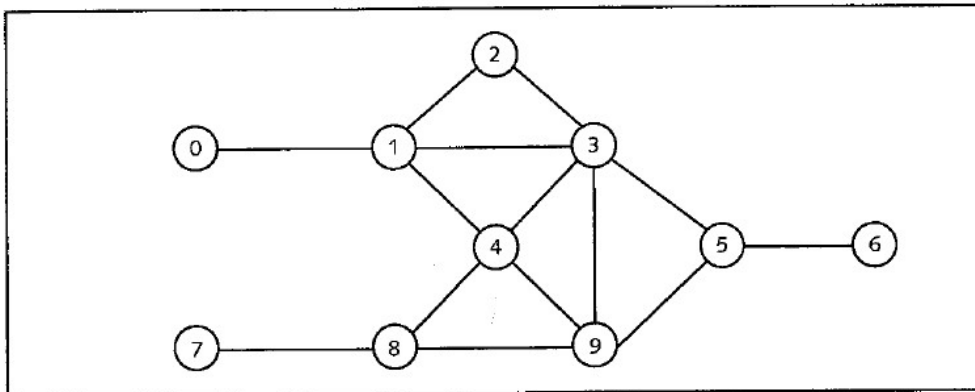
شكل 12-31: رسم بياني للتمرين رقم 6

7. اوجد شجرة امتداد في الرسم البياني في شكل 12-32.



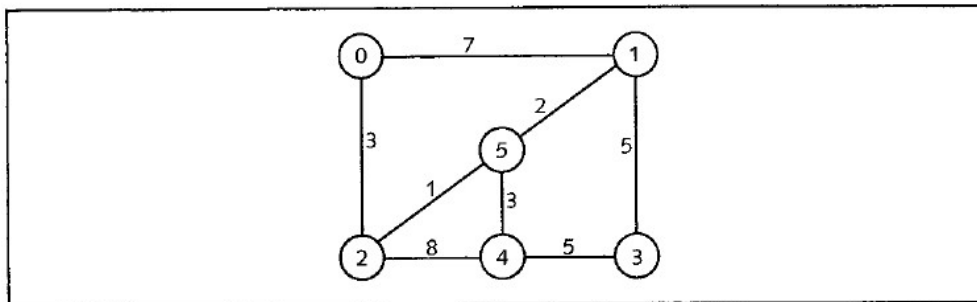
شكل 12-32: رسم بياني لتمارين رقم 7

8. اوجد شجرة امتداد في الرسم البياني في شكل 12-33.



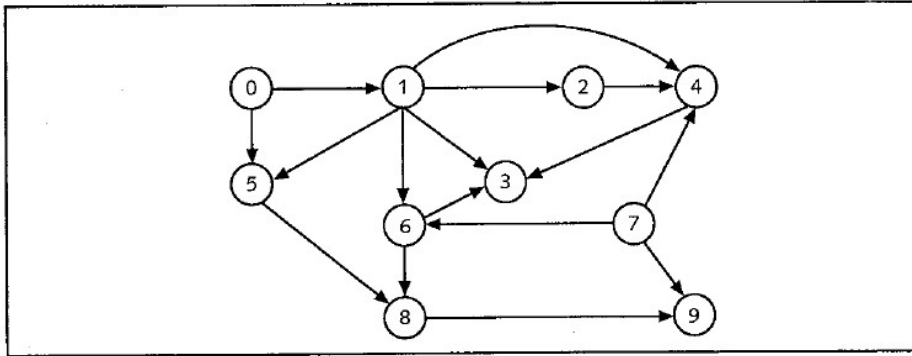
شكل 12-33: رسم بياني لتمارين رقم 8

9. اوجد شجرة الامتداد الأدنى للرسم البياني في الشكل 12-34 مستخدماً الخوارزمية المعطاة في هذا الفصل.



شكل 12-34: رسم بياني لتمارين رقم 9

10. اذكر عقد الرسم البياني في شكل 12.35 بالترتيب الطوبوغرافي الأول للعرض.



شكل 12-35: رسم بياني لتمارين رقم 10

تمارين برمجة

1. اكتب برنامج يقوم باخراج عقد الرسم البياني في الاجتياز الأول للعمق.
2. اكتب برنامج يقوم باخراج عقد الرسم البياني في الاجتياز الأول للعرض.
3. اكتب برنامج يقوم باخراج أقصر مسافة من عقدة معطاة الى كل عقدة أخرى في الرسم البياني.
4. اكتب برنامج يقوم باخراج شجرة الامتداد الأدنى لرسم بياني معطى.
5. خوارزمية تحديد شجرة الامتداد الأدنى المعطاة في هذا الفصل من النوع $O(n^3)$ ي بديل $O(n^2)$.
لخوارزمية بريم وهو من النوع $O(n^2)$.
المدخلات: رسم بياني موزون متصل $G = (V, E)$ به عدد n من العقد مرقمة 0، 1، ...، $n - 1$ بدءاً من القمة s بمصفوفة وزن من W .

المخرجات: شجرة الامتداد الأدنى

```
Prim2(G, W, n, s)
Let T = (V, E), where E =  $\phi$ .
for(j = 0; j < n; j++)
{
    edgeWeight[j] = W[s][j];
    edges[j] = s;
    visited[s] = false;
}
edgeWeight[s] = 0;
visited[s] = true;
while(not all nodes are visited)
{
```

اختر العقدة التي لم تتم زيارتها والتي تمتلك أقل وزن واطلق عليها k .

```

visited[k] = true;
E = E ∪ {(k, edges[k])}
V = V ∪ {k}
for each node j that is not visited
    if(W[k][j] < edgeWeight[k])
    {
        edgeWeight[k] = W[k][j];
        edges[j] = k;
    }
}
return T

```

اكتب تعريف الدالة prim2 لتطبيق هذه الخوارزمية وأضف هذه الدالة الى الفئة msTreeType.
فضلاً عن هذا اكتب برنامج لاختبار هذه النسخة من خوارزمية بريم.

6. اكتب برنامج لاختبار خوارزمية الترتيب الطوبوغرافي الأول بالرخص.
7. لتكن G رسم بياني و $V(G) = \{v_1, v_2, \dots, v_n\}$ حيث $n > 0$. تذكر أن الترتيب الطوبوغرافي لـ $V(G)$ عبارة عن ترتيب طولي $v_{i1}, v_{i2}, \dots, v_{in}$ حيث أن إذا كان v_{ij} عنصر يسبق v_{ik} فإن $1 \leq j < k \leq n$ و $1 \leq i \leq n$. إذن تسبق أي v_{ij} v_{ik} في هذا ترتيب الطولي. افترض أن G ليس به دورات. الخوارزمية التالية الخاصة بالترتيب الطوبوغرافي الأول للعمق تحدد عقد الرسم البياني بترتيب طوبوغرافي. في الترتيب الطوبوغرافي الأول للعمق نقوم أولاً بإيجاد قمة لا يليها شيء (مثل هذه القمة تتواجد لأن الرسم البياني ليس به دورات) ونجعلها الأخيرة في الترتيب الطوبوغرافي. بعد أن وضعنا جميع ما يلي القمة بالترتيب الطوبوغرافي نضع القمة بالترتيب الطوبوغرافي قبل أي مما يليها. من الواضح أننا في الترتيب الطوبوغرافي الأول للعمق نوجد أولاً القمة التي يتم وضعها في $topologicalOrder[n-1]$ ثم $topologicalOrder[n-2]$ وهكذا.
- اكتب تعريفات دالات C++ لتطبيق الترتيب الطوبوغرافي الأول للعمق وأضف هذه الدالات الى الفئة topologicalOrderT المستمدة من الفئة graphType. اكتب كذلك برنامج لاختبار ترتيبك الطوبوغرافي الأول للعمق.

الفصل

13

مكتبة القالب المعياري

في هذا الفصل سوف:

- تعلم المزيد عن مكتبة القالب المعياري.
- تصبح على علم بالحاويات المترابطة.
- تستكشف كيفية استخدام الحاويات المترابطة للتحكم في البيانات في برنامج ما.
- تتعلم خوارزميات عامة متنوعة.

قام الفصل رقم 4 بتقديم مكتبة القالب المعياري. تذكر أن المكونات الأساسية لمكتبة القالب المعياري هي الحاويات والمكررات والخوارزميات. الفئات الثلاث من الحاويات هي الحاويات التتابعية، والحاويات المترابطة، ومكيفات الحاويات. قام الفصل 4 بوصف الحاويات التتابعية `vector` و `deque` وقام فصل 5 بوصف الحاوية التتابعية `list`. كما تم وصف مكيف الحاوية `stack` (مرصوصة) في الفصل رقم 7 ومكيفات الحاوية `queue` (طابور) و `priority_queue` (طابور الأسبقية) تم وصفهما في فصل رقم 8. الفصل 4 ناقش المكررات وهذا الفصل يناقش مكونات مكتبة القالب المعياري التي لم تتم مناقشتها في الفصول السابقة خاصة الحاويات المترابطة والخوارزميات. قبل مناقشة الحاويات المترابطة نقوم أولاً بمناقشة الفئة `pair` التي تقوم باستخدامها بعض من الحاويات المترابطة.

الفئة pair:

بمساعدة الفئة pair يمكن دمج قيمتان في وحدة واحدة وبهذا يمكن معاملتهما كوحدة واحدة. بهذا يمكن لدالة ما ارجاع قيمتين باستخدام الفئة pair. يتم استخدام هذه الفئة في العديد من الأماكن الأخرى في مكتبة القالب المعياري. على سبيل المثال الفئة map و multimap الاتين يتم وصفهما فيما بعد في هذا الفصل تستخدمان الفئة pair لادارة عناصرها.

تعريف الفئة pair موجود في الملف الرئيسي utility. بهذا يجب على البرنامج لاستخدام الفئة pair أن يتضمن البيان التالي:

```
#include <utility>
```

الفئة pair لها مقومان: المقوم الافتراضي ومقوم له عاملان. بهذا يكون التركيب العام لاعلان هدف من النوع pair هو:

```
pair<Type1, Type2> pElement;
```

أو:

```
pair<Type1, Type2> pElement(expr1, expr2);
```

حيث expr1 من النوع Type1 و expr2 من النوع Type2. كل هدف من النوع pair له عنصرين بيانات هما first و second وعنصري البيانات هذين عناصر عامة. بما أن عنصر البيانات first و second لهدف من النوع pair يكونا عناصر عامة اذن يمكن للهدف من النوع pair أن يتناول عناصر البيانات هذه بشكل مباشر في برنامج ما. المثال 1-13 يوضح استخدام الفئة pair.

مثال 1-13:

انظر الى البيانات التالية:

```
pair<int, double> x; //Line 1
pair<int, double> y(13, 45.9); //Line 2
pair<int, int> z(10, 20); //Line 3
pair<string, string> name("Bill", "Brown"); //Line 4
pair<string, double> employee("John Smith", 45678.50); //Line 5
```

البيان في الصف 1 يعلن أن x هدف من النوع pair والمكون الأول من x من النوع int والمكون الثاني من النوع double. بما أنه لم يتم تحديد أية قيم في اعلان x فان المقوم الافتراضي للفئة pair يتم تنفيذه وتتم تهيئة عناصر البيانات first و second عند قيمها الافتراضية التي تكون في هذه الحالة صفر.

البيان في الصف 2 يعلن أن y هدف من النوع pair. المكون الأول من y يكون من النوع int والمكون الثاني يكون من النوع double. تتم تهيئة المكون الأول من y أي first عند 13 والمكون الثاني وهو second تتم تهيئته عند 45.9.

البيان في الصف 3 يعلن أن z هدف من النوع pair وكل من مكونان z من النوع int. المكون الأول من z وهو first تتم تهيئته عند 10 والمكون الثاني وهو second تتم تهيئته عند 20.

البيان في الصف 4 يعلن أن name (الاسم) هدف من النوع pair وكل من مكونان الاسم يكون من النوع string. المكون الأول من name وهو first تتم تهيئته عند "Bill" والمكون الثاني وهو second تتم تهيئته عند "Brown".

البيان في الصف 5 يعلن أن employee هدف من النوع pair. المكون الأول من employee يكون من النوع string والمكون الثاني يكون من النوع double. تتم تهيئة المكون الأول من employee أي first عند "John Smith" والمكون الثاني وهو second تتم تهيئته عند 45678.50.

البيان:

```
x first = 50;
```

يحدد 50 لعنصر البيانات first من x وبالمثل يقوم البيان:

```
name.second = "Calvert";
```

بتحديد "Calvert" لعنصر البيانات second من الاسم.

البيانات التالية توضح كيفية اخراج قيمة عنصر من النوع pair. افترض أن لدينا الاعلانات الموجودة في الصفوف من 1 وحتى 5.

| البيان | التأثير |
|--|--|
| <pre>cout<<y.first<<" "<<y.second<<endl;</pre> | يخرج: 13 45.9 |
| <pre>cout<<name.first<<" "<<name.second<<endl;</pre> | يخرج: بيل براون "Bill Brown" |
| <pre>cout<<employee.first<<" "<<employee.second<<endl;</pre> | يخرج: جون سميث 45678.50 John Smith 45678.50 |

مقارنة أهداف من النوع pair:

لقد تم ائصال العلاقات المترابطة للفئة pair وتتم مقارنة الأهداف المتشابهة من النوع pair كما يلي:

افترض أن x و y أهداف من النوع pair وعناصر البيانات المقابلة لكل من x و y من نفس النوع. (إذا لم تكن عناصر بيانات كل من x و y من النوع المدمج اذن يجب أن يتم تعريف العوامل المترابطة بشكل صحيح لكل من x و y). الجدول 1-13 يصف كيفية تعريف العوامل المترابطة للفئة pair.

جدول 13-1: العوامل المترابطة للفئة pair

| المقارنة | الوصف |
|----------|---|
| $x == y$ | إذا $(x.first == y.first)$ و $(x.second == y.second)$ |
| $x < y$ | إذا $(x.first < y.first)$ أو $(x.second >= y.second)$ و $(x.second < y.second)$ |
| $x <= y$ | إذا $(x < y)$ أو $(x == y)$ |
| $x > y$ | إذا لم تكن $(x <= y)$ |
| $x >= y$ | إذا لم تكن $(x < y)$ |
| $x != y$ | إذا لم تكن $(x == y)$ |

النوع pair والدالة make_pair:

يحتوي الملف الرئيسي utility على تعريف قالب الدالة make_pair. بمساعدة الدالة make_pair يمكننا عمل أزواج دون تحديد النوع pair بوضوح. تعريف قالب الدالة make_pair مماثل لما يلي:

```
template<class T1, class T2>
pair<T1, T2> make_pair(const T1& X, const T2& Y)
{
    return {pair<T1, T2>(X, Y)};
}
```

ان تعريف قالب الفئة make_pair يتضح أن قالب الدالة make_pair عبارة عن دالة منتجة للقيمة وتنتج قيمة من النوع pair. مكونات القيمة المنتجة بواسطة قالب الدالة make_pair يتم تمريرها كعوامل لقالب الدالة make_pair. التعبير:

make_pair (75, 'A')

ينتج قيمة من النوع pair. قيمة المكون الأول هي 75 وقيمة المكون الثاني هي الحرف A. تكون الدالة make_pair مفيدة بوجه خاص اذا كان هناك زوج يتم تمريره كمعطى لدالة ما. المثال 13-2 يوضح استخدام الدالة make_pair.

```

#include <iostream>
#include <utility>
#include <string>

using namespace std;

void funcExp(pair<int, int>);
void funcExp1(pair<int, char>);
void funcExp2(pair<int, string> x);
void funcExp3(pair<int, char *> x);

int main()
{
    pair<int, double> x(50, 87.67);           //Line 1
    pair<string, string> name("John", "Johnson"); //Line 2

    cout<<"Line 3: "<<x.first<<" "<<x.second<<endl; //Line 3
    cout<<"Line 4: "<<name.first<<" "<<name.second
        <<endl;                               //Line 4

    pair<int, int> y;                          //Line 5
    cout<<"Line 6: "<<y.first<<" "<<y.second<<endl; //Line 6

    pair<string, string> name2;                //Line 7
    cout<<"Line 8: "<<name2.first<<"***"
        <<name2.second<<endl;                //Line 8

    funcExp(make_pair(75, 80));                //Line 9
    funcExp1(make_pair(87, 'H'));              //Line 10
    funcExp1(pair<int, char>(198, 'K'));        //Line 11
    funcExp2(pair<int, string>(250, "Hello")); //Line 12
    funcExp2(make_pair(65, string("Hello There"))); //Line 13
    funcExp3(pair<int, char *>(35, "Hello World")); //Line 14
    funcExp3(make_pair(22, (char *)("Sunny"))); //Line 15

    return 0;                                //Line 16
}

void funcExp(pair<int, int> x)
{
    cout<<"Line 17: "<<"In funcExp: "<<x.first
        <<" "<<x.second<<endl;                //Line 17
}

void funcExp1(pair<int, char> x)
{
    cout<<"Line 18: "<<"In funcExp1: "<<x.first
        <<" "<<x.second<<endl;                //Line 18
}

```

```

void funcExp2(pair<int, string> x)
{
    cout<<"Line 19: "<<"In funcExp2: "<<x.first
        <<" "<<x.second<<endl;           //Line 19
}

void funcExp3(pair<int, char *> x)
{
    cout<<"Line 20: "<<"In funcExp3: "<<x.first
        <<" "<<x.second<<endl;           //Line 20
}

```

المخرجات:

```

الصف 3: 87.67 50
الصف 4: جون جونسون
الصف 6: 0 0
الصف 8: ***
الصف 17: In funcExp 80 75
الصف 18: In funcExp H 87
الصف 18: In funcExp K 198
الصف 19: In funcExp Hello 250
الصف 19: In funcExp Hello There 65
الصف 20: In funcExp Hello World 35
الصف 20: In funcExp Sunny 22

```

الحاويات المترابطة:

العناصر الموجودة في حاوية مترابطة يتم تلقائياً ترتيبها وفقاً لبعض معايير الترتيب. معيار الترتيب الافتراضي هو العامل المترابط < (أصغر من). المستخدمون لهم كذلك اختيار تحديد معيار الترتيب الخاص بهم.

بما أن العناصر في الحاوية المترابطة يتم ترتيبها بشكل آلي اذن عندما يتم ادخال عنصر جديد في الحاوية يتم ادخاله في المكان الصحيح. هناك طريقة مناسبة وسريعة لتطبيق هذا النوع من بنية البيانات وهي استخدام شجرة بحث ثنائية. هذا يعني في الحقيقة كيفية تطبيق الحاويات المترابطة. بهذا يكون لكل عنصر في الحاوية عقدة أصلية (عدا العقدة الجذرية) وعلى الأكثر ثمرتين. بالنسبة الى كل عنصر يكون المفتاح في العقدة الأصلية أكبر من المفتاح في الثمرة اليسرى وأصغر من المفتاح في الثمرة اليمنى.

الحاويات المترابطة المسبقة التعريف في مكتبة القالب المعياري هي:

- sets
- multisetes
- maps
- multimaps

الأقسام التالية توضح هذه الحاويات.

الحاويات المترابطة: set و multiset:

كما وضح من قبل تقوم كل من الحاويتين set و multiset تلقائياً بترتيب عناصرها بناءً على بعض من معيار الترتيب. معيار الترتيب الافتراضي هو العامل المترابط < (أصغر من). أي أن العناصر يتم ترتيبها ترتيباً تصاعدياً. يمكن أن يقوم المستخدم أيضاً بتحديد معيار ترتيب آخر. بالنسبة إلى أنواع البيانات المحددة بواسطة المستخدم مثل الفئات يجب أن يتم ائصال العوامل المترابطة بشكل صحيح.

الفرق الوحيد بين الحاويتين set و multiset هو أن الحاوية multiset تسمح بوجود مكررات بينما لا تسمح بذلك الحاوية multiset.

اسم الفئة التي تقوم بتعريف الحاوية set هو set واسم الفئة التي تقوم بتعريف الحاوية multiset هو multiset. اسم الملف الرئيسي المحتوي على تعريفات الفئات set و multiset وتعريفات الدالات لتطبيق عمليات متنوعة على هذه الحاويات هو set. لهذا يجب أن يشتمل البرنامج على البيان التالي من أجل استخدام أي من هذه الحاويات:

```
#include <set>
```

اعلان الحاويات المترابطة set و multiset:

الفئات set و multiset تحتوي على عدة مقومات لاعلان وتهيئة حاويات من هذه الأنواع. يقوم هذا القسم بمناقشة الطرق المتعددة التي يتم بها اعلان وتهيئة هذه الأنواع من الحاويات المترابطة. الجدول 2-13 يوضح كيفية اعلان وتهيئة حاوية set / multiset من نوع معين. (في الجدول 2-13 تكون

ctType اما set أو multiset).

جدول 2-13: طرق عديدة لاعلان حاوية set / multiset

| البيان | التأثير |
|--|--|
| <code>ctType<elmType> ct;</code> | يصنع حاوية set أو multiset فارغة اسمها ct ومعيار الترتيب هو > |
| <code>ctType<elmType, sortOp> ct;</code> | يصنع حاوية set أو multiset فارغة اسمها ct ومعيار الترتيب يتحدد بواسطة sortOp |
| <code>ctType<elmType> ct (otherCt);</code> | يصنع حاوية set أو multiset اسمها ct. يتم نسخ عناصر otherCt داخل ct ومعيار الترتيب هو >. كل من ct و otherCt تكونا من نفس النوع. |
| <code>ctType<elmType, sortOp> ct (otherCt);</code> | يصنع حاوية set أو multiset اسمها ct. يتم نسخ عناصر otherCt داخل ct ومعيار الترتيب يتحدد بواسطة sortOp. كل من ct و otherCt تكونا من نفس النوع. لاحظ أن معيار ترتيب ct و otherCt يجب أن يكون واحداً. |

جدول 2-13: طرق عديدة لاعلان حاوية set / multiset

| البيان | التأثير |
|---|---|
| <code>ctType<elmType> ct (beg, end);</code> | يصنع حاوية <code>set</code> أو <code>multiset</code> اسمها <code>ct</code> ويتم نسخ العناصر بدءاً من الموضع <code>beg</code> حتى الموضع <code>end-1</code> داخل <code>ct</code> . كل من <code>beg</code> و <code>end</code> عبارة عن مكررات. |
| <code>ctType<elmType, sortOp> ct (beg, end);</code> | يصنع حاوية <code>set</code> أو <code>multiset</code> اسمها <code>ct</code> ويتم نسخ العناصر بدءاً من الموضع <code>beg</code> حتى الموضع <code>end-1</code> داخل <code>ct</code> . كل من <code>beg</code> و <code>end</code> عبارة عن مكررات ومعيار الترتيب يتحدد بواسطة <code>sortOp</code> . |

إذا كنت تريد استخدام معيار ترتيب بخلاف المعيار الافتراضي يجب عليك أن تقم بتحديد هذا الخيار عندما يتم اعلان الحاوية. على سبيل المثال انظر الى البيانات التالية:

```
set<int> intSet; //Line 1
set<int, greater<int> > otherIntSet; //Line 2
multiset<string> stringMultiSet; //Line 3
multiset<string, greater<string> > otherStringMultiSet; //Line 4
```

البيان في الصف 1 يعلن أن `intset` حاوية `set` فارغة ونوع العنصر هو `int` ومعيار الترتيب هو معيار الترتيب الافتراضي. البيان في الصف 2 يعلن أن `otherIntSet` حاوية `set` فارغة ونوع العنصر هو `int` ومعيار الترتيب هو أكبر من (أي أنه سوف يتم ترتيب العناصر الموجودة في الحاوية `otherIntSet` ترتيباً تنازلياً). البيان في الصفين 3 و4 لهما نفس المعاني. البيانات في الصفين 2 و4 توضح كيفية تحديد معيار الترتيب التنازلي.

في البيانات في الصفين 2 و4 لاحظ المسافة الموجودة بين العلامتين `<` أي المسافة الموجودة بين `greater<int>` و العلامة `<`. هذه المسافة ضرورية لأن `<<` تعتبر أيضاً معامل تحويل في `C++`. فضلاً عن هذا فإن `greater` في الصفين 2 و4 تكون هدف دالة تم وصفه في قسم "أهداف الدالة" لاحقاً في هذا الفصل.

تدخل وحذف عنصر من حاوية `set` أو `multiset`:

افترض أن `ct` حاوية اما من النوع `set` أو من النوع `multiset`. الجدول 3-13 يوضح العمليات التي يمكن استخدامها لادخال أو حذف أو ايجاد العناصر في مجموعة. كما أن الجدول 3-13 يوضح أيضاً كيفية استخدام هذه العمليات. يتم توضيح اسم الدالة بالخط العريض. في هذا الجدول تكون `ct` اما حاوية `set` أو حاوية `multiset`.

جدول 3-13: عمليات لادخال أو حذف أو ايجاد العناصر في حاوية `set` أو `multiset`

| التعبير | التأثير |
|--|---|
| <code>ct.insert (elem);</code> | يدخل نسخة من العنصر في ct وفي حالة المجموعات تنتج ما اذا كانت عملية الإدخال نجحت. |
| <code>ct.insert (position, elem);</code> | تدخل نسخة من العنصر في ct ويتم انتاج الموضع الذي يتم ادخال العنصر فيه. المعامل الأول position يشير الى المكان الذي يبدأ منه البحث من أجل الإدخال. المعامل position عبارة عن مكرر. |
| <code>ct.insert (beg, end);</code> | يدخل نسخة من جميع العناصر داخل ct بدءاً من الموضع beg وحتى end-1 وكل من beg و end عبارة عن مكررات. |
| <code>ct.erase (elem);</code> | يحذف جميع العناصر ذات القيمة elem ويتم انتاج عدد العناصر التي يتم حذفها. |
| <code>ct.erase (position);</code> | يحذف العناصر الموجودة في الموضع المحدد بواسطة المكرر position ولا يتم انتاج أي قيمة. |
| <code>ct.insert (beg, end);</code> | يحذف جميع العناصر بدءاً من الموضع beg وحتى end - 1 وكلا من beg و end عبارة عن مكررات ولا يتم انتاج أي قيمة. |
| <code>ct.clear ();</code> | يحذف جميع العناصر من الحاوية ct. وبعد هذه العملية تكون الحاوية ct فارغة. |
| <code>ct.find (elem);</code> | ينتج مكرر للعنصر الأول في ct مساوي لelem. اذا لم يتم العثور على هذه العناصر ينتج () ct.end. |

المثال 3-13 يوضح عمليات متنوعة على حاوية set أو multiset:

مثال 3-13:

```
#include <iostream>
#include <set>
#include <string>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    set<int> intSet; //Line 1
    set<int, greater<int> > intSetA; //Line 2

    set<int, greater<int> >::iterator intGtIt; //Line 3

    ostream_iterator<int> screen(cout, " "); //Line 4

    intSet.insert(16); //Line 5
    intSet.insert(8); //Line 6
```

```

intSet.insert(20); //Line 7
intSet.insert(3); //Line 8

cout<<"Line 9: intSet: "; //Line 9
copy(intSet.begin(), intSet.end(), screen); //Line 10
cout<<endl; //Line 11

intSetA.insert(36); //Line 12
intSetA.insert(84); //Line 13
intSetA.insert(30); //Line 14
intSetA.insert(39); //Line 15
intSetA.insert(59); //Line 16
intSetA.insert(238); //Line 17
intSetA.insert(156); //Line 18

cout<<"Line 19: intSetA: "; //Line 19
copy(intSetA.begin(), intSetA.end(), screen); //Line 20
cout<<endl; //Line 21

intSetA.erase(59); //Line 22

cout<<"Line 23: After removing 59, intSetA: "; //Line 23
copy(intSetA.begin(), intSetA.end(), screen); //Line 24
cout<<endl; //Line 25

intGtIt = intSetA.begin(); //Line 26
++intGtIt; //Line 27
++intGtIt; //Line 28
++intGtIt; //Line 29

intSetA.erase(intGtIt); //Line 30

cout<<"Line 31: After removing the fourth element, "
    <<endl<<"          intSetA: "; //Line 31
copy(intSetA.begin(), intSetA.end(), screen); //Line 32
cout<<endl; //Line 33

set<int, greater<int> > intSetB(intSetA); //Line 34

cout<<"Line 35: intSetB: "; //Line 35
copy(intSetB.begin(), intSetB.end(), screen); //Line 36
cout<<endl; //Line 37

intSetB.clear(); //Line 38

cout<<"Line 39: After removing all the elements, "
    <<endl<<"          intSetB: "; //Line 39
copy(intSetB.begin(), intSetB.end(), screen); //Line 40
cout<<endl; //Line 41

```

```

multiset<string, greater<string> > namesMultiSet; //Line 42
multiset<string, greater<string> >::iterator iter; //Line 43

ostream_iterator<string> pScreen(cout, " "); //Line 44

namesMultiSet.insert("Donny"); //Line 45
namesMultiSet.insert("Zippy"); //Line 46
namesMultiSet.insert("Goofy"); //Line 47
namesMultiSet.insert("Hungry"); //Line 48
namesMultiSet.insert("Goofy"); //Line 49
namesMultiSet.insert("Donny"); //Line 50

cout<<"Line 51: namesMultiSet: "; //Line 51
copy(namesMultiSet.begin(), namesMultiSet.end(),
      pScreen); //Line 52
cout<<endl; //Line 53

return 0;
}

```

المخرجات:

الصف 9: intset: 20 16 8 3

الصف 19: intsetA: 30 36 39 59 84 456 238

الصف 23: بعد ازالة 59 تكون intsetA: 30 36 39 84 156 238

الصف 31: بعد ازالة العنصر الرابع:

30 36 84 156 238 :Intset

الصف 35: intsetB: 30 36 84 156 238

الصف 39: بعد ازالة جميع العناصر

:intsetB

الصف 51: namesMultiSet: زيبي هنجري جوفي جوفي دوني دوني

البيان في الصف 1 يعلن أن intset حاوية من النوع set. البيان في الصف 2 يعلن أن intsetA حاوية set عناصرها مرتبة ترتيباً تنازلياً. البيان في الصف 3 يعلن أن intGtIt حاوية من النوع set. المكرر intGtIt يمكنه معالجة عناصر أي حاوية set عناصرها من النوع int مرتبة ترتيباً تنازلياً. البيان في الصف 4 يعلن أن screen مكرر من النوع ostream يقوم باخراج عناصر أي حاوية عناصرها من النوع int.

البيانات في الصفوف من 5 وحتى 8 تقوم بادخال 16، و8، و20، و3 في intSet والبيان في الصف 10 يخرج عناصر intSet. في المخرجات انظر الى الخط الذي يضع علامة على الصف رقم 9 الذي يحتوي على مخرجات البيانات في الصفوف من 9 وحتى 11 من البرنامج.

البيانات في الصفوف من 12 وحتى 18 تقوم بادخال 36، و84، و30، و39، و59، و238، و156 في intSetA والبيان في الصف 20 يخرج عناصر intSetA. في المخرجات انظر الى الخط الذي يضع علامة على الصف رقم 19 الذي يحتوي على مخرجات البيانات في الصفوف من 19 وحتى 21 من البرنامج. لاحظ أن عناصر intSetA تظهر في ترتيب تنازلي.

البيان في الصف 22 يزيل 59 من intSetA. بعد تنفيذ البيان في الصف 26 تشير intGtIt الى أول عنصر من intSetA. البيانات في الصفوف 27 و28 و29 يقوم كل منها بتقديم intGtIt الى العنصر التالي من intSetA. بعد تنفيذ البيان الموجود في الصف 29 تشير intGtIt الى العنصر الرابع من intSetA.

البيان في الصف 30 يزيل عنصر intSetA المشار اليه بواسطة intGtIt. معاني البيانات الموجودة في الصفوف من 34 وحتى 41 متشابهة.

البيان في الصف 42 يعلن أن namesMultiSet حاوية من النوع multiset. العناصر في namesMultiSet من النوع string مرتبة ترتيباً تنازلياً. البيان في الصف 43 يعلن أن iter مكرر من النوع multiset.

البيانات في الصفوف من 45 وحتى 50 تقوم بادخال Donny، وZippy، وGoofy، وHungry، وGoofy، وDonny بداخل namesMultiSet. البيان في الصف 52 يخرج عناصر namesMultiSet.

الحاويات المترابطة: map وmultimap:

الحاويات map وmultimap تدير عناصرها في صيغة مفتاح / قيمة. يتم ترتيب العناصر بشكل تلقائي وفقاً لمعيار ترتيب مطبق على المفتاح. معيار الترتيب الافتراضي هو المعامل الترابطي > (أصغر من أي أن العناصر يتم ترتيبها ترتيباً تصاعدياً). يمكن أن يقوم المستخدم كذلك بتحديد معيار ترتيب آخر. بالنسبة الى أنواع البيانات المحددة بواسطة المستخدم مثل الفئات يجب ان يتم اثقال عوامل الارتباط بشكل صحيح.

الفرق الوحيد بين الحاويتين map وmultimap هو أن الحاوية multimap تسمح بوجود مكررات بينما لا تسمح بذلك الحاوية map.

اسم الفئة التي تقوم بتعريف الحاوية map هو map واسم الفئة التي تقوم بتعريف الحاوية multimap هو multimap. اسم الملف الرئيسي المحتوي على تعريفات الفئات map و multimap وتعريفات الدالات لتطبيق عمليات متنوعة على هذه الحاويات هو map. لهذا يجب أن يشتمل البرنامج على البيان التالي من أجل استخدام أي من هذه الحاويات:

```
#include <map>
```

اعلان الحاويات المترابطة map و multimap:

الفئات map و multimap تحتوي على عدة مقومات لاعلان وتهيئة حاويات من هذه الأنواع. يقوم هذا القسم بمناقشة الطرق المتعددة التي يتم بها اعلان وتهيئة هذه الأنواع من الحاويات المترابطة. الجدول 4-13 يوضح كيفية اعلان وتهيئة حاوية map / multimap من نوع معين. (في الجدول

4-13 تكون ctType اما map أو multimap).

جدول 4-13: طرق عديدة لاعلان حاوية map / multimap

| البيان | التأثير |
|---|---|
| <code>ctType<key, elmType> ct;</code> | يصنع حاوية map أو multimap فارغة ct ومعيار الترتيب هو >. |
| <code>ctType<key, elmType, sortOp> ct;</code> | يصنع حاوية map أو multimap فارغة ct ومعيار الترتيب يتحدد من sortOp. |

جدول 4-13: طرق عديدة لاعلان حاوية map / multimap

| البيان | التأثير |
|---|--|
| <code>ctType<key, elmType> ct (otherCt);</code> | يصنع حاوية map أو multimap اسمها ct. يتم نسخ عناصر otherCt داخل ct ومعيار الترتيب هو >. كل من ct و otherCt تكونا من نفس النوع. |
| <code>ctType<key, elmType, sortOp> ct (otherCt);</code> | يصنع حاوية map أو multimap اسمها ct. يتم نسخ عناصر otherCt داخل ct ومعيار الترتيب يتحدد بواسطة sortOp. كل من ct و otherCt تكونا من نفس النوع. لاحظ أن معيار ترتيب ct و otherCt يجب ان يكون واحداً. |
| <code>ctType<elmType> ct (beg, end);</code> | يصنع حاوية map أو multimap اسمها ct ويتم نسخ العناصر بدءاً من الموضع beg حتى الموضع end-1 داخل ct. كل من beg و end عبارة عن مكررات. |
| <code>ctType<elmType, sortOp> ct (beg, end);</code> | يصنع حاوية map أو multimap اسمها ct ويتم نسخ العناصر بدءاً من الموضع beg حتى الموضع end-1 داخل ct. كل من beg و end عبارة عن مكررات ومعيار الترتيب يتحدد بواسطة sortOp. |

إذا كنت تريد استخدام معيار ترتيب بخلاف المعيار الافتراضي يجب عليك أن تقوم بتحديد هذا الخيار عندما يتم اعلان الحاوية. على سبيل المثال انظر الى البيانات التالية:

```
map<int, int> intMap; //Line 1
map<int, int, greater<int> > otherIntMap; //Line 2
multimap<int, string> stringMultiMap; //Line 3
multimap<int, string, greater<string> > otherStringMultiMap; //Line 4
```

البيان في الصف 1 يعلن أن `intmap` حاوية `map` فارغة ونوع المفتاح ونوع القيمة هو `int` ومعيار الترتيب هو معيار الترتيب الافتراضي. البيان في الصف 2 يعلن أن `otherIntMap` حاوية `map` فارغة ونوع المفتاح ونوع القيمة هو `int` ومعيار الترتيب هو أكبر من (أي أنه سوف يتم ترتيب العناصر الموجودة في الحاوية `otherIntMap` ترتيباً تنازلياً). البيان في الصفين 3 و4 لهما نفس المعاني. البيانات في الصفين 2 و4 توضح كيفية تحديد معيار الترتيب التنازلي. في البيانات في الصفين 2 و4 لاحظ المسافة الموجودة بين علامتين `<` أي المسافة الموجودة بين `greater<int>` والعلامة `<`. هذه المسافة ضرورية لأن `<<` تعتبر أيضاً معامل تحويل في `C++`. فضلاً عن هذا فإن `greater` في الصفين 2 و4 تكون هدف دالة تم وصفه في قسم "أهداف الدالة" لاحقاً في هذا الفصل.

تدخل وجذف عنصر من حاوية `map` أو `multimap`:

افترض أن `ct` حاوية اما من النوع `map` أو من النوع `multimap`. الجدول 5-13 يوضح العمليات التي يمكن استخدامها لادخال أو حذف أو ايجاد العناصر في حاوية `map` أو `multimap`. كما أن الجدول 5-13 يوضح أيضاً كيفية استخدام هذه العمليات. يتم توضيح اسم الدالة بالخط العريض. في هذا الجدول تكون `ct` اما حاوية `map` أو حاوية `multimap`.

جدول 5-13: عمليات لادخال أو حذف أو ايجاد العناصر في حاوية `map` أو `multimap`

| التعبير | التأثير |
|--|--|
| <code>ct.insert (elem);</code> | يدخل نسخة من العنصر في <code>ct</code> وفي حالة الحاويات <code>map</code> تنتج ما اذا كانت عملية الادخال نجحت. |
| <code>ct.insert (position, elem);</code> | تدخل نسخة من العنصر في <code>ct</code> ويتم انتاج الموضع الذي يتم ادخال العنصر فيه. المعامل الأول <code>position</code> يشير الى المكان الذي يبدأ منه البحث من أجل الادخال. المعامل <code>position</code> عبارة عن مكرر. |
| <code>ct.insert (beg, end);</code> | يدخل نسخة من جميع العناصر داخل <code>ct</code> بدءاً من الموضع <code>beg</code> وحتى <code>end-1</code> وكل من <code>beg</code> و <code>end</code> عبارة عن تكرارات. |
| <code>ct.erase (elem);</code> | يحذف جميع العناصر ذات القيمة <code>elem</code> ويتم انتاج عدد العناصر التي يتم حذفها. |
| <code>ct.erase (position);</code> | يحذف العناصر الموجودة في الموضع المحدد بواسطة المكرر <code>position</code> ولا يتم انتاج أي قيمة. |
| <code>ct.insert (beg, end);</code> | يحذف جميع العناصر بدءاً من الموضع <code>beg</code> وحتى <code>end-1</code> وكلا من <code>beg</code> و <code>end</code> عبارة عن تكرارات ولا يتم انتاج أي قيمة. |
| <code>ct.clear ();</code> | يحذف جميع العناصر من الحاوية <code>ct</code> . وبعد هذه العملية تكون الحاوية <code>ct</code> فارغة. |
| <code>ct.find (key);</code> | ينتج مكرر للعنصر الأول في <code>ct</code> الذي يمتلك كفتاح مساوي لـ <code>key</code> . اذا لم يتم العثور على هذه العناصر ينتج <code>(ct.end)</code> . |

المثال 4-13 يوضح عمليات متنوعة على حاوية `map` أو `multimap`:

```

#include <iostream>
#include <map>
#include <utility>
#include <string>
#include <iterator>

using namespace std;

int main()
{
    map<int, int> intMap; //Line 1
    map<int, int>::iterator mapItr; //Line 2

    intMap.insert(make_pair(1, 16)); //Line 3
    intMap.insert(make_pair(2, 8)); //Line 4
    intMap.insert(make_pair(4, 20)); //Line 5
    intMap.insert(make_pair(3, 3)); //Line 6
    intMap.insert(make_pair(1, 23)); //Line 7
    intMap.insert(make_pair(20, 18)); //Line 8
    intMap.insert(make_pair(8, 28)); //Line 9
    intMap.insert(make_pair(15, 60)); //Line 10
    intMap.insert(make_pair(6, 43)); //Line 11
    intMap.insert(pair<int, int>(12, 16)); //Line 12

    cout<<"Line 13: The elements of intMap:"<<endl; //Line 13
    for(mapItr = intMap.begin(); mapItr != intMap.end();
        mapItr++) //Line 14
        cout<<mapItr->first<<"\t"<<mapItr->second
            <<endl; //Line 15
    cout<<endl; //Line 16

    intMap.erase(12); //Line 17

    mapItr = intMap.begin(); //Line 18
    ++mapItr; //Line 19
    ++mapItr; //Line 20
    intMap.erase(mapItr); //Line 21

    cout<<"Line 22: After deleting, the elements of intMap:"
        <<endl; //Line 22
    for(mapItr = intMap.begin(); mapItr != intMap.end();
        mapItr++) //Line 23
        cout<<mapItr->first<<"\t"<<mapItr->second
            <<endl; //Line 24
    cout<<endl; //Line 25

    multimap<string, string> namesMultiMap; //Line 26
    multimap<string, string>::iterator nameItr; //Line 27

    namesMultiMap.insert(make_pair("A1", "Donny")); //Line 28
    namesMultiMap.insert(make_pair("B1", "Zippy")); //Line 29
    namesMultiMap.insert(make_pair("K1", "Goofy")); //Line 30
    namesMultiMap.insert(make_pair("A2", "Hungry")); //Line 31
    namesMultiMap.insert(make_pair("D1", "Goofy")); //Line 32
    namesMultiMap.insert(make_pair("A1", "Dumpy")); //Line 33

```

```

cout<<"Line 34: namesMultiMap: "<<endl;           //Line 34
for(nameItr = namesMultiMap.begin();
    nameItr != namesMultiMap.end();
    nameItr++)                                     //Line 35
    cout<<nameItr->first<<"\t"<<nameItr->second
        <<endl;                                   //Line 36
cout<<endl;                                       //Line 37

return 0;                                         //Line 38
}

```

المخرجات:

الصف 13: عناصر intMap:

| | |
|----|----|
| 16 | 1 |
| 8 | 2 |
| 3 | 3 |
| 20 | 4 |
| 43 | 6 |
| 28 | 8 |
| 16 | 12 |
| 60 | 15 |
| 18 | 20 |

الصف 22: بعد الحذف تكون عناصر intMap:

| | |
|----|----|
| 16 | 1 |
| 8 | 2 |
| 20 | 4 |
| 43 | 6 |
| 28 | 8 |
| 60 | 15 |
| 18 | 20 |

الصف 34: namesMultiMap:

| | |
|--------|----|
| Donny | A1 |
| Dumpy | A1 |
| Hungry | A2 |
| Zippy | B1 |
| Goofy | D1 |
| Goofy | K1 |

البيان في الصف 1 يعلن أن intMap حاوية من النوع map والبيان في الصف 2 يعلن أن mapItr مكرر من النوع map. المكرر mapItr يمكنه معالجة عناصر أي حاوية map تمتلك عناصرها نوع المفتاح ونوع القيمة int.

البيانات في الصفوف من 3 وحتى 12 تقوم بادخال العناصر بمفاتيحها. على سبيل المثال يتم ادخال 16 بالمفتاح 1. البيانات في الصفوف من 3 وحتى 11 تستخدم الدالة make_pair لادخال العناصر والبيان في الصف رقم 12 يستخدم الفئة pair كعامل تكلفة لادخال العناصر. الحلقة for في الصف رقم 14 تخرج عناصر الحاوية intMap.

البيان في الصف 17 يزيل العنصر ذو المفتاح 12 من intMap والبيان في الصف 18 يقوم بتهيئة mapItr عند العنصر الأول في الحاوية intMap. البيانات في الصفين 19 و 20 يقوم كل منهما بتقديم mapItr الى العنصر التالي في intMap. بعد تنفيذ البيان في الصف 20 يشير mapItr الى العنصر الثالث من intMap. البيان في الصف 21 يزيل عنصر intMap المشار اليه بواسطة mapItr. الحلقة for في الصف 23 تقوم باخراج عناصر الحاوية intMap. البيان في الصف 26 يعلن أن namesMultiMap حاوية من النوع multimap. المفاتيح والقيم الموجودة في namesMultiMap تكون من النوع string (مقطع). البيان في الصف 27 يعلن أن nameItr حاوية من النوع multimap. البيانات في الصفوف من 28 وحتى 33 تقوم بادخال العناصر في namesMultiMap. الحلقة for في الصف 35 تقوم باخراج عناصر الحاوية namesMultiMap.

الحاويات، والملفات الرئيسية المرتبطة بها، ودعم المكررات:

قام الفصلان 4 و 5 والأقسام السابقة بمناقشة أنواع متعددة من الحاويات. تذكر أن كل حاوية عبارة عن فئة وتعريف الفئة التي تقوم بتطبيق حاوية معينة موجود في الملف الرئيسي. الجدول 6-13 يصف الحاوية والملف الرئيسي المرتبط بها ونوع المكرر الذي تدعمه الحاوية. جدول 6-13: الحاوية والملف الرئيسي المرتبط بها ونوع المكرر الذي تدعمه كل حاوية.

| حاويات متتابعة | الملف الرئيسي المرتبط بها | نوع دعم المكرر |
|----------------|---------------------------|----------------|
| vector | <vector> | تناول عشوائي |
| deque | <deque> | تناول عشوائي |
| list | <list> | ثنائية الاتجاه |
| حاويات مترابطة | الملف الرئيسي المرتبط بها | نوع دعم المكرر |
| set | <set> | ثنائية الاتجاه |
| multiset | <set> | ثنائية الاتجاه |
| map | <map> | ثنائية الاتجاه |
| multimap | <map> | ثنائية الاتجاه |
| مكيفات | الملف الرئيسي المرتبط بها | نوع دعم المكرر |

| | | |
|-------------------------------|---------|----------------------|
| stack (مصفوفة) | <stack> | لا يوجد دعم للمكررات |
| queue (طابور) | <queue> | لا يوجد دعم للمكررات |
| priority_queue (طابور أسبقية) | <queue> | لا يوجد دعم للمكررات |

الخوارزميات:

هناك عدة عمليات يمكن تعريفها من أجل الحاويات. هناك بعض من العمليات يكون خاص بحاوية ما ولهذا يتم تقديمه كجزء من تعريف الحاوية (أي كدالات عنصر للفئة التي تطبق الحاوية). بالرغم من هذا هناك عدة عمليات – مثل الایجاد والترتيب والدمج – مشتركة بين جميع الحاويات. يتم تقديم هذه العمليات كخوارزميات عامة ويمكن تطبيقها على جميع الحاويات وكذلك نوع المصفوفات المدمجة. الخوارزميات تكون مرتبطة بحاوية معينة عن طريق زوج من المكررات. يتم وضع الخوارزميات العامو في الملف الرئيسي algorithm. الأقسام التالية تصف العديد من تلك الخوارزميات وتوضح كيفية استخدامها في برنامج ما. بما أن الخوارزميات يتم تطبيقها بمساعدة الدالات اذن فان المصطلحين دالة وخوارزمية تعني الشئ ذاته في الأقسام التالية.

تصنيف خوارزميات مكتبة القالب المعياري:

لقد قمنا في الفصلين 4 و 5 وفي أقسام سابقة من هذا الفصل بتطبيق عمليات عديدة على الحاويات المتتابعة مثل المسح والترتيب والدمج. بالرغم من هذا فان هذه الخوارزميات كانت مرتبطة بحاوية معينة على أساس عناصر فئة معينة. جميع تلك الخوارزميات والمزيد منها متوفر في أشكال أكثر عامية يطلق عليها **خوارزميات عامة** ويمكن تطبيقها في مجموعة متنوعة من المواقف. يقوم هذا القسم بمناقشة بعض من هذه الخوارزميات العامة.

تحتوي مكتبة القالب المعياري على خوارزميات تنتظر فقط الى العناصر الموجودة في الحاوية وخوارزميات تنقل عناصر الحاوية. مكتبة القالب المعياري تحتوي كذلك على خوارزميات يمكنها أن تقوم بحسابات معينة مثل ايجاد مجموع عناصر حاوية عديدة. بالاضافة الى هذا تحتوي مكتبة القالب المعياري على خوارزميات من أجل عمليات نظرية المجموعة الأساسية مثل تحديد الاتحاد والتقاطع. لقد واجهت بالفهل بعض من الخوارزميات العامة مثل خوارزمية النسخ copy التي تقوم بنسخ العناصر من نطاق معطى من العناصر الى مكان آخر مثل حاوية أخرى أو الى الشاشة. يمكن تصنيف الخوارزميات في مكتبة القالب المعياري الى الفئات التالية:

- خوارزميات غير تعديلية.
- خوارزميات تعديلية.
- خوارزميات رقمية.

■ خوارزميات مكدسة.

الأقسام الأربعة التالية تصف هذه الخوارزميات. الجزء الأكبر من الخوارزميات العامة موجود في الملف الرئيسي algorithm وهناك خوارزميات معينة مثل الخوارزميات الرقمية موجودة في الملف الرئيسي numeric.

الخوارزميات الغير تعديلية:

الخوارزميات الغير تعديلية لا تقوم بتعديل عناصر الحاوية ولكنها تقوم فقط بفحص العناصر. الجدول 7-13 يحدد الخوارزميات الغير تعديلية.

جدول 7-13: خوارزميات غير تعديلية:

| | | |
|-------------|---------------|---------------|
| max_element | find_end | adjacent_find |
| min | find_first_of | binary_search |
| min_element | find_if | count |
| mismatch | for_each | count_if |
| search | includes | equal |
| search_n | lower_bound | equal_range |
| upper_bound | max | find |

الخوارزميات التعديلية:

الخوارزميات التعديلية مثلما يتضح من الاسم تقوم بتعديل عناصر الحاوية عن طريق اعادة ترتيب أو حذف أو تغيير قيم العناصر. الجدول 8-13 يذكر الخوارزميات التعديلية. جدول 8-13 : خوارزميات تعديلية:

| | | |
|--------------------------|------------------|-------------------|
| rotate_copy | prev_permutation | copy |
| set_difference | random_shuffle | copy_backward |
| set_intersection | remove | fill |
| set_symmetric_difference | remove_copy | fill_n |
| set_union | remove_copy_if | generate |
| sort | remove_if | generate_n |
| stable_partition | replace | inplace_merge |
| stable_sort | replace_copy | iter_swap |
| swap | replace_copy_if | merge |
| swap_ranges | replace_if | next_permutation |
| transform | reverse | nth_element |
| unique | reverse_copy | partial_sort |
| Unique_copy | rotate | partial_sort_copy |
| | | partition |

الخوارزميات التعديلية التي تقوم بتغيير ترتيب العناصر وليس قيمها يطلق عليها أيضاً **خوارزميات التغير**. على سبيل المثال `next_permutation`، `partition`، و `prev_permutation`، و `random_shuffle`، و `reverse`، و `reverse_copy`، و `rotate`، و `rotate_copy`، و `stable_partition` عبارة عن خوارزميات تغير.

الخوارزميات الرقمية:

بتم تصميم الخوارزميات الرقمية للقيام بحسابات رقمية على عناصر الحاوية. الجدول 9-13 يذكر هذه الخوارزميات.

جدول 9-13: خوارزميات رقمية:

| | |
|---------------|---------------------|
| inner_product | accumulate |
| partial_sum | adjacent_difference |

خوارزميات مكدسة:

قام الفصل رقم 10 بوصف خوارزمية الترتيب بالتكدس. تذكر أنه في خوارزمية الترتيب بالتكدس يتم عرض المصفوفة المحتوية على البيانات كشجرة ثنائية. لهذا فان التكدس عبارة عن شكل من شجرة نائية يتم تمثيلها كمصفوفة. في التكدس يكون العنصر الأول هو أكبر عنصر والعنصر الموجود في الموضع i (إذا وجد) يكون أكبر من العناصر الموجودة في الموضع $2i$ أو $2i + 1$ (إذا وجد). في خوارزمية الترتيب بالتكدس يتم أولاً تحويل المصفوفة المحتوية على البيانات الى تكدس ثم يتم ترتيب المصفوفة باستخدام نوع خاص من خوارزمية الترتيب. الجدول 10-13 يذكر الخوارزميات المقدمة من مكتبة القالب المعياري لتطبيق خوارزمية الترتيب بالتكدس.

جدول 10-13: خوارزميات التكدس:

| | |
|-----------|-----------|
| push_heap | make_heap |
| sort_heap | pop_heap |

غالبية خوارزميات مكتبة القالب المعياري موضحة قرب نهاية هذا القسم. بالنسبة الى الجزء الأكبر من قوالب الدالة لهذه الخوارزميات فهي معطاة مع توضيح مختصر لما تقوم به كل خوارزمية. بعد هذا نتعلم كيفية استخدام هذه الخوارزميات بمساعدة مثال أو برنامج من لغة C++. ان خوارزميات مكتبة القالب المعياري مفيدة للغاية وتحقق نتائج مذهلة. فضلاً عن هذا فقد تم جعلها خوارزميات عامة حتى تسمح للمستخدم بتحديد معيار التحكم بدلاً من استخدام العمليات العادية للتحكم في الحاويات. على سبيل المثال يكون الترتيب الطبيعي هو الترتيب التصاعدي ولكن يمكن للمستخدم تحديد معيار آخر لترتيب الحاوية ترتيبياً تنازلياً. بهذا يتم تطبيق كل خوارزمية بمساعدة الدالات المثقلة. قبل البدء في وصف هذه الخوارزميات نقوم بمناقشة أهداف الدالة التي تسمح للمستخدم بتحديد معيار التحكم.

أهداف الدالة:

لجعل الخوارزمية العامة مرنة تقوم عادةً مكتبة القالب المعياري بتقديم صيغتان من الخوارزمية باستخدام آلية أثقال الدالة. الصيغة الأولى من الخوارزمية تستخدم العملية العادية لتحقيق هذا الهدف وفي الصيغة الثانية يمكن للمستخدم تحديد المعيار بناءً على الخوارزمية التي تعالج العناصر. على سبيل المثال تقوم الخوارزمية `adjacent_find` بالبحث الحاسوبية وانتاج موضع أول عنصرين متساويين. في الصيغة الثانية من هذه الخوارزمية يمكننا تحديد المعيار (ليكن أصغر من) للبحث عن أول عنصرين يكون فيهما العنصر الثاني أصغر من العنصر الأول. يتم تمرير هذا المعيار كأهداف للدالة. بشكل أكثر رسمية يحتوي **هدف الدالة** على دالة يمكن معاملتها كدالة باستخدام معامل استدعاء الدالة (). في الحقيقة ان هدف الدالة عبارة عن قالب فئة يقوم بأثقال معامل استدعاء الدالة (). بالإضافة الى السماح لك بعمل أهداف دالة خاصة بك تقدم مكتبة القالب المعياري أهداف دالة حسابية و مترابطة ومنطقية يتم وصفها في الجدول 11-13. تكون أهداف دالة مكتبة القالب المعياري موجودة في الملف الرئيسي `functional`.

جدول 11-13: أهداف الدالة الحسابية لمكتبة القالب المعياري:

| الوصف | اسم هدف الدالة |
|-------|-------------------------------------|
| ***** | <code>plus <Type></code> |
| ***** | <code>minus<Type></code> |
| ***** | <code>multiplies<Type></code> |
| ***** | <code>divides<Type></code> |
| ***** | <code>modulus<Type></code> |
| ***** | <code>negate<Type></code> |

المثال 5-13 يوضح كيفية استخدام أهداف دالة مكتبة القالب المعياري.

//Function Objects

```

#include <iostream>
#include <string>
#include <algorithm>
#include <numeric>
#include <iterator>
#include <vector>
#include <functional>

using namespace std;

int funcAdd(plus<int>, int, int);

int main()
{
    plus<int> addNum; //Line 1
    int num = addNum(34, 56); //Line 2

    cout<<"Line 3: num = "<<num<<endl; //Line 3

    plus<string> joinString; //Line 4

    string str1 = "Hello "; //Line 5
    string str2 = "There"; //Line 6

    string str = joinString(str1, str2); //Line 7

    cout<<"Line 8: str = "<<str<<endl; //Line 8

    cout<<"Line 9: Sum of 34 and 26 = "
        <<funcAdd(addNum, 34, 26)<<endl; //Line 9

    int list[8] = {1, 2, 3, 4, 5, 6, 7, 8}; //Line 10

    vector<int> intList(list, list + 8); //Line 11
    ostream_iterator<int> screenOut(cout, " "); //Line 12

    cout<<"Line 13: intList: "; //Line 13
    copy(intList.begin(), intList.end(), screenOut); //Line 14
    cout<<endl; //Line 15

    //accumulate
    int sum = accumulate(intList.begin(),
        intList.end(), 0); //Line 16

```

```

cout<<"Line 17: Sum of the elements of intList = "
    <<sum<<endl;                                     //Line 17

int product = accumulate(intList.begin(),
    intList.end(),
    1, multiplies<int>()); //Line 18

cout<<"Line 19: Product of the elements of intList = "
    <<product<<endl;                                     //Line 19

return 0;
}

int funcAdd(plus<int> sum, int x, int y)
{
    return sum(x, y);
}

```

المخرجات:

الصف 3: num = 90

الصف 8: str = Hello There

الصف 9: مجموع 34 و 26 = 60

الصف 13: intList: 1 2 3 4 5 6 7 8

الصف 17: مجموع عناصر intList = 36

الصف 19: حاصل ضرب عناصر intList = 40320

الجدول 12-13 يصف أهداف دالة مكتبة القالب المعياري المرتبطة:
 جدول 12-13: أهداف دالة مكتبة القالب المعياري المرتبطة

| الوصف | اسم هدف الدالة |
|---|--------------------|
| تنتج true إذا كان المعطيان متساويين وبخلاف هذا تنتج false. على سبيل المثال equal_to<int>compare; bool isEqual = compare (5, 5); قيمة isEqual تكون true. | equal_to<Type> |
| تنتج true إذا لم يكن المعطيان متساويين وبخلاف هذا تنتج false. على سبيل المثال not_equal_to<int>compare; bool isNotEqual = compare (5, 6) قيمة isNotEqual تكون True. | not_equal_to<Type> |
| تنتج true إذا كان المعطى الأول أكبر من المعطى الثاني وبخلاف هذا تنتج false. على سبيل المثال great<int>compare; bool isGreater = compare (8, 5); قيمة isGreater تكون true. | greater<Type> |

جدول 12-13: أهداف دالة مكتبة القالب المعياري المرتبطة

| الوصف | اسم هدف الدالة |
|--|--|
| تنتج true إذا كان المعطى الأول أكبر من أو يساوي المعطى الثاني وبخلاف هذا تنتج false. على سبيل المثال <code>greater_equal<int>compare;</code> <code>bool isGreaterEqual = compare (8, 5);</code> قيمة isGreaterEqual تكون true. | <code>Greater_equal<Type></code> |
| تنتج true إذا كان المعطى الأول أصغر من المعطى الثاني وبخلاف هذا تنتج false. على سبيل المثال <code>less<int>compare;</code> <code>bool isless = compare (3, 5);</code> قيمة isless تكون true. | <code>less<Type></code> |
| تنتج true إذا كان المعطى الأول أصغر من أو يساوي المعطى الثاني وبخلاف هذا تنتج false. على سبيل المثال <code>less_equal<int>compare;</code> <code>bool isless = compare (8, 15);</code> قيمة isless تكون true. | <code>less_equal<Type></code> |

يمكن كذلك تطبيق أهداف دالة مكتبة القالب المعياري المرتبطة على الحاويات كما هو موضح فيما بعد. ان خوارزمية مكتبة القالب المعياري `adjacent_find` تقوم ببحث الحاوية وانتاج الموضع في الحاوية الذي يوجد به العنصران المتساويان. هذه الخوارزمية لها صيغة ثانية تسنح للمستخدم بتحديد معيار المقارنة. على سبيل المثال انظر الى الحاوية التالية `vecList`:

`vecList = {2, 3, 4, 5, 1, 7, 8, 9}`

من المفترض أن تكون عناصر `vecList` مرتبة ترتيباً تصاعدياً. لرؤية ما اذا كانت العناصر غير مرتبة يمكننا استخدام خوارزمية `adjacent_find` كما يلي:

```
intItr = adjacent_find(vecList.begin(), vecList.end(),
                      greater<int>());
```

حيث `intItr` مكرر من النوع `vector`. تبدأ الدالة `adjacent_find` عند الموضع `vecList.begin` (أي أول عنصر من `vecList` وتبحث عن المجموعة الأولى من العناصر التي يكون فيها العنصر الأول أكبر من الثاني. تقوم الدالة بانتاج مؤشر الى العنصر 5 الذي يتم تخزينه في `intItr`. البرنامج الموجود في مثال 13-6 يوضح كيفية استخدام أهداف الدالة المرتبطة.

مثال 13-6:

//STL Predicates

```
#include <iostream>
#include <string>
#include <algorithm>
#include <iterator>
```

```

#include <vector>
#include <functional>

using namespace std;

int main()
{
    equal_to<int> compare; //Line 1
    bool isEqual = compare(6, 6); //Line 2

    cout<<"Line 3: isEqual = "<<isEqual<<endl; //Line 3

    greater<string> greaterStr; //Line 4

    string str1 = "Hello"; //Line 5
    string str2 = "There"; //Line 6

    if(greaterStr(str1, str2)) //Line 7
        cout<<"Line 8: \""<<str1<<"\" is greater "
            <<"than \""<<str2<<"\"<<endl; //Line 8
    else //Line 9
        cout<<"Line 10: \""<<str1<<"\" is not "
            <<"greater than \""<<str2<<"\"<<endl; //Line 10

    int temp[8] = {2, 3, 4, 5, 1, 7, 8, 9}; //Line 11

    vector<int> vecList(temp, temp+8); //Line 12
    vector<int>::iterator intItr1, intItr2; //Line 13
    ostream_iterator<int> screen(cout, " "); //Line 14

    cout<<"Line 15: vecList: "; //Line 15
    copy(vecList.begin(), vecList.end(), screen); //Line 16
    cout<<endl; //Line 17

    intItr1 = adjacent_find(vecList.begin(),
                           vecList.end(),
                           greater<int>()); //Line 18
    intItr2 = intItr1 + 1; //Line 19

    cout<<"Line 20: In vecList, the first set of "
        <<"out of order elements are: "<<*intItr1
        <<" "<<*intItr2<<endl; //Line 20
    cout<<"Line 21: In vecList, the first out of "
        <<"order element is at position: "
        <<vecList.end() - intItr2<<endl; //Line 21

    return 0;
}

```

المخرجات:

الصف 3: isEqual = 1

الصف 10: "Hello" ليست أكبر من "There".

الصف 15: vecList: 2 3 4 5 1 7 8 9

الصف 20: في vecList المجموعة الأولى من العناصر الخارجة عن الترتيب: 5 1

الصف 21: في vecList أول عنصر خارج عن الترتيب هو عند

الموضع: 4

جدول 13-13 يصف أهداف دالة مكتبة القالب المعياري المنطقية.

جدول 13-13 أهداف دالة مكتبة القالب المعياري المنطقية

| اسم هدف الدالة | التأثير |
|--------------------------------------|---|
| <code>logical_not<Type></code> | تنتج true إذا تم تقييم معاملها عند false وبخلاف هذا تنتج false. هذا عنصر دالة أحادي. |
| <code>logical_and<Type></code> | تنتج true إذا تم تقييم كل من معاملها عند true وبخلاف هذا تنتج false. هذا هدف دالة ثنائي. |
| <code>logical_or<Type></code> | تنتج true إذا تم تقييم واحد على الأقل من معاملاتها عند true وخلاف هذا تنتج false. هذا هدف دالة ثنائي. |

الأصول:

الأصول نوع خاص من أهداف الدالة يقوم بإنتاج قيم منطقية. هناك نوعان من الأصول – أحادي وثنائي. الأصول الأحادية تتحقق من خاصية واحدة لمعطى واحد والأصول الثنائية تتحقق من خاصية واحدة لزوج من – أي اثنين من- المعطيات. عادةً يتم استخدام الأصول لتحديد معيار البحث أو الترتيب. في مكتبة القالب المعياري يجب أن يقوم الأصل دائماً بإنتاج نفس النتيجة للقيمة ذاتها. لهذا فإن الدالات التي تقوم بتعديل حالاتها الداخلية لا يمكن اعتبارها أصول.

ادخال مكرر:

انظر الى البيانات التالية:

```
int list[5] = {1, 3, 6, 9, 12}; //Line 1
vector<int> vList; //Line 2
```

البيان في الصف 1 يقوم بإعلان وتهيئة list لكي تكون مصفوفة من 5 مكونات والبيان في الصف 2 يعلن أن vList حاوية vector. بما أنه لم يتم تحديد أي حجم من أجل vList إذن لا يتم حجز مساحة ذاكرة لعناصر vList. افترض الآن أننا نريد نسخ عناصر list داخل vList. البيان:

```
copy(list, list + 8, vList.begin());
```

لن يعمل لأنه لم يتم تخصيص مساحة ذاكرة من أجل عناصر vList والدالة copy تستخدم معامل الاسناد لنسخ العناصر من المصدر الى المكان المقصود.

من أحد حلول هذه المشكلة استخدام حلقة for للانتقال خلال عناصر القائمة واستخدام الدالة push_back من vecList لنسخ عناصر القائمة. بالرغم من هذا هناك حل أفضل وهو يكون مناسب

وسهل التطبيق عندما لا تكون هناك مساحة ذاكرة مخصصة عند المكان المقصود. مكتبة القالب المعياري تقدم ثلاث مكررات تسمى **مكررات ادخال** لادخال العناصر في المكان المقصود وهي: `back_inserter`، `front_inserter`، و `inserter`.

▪ `back_inserter`: هذا المدخل يستخدم العملية `push_back` من الحاوية في مكان معامل الاسناد. المعطى لهذا المكرر هو الحاوية ذاتها. على سبيل المثال يمكننا في المشكلة السابقة أن نقوم بنسخ عناصر القائمة داخل `vList` باستخدام `back_inserter` كما يلي:

▪ `inserter`: هذا المدخل يستخدم العملية `insert` من الحاوية في مكان معامل الاسناد. هذا المكرر له معطيان: المعطى الأول هو الحاوية ذاتها والمعطى الثاني هو مكرر للحاوية يحدد الموضع الذي يجب أن يبدأ منه الادخال.

البرنامج في المثال 7-13 يوضح آثار المدخلين على الحاوية:

مثال 7-13:

```
//Inserters

#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>
#include <list>

using namespace std;

int main()
{
    int temp[8] = {1, 2, 3, 4, 5, 6, 7, 8}; //Line 1

    vector<int> vecList1; //Line 2
    vector<int> vecList2; //Line 3

    ostream_iterator<int> screenOut(cout, " "); //Line 4

    copy(temp, temp + 8, back_inserter(vecList1)); //Line 5
```

```

cout<<"Line 6: vecList1: "; //Line 6
copy(vecList1.begin(), vecList1.end(), //Line 7
      screenOut); //Line 8
cout<<endl;

copy(vecList1.begin(), vecList1.end(), //Line 9
      inserter(vecList2, vecList2.begin()));

cout<<"Line 10: vecList2: "; //Line 10
copy(vecList2.begin(), vecList2.end(), //Line 11
      screenOut); //Line 12
cout<<endl;

list<int> tempList; //Line 13

copy(vecList2.begin(), vecList2.end(), //Line 14
      front_inserter(tempList));

cout<<"Line 15: tempList: "; //Line 15
copy(tempList.begin(), tempList.end(), //Line 16
      screenOut); //Line 17
cout<<endl;

return 0;
}

```

المخرجات:

```

الصف 6: vecList1: 8 7 6 5 4 3 2 1
الصف 10: vecList2: 8 7 6 5 4 3 2 1
الصف 15: tempList: 1 2 3 4 5 6 7 8

```

خوارزميات مكتبة القالب المعياري:

يقوم هذا القسم بوصف غالبية خوارزميات مكتبة القالب المعياري. تشتمل كل خوارزمية على نماذج الدالة وعلى وصف موجز لما تقوم به الخوارزمية ومثال أو برنامج يوضح كيفية استخدامها. في نماذج الدالة تشير أنواع العوامل الى نوع الحاوية الذي يمكن تطبيق الخوارزمية عليه. على سبيل المثال اذا كان العامل من نوع مكررات التناول العشوائي فان الخوارزمية يمكن تطبيقها على الحاويات التي تكون من نوع التناول العشوائي فقط مثل الحاويات vector. طوال الوقت نستخدم مختصرات مثل outputItr لكي نقصد مكرر اخراج و inputItr لكي نقصد مكرر ادخال و forwardItr لكي نقصد مكرر أمامي وهكذا.

الدالات fill و fill_n:

يتم استخدام الدالة fill لملاً الحاوية بالعناصر زيتم استخدام الدالة fill_n لملاً العناصر n التالية. العنصر الذي يتم استخدامه كعنصر ملاً يتم تمريره كمعامل لهذه الدالات.

يتم تعريف كلا من هاتين الدالتين في الملف الرئيسي algorithm ونماذج هذه الدالات تكون:

```
template<class forwardItr, class Type>
void fill(forwardItr first, forwardItr last, const Type& value);
```

```
template<class forwardItr, class size, class Type>
void fill_n(forwardItr first, size n, const Type& value);
```

أول معاملين من الدالة fill يكونان مكررات أمامية تقوم بتحديد واضع بداية ونهاية الحاوية والمعامل الثالث هو عنصر المملأ. المعامل الأول للدالة fill_n عبارة عن مكرر أمامي يقوم بتحديد موضع بداية الحاوية والمعامل الثاني يقوم بتحديد عدد العناصر التي يتم ملأها والمعامل الثالث يقوم بتحديد عنصر المملأ. البرنامج الموجود في المثال 8-13 يوضح كيفية استخدام هذه الدالات.

مثال 8-13:

```
//STL functions fill and fill_n

#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

int main()
{
    vector<int> vecList(8); //Line 1
    ostream_iterator<int> screen(cout, " "); //Line 2

    fill(vecList.begin(), vecList.end(), 2); //Line 3

    cout<<"Line 4: After filling vecList with 2's: "; //Line 4
    copy(vecList.begin(), vecList.end(), screen); //Line 5
    cout<<endl; //Line 6

    fill_n(vecList.begin(), 3, 5); //Line 7

    cout<<"Line 8: After filling the first three "
    <<"elements with 5's: "<<endl<<" "; //Line 8
    copy(vecList.begin(), vecList.end(), screen); //Line 9
    cout<<endl; //Line 10

    return 0;
}
```

المخرجات:

الصف 4: بعد ملأ vecList برقم 2 تكون: 2 2 2 2 2 2 2 2

الصف 8: بعد ملأ أول ثلاثة عناصر برقم 5 تكون:

2 2 2 2 2 5 5 5

البيانات في الصفوف 1 و2 تعلن أن vecList حاوية تتابعية حجمها 8 وأن screen مكرر ostream مهياً عند cout مع مسافة رموز معينة للحد. البيان في الصف 3 يستخدم الدالة fill لملأ vecList

بالرقم 2 أي أن جميع العناصر الثمانية من vecList محددة عند 2. تذكر أن () vecList.begin ينتج مكرر للعنصر الأول من vecList ويقوم () vecList.end بانتاج مكرر للعنصر الأخير من vecList. البيان في الصف 5 يقوم باخراج عناصر vecList باستخدام الدالة copy والبيان في الصف 7 يستخدم الدالة fill_n لتخزين 5 في عناصر vecList. المعامل الأول للدالة fill_n هو () vecList.begin الذي يحدد موضع بداية النسخ. المعامل الثاني للدالة fill_n هو 3 الذي يحدد عدد العناصر التي يتم ملأها. المعامل الثالث وهو 5 يحدد رمز الملاء. لهذا يتم نسخ 5 داخل أول ثلاثة عناصر من vecList. البيان في الصف 9 يقوم باخراج عناصر vecList.

الدالات generate و generate_n:

يتم استخدام الدالة generate والدالة generate_n لتوليد عناصر وملاء التتابع. يتم تعريف كلا من هاتين الدالتين في الملف الرئيسي algorithm ونماذج هذه الدالات تكون:

```
template<class forwardItr, class function>
void generate(forwardItr first, forwardItr last, function gen);

template<class forwardItr, class size, class function>
void generate_n(forwardItr first, size n, function gen);
```

الدالة generate تملأ تتابع في النطاق $first \dots last - 1$ مع استدعاءات متتالية للدالة () gen. الدالة generate_n تملأ تتابع في النطاق $first \dots first + n - 1$ أي بدءاً من الموضع first مع استدعاءات متتالية للدالة () gen. لاحظ أن gen قد تكون كذلك مؤشر الى دالة ما. فضلاً عن هذا اذا كانت gen دالة اذن يجب ان تكون دالة منتجة للقيمة بدون عوامل. البرنامج في المثال 9-13 يوضح كيفية استخدام هذه الدالات.

مثال 9-13:

//STL Functions generate and generate_n

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;
```

```

int nextNum();

int main()
{
    vector<int> vecList(8); //Line 1

    ostream_iterator<int> screen(cout, " "); //Line 2

    generate(vecList.begin(), vecList.end(), nextNum); //Line 3

    cout<<"Line 4: vecList after filling with "
         <<"numbers: "; //Line 4

    copy(vecList.begin(), vecList.end(), screen); //Line 5
    cout<<endl; //Line 6

    generate_n(vecList.begin(), 3, nextNum); //Line 7

    cout<<"Line 8: vecList after filling the first three "
         <<"elements " <<endl
         <<"          with the next number: "; //Line 8
    copy(vecList.begin(), vecList.end(), screen); //Line 9
    cout<<endl; //Line 10

    return 0;
}

int nextNum()
{
    static int n = 1;

    return n++;
}

```

المخرجات:

الصف 4: vecList بعد ملأها بالأعداد: 1 2 3 4 5 6 7 8

الصف 8: vecList بعد ملأ أول ثلاثة عناصر

بالعدد التالي: 9 10 11 4 5 6 7 8

يحتوي هذا البرنامج على الدالة المنتجة للقيمة nextNum التي تحتوي على متغير ساكن n مهياً عند استدعاء هذه الدالة ينتج القيمة الحالية لـ n ثم يزيد قيمة n. لهذا يقوم الاستدعاء الأول للدالة nextNum بانتاج 1 والاستدعاء الثاني ينتج 2 وهكذا.

البيانات في الصفوف 1 و2 تعلن أن vecList حاوية تتابعية حجمها 8 وأن screen مكرر ostream مهياً عند cout مع مسافة رموز معينة للحد. البيان في الصف 3 يستخدم الدالة generate لملأ vecList عن طريق الاستدعاء المتتالي للدالة nextNum. لاحظ أنه بعد تنفيذ البيان في الصف 3 تكون قيمة المتغير الساكن n من nextNum هي 9. البيان في الصف 5 يقوم باخراج عناصر .vecList

البيان في الصف 7 يستدعي الدالة generate_n لملاً أول ثلاثة عناصر من vecList عن طريق استدعاء الدالة nextNum ثلاث مرات. موضع البداية هو () vecList.begin وهو أول عنصر من vecList وعدد العناصر التي يتم مملأها هو 3 ويتم اعطائه عن طريق المعامل الثاني للدالة generate_n (انظر الصف 7). البيان في الصف 9 يقوم باخراج عناصر vecList.

الدالات find، find_if، find_end، و find_first_of:

يتم استخدام الدالات find، find_if، find_end، و find_first_of لايجاد العناصر في نطاق محدد. يتم تعريف هذه الدالات في الملف الرئيسي algorithm. نماذج الدالتين find و find_if تكون كما يلي:

```
template<class inputItr, class size, class Type>
inputItr find(inputItr first, inputItr last,
              const Type& searchValue);

template<class inputItr, class unaryPredicate>
inputItr find_if(inputItr first, inputItr last, unaryPredicate op);
```

تقوم الدالة find ببحث نطاق العناصر first...last - 1 من أجل عنصر البحث searchValue. اذا تم العثور على عنصر البحث في هذا النطاق تنتج الدالة موضع العثور على searchValue في النطاق وبخلاف هذا تنتج last. تقوم الدالة find_if ببحث نطاق العناصر first...last - 1 من أجل العنصر الذي يكون له op (عنصر النطاق) true. اذا كان العنصر الذي يلي (rangeElement) op صحيح وتم العثور عليه تقوم الدالة بانتاج الموضع في نطاق محدد حيث تم العثور على العنصر وبخلاف هذا تنتج last.

المثال 10-13 يوضح كيفية استخدام الدالتين find و find_if:

مثال 10-13:

انظر الى البيانات التالية:

```
char cList[10] = {'a', 'i', 'C', 'd', 'e', 'f',
                 'o', 'H', 'u', 'j'};           //Line 1
vector<char> charList(cList, cList + 10);       //Line 2
vector<char>::iterator position;                //Line 3
```

بعد تنفيذ البيان في الصف 2 تكون الحاوية charList التي من النوع vector كما يلي:

```
charList = {'a', 'i', 'C', 'd', 'e', 'f', 'o', 'H', 'u', 'j'};
```

انظر الى البيان التالي:

```
position = find(charList.begin(), charList.end(), 'd');
```

يقوم هذا البيان ببحث charList من أجل الظهور الأول من 'd' وينتج مكرراً يتم تخزينه في position. بما أن 'd' هي الرمز الرابع في charList اذن موضعه يكون 3. لهذا يشير position الى العنصر الموجود في الموضع 3 في charList.

انظر الآن الى البيان التالي:

```
position = find_if(charList.begin(), charList.end(), isupper);
```

هذا البيان يستخدم الدالة find_if للعثور على أول حرف كبير في charList. لاحظ أن الدالة isupper من الملف الرئيسي ctype يتم تمريرها كمعامل ثالث للدالة find_if. أول جرف كبير في charList هو العنصر الثالث. لهذا بعد تنفيذ هذا البيان يشير position الى العنصر الثالث من charList.

نترك لك كتمرين كتابة برنامج يقوم باختبار الدالتين find و find_if (انظر تمرين البرمجة 1 في نهاية هذا الفصل).

بعد هذا نقوم بوصف الدالتين find_end و find_first_of. كل من هاتين الدالتين لهما صيغتين ونماذج الدالة find_end هي:

```
template<class forwardItr1, class forwardItr2>
forwardItr1 find_end(forwardItr1 first1, forwardItr1 last1,
                    forwardItr2 first2, forwardItr2 last2);
```

```
template<class forwardItr1, class forwardItr2,
         class binaryPredicate>
forwardItr1 find_end(forwardItr1 first1, forwardItr1 last1,
                    forwardItr2 first2, forwardItr2 last2,
                    binaryPredicate op);
```

كل من صيغتي الدالة find_end تقوم ببحث النطاق first1...last1-1 من أجل الظهور الأخير من النطاق first2...last2-1. اذا كان البحث ناجحاً تقوم الدالة بانتاج الموضع في first2...last1-1 حيث يظهر ما يناسبه وبخلاف هذا تنتج last1. هذا يعني أن الدالة find_end تنتج موضع العنصر الأخير في النطاق first1...last1-1 حيث يكون النطاق first2...last2-1 نطاق فرعي من first1...last1-1. في الصيغة الأولى تتم مقارنة العناصر من أجل التساوي وفي الصيغة التالية يجب ان تكون المقارنة (elementFirstRange, elementSecondRange) صحيحة.

نماذج الدالة find_first_of هي:

```

template<class forwardItr1, class forwardItr2>
forwardItr1 find_first_of(forwardItr1 first1, forwardItr1 last1,
                          forwardItr2 first2, forwardItr2 last2);

template<class forwardItr1, class forwardItr2,
         class binaryPredicate>
forwardItr1 find_first_of(forwardItr1 first1, forwardItr1 last1,
                          forwardItr2 first2, forwardItr2 last2,
                          binaryPredicate op);

```

ي الصيغة الا

يكون موجود كذلك في النطاق first1...last1-1.

الصيغة الثانية تنتج موضع أول عنصر من first2...last2-1 ضمن النطاق first1...last1-1 الذي تكون له (elemRange1, elemRange2) op صحيح. اذا لم يتم العثور على ما يناسب البحث تقوم كل من الصيغتين بانتاج last1-1.

المثال 11-13 يوضح كيفية استخدام الدالات find_end و find_first_of.

مثال 11-13:

افترض أن لديك البيانات التالية:

```

int list1[10] = {12, 34, 56, 21, 34, 78, 34, 56, 12, 25};
int list2[2] = {34, 56};
int list3[5] = {33, 48, 21, 34, 73};
vector<int>::iterator location;

```

انظر الى البيان التالي:

```
location = find_end(list1, list1 + 10, list2, list2 + 2);
```

هذا البيان يستخدم الدالة find_end لايجاد آخر ظهور من list2 كتتابع فرعي من list1. الظهور الأخير من list2 في list1 يبدأ عند الموضع 6 (أي عند العنصر السابع). لهذا بعد تنفيذ هذا البيان يشير location الى العنصر الموجود في الموضع 6 في list1 وهو سابع عنصر من list1. انظر الآن الى البيان التالي:

```
location = find_first_of(list1, list1 + 10, list3, list3 + 5);
```

هذا البيان يستخدم الدالة find_first_of لايجاد الموضع في list1 حيث يكون العنصر الأول من list3 عنصر أيضاً من list1. العنصر الأول من list3 والذي هو عنصر أيضاً من list1 هو 34 وموضعه في list1 هو 1 وهو ثاني عنصر من list1. لهذا بعد تنفيذ هذا البيان يشير location الى العنصر الموجود في الموضع 1 في list1 وهو العنصر الثاني من list1. نترك لك كتمرين كتابة برنامج يقوم باختبار الدالتين find_end و find_first_of (انظر تمرين البرمجة 2 في نهاية هذا الفصل).

الدالات `remove`، `remove_if`، `remove_copy`، و `remove_copy_if`:

يتم استخدام الدالة `remove` لازالة عناصر معينة من تتابع ما ويتم استخدام الدالة `remove_if` لازالة العناصر من تتابع ما باستخدام بعض المعايير. الدالة `remove_copy` تقوم بنسخ عناصر التتابع داخل تتابع آخر عن طريق استبعاد عناصر معينة من التتابع الأول. بالمثل تقوم الدالة `remove_copy_if` بنسخ عناصر التتابع داخل تتابع آخر عن طريق استبعاد عناصر معينة من التتابع الأول باستخدام بعض المعايير. يتم تعريف هذه الدالات في الملف الرئيسي `algorithm`. نماذج الدالتين `remove` و `remove_if` هي:

```
template<class forwardItr, class Type>
forwardItr remove(forwardItr first, forwardItr last,
                  const Type& value);

template<class forwardItr, class unaryPredicate>
forwardItr remove_if(forwardItr first, forwardItr last,
                    unaryPredicate op);
```

تقوم الدالة `remove` بازالة كل ظهور من عنصر محدد في النطاق `first...last-1`. يتم تمرير العنصر الذي يتم حذفه كمعامل ثالث لهذه الدالة. الدالة `remove_if` تزيل هذه العناصر في النطاق `first...last-1` التي يكون الأصل `(element)` صحيح بالنسبة اليها. كل من هاتين الدالتين تنتج مكرراً أمامياً يشير الى الموضع التالي للعنصر الأخير من النطاق الجديد للعناصر. هذه الدالات لا تقوم بتعديل حجم الحاوية ولكن في الحقيقة يتم نقل العناصر الى بداية الحاوية. على سبيل المثال اذا كان التتابع هو (3، 7، 2، 5، 7، 9) والعنصر الذي يتم حذفه هو 7. اذن بعد حذف 7 يكون التتابع الناتج هو: (3، 2، 5، 9، 7). تقوم الدالة بانتاج مؤشر الى العنصر 9 (الذي يأتي بعد 5). البرنامج في مثال 12-13 يوضح أهمية هذا المكرر الامامي الذي تم انتاجه. (انظر الصفوف 8، و10، و12، و14).

لنقم الآن بالنظر الى نماذج الدالات `remove_copy`، `remove_copy_if`.

```
template<class inputItr, class outputItr, class Type>
outputItr remove_copy(inputItr first1, inputItr last1,
                     outputItr destFirst, const Type& value);

template<class inputItr, class outputItr, class unaryPredicate>
outputItr remove_copy_if(inputItr first1, inputItr last1,
                        outputItr destFirst,
                        unaryPredicate op);
```

تقوم الدالة `remove_copy` بنسخ جميع العناصر الموجودة في النطاق `first1...last1-1` فيما عدا العناصر التي تحددها `value` داخل المتتابع بدءاً من الموضع `destFirst`. بالمثل تقوم الدالة `remove_copy_if` بنسخ جميع العناصر الموجودة في النطاق `first1...last1-1` فيما عدا العناصر التي يكون `op(element)` صحيح بالنسبة إليها داخل المتتابع بدءاً من الموضع `destFirst`. كل من هاتين الدالتين تنتجان مكرر اخراج يشير الى الموضع التالي لآخر عنصر تم نسخه. البرنامج في مثال 12-13 يوضح كيفية استخدام الدالات `remove` و `remove_if` و `remove_copy` و `remove_copy_if`.

مثال 12-13:

```
//STL Functions remove, remove_if, remove_copy, and
//remove_copy_if

#include <iostream>
#include <cctype>
```

```
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

bool lessThanEqualTo50(int num);
int main()
{
    char cList[10] = {'A', 'a', 'A', 'B', 'A',
                     'c', 'D', 'e', 'F', 'A'};           //Line 1

    vector<char> charList(cList, cList + 10);           //Line 2
    vector<char>::iterator lastElem, newLastElem;       //Line 3

    ostream_iterator<char> screen(cout, " ");           //Line 4

    cout<<"Line 6: Character list: ";                   //Line 5
    copy(charList.begin(), charList.end(), screen);     //Line 6
    cout<<endl;                                          //Line 7

    //remove
    lastElem = remove(charList.begin(),
                     charList.end(), 'A');               //Line 8

    cout<<"Line 9: Character list after removing A: ";   //Line 9
    copy(charList.begin(), lastElem, screen);           //Line 10
    cout<<endl;                                          //Line 11

    //remove_if
    newLastElem = remove_if(charList.begin(),
                           lastElem, isupper);          //Line 12
    cout<<"Line 13: Character list after removing "
    <<"the uppercase letters: "<<endl;                  //Line 13
    copy(charList.begin(), newLastElem, screen);        //Line 14
    cout<<endl<<endl;                                    //Line 15

    int list[10] = {12, 34, 56, 21, 34,
                   78, 34, 55, 12, 25};                 //Line 16

    vector<int> intList(list, list + 10);               //Line 17
    vector<int>::iterator endElement;                   //Line 18

    ostream_iterator<int> screenOut(cout, " ");         //Line 19

    cout<<"Line 20: intList: ";                         //Line 20
    copy(intList.begin(), intList.end(), screenOut);    //Line 21
    cout<<endl;                                          //Line 22
}
```

```

vector<int> templ(10); //Line 23

//remove_copy
endElement = remove_copy(intList.begin(), intList.end(),
                           templ.begin(), 34); //Line 24

cout<<"Line 25: templ after copying all the "
    <<"elements of intList except 34: "<<endl; //Line 25
copy(templ.begin(), endElement, screenOut); //Line 26
cout<<endl; //Line 27

vector<int> temp2(10, 0); //Line 28

//remove_copy_if
remove_copy_if(intList.begin(), intList.end(),
               temp2.begin(), lessThanEqualTo50); //Line 29

cout<<"Line 30: temp2 after copying all the elements of "
    <<"intList except \nnumbers less than 50: "; //Line 30
copy(temp2.begin(), temp2.end(), screenOut); //Line 31
cout<<endl; //Line 32

return 0;
}

bool lessThanEqualTo50(int num)
{
    return (num <= 50);
}

```

المخرجات:

الصف 6: قائمة الحروف: A F e D c A B A a A

الصف 9: قائمة الحروف بعد حذف A: F e D c B a

الصف 13: قائمة الحروف بعد حذف الاحرف الكبيرة: e c a

الصف 20: intList: 25 12 55 34 78 34 21 56 34 12

الصف 25: templ بعد نسخ جميع عناصر intList عدا 34:

25 12 55 78 21 56 12

الصف 30: temp2 بعد نسخ جميع عناصر intList عدا

الأعداد الأصغر من 50: 0 0 0 0 0 0 55 78 56

البيان في الصف 2 يصنع قائمة vector اسمها charList من النوع char ويقوم بتهيئة charList باستخدام المصفوفة cList التي تم عملها في الصف 1. البيان في الصف 3 يعلن مكررين vector هما lastElem، وnewLastElem. البيان في الصف 4 يعلن مكرر ostream اسمه screen. البيان في الصف 6 يقوم باخراج قيمة charList والبيان في الصف 8 يستخدم الدالة remove لازالة جميع حالات ظهور 'A' من charList. تقوم الدالة بانتاج مكرر موجود في موضع يلي آخر عنصر من

النطاق الجديد والذي يتم تخزينه في lastElem. البيان في الصف 10 يقوم باخراج عناصر النطاق الجديد.

(لاحظ أن البيان في الصف 10 يخرج العناصر في النطاق lastElem... charList.begin ()
1. البيان في الصف 12 يستخدم الدالة remove_if لازالة الحروف الكبيرة من القائمة charList
ويقوم بتخزين المؤشر الناتج من الدالة remove_if في newLastElem. البيان في الصف 14 يقوم
باخراج عناصر النطاق الجديد.

البيان في الصف 17 يصنع حاوية vector تسمى intList من النوع int ويقوم بتهيئة intList
باستخدام المصفوفة list التي تم عملها في الصف 16. البيان في الصف 21 يقوم باخراج عناصر
intList. البيان في الصف 24 يقوم بنسخ جميع العناصر عدا حالات ظهور 34 من intList الى
داخل temp1. القائمة intList لم يتم تعديلها. البيان في الصف 26 يقوم باخراج عناصر temp1
والبيان في الصف 28 يصنع حاوية vector تسمى temp2 من النوع int وبها 10 مكونات ويقوم
بتهيئة جميع عناصر temp2 عند صفر. البيان في الصف 29 يستخدم الدالة remove_copy_if
لنسخ هذه العناصر من intList التي تكون أثل من أو تساوي 50. البيان في الصف 31 يقوم باخراج
عناصر temp2.

الدالات replace، و replace_if، و replace_copy، و replace_copy_if:

الدالة replace يتم استخدامها لاستبدال جميع حالات ظهور عنصر محدد ضمن نطاق محدد بقيمة
جديدة ويتم استخدام الدالة replace_if لاستبدال قيم العناصر التي تلبي معيار معين ضمن نطاق محدد
بقيمة جديدة. نماذج هذه الدالات هي:

```
template<class forwardItr, class Type>
void replace(forwardItr first, forwardItr last,
             const Type& oldValue, const Type& newValue);

template<class forwardItr, class unaryPredicate, class Type>
void replace_if(forwardItr first, forwardItr last,
                unaryPredicate op, const Type& newValue);
```

تقوم الدالة replace باستبدال جميع العناصر الموجودة في النطاق first...last-1 التي تكون قيمها
متساوية مع oldValue (القيمة القديمة) بالقيمة التي تحددها newValue (القيمة الجديدة). تقوم الدالة
replace_if باستبدال جميع العناصر في النطاق first...last-1 التي يكون op(element) صحيح
بالنسبة اليها بالقيمة المحددة بواسطة newValue (القيمة الجديدة).

الدالة `replace_copy` عبارة عن اندماج من الدالة `replace` والدالة `copy`. بالمثال الدالة `replace_copy_if` عبارة عن اندماج من الدالة `replace_if` والدالة `copy`. لنقم أولاً بالنظر الى نماذج الدالات `replace_copy`، و `replace_copy_if`:

```
template<class inputItr, class outputItr, class Type>
outputItr replace_copy(forwardItr first, forwardItr last,
                       outputItr destFirst,
                       const Type& oldValue,
                       const Type& newValue);

template<class forwardItr, class outputItr,
         class unaryPredicate, class Type>

outputItr replace_copy_if(forwardItr first, forwardItr last,
                          outputItr destFirst,
                          unaryPredicate op,
                          const Type& newValue);
```

تقوم الدالة `replace_copy` بنسخ جميع العناصر الموجودة في النطاق `first...last-1` داخل الحاوية بدءاً من `destFirst`. اذا كانت قيمة عنصر ما في هذا النطاق متساوية مع القيمة القديمة يتم استبدالها بالقيمة الجديدة. الدالة `replace_copy_if` تقوم بنسخ جميع العناصر في النطاق `first...last-1` داخل الحاوية بدءاً من `destFirst`. اذا كانت `op (element)` صحيحة بالنسبة الى أي عنصر في هذا النطاق فان قيمته يتم استبدالها بالقيمة الجديدة. هذه الدالات تنتج مكرر اخراج (مؤشر) يكون موضعه بعد آخر عنصر تم نسخه في المكان المقصود.

المثال 13-13 يوضح كيفية استخدام الدالات `replace` و `replace_if` و `replace_copy` و `replace_copy_if`.

مثال 13-13:

انظر الى البيانات التالية:

```
char cList[10] = {'A', 'a', 'A', 'B', 'A',           //Line 1
                 'c', 'D', 'e', 'F', 'A'};
vector<char> charList(cList, cList + 10);           //Line 2
```

بعد تنفيذ البيان في الصف 2 تكون الحاوية `charList` كما يلي:

```
charList = {'A', 'a', 'A', 'B', 'A', 'c',           //Line 3
            'D', 'e', 'F', 'A'}
```

انظر الآن الى البيان التالي:

```
replace(charList.begin(), charList.end(), 'A', 'Z'); //Line 4
```

هذا البيان يستخدم الدالة replace لاستبدال جميع حالات ظهور 'A' بالحرف 'Z' في charList. بعد تنفيذ هذا البيان تكون charList هي:

```
charList ={'Z', 'a', 'Z', 'B', 'Z', 'c', 'D',  
           'e', 'F', 'Z'} //Line 5
```

انظر الآن الى البيان التالي:

```
replace_if(charList.begin(), charList.end(),  
           isupper, '*'); //Line 6
```

هذا البيان يستخدم الدالة replace_if لاستبدال الحروف الكبيرة بالرمز '*' في القائمة charList. بعد تنفيذ هذا البيان تكون charList هي:

```
charList ={'*', 'a', '*', '*', '*', 'c', '*',  
           'e', '*', '*'} //Line 7
```

بعد هذا افترض أن لديك البيانات التالية:

```
int list[10] = {12, 34, 56, 21, 34, 78, 34, 55, 12, 25}; //Line 8  
vector<int> intList(list, list + 10); //Line 9  
vector<int> temp(10); //Line 10
```

البيان في الصف 9 يصنع حاوية vector تسمى intList ومن النوع int ويقوم بتهيئة intList باستخدام المصفوفة list التي تم عملها في الصف 8. بعد تنفيذ البيان في الصف 9 تكون intList:

```
intList = {12, 34, 56, 21, 34, 78, 34, 55, 12, 25}
```

البيان في الصف 10 يعلن أن الحاوية temp من النوع int والآن انظر الى البيان التالي:

```
replace_copy(intList.begin(), intList.end(),  
            temp.begin(), 34, 0); //Line 11
```

هذا البيان يقوم بنسخ جميع عناصر intList داخل temp ويتم استبدال 34 بالقيمة صفر. القائمة intList غير معدلة وبعد هذا البيان temp يكون:

```
temp = {12, 0, 56, 21, 0, 78, 0, 55, 12, 25}
```

الآن افترض أن لديك تعريف الدالة التالي:

```
bool lessThanOrEqualTo50(int num) //Line 12  
{  
    return (num <= 50);  
}
```

الدالة `lessThanEqualTo50` تنتج `true` اذا كان `num` (العدد) أصغر من أو يساوي 50 وبخلاف هذا تنتج `false`. انظر الى البيان التالي:

```
replace_copy_if(intList.begin(), intList.end(),
               temp.begin(), lessThanEqualTo50, 50);    //Line 13
```

هذا البيان يستخدم الدالة `replace_copy_if` لنسخ عناصر `intList` داخل `temp` ويستبدل جميع العناصر الأصغر من أو تساوي 50 بالعدد 50. لاحظ أن المعامل الرابع للدالة `replace_copy_if` هو الدالة `lessThanEqualTo50`. بعد تنفيذ البيان في الصف 13 يكون `temp` كما يلي:

نترك لك كتمرير `temp = {50, 50, 56, 50, 50, 78, 50, 55, 50, 50}` و `replace_copy`، و `replace_copy_if` (انظر تمرين البرمجة رقم 3 في نهاية هذا الفصل).

الدالات `swap`، `iter_swap` و `swap_ranges`:

يتم استخدام الدالات `swap`، `iter_swap` و `swap_ranges` لتبادل العناصر. يتم تعريف هذه الدالات في الملف الرئيسي `algorithm`. نماذج هذه الدالات هي:

```
template<class Type>
void swap(Type& object1, Type& object2);

template<class forwardItr1, class forwardItr2>
void iter_swap(forwardItr1 first, forwardItr2 second);

template<class forwardItr1, class forwardItr2>
forwardItr2 swap_ranges(forwardItr1 first1, forwardItr1 last1,
                       forwardItr2 first2);
```

تقوم الدالة `swap` بتبادل قيم الهدف الاول والهدف الثاني وتقوم الدالة `iter_swap` بتبادل القيم التي يشير اليها المكرران `first` و `second`. تقوم الدالة `swap_ranges` بتبادل عناصر النطاق `first1...last1-1` بالعناصر المتتالية بدءاً من الموضع `first2`. انها تنتج مكرر النطاق الثاني الموجود بعد آخر عنصر يتم تبادله بموضع واحد. البرنامج الموجود في المثال 13-14 يوضح كيفية استخدام هذه الدالات.

```

//STL functions swap, iter_swap, and swap_ranges

#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>

using namespace std;

int main()
{
    char cList[10] = {'A', 'B', 'C', 'D', 'F',
                     'G', 'H', 'I', 'J', 'K'};           //Line 1

    vector<char> charList(cList, cList + 10);           //Line 2
    vector<char>::iterator charItr;                     //Line 3

    ostream_iterator<char> screen(cout, " ");           //Line 4

    cout<<"Line 5: Character list: ";                   //Line 5
    copy(charList.begin(), charList.end(), screen);     //Line 6
    cout<<endl;                                          //Line 7

    //swap
    swap(charList[0], charList[1]);                     //Line 8

    cout<<"Line 9: Character list after swapping the "
        <<"first and second elements: "<<endl;         //Line 9
    copy(charList.begin(), charList.end(), screen);     //Line 10
    cout<<endl;                                          //Line 11

    //iter_swap
    iter_swap(charList.begin() + 2,
              charList.begin() + 3);                   //Line 12

    cout<<"Line 13: Character list after swapping the "
        <<"third and fourth elements: "<<endl;         //Line 13
}

```

```

copy(charList.begin(), charList.end(), screen); //Line 14
cout<<endl; //Line 15

charItr = charList.begin() + 4; //Line 16
iter_swap(charItr, charItr + 1); //Line 17

cout<<"Line 18: Character list after swapping the "
    <<"fifth and sixth elements: "<<endl; //Line 18
copy(charList.begin(), charList.end(), screen); //Line 19
cout<<endl<<endl; //Line 20

int list[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; //Line 21

vector<int> intList(list, list + 10); //Line 22

ostream_iterator<int> screenOut(cout, " "); //Line 23

cout<<"Line 24: intList: "; //Line 24
copy(intList.begin(), intList.end(), screenOut); //Line 25
cout<<endl; //Line 26

    //swap_ranges
swap_ranges(intList.begin(), intList.begin() + 4,
            intList.begin() + 5); //Line 27

cout<<"Line 28: intList after swapping the first "
    <<"four elements with the \n         four elements "
    <<"starting at the sixth element of intList: "
    <<endl; //Line 28
copy(intList.begin(), intList.end(), screenOut); //Line 29
cout<<endl; //Line 30

swap_ranges(list, list + 10, intList.begin()); //Line 31

cout<<"Line 32: list and intList after swapping "
    <<"the elements with each other: "<<endl; //Line 32
cout<<"Line 33: list: "; //Line 33
copy(list, list + 10, screenOut); //Line 34
cout<<endl; //Line 35
cout<<"Line 36: intList: "; //Line 36
copy(intList.begin(), intList.end(), screenOut); //Line 37
cout<<endl; //Line 38

return 0;
}

```

المخرجات:

الصف 5: قائمة الحروف: K J I H G F E D C B A

الصف 9: قائمة الحروف بعد تبادل العنصرين الأول والثاني:

K J I H G F D C A B

الصف 13: قائمة الحروف بعد تبادل العنصرين الثالث والرابع:

K J I H G F C D A B

الصف 18: قائمة الحروف بعد تبادل العنصرين الخامس والسادس:

K J I H F G C D A B

الصف 24: intList: 10 9 8 7 6 5 4 3 2 1

الصف 28: intList بعد تبادل الأربعة عناصر الأولى بأربعة عناصر بدءاً من العنصر السادس من

intList: 10 4 3 2 1 5 9 8 7 6

الصف 32: القائمة و intList بعد تبادل عناصرهما مع بعضهما البعض:

الصف 33: 10 4 3 2 1 5 9 8 7 6

الصف 36: intList: 10 9 8 7 6 5 4 3 2 1

البيان في الصف 2 يصنع حاوية vector تسمى charList ويقوم بتهيئتها باستخدام المصفوفة cList

التي تم اعلانها في الصف 1. البيان في الصف 6 يقوم باخراج قيم charList والبيان في الصف 8

يبادل بين العنصرين الأول والثاني من charList. البيان في الصف 12 يقوم باستخدام الدالة

iter_swap بمبادلة العنصرين الثالث والرابع من charList. (تذكر أن موضع العنصر الأول في

charList هو 0). بعد تنفيذ البيان الموجود في الصف 16 تشير charItr الى العنصر الخامس من

charList. البيان في الصف 17 يستخدم المكرر charItr لتبادل العنصرين الخامس والسادس من

charList. البيان في الصف 19 يقوم باخراج قيم عناصر charList. (في المخرجات يحتوي السطر

الذي يميز الصف 18 على مخرجات الصفوف من 18 حتى 20 من البرنامج.)

البيان في الصف 22 يصنع الحاوية vector التي تسمى intList ويقوم بتهيئتها باستخدام المصفوفة

المعلنة في الصف 21. البيان في الصف 25 يقوم باخراج قيم عناصر intList. البيان في الصف 27

يستخدم الدالة swap_ranges لتبادل أول أربعة عناصر من intList بأربعة عناصر من intList

بدءاً من العنصر السادس من intList. البيان في الصف 29 يقوم باخراج عناصر intList. (في

المخرجات يحتوي السطر الذي يميز الصف 28 على مخرجات الصفوف من 28 حتى 30 من

البرنامج.)

البيان في الصف 31 يقوم بتبادل عناصر المصفوفة list مع عناصر الحاوية intList. البيان في

الصف 34 يقوم باخراج عناصر المصفوفة list والبيان في الصف 37 يقوم باخراج intList.

الدالات search_n، sort، وbinary_search:

يتم استخدام الدالات search_n، sort، وbinary_search لبحث العناصر. يتم تعريف

هذه الدالات في الملف الرئيسي algorithm.

نماذج الدالة search هي:

```
template<class forwardItr1, class forwardItr2>
forwardItr1 search(forwardItr1 first1, forwardItr1 last1,
                  forwardItr2 first2, forwardItr2 last2);
```

```
template<class forwardItr1, class forwardItr2,
        class binaryPredicate>
```

```
forwardItr1 search(forwardItr1 first1, forwardItr1 last1,
                  forwardItr2 first2, forwardItr2 last2,
                  binaryPredicate op);
```

ان الدالة search باعطائها نطاقين هما 1- first1...last1 والنطاق 1- first2...last2 تقوم
 ببحث العنصر الأول في النطاق 1- first1...last1 حيث يظهر النطاق 1- first2...last2 كنطاق
 فرعي من 1- first1...last1. الصيغة الأولى تقوم بعمل مقترنة تساوي بين عناصر النطاقين.
 بالنسبة الى الصيغة الثانية يجب أن تكون المقارنة
 op
 (elemFirstRange, elemSecindRange) صحيحة. اذا تم العثور على عنصر مناسب تقوم
 الدالة بانتاج الموضع في النطاق 1- first1...last1 حيث يظهر العنصر المناسب وبخلاف هذا تقوم
 الدالة بانتاج false.

نماذج الدالة search_n هي:

```
template<class forwardItr, class size, class Type>
forwardItr search_n(forwardItr first, forwardItr last,
                   size count, const Type& value);
```

```
template<class forwardItr, class size, class Type,
        class binaryPredicate>
forwardItr search_n(forwardItr first, forwardItr last,
                   size count, const Type& value,
                   binaryPredicate op);
```

الدالة search_n باعطائها نطاق من العناصر 1- first...last تقوم بالبحث عن حالات الظهور
 المتتالية من القيمة. تقوم الصيغة الأولى بانتاج الموضع الموجود في النطاق 1- first...last حيث
 يكون للتتابع الفرعي من العناصر المتتالية count قيم متساوية مع value. الصيغة الثانية تنتج
 الموضع الموجود في النطاق 1- first...last حيث يتواجد التتابع الفرعي من العناصر المتتالية
 count وهذا التتابع تكون (elemRange, value) op صحيحة بالنسبة اليه. اذا لم يتم العثور على
 عنصر مناسب تقوم كلتا الدالتين بانتاج last.

نماذج الدالة sort هي:

```
template<class randomAccessItr>
void sort(randomAccessItr first, randomAccessItr last);
```

```
template<class randomAccessItr, class compare>
void sort(randomAccessItr first, randomAccessItr last,
          compare op);
```

الصيغة الأولى من الدالة sort باعادة ترتيب العناصر في النطاق first...last-1 ترتيباً تصاعدياً والصيغة التالية تقوم باعادة ترتيب العناصر وفقاً لمعيار يتحدد بواسطة op. نماذج الدالة binary_search هي:

```
template<class forwardItr, class Type>
bool binary_search(forwardItr first, forwardItr last,
                  const Type& searchValue);

template<class forwardItr, class Type, class compare>
bool binary_search(forwardItr first, forwardItr last,
                  const Type& searchValue, compare op);
```

الصيغة الأولى تقوم بإنتاج true إذا تم العثور على searchValue (قيمة البحث) في النطاق first...last-1 وبخلاف هذا تنتج false. الصيغة الثانية تستخدم هدف الدالة op الذي يقوم بتحديد معيار البحث.

المثال 13-15 يوضح كيفية استخدام خوارزميات البحث والترتيب هذه.

مثال 13-15:

```
//STL Functions search, search_n, sort, and binary_search

#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

int main()
{
    int intList[15] = {12, 34, 56, 34, 34,
                      78, 38, 43, 12, 25,
                      34, 56, 62, 5, 49}; //Line 1

    vector<int> vecList(intList, intList + 15); //Line 2
    int list[2] = {34, 56}; //Line 3

    vector<int>::iterator location; //Line 4

    ostream_iterator<int> screenOut(cout, " "); //Line 5

    cout<<"Line 6: vecList: "; //Line 6
    copy(vecList.begin(), vecList.end(), screenOut); //Line 7
    cout<<endl; //Line 8

    cout<<"Line 9: list: "; //Line 9
    copy(list, list + 2, screenOut); //Line 10
    cout<<endl; //Line 11

    //search
    location = search(vecList.begin(), vecList.end(), //Line 12
                    list, list + 2);

    if(location != vecList.end()) //Line 13
        cout<<"Line 14: list found in vecList. The "
            <<"first occurrence of \n    list in vecList "
            <<"is at position: "
            <<(location - vecList.begin())<<endl; //Line 14
```

```

else //Line 15
    cout<<"Line 16: list is not in vecList"<<endl; //Line 16

    //search_n
    location = search_n(vecList.begin(), vecList.end(), //Line 17
        2, 34);

    if(location != vecList.end()) //Line 18
        cout<<"Line 19: Two consecutive occurrences of "
            <<"34 found in \n    vecList at position: "
            <<(location - vecList.begin())<<endl; //Line 19
    else //Line 20
        cout<<"Line 21: Two consecutive occurrences of "
            <<"34 not in vecList."<<endl; //Line 21

    //sort
    sort(vecList.begin(), vecList.end()); //Line 22

    cout<<"Line 23: vecList after sorting:"
        <<endl<<"    "; //Line 23
    copy(vecList.begin(), vecList.end(), screenOut); //Line 24
    cout<<endl; //Line 25

    //binary_search
    bool found; //Line 26

    found = binary_search(vecList.begin(), //Line 27
        vecList.end(), 43);

    if(found) //Line 28
        cout<<"Line 29: 43 found in vecList"<<endl; //Line 29
    else //Line 30
        cout<<"Line 31: 43 not in vecList"<<endl; //Line 31

    return 0;
}

```

المخرجات:

الصف 6: vecList: 12 34 34 78 38 43 12 25 34 56 62 5 49
 الصف 9: list: 34 56
 الصف 14: تم العثور على list داخل vecList. الظهور الأول من first في vecList عند الموضع:
 1

الصف 19: تم العثور على ظهورين متتاليين من 34 في vecList عند الموضع: 3
 الصف 23: vecList بعد الترتيب:

5 12 25 34 34 38 43 49 56 56 62 78

الصف 29: تم العثور على 43 في vecList.

البيان في الصف 2 يصنع حاوية vector تسمى vecList ويقوم بتهيئتها باستخدام المصفوفة intList التي تم عملها في الصف 1. البيان في الصف 3 يصنع مصفوفة تسمى list ومكونة من مكونين يقوم بتهيئتها.

البيان في الصف 7 يقوم باخراج vecList والبيان في الصف 12 يستخدم الدالة search ويقوم ببحث vecList للعثور على الموضع (أول ظهور) الموجود في vecList والذي يظهر به list كتتابع

فرعي. البيانات في الصفوف من 13 وحتى 16 تقوم باخراج نتيجة البحث وانظر الخط الذي يميز الصف 14 في المخرجات.

البيان في الصف 17 يستخدم الدالة search_n للعثور على الموضع الموجود في vecList حيث يظهر تتابعين متتاليين من 34. البيانات في الصفوف من 18 وحتى 21 تخرج نتيجة البحث.

البيان في الصف 22 يستخدم الدالة sort لترتيب vecList والبيان في الصف 24 يخرج vecList. في المخرجات يحتوى الخط الذي يميز الصف 23 على مخرجات البيانات في الصفوف من 23 الى 25 من البرنامج.

البيان في الصف 27 يستخدم الدالة binary_search لبحث vecList. البيانات في الصفوف من 28 وحتى 31 تقوم باخراج نتيجة البحث.

الدالات adjacent_find، merge، و inplace_merge:

يتم استخدام الخوارزمية adjacent_find للعثور على أول ظهور من العناصر المتتالية التي تلبي معيار معين. نماذج الدالات التي تقوم بتطبيق هذه الخوارزمية هي:

```
template<class forwardItr>
forwardItr adjacent_find(forwardItr first, forwardItr last);

template<class forwardItr, class binaryPredicate>
forwardItr adjacent_find(forwardItr first, forwardItr last,
                        binaryPredicate op);
```

الصيغة الأولى من adjacent_find تستخدم معيار التساوي أي أنها تبحث عن أول ظهور متتالي من نفس العنصر. في الصيغة الثانية تنتج الخوارزمية مكرر الى العنصر الموجود في النطاق first...last-1 والذي تكون له (elem, nextElem) op صحيحة حيث elem عنصر في النطاق first...last-1 وحيث nextElem عنصر في هذا النطاق يلي العنصر elem. اذا لم يتم العثور على عنصر مناسب تقوم كلتا الدالتين بإنتاج last.

افترض أن intList حاوية قائمة من النوع int. وافترض كذلك أن intList تكون:

```
intList = {0, 1, 1, 2, 3, 4, 4, 5, 6, 6}; //Line 1
```

انظر الى البيانات التالية:

```
list<int>::iterator listItr; //Line 2
listItr = adjacent_find(intList.begin(), intList.end()); //Line 3
```

البيان في الصف 2 يعلن أن listItr مكرر قائمة يمكنه أن يشير الى أي حاوية قائمة من النوع int. البيان في الصف 3 يستخدم الدالة adjacent_find للعثور على موضع (أول مجموعة من) العناصر المتطابقة المتتالية. تقوم الدالة بإنتاج مؤشر الى أول مجموعة من العناصر المتتالية التي يتم تخزينها في listItr. بعد تنفيذ البيان في الصف 3 تشير listItr الى العنصر الثاني من intList.

افترض الآن أن vecList حاوية vector من النوع int. افترض كذلك أن vecList تكون:

```
vecList = {1, 3, 5, 7, 9, 0, 2, 4, 6, 8}; //Line 4
//Line 5
vector<int>::iterator intItr;
intItr = adjacent_find(vecList.begin(), vecList.end(),
                       greater<int>()); //Line 6
```

البيان في الصف يعلن أن intItr مكرر vector يمكنه أن يشير إلى أي حاوية vector من النوع int. البيان في الصف 6 يستخدم الصيغة الثانية من الدالة adjacent_find للعثور على أول عنصر من vecList أكبر من العنصر التالي من vecList. لاحظ أن المعامل الثالث للدالة adjacent_find هو الأصل الثنائي greater (أكبر) الذي ينتج الموضع الموجود في vecList حيث يكون أول عنصر أكبر من العنصر الثاني. يتم تخزين الموضع الناتج في المكرر intItr. بعد تنفيذ البيان في الصف 6 يشير intItr إلى العنصر 9.

بعد هذا نناقش الخوارزمية merge (الدمج). تقوم الخوارزمية merge بدمج القوائم المرتبة ويتم تخزين النتيجة في قائمة مرتبة يجب أن يتم ترتيب كلتا القائمتين وفقاً لنفس المعيار. على سبيل المثال سوف تكون القائمتان مرتبتين ترتيباً تصاعدياً. نماذج الدالات لتطبيق خوارزميات merge هي:

```
template<class inputItr1, class inputItr2,
         class outputItr>
outputItr merge(inputItr1 first1, inputItr1 last1,
                inputItr2 first2, inputItr2 last2,
                outputItr destFirst);

template<class inputItr1, class inputItr2,
         class outputItr, class binaryPredicate>
outputItr merge(inputItr1 first1, inputItr1 last1,
                inputItr2 first2, inputItr2 last2,
                outputItr destFirst, binaryPredicate op);
```

كلتا الصيغتين من الخوارزمية merge تدمج عناصر النطاقات المرتبة 1- first1...last1 و 1- first2...last2. النطاق المقصود الذي يبدأ بالمكرر destFirst يحتوي على العناصر المدمجة. الصيغة الأولى تستخدم المعامل أصغر من > لترتيب العناصر. الصيغة الثانية تستخدم الأصل الثنائي op لترتيب العناصر أي أن op (elemRange1, elemRange2) يجب أن تكون true. تقوم كلتا الصيغتين بإنتاج الموضع التالي لآخر عنصر تم نسخه في النطاق المقصود. فضلاً عن هذا لا يتم تعديل نطاقات المصدر ويجب ألا يتداخل النطاق المقصود مع نطاقات المصدر.

انظر إلى البيانات التالية:

```
int list1[5] = {0, 2, 4, 6, 8}; //Line 7
int list2[5] = {1, 3, 5, 7, 9}; //Line 8
```

```
list<int> intList; //Line 9
merge(list1, list1 + 5, list2, list2 + 5,
      back_inserter(intList)); //Line 10
```

البيانات في الصفوف 7 و 8 تصنع المصفوفات المرتبة list1 و list2. البيانات في الصف 9 تعلن أن intList حاوية قائمة من النوع int. البيان في الصف 10 يستخدم الدالة merge لدمج list1 و list2. المعامل الخامس للدالة merge في الصف 10 هو استدعاء back_inserter (المدخل الخلفي) الذي يضع القائمة المدمجة في intList. بعد تنفيذ البيان في الصف 10 تحتوي intList على القائمة المدمجة أي أن:

```
intList = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

يتم استخدام الخوارزمية inplace_merge لدمج تتابعين متتاليين مرتبين. نماذج الدالات التي تطبق هذه الخوارزمية هي:

```
template<class biDirectionalItr>
void inplace_merge(biDirectionalItr first,
                  biDirectionalItr middle,
                  biDirectionalItr last);

template<class biDirectionalItr, class binaryPredicate>
void inplace_merge(biDirectionalItr first,
                  biDirectionalItr middle,
                  biDirectionalItr last,
                  binaryPredicate op);
```

كل من الصيغتين تقومان بدمج التتابعات المتتالية المرتبة first...middle-1 و middle...last-1. العناصر المدمجة تستبدل النطاقين التي تبدأ عند first. الصيغة الأولى تستخدم معيار أصغر من لدمج التتابعين المتتاليين والصيغة الثانية تستخدم الأصل الثنائي op لدمج التتابعات أي أنه بالنسبة الى عناصر التتابعين يجب أن تكون op(elemSeq1, elemSeq2) صحيحة. على سبيل المثال افترض أن:

```
vecList = {1, 3, 5, 7, 9, 2, 4, 6, 8}
```

حيث تكون vecList حاوية vector. افترض كذلك أن vecItr مكرر vector الذي يشير الى العنصر 2. اذن بعد تنفيذ البيان:

```
inplace_merge(vecList.begin(), vecItr, vecList.end());
```

تكون العناصر الموجودة في vecList مرتبة كما يلي:

```
vecList = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

نترك لك كتمرين كتابة برنامج يوضح كيفية استخدام الدالات adjacent_find و merge و inplace_merge (انظر تمرين البرمجة 4 في نهاية هذا الفصل).

الدالات reverse و reverse_copy و rotate و rotate_copy:
الخوارزمية reverse تعكس ترتيب العناصر في نطاق معطى. نموذج الدالة لتطبيق الخوارزمية reverse هو:

```
template<class biDirectionalItr>
void reverse(biDirectionalItr first, biDirectionalItr last);
```

يتم عكس العناصر في النطاق first...last-1 على سبيل المثال اذا كانت vecList = (1, 2, 5, 3, 4) اذن العناصر ذات الترتيب العكس هي vecList = (4, 3, 5, 2, 1).
تقوم الخوارزمية reverse_copy بعكس ترتيب العناصر في نطاق معطى أثناء نسخهم في النطاق المقصود. المصدر لا يتم تعديله ونموذج الدالة التي تقوم بتطبيق الخوارزمية reverse_copy هو:

```
template<class biDirectionalItr, class outputItr>
outputItr reverse_copy(biDirectionalItr first,
                      biDirectionalItr last,
                      outputItr destFirst);
```

يتم نسخ العناصر في النطاق first...last -1 بترتيب عكسي عند المكان المقصود الذي يبدأ بdestFirst. كما تقوم الدالة بانتاج مكرر موجود بعد آخر عنصر تم نسخه بالمكان المقصود.
تقوم الخوارزمية rotate بدوران عناصر نطاق محدد ونموذجها يكون:

```
template<class forwardItr>
void rotate(forwardItr first, forwardItr newFirst,
           forwardItr last);
```

يتم نقل العناصر الموجودة في النطاق first...newFirst-1 الى نهاية النطاق. العنصر المحدد بواسطة newFirst يصبح أول عنصر من النطاق. على سبيل المثال افترض أن:

```
vecList = {3, 5, 4, 0, 7, 8, 2, 5}
```

ويشير المكرر vecItr الى صفر. اذن بعد تنفيذ البيان:

```
rotate(vecList.begin(), vecItr, vecList.end());
```

تكون vecList كما يلي:

```
vecList = {0, 7, 8, 2, 5, 3, 5, 4}
```



```

listItr = intList.begin(); //Line 17
listItr++; //Line 18
listItr++; //Line 19

cout<<"Line 20: intList before rotating: "; //Line 20
copy(intList.begin(), intList.end(), screen); //Line 21
cout<<endl; //Line 22

//rotate
rotate(intList.begin(), listItr, intList.end()); //Line 23

cout<<"Line 24: intList after rotating: "; //Line 24
copy(intList.begin(), intList.end(), screen); //Line 25
cout<<endl; //Line 26

resultList.clear(); //Line 27

//rotate_copy
rotate_copy(intList.begin(), listItr, intList.end(), //Line 28
            back_inserter(resultList));

cout<<"Line 29: intList after rotating and "
    <<"copying: "; //Line 29
copy(intList.begin(), intList.end(), screen); //Line 30
cout<<endl; //Line 31

cout<<"Line 32: resultList after rotating and "
    <<"copying: "; //Line 32
copy(resultList.begin(), resultList.end(), //Line 33
    screen); //Line 34
cout<<endl; //Line 34

resultList.clear(); //Line 35

rotate_copy(intList.begin(),
            find(intList.begin(), intList.end(), 6), //Line 36
            intList.end(),
            back_inserter(resultList));

cout<<"Line 37: resultList after rotating and "
    <<"copying: "; //Line 37
copy(resultList.begin(), resultList.end(), //Line 38
    screen); //Line 39
cout<<endl; //Line 39

return 0;
}

```

المخرجات:

الصف 6: intList: 1 3 5 7 9 0 2 4 6 8
الصف 10: intList بعد العكس: 8 6 4 2 0 9 7 5 3 1
الصف 14: القائمة الناتجة: 1 3 5 7 9 0 2 4 6 8
الصف 20: intList قبل الدوران: 8 6 4 2 0 9 7 5 3 1
الصف 24: intList بعد الدوران: 6 8 1 3 5 7 9 0 2 4

الصف 29: intList بعد الدوران والنسخ: 6 8 1 3 5 7 9 0 2 4
الصف 32: القائمة الناتجة بعد الدوران والنسخ: 2 4 6 8 1 3 5 7 9 0
الصف 37: القائمة الناتجة بعد الدوران والنسخ: 8 1 3 5 7 9 0 2 4 6
المخرجات السابقة واضحة والتفاصيل متروكة كتمرين لك.

الدالات count، count_if، max، max_element، min، min_element، وrandom_shuffle:

تقوم الخوارزمية count باحصاء حالات ظهور قيمة معينة في نطاق معطى. نموذج الدالة التي تقوم بتطبيق هذه الخوارزمية هي:

```
template<class inputItr, class type>
iterator_traits<inputItr>::difference_type
count(inputItr first, inputItr last, const Type& value);
```

الدالة count تنتج عدد المرات التي تظهر بها القيمة المحددة بواسطة المعامل value في النطاق first...last-1.

تقوم الخوارزمية count_if باحصاء حالات ظهور قيمة معينة في نطاق معطى تلبي معيار معين. نموذج الدالة التي تقوم بتطبيق هذه الخوارزمية هو:

```
template<class inputItr, class unaryPredicate>
iterator_traits<inputItr>::difference_type
count_if(inputItr first, inputItr last, unaryPredicate op);
```

الدالة count_if تنتج عدد المرات في النطاق first...last-1 التي يكون لها (elemRange) op صحيح.

يتم استخدام الخوارزمية max لتحديد الحد الأقصى من بين قيمتين. هذه الخوارزمية لها صيغتان كما هو موضح عن طريق النماذج التالية:

```
template<class Type>
const Type& max(const Type& aVal, const Type& bVal);

template<class Type, class compare>
const Type& max(const Type& aVal, const Type& bVal, compare comp);
```

في الصيغة الأولى يتم استخدام المعامل أكبر من المتربط بType. في الصيغة الثانية يتم استخدام عملية المقارنة المحددة بواسطة comp.

يتم استخدام الخوارزمية max_element لتحديد أكبر عنصر في نطاق معطى. هذه الخوارزمية لها صيغتان كما هو موضح بواسطة النماذج التالية:

```
template<class forwardItr>
forwardItr max_element(forwardItr first, forwardItr last);

template<class forwardItr, class compare>
forwardItr max_element(forwardItr first, forwardItr last,
                      compare comp);
```

الصيغة الأولى تستخدم المعامل أكبر من المرتبط بنوع بيانات العناصر في النطاق `first...last-1` في الصيغة الثانية يتم استخدام عملية المقارنة المحددة بواسطة `comp`. تقوم كلتا الصيغتين بإنتاج مكرر إلى العنصر المحتوي على أكبر قيمة في النطاق `first...last-1`. يتم استخدام الخوارزمية `min` لتحديد الحد الأدنى من بين قيمتين. هذه الخوارزمية لها صيغتان كما هو موضح عن طريق النماذج التالية:

```
template<class Type>
const Type& min(const Type& aVal, const Type& bVal);

template<class Type, class compare>
const Type& min(const Type& aVal, const Type& bVal, compare comp);
```

في الصيغة الأولى يتم استخدام المعامل أصغر من المترابط بـ `Type`. في الصيغة الثانية يتم استخدام عملية المقارنة المحددة بواسطة `comp`. يتم استخدام الخوارزمية `min_element` لتحديد أصغر عنصر في نطاق معطى. هذه الخوارزمية لها صيغتان كما هو موضح بواسطة النماذج التالية:

```
template<class forwardItr>
forwardItr min_element(forwardItr first, forwardItr last);

template<class forwardItr, class compare>
forwardItr min_element(forwardItr first, forwardItr last,
                      compare comp);
```

الصيغة الأولى تستخدم المعامل أصغر من المرتبط بنوع بيانات العناصر في النطاق `first...last-1`. في الصيغة الثانية يتم استخدام عملية المقارنة المحددة بواسطة `comp`. تقوم كلتا الصيغتين بإنتاج مكرر إلى العنصر المحتوي على أصغر قيمة في النطاق `first...last-1`. يتم استخدام الخوارزمية `random_shuffle` لترتيب العناصر بشكل عشوائي في نطاق معطى. هناك صيغتان من هذه الخوارزمية كما هو موضح من النماذج التالية:

```
template<class randomAccessItr>
void random_shuffle(randomAccessItr first,
                  randomAccessItr last);

template<class randomAccessItr, class randomAccessGenerator>
void random_shuffle(randomAccessItr first,
                  randomAccessItr last,
                  randomAccessGenerator rand);
```

الصيغة الأولى تقوم باعادة ترتيب العناصر الموجودة في النطاق first...last-1 باستخدام مولد أعداد عشوائية موحد التوزيع. الصيغة الثانية تعيد ترتيب العناصر الموجودة في النطاق first...last-1 باستخدام هدف دالة مولد لأعداد عشوائية أو مؤشر الى دالة محددة بواسطة rand.

المثال 17-13 يوضح كيفية استخدام هذه الدالات:

مثال 17-13:

```
//STL Functions count, count_if, max_element,
//min_element, and random_shuffle

#include <iostream>
#include <cctype>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

int main ()
{
    char cList[10] = {'Z', 'a', 'Z', 'B', 'Z',
                     'c', 'D', 'e', 'F', 'Z'};           //Line 1

    vector<char> charList(cList, cList + 10);           //Line 2

    ostream_iterator<char> screen(cout, " ");           //Line 3

    cout<<"Line 4: charList: ";                          //Line 4
    copy(charList.begin(), charList.end(), screen);     //Line 5
    cout<<endl;                                           //Line 6

    //count
    int noOfZs = count(charList.begin(), charList.end(),
                       'Z');                             //Line 7

    cout<<"Line 8: Number of Z\'s in charList:"           //Line 8
    <<noOfZs<<endl;

    //count_if
    int noOfUpper = count_if(charList.begin(),
                             charList.end(), isupper); //Line 9

    cout<<"Line 10: Number of uppercase letters in "
    <<"charList: " <<noOfUpper<<endl;                   //Line 10
```

```

int list[10] = {12, 34, 56, 21, 34,
               78, 34, 55, 12, 25};           //Line 11

ostream_iterator<int> screenOut(cout, " ");    //Line 12

cout<<"Line 13: list: ";                      //Line 13
copy(list, list + 10, screenOut);             //Line 14
cout<<endl;                                   //Line 15

//max_element
int *maxLoc = max_element(list, list + 10);    //Line 16

cout<<"Line 17: Largest element in list: "
    <<*maxLoc<<endl;                          //Line 17

//min_element
int *minLoc = min_element(list, list + 10);    //Line 18

cout<<"Line 19: Smallest element in list: "
    <<*minLoc<<endl;                          //Line 19

//random_shuffle
random_shuffle(list, list + 10);               //Line 20

cout<<"Line 21: list after random shuffle: ";  //Line 21
copy(list, list + 10, screenOut);             //Line 22
cout<<endl;                                   //Line 23

return 0;
}

```

المخرج

الصف 4: charList: Z F e D c Z B Z a Z
 الصف 8: عدد حرف 'Z' في charList: 4
 الصف 10: عدد الحروف الكبيرة في charList: 7
 الصف 13: القائمة: 25 12 55 34 78 34 21 56 34 12
 الصف 17: أكبر عنصر في القائمة: 78
 الصف 19: أصغر عنصر في القائمة: 12
 الصف 21: القائمة بعد الخلط العشوائي: 34 34 21 55 78 12 56 25 34 12
 المخرجات السابقة واضحة والتفاصيل متروكة لك كتمرين.

الدالات for_each و transform:

يتم استخدام الخوارزمية for_each لتناول ومعالجة كل عنصر في نطاق معين عن طريق تطبيق دالة يتم تمريرها كمعامل. نموذج الدالة التي تطبق هذه الخوارزمية هو:

```

template<class inputItr, class function>
function for_each(inputItr first, inputItr last, function func);

```

الدالة المحددة بواسطة المعامل func يتم تطبيقها على كل عنصر في النطاق first...last-1. يمكن للدالة func تعديل العنصر وغالباً يتم إهمال القيمة الناتجة من الدالة for_each. الخوارزمية transform لها صيغتان. نماذج الدالات التي تقوم بتطبيق هذه الخوارزمية هي:

```
template<class inputItr, class outputItr,
        class unaryOperation>
outputItr transform(inputItr first, inputItr last,
                   outputItr destFirst,
                   unaryOperation op);

template<class inputItr1, class inputItr2,
        class outputItr, class binaryOperation>
outputItr transform(inputItr1 first1, inputItr1 last,
                   inputItr2 first2,
                   outputItr destFirst,
                   binaryOperation bOp);
```

الصيغة الأولى من الدالة transform لها أربع معاملات. تقوم هذه الدالة بعمل تتابع من العناصر عند المكان المقصود التي تبدأ ب destFirst عن طريق تطبيق العملية الأحادية op على كل عنصر في النطاق first1...last-1. تقوم هذه الدالة بإنتاج مكرر موجود بعد آخر عنصر تم نسخه عند المكان المقصود.

الصيغة الثانية من الدالة transform لها خمس معاملات. تقوم الدالة بعمل تتابع من العناصر عن طريق تطبيق العملية الثنائية bOp - أي (elemRange, elemRange2) bOp - على العناصر المقابلة في النطاق first1...last1 والنطاق الذي يبدأ ب destFirst. يتم وضع النتائج عند المكان المقصود الذي يبدأ ب destFirst. الدالة تنتج مكرر موجود بعد آخر عنصر تم نسخه عند المكان المقصود.

المثال 13- 18 يوضح كيفية استخدام هذه الدالات.

مثال 13- 18:

//STL Functions for_each and transform

```
#include <iostream>
#include <cctype>
#include <algorithm>
#include <iterator>
#include <vector>
```

```

using namespace std;

void doubleNum(int& num);

int main()
{
    char cList[5] = {'a', 'b', 'c', 'd', 'e'};           //Line 1
    vector<char> charList(cList, cList + 5);             //Line 2
    ostream_iterator<char> screen(cout, " ");           //Line 3
    cout<<"Line 4: charList: ";                          //Line 4
    copy(charList.begin(), charList.end(), screen);     //Line 5
    cout<<endl;                                           //Line 6

    transform(charList.begin(), charList.end(),         //Line 7
              charList.begin(), toupper);

    cout<<"Line 8: charList after changing all lowercase"
        <<" letters to \n      uppercase: ";           //Line 8
    copy(charList.begin(), charList.end(), screen);     //Line 9
    cout<<endl;                                           //Line 10

    int list[7] = {2, 8, 5, 1, 7, 11, 3};              //Line 11
    ostream_iterator<int> screenOut(cout, " ");         //Line 12
    cout<<"Line 13: list: ";                             //Line 13
    copy(list, list + 7, screenOut);                   //Line 14
    cout<<endl;                                           //Line 15

    cout<<"Line 16: The effect of the for_each "
        <<"function: ";                                 //Line 16
    for_each(list, list + 7, doubleNum);               //Line 17
    cout<<endl;                                           //Line 18

    cout<<"Line 19: list after a call to the for_each "
        <<"function: ";                                 //Line 19
    copy(list, list + 7, screenOut);                   //Line 20
    cout<<endl;                                           //Line 21

    return 0;
}

void doubleNum(int& num)
{
    num = 2 * num;
    cout<<num<<" ";
}

```

المخرجات:

الصف 4: charList: a b c d e

الصف 8: charList بعد تغيير جميع الحروف الصغيرة الى

حروف كبيرة: A B C D E

الصف 13: القائمة: 2 8 1 7 5 11 3

الصف 16: تأثير الدالة for_each: 4 16 10 2 14 22 6

الصف 19: القائمة بعد استدعاء الدالة `for_each`: 6 22 14 2 10 16 4

البيان في الصف 7 يستخدم الدالة `transform` لتغيير كل حرف صغير من `charList` الى نظيره الكبير. في المخرجات يحتوى السطر الذي يحدد الصف 8 على مخرجات البيانات في الصفوف من 8 وحتى 10 من البرنامج. لاحظ أن المعامل الرابع للدالة `transform` (في الصف 7) هو الدالة `toupper` من الملف الرئيسي `ccType`.

البيان في الصف 17 يستدعي الدالة `for_each` لمعالجة كل عنصر في القائمة باستخدام الدالة `doubleNum`. الدالة `doubleNum` لها معامل اشارة هو `num` من النوع `int`. فضلاً عن هذا تقوم هذه الدالة بمضاعفة قيمة `num` ثم يخرج قيمة `num`. بما أن `num` معامل اشارة ان يتم تغيير قيمة المعامل الحقيقي. في المخرجات يحتوي الخط الذي يحدد الصف 16 على المخرجات الناتجة من البيان `cout` في الدالة `doubleNum` الذي يتم تمريره كمعامل ثالث للدالة `for_each` (انظر الصف 17).

البيان في الصف 20 يخرج قيم عناصر القائمة. في المخرجات يحتوي الصف 19 على مخرجات البيانات في الصفوف من 19 الى 20 من البرنامج.

الـدالات `includes`، `set_intersection`، `set_union`، و `set_difference` و `set_symmetric_difference`:

يقوم هذا القسم بوصف عمليات نظرية المجموعة `includes` (مجموعة فرعية)، و `set_intersection`، `set_union`، و `set_difference` و `set_symmetric_difference`. تفترض هذه الخوارزميات أن العناصر داخل كل نطاق معطى تكون مخزنة بالفعل. تقوم الخوارزمية `includes` بتحديد ما اذا كانت العناصر الموجودة في نطاق واحد تظهر في نطاق آخر. هذه الدالة لها صيغتان كما هو موضح بواسطة النماذج التالية:

```
template<class inputItr1, class inputItr2>
bool includes(inputItr1 first1, inputItr1 last1,
              inputItr2 first2, inputItr2 last2);

template<class inputItr1, class inputItr2,
          class binaryPredicate>
bool includes(inputItr1 first1, inputItr1 last1,
              inputItr2 first2, inputItr2 last2,
              binaryPredicate op);
```

كل من صيغتين الدالة `includes` تفترض أن العناصر في النطاقات `first1...last1-1` و `first2...last2-1` مرتبة وفقاً لنفس معيار الترتيب. تقوم الدالة بإنتاج `true` اذا كانت جميع العناصر في النطاق `first2...last2-1` موجودة كذلك في `first1...last1-1`. بمعنى آخر تقوم الدالة بإنتاج `true` اذا احتوت `first1...last1-1` على جميع العناصر في النطاق `first2...last2-1`. الصيغة

الأولى تفترض أن العناصر في كل من النطاقين مرتبين ترتيب تصاعدي. الصيغة الثانية تستخدم العملية op لتحديد ترتيب العناصر.

المثال 19-13 يوضح كيفية عمل الدالة includes:

مثال 19-13:

```
//STL function includes
//This function assumes that the elements in the given ranges
//are ordered according to some sorting criterion.

#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    char setA[5] = {'A', 'B', 'C', 'D', 'E'};           //Line 1
    char setB[10] = {'A', 'B', 'C', 'D', 'E',           //Line 2
                    'F', 'I', 'J', 'K', 'L'};
    char setC[5] = {'A', 'E', 'I', 'O', 'U'};          //Line 3

    ostream_iterator<char> screen(cout, " ");          //Line 4

    cout<<"Line 5: setA: ";                             //Line 5
    copy(setA, setA + 5, screen);                       //Line 6
    cout<<endl;                                           //Line 7

    cout<<"Line 8: setB: ";                             //Line 8
    copy(setB, setB + 10, screen);                      //Line 9
    cout<<endl;                                           //Line 10

    cout<<"Line 11: setC: ";                             //Line 11
    copy(setC, setC + 5, screen);                       //Line 12
    cout<<endl;                                           //Line 13

    if(includes(setB, setB + 10, setA, setA + 5))       //Line 14
        cout<<"Line 15: setA is a subset of setB"       //Line 15
        <<endl;                                           //Line 16
    else
        cout<<"Line 17: setA is not a subset of setB"   //Line 17
        <<endl;

    if(includes(setB, setB + 10, setC, setC + 5))       //Line 18
        cout<<"Line 19: setC is a subset of setB"       //Line 19
        <<endl;                                           //Line 20
    else
        cout<<"Line 21: setC is not a subset of setB"   //Line 21
        <<endl;

    return 0;
}
```

المخرجات:

الصف 5: setA: E D C B A

الصف 8: setB: L K J I F E D C B A

الصف 11: setC : A O I U

الصف 15: setA مجموعة فرعية من setB

الصف 21: setC ليست مجموعة فرعية من setB

المخرجات السابقة واضحة والتفاصيل متروكة لك كتمرين.

يتم استخدام الخوارزمية set_intersection ليجاد العناصر المشتركة بين نطاقين من العناصر. هذه الخوارزمية لها صيغتان كما هو موضح بواسطة النماذج التالية:

```
template<class inputItr1, class inputItr2,
         class outputItr>
outputItr set_intersection(inputItr1 first1, inputItr1 last1,
                           inputItr2 first2, inputItr2 last2,
                           outputItr destFirst);
```

```
template<class inputItr1, class inputItr2,
         class outputItr, class binaryPredicate>
outputItr set_intersection(inputItr1 first1, inputItr1 last1,
                           inputItr2 first2, inputItr2 last2,
                           outputItr destFirst,
                           binaryPredicate op);
```

تقوم كلتا الصيغتين بعمل تتابع من العناصر المرتبة المشتركة بين نطاقين مرتبين هما first1...last1-1 و first2...last2-1. يتم وضع التتابع الذي تم عمله في الحاوية التي تبدأ ب destFirst. كلتا الصيغتان تقومان بانتاج مكرر موجود بعد آخر عنصر يتم نسخه عند النطاق المقصودتفترض الصيغة الأولى أن العناصر في كلا النطاقين مرتبة ترتيباً تصاعدياً. الصيغة الثانية تفترض أن كلا النطاقين مرتبين باستخدام العملية المحددة بواسطة op. العناصر الموجودة في نطاقات المصدر لا يتم تعديلها.

افترض أن:

```
setA = {2, 4, 5, 7, 8}
setB = {1, 2, 3, 4, 5, 6, 7}
setC = {2, 5, 8, 8, 15}
setD = {1, 4, 4, 6, 7, 12}
setE = {2, 3, 4, 4, 5, 6, 10}
```

اذن:

```
AintersectB = {2, 4, 5, 7}
AintersectC = {2, 5, 8}
DintersectE = {4, 4, 6}
```

لاحظ أنه بما أن 8 يظهر مرة واحدة فقط في setA فإن 8 تظهر مرة واحدة فقط في AintersectC بالرغم من ظهور 8 مرتين في setC. بالرغم من هذا تظهر 4 مرتين في DintersectE لأن 4 تظهر مرتين في كل من setD و setE.

يتم استخدام الخوارزمية set_union للعثور على العناصر الموجودة في نطاقين من العناصر. هذه الخوارزمية لها صيغتان كما هو موضح بواسطة النماذج التالية:

```
template<class inputItr1, class inputItr2,
        class outputItr>
outputItr set_union(inputItr1 first1, inputItr1 last1,
                   inputItr2 first2, inputItr2 last2,
                   outputItr destFirst);

template<class inputItr1, class inputItr2,
        class outputItr, class binaryPredicate>
outputItr set_union(inputItr1 first1, inputItr1 last1,
                   inputItr2 first2, inputItr2 last2,
                   outputItr result,
                   binaryPredicate op);
```

كل من الصيغتان تقوم بعمل تتابع من العناصر المرتبة التي تظهر في أي من النطاقين المرتبين وهما first1...last1-1 و first2...last2-1. يتم وضع التتابع الذي تم عمله في الحاوية التي تبدأ بـ destFirst. كلتا الصيغتان تقومان بإنتاج مكرر موجود بعد آخر عنصر يتم نسخه عند النطاق المقصودتفترض الصيغة الأولى أن العناصر في كلا النطاقين مرتبة ترتيبياً تصاعدياً. الصيغة الثانية تفترض أن كلا النطاقين مرتبين باستخدام العملية المحددة بواسطة op. العناصر الموجودة في نطاقات المصدر لا يتم تعديلها.

افترض أن لديك setA، و setB، و setC، و setD، و setE كما تم تعريفها من قبل. اذن:

```
AunionB = {1, 2, 3, 4, 5, 6, 7, 8}
AunionC = {2, 4, 5, 7, 8, 8, 15}
BunionD = {1, 2, 3, 4, 4, 5, 6, 7, 12}
DunionE = {1, 2, 3, 4, 4, 5, 6, 7, 10, 12}
```

لاحظ أن بما أن 8 تظهر مرتين في setC فإنها تظهر مرتين في AunionC. بما أن 4 تظهر مرتين في setD وفي setE اذن فإنها تظهر مرتين في DunionE.

المثال 13-20 يوضح كيفية عمل الدالتين set_union و set_intersection.

مثال 13-20:

// هذه الدالات تفترض أن العناصر في النطاقات المعطاة

//مرتبة وفقاً لنفس معيار الترتيب.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int setA[5] = {2, 4, 5, 7, 8}; //Line 1
    int setB[7] = {1, 2, 3, 4, 5, 6, 7}; //Line 2
    int setC[5] = {2, 5, 8, 8, 15}; //Line 3
    int setD[6] = {1, 4, 4, 6, 7, 12}; //Line 4

    int AunionB[10]; //Line 5
    int AunionC[10]; //Line 6
    int BunionD[15]; //Line 7
    int AintersectB[10]; //Line 8
    int AintersectC[10]; //Line 9

    int *lastElem; //Line 10

    ostream_iterator<int> screen(cout, " "); //Line 11

    cout<<"Line 12: setA = "; //Line 12
    copy(setA, setA + 5, screen); //Line 13
    cout<<endl; //Line 14

    cout<<"Line 15: setB = "; //Line 15
    copy(setB, setB + 7, screen); //Line 16
    cout<<endl; //Line 17

    cout<<"Line 18: setC = "; //Line 18
    copy(setC, setC + 5, screen); //Line 19
    cout<<endl; //Line 20

    cout<<"Line 21: setD = "; //Line 21
    copy(setD, setD + 6, screen); //Line 22
    cout<<endl; //Line 23

    lastElem = set_union(setA, setA + 5,
                        setB, setB + 7,
                        AunionB); //Line 24

    cout<<"Line 25: Set AunionB: "; //Line 25
    copy(AunionB, lastElem, screen); //Line 26
    cout<<endl; //Line 27

    lastElem = set_union(setA, setA + 5,
                        setC, setC + 5,
                        AunionC); //Line 28
```

```

cout<<"Line 29: Set AunionC: ";           //Line 29
copy(AunionC, lastElem, screen);         //Line 30
cout<<endl;                             //Line 31

lastElem = set_union(setB, setB + 7,
                    setD, setD + 6,
                    BunionD);           //Line 32

cout<<"Line 33: Set BunionD: ";           //Line 33
copy(BunionD, lastElem, screen);         //Line 34
cout<<endl;                             //Line 35

lastElem = set_intersection(setA, setA + 5,
                          setB, setB + 7,
                          AintersectB); //Line 36

cout<<"Line 37: Set AintersectB: ";       //Line 37
copy(AintersectB, lastElem, screen);     //Line 38
cout<<endl;                             //Line 39

lastElem = set_intersection(setA, setA + 5,
                          setC, setC + 5,
                          AintersectC); //Line 40

cout<<"Line 41: Set AintersectC: ";       //Line 41
copy(AintersectC, lastElem, screen);     //Line 42
cout<<endl;                             //Line 43

return 0;
}

```

المخرجات:

الصف 12:

الصف 15: setB = 1 2 3 4 5 6 7

الصف 18: setC = 2 5 8 8 15

الصف 21: setD = 1 4 4 6 7 12

الصف 25: تحديد AunionB: 1 2 3 4 5 6 7 8

الصف 29: تحديد AunionC: 2 4 5 7 8 8 15

الصف 33: تحديد BunionD: 1 2 3 4 4 6 7 12

الصف 37: تحديد AintersectB: 2 4 5 7

الصف 41: تحديد AintersectC: 2 5 8

المخرجات السابقة واضحة والتفاصيل متروكة لك كتمرين.

يتم استخدام الخوارزمية `set_difference` لإيجاد العناصر في نطاق واحد من العناصر التي لا تظهر في نطاق آخر من العناصر. هذه الخوارزمية لها صيغتان كما هو موضح بواسطة النماذج التالية:

كلتا الصيغتان تقومان بعمل تتابع من عناصر مرتبة موجودة في النطاق المرتب 1-last1...first1 ولكن ليست موجودة في النطاق المرتب 1-last2...first2 أو عناصر موجودة في النطاق المرتب 1-last2...first2 ولكن غير موجودة في 1-last1...first1. بمعنى آخر يحتوي تتابع العناصر التي تم عملها عن طريق set_symmetric_difference على العناصر الموجودة في range1_difference_range1 اتحاد range2_difference_range1. يتم وضع التتابع الذي تم عمله في الحاوية التي تبدأ بـ destFirst. كلتا الصيغتان تقومان بإنتاج مكرر موجود بعد آخر عنصر يتم نسخه عند النطاق المقصود. تفترض الصيغة الأولى أن العناصر في كلا النطاقين مرتبة ترتيباً تصاعدياً. الصيغة الثانية تفترض أن كلا النطاقين مرتبين باستخدام العملية المحددة بواسطة op. العناصر الموجودة في نطاقات المصدر لا يتم تعديلها. يمكن توضيح أن التتابع الذي تم عمله بواسطة set_symmetric_difference يحتوي على العناصر الموجودة في range1_union_range2 وليس في range1_intersection_range2.

افترض أن:

```
setB = {3, 4, 5, 6, 7, 8, 10}
setC = {1, 5, 6, 8, 15}
setD = {2, 5, 5, 6, 9}
```

لاحظ أن Bdifference = {3, 4, 7, 10} وأن Cdifference = {1, 15}. لهذا تكون:

```
BsymDiffC = {1, 3, 4, 7, 10, 15}
```

الآن CdifferenceD = {1, 8, 15} DdifferenceC = {2, 5, 9}

```
DsymDiffC = {1, 2, 5, 8, 9, 15}
```

المثال 13-21 يوضح كيفية عمل الدالات set_difference و set_symmetric_difference:

مثال 13-21:

افترض أن لدينا البيانات التالية:

```
int setA[5] = {2, 4, 5, 7, 8};           //Line 1
int setB[7] = {3, 4, 5, 6, 7, 8, 10};    //Line 2
int setC[5] = {1, 5, 6, 8, 15};          //Line 3

int AdifferenceC[5];                     //Line 4
int BsymDiffC[10];                       //Line 5
```

انظر الى البيان التالي:

```
set_difference(setA, setA + 5, setC, setC + 5,
               AdifferenceC);           //Line 6
```

بعد تنفيذ هذا البيان تحتوي Adifference على العناصر الموجودة في setA وغير موجودة في setC وهي:

```
AdifferenceC = {2, 4, 7} //Line 7
```

انظر الآن الى البيان التالي:

```
set_symmetric_difference(setB, setB + 7, setC, setC + 5,
                          BsymDiffC); //Line 8
```

بعد تنفيذ هذا البيان تحتوي BsymDiffC على العناصر الموجودة في setB وغير موجودة في setC أو العناصر الموجودة في setC وغير موجودة في setB وهي:

```
BsymDiffC = {1, 3, 4, 7, 10, 15} //Line 9
```

نترك لك كتمرين كتابة برنامج يوضح كيفية استخدام الدالات set_difference و set_symmetric_difference (انظر تمرين البرمجة 5 في نهاية هذا الفصل).

الدالات accumulate، adjacent_difference، و inner_product، و partial_sum:

الخوارزميات accumulate، و adjacent_difference، و inner_product، و partial_sum عبارة عن دالات رقمية وبهذا تتحكم في البيانات الرقمية. كل من هذه الدالات لها صيغتان. الصيغة الأولى تستخدم العملية العادية للتحكم في البيانات. على سبيل المثال تقوم الخوارزمية accumulate بإيجاد مجموع جميع العناصر في نطاق معطى. في الصيغة التالية يمكننا تحديد العملية التي يتم تطبيقها على عناصر النطاق. على سبيل المثال يمكننا بدلاً من اضافة عناصر نطاق معطى تحديد عمليات الضرب للخوارزمية accumulate لضرب عناصر النطاق. بعد هذا نقوم كالمعتاد باعطاء نماذج كل من هذه الخوارزميات يليها شرح موجز. الخوارزميات موجودة في الملف الرئيسي .numeric

```
template<class inputItr, class Type>
Type accumulate(inputItr first, inputItr last, Type init);

template<class inputItr, class Type, class binaryOperation>
Type accumulate(inputItr first, inputItr last,
                Type init, binaryOperation op);
```

الصيغة الأولى من الخوارزمية accumulate تقوم باضافة جميع العناصر الى قيمة أولية محددة بواسطة المعامل init في النطاق first...last-1. على سبيل المثال اذا كانت قيمة init تساوي صفراً فإن الخوارزمية تنتج مجموع جميع العناصر. في الصيغة التالية يمكننا تحديد عملية ثنائية مثل

الضرب لكي يتم تطبيقها على عناصر النطاق. على سبيل المثال اذا كانت قيمة init تساوي 1 وكانت العملية الثنائية هي الضرب فان الخوارزمية تنتج ناتج العناصر في النطاق. بعد هذا نصف الخوارزمية adjacent_difference ونماذجها هي:

```
template<class inputItr, class outputItr>
outputItr adjacent_difference(inputItr first, inputItr last,
                              outputItr destFirst);
```

```
template<class inputItr, class outputItr,
         class binaryOperation>
outputItr adjacent_difference(inputItr first, inputItr last,
                              outputItr destFirst,
                              binaryOperation op);
```

الصيغة الأولى تصنع تتابع من العناصر يكون فيه العنصر الأول هو نفسه العنصر الأول في النطاق first...last-1 وجميع العناصر الأخرى هي الفروق بين العناصر الحالية والسابقة. على سبيل المثال اذا كان نطاق العناصر هو:

{2, 5, 6, 8, 3, 7}

اذن فان التتابع الذي يتم عمله بواسطة الدالة adjacent_difference هو:

{2, 3, 1, 2, -5, 4}

العنصر الأول هو نفس العنصر الأول في النطاق الأصلي. العنصر الثاني متساوي مع العنصر الثاني في النطاق الأصلي ناقص العنصر الأول في النطاق الأصلي. بالمثل يكون العنصر الثالث متساوي مع العنصر الثالث في النطاق الأصلي ناقص العنصر الثاني في النطاق الأصلي وهكذا.

في الصيغة الثانية من adjacent_difference يتم تطبيق العملية الثنائية op على العناصر الموجودة في النطاق. يتم نسخ التتابع الناتج في المكان المحدد بواسطة destFirst. على سبيل المثال اذا كان التتابع هو (2, 5, 6, 8, 3, 7) والعملية كانت عملية ضرب فان التتابع الناتج يكون (2, 10, 30, 48, 24, 12).

كلتا الصيغتان تنتج مكرر موجود بعد آخر عنصر تم نسخه عند المكان المقصود. المثال 13-22 يوضح كيفية استخدام الدالات accumulate و adjacent_difference.

```
//Numeric algorithms accumulate and adjacent_difference

#include <iostream>
#include <algorithm>
#include <numeric>
#include <iterator>
#include <vector>
#include <functional>

using namespace std;

void print(vector<int> vList);

int main()
{
    int list[8] = {1, 2, 3, 4, 5, 6, 7, 8};           //Line 1

    vector<int> vecList(list, list + 8);             //Line 2
    vector<int> newVList(8);                          //Line 3

    cout<<"Line 4: vecList: ";                       //Line 4
```

```

print(vecList); //Line 5

//accumulate
int sum = accumulate(vecList.begin(),
                    vecList.end(), 0); //Line 6

cout<<"Line 7: Sum of the elements of vecList = "
    <<sum<<endl; //Line 7

int product = accumulate(vecList.begin(),
                        vecList.end(),
                        1, multiplies<int>()); //Line 8

cout<<"Line 9: Product of the elements of "
    <<"vecList = "<<product<<endl; //Line 9

//adjacent_difference
adjacent_difference(vecList.begin(),
                  vecList.end(),
                  newVList.begin()); //Line 10

cout<<"Line 11: newVList: "; //Line 11
print(newVList); //Line 12

adjacent_difference(vecList.begin(), vecList.end(),
                  newVList.begin(),
                  multiplies<int>()); //Line 13

cout<<"Line 14: newVList: "; //Line 14
print(newVList); //Line 15

return 0;
}

void print(vector<int> vList)
{
    ostream_iterator<int> screenOut(cout, " "); //Line 16

    copy(vList.begin(), vList.end(), screenOut); //Line 17
    cout<<endl; //Line 18
}

```

المخرجات:

الصف 4: vecList: 1 2 3 4 5 6 7 8

الصف 7: مجموع عناصر vecList = 36

الصف 9: ناتج ضرب عناصر vecList = 40320

الصف 11: newList: 1 1 1 1 1 1 1 1

الصف 14: newList: 1 2 6 12 20 30 42 56

المخرجات السابقة واضحة والتفاصيل يتم تركها لك كتمرين.

يتم استخدام الخوارزمية inner_product للتحكم في عناصر النطاقين. نماذج هذه الخوارزمية تكون كما يلي:

```
template<class inputItr1, class inputItr2, class Type>
Type inner_product(inputItr1 first1, inputItr1 last,
                  inputItr2 first2, Type init);

template<class inputItr1, class inputItr2, class Type
        class binaryOperation1, class binaryOperation2>
Type inner_product(inputItr1 first1, inputItr1 last,
                  inputItr2 first2, Type init,
                  binaryOperation1 op1, binaryOperation2 op2);
```

الصيغة الأولى تقوم بضرب العناصر المقابلة في النطاق first1...last-1 ونطاق العناصر الذي يبدأ من first2 ثم يتم جمع نواتج ضرب هذه العناصر الى القيمة المحددة بواسطة المعامل init. لكي نكون محددين افترض أن elem1 يتراوح فوق النطاق الأول وأن elem2 يتراوح فوق النطاق الثاني الذي يبدأ بـ first2.

الصيغة الأولى تحسب:

$$\text{elem2} + \text{elem1} + \text{init} = \text{init}$$

بالنسبة الى جميع العناصر المقابلة. على سبيل المثال افترض أن النطاقين هما (2، 4، 7، 8) و (1، 4، 6، 9) وأن init تساوي صفر. تقوم الدالة بحساب و انتاج:

$$132 = 9 * 8 + 6 * 7 + 4 * 4 + 1 * 2 + 0$$

في الصيغة التالية يمكن استبدال الجمع الافتراضي بالعملية المحددة بواسطة op1 ويمكن استبدال الضرب الافتراضي بالعملية المحددة بواسطة op2. في الحقيقة تقوم هذه الصيغة بحساب:

$$\text{init op1 (elem1 op2 elem2)} = \text{init}$$

الخوارزمية partial_sum لها صيغتان كما هو موضح بواسطة النماذج التالية:

```
template<class inputItr, class outputItr>
outputItr partial_sum(inputItr first, inputItr last,
                    outputItr destFirst);

template<class inputItr, class randomAccessItr,
        class binaryOperation>
outputItr partial_sum(inputItr first, inputItr last,
                    outputItr destFirst, binaryOperation op);
```

الصيغة الأولى تصنع تتابعاً من العناصر يكون فيه كل عنصر عبارة عن مجموع جميع العناصر السابقة في النطاق first...last-1 حتى موضع العنصر. على سبيل المثال يكون العنصر الأول من التتابع الجديد هو نفسه العنصر الأول في النطاق first...last-1 ويكون العنصر الثاني هو مجموع أول عنصرين في النطاق first...last-1 ويكون العنصر الثالث من التتابع الجديد هو مجموع أول ثلاثة عناصر في النطاق first...last-1 وهكذا. على سبيل المثال بالنسبة الى تتابع العناصر التالي:

$$(1, 3, 4, 6)$$

تقوم الدالة partial_sum بتوليد المتتابع التالي:

(1, 4, 8, 14)

في الصيغة التالية يمكن استبدال الجمع الافتراضي بالعملية المحددة بواسطة op.

على سبيل المثال اذا كان المتتابع هو:

(1, 3, 4, 6)

وكانت العملية عملية ضرب فان الدالة partial_sum تقوم بتوليد المتتابع التالي:

(1, 3, 12, 72)

يتم نسخ المتتابع الذي تم عمله عند المكان المحدد بواسطة destFirst ويقوم بانتاج مكرر موجود بعد آخر عنصر تم نسخه عند المكان المقصود.

المثال 13-23 يوضح كيفية عمل الدالتين inner_product و partial_sum.

مثال 13-23:

افترض أن لديك البيانات التالية:

```
int list1[8] = {1, 2, 3, 4, 5, 6, 7, 8};           //Line 1
int list2[8] = {2, 4, 5, 7, -9, 11, 12, 14};       //Line 2

vector<int> vecList(list1, list1 + 8);              //Line 3
vector<int> newVList(list2, list2 + 8);             //Line 4

int sum;                                           //Line 5
```

بعد تنفيذ البيانات في الصفوف 3 و 4:

```
vecList = {1, 2, 3, 4, 5, 6, 7, 8}                //Line 6
newVList = {2, 4, 5, 7, -9, 11, 12, 14}            //Line 7
```

انظر الآن الى البيان التالي:

```
sum = inner_product(vecList.begin(), vecList.end(),
                    newVList.begin(), 0);          //Line 8
```

يقوم هذا البيان بحساب حاصل الضرب الداخلي لكل من vecList و newVList ويتم تخزين النتيجة في sum وهي:

```
sum = 0 + 1 * 2 + 2 * 4 + 3 * 5 + 4 * 7 + 5 * (-9)
      + 6 * 11 + 7 * 12 + 8 * 14
      = 270
```

انظر الآن الى البيان التالي:

```
sum = inner_product(vecList.begin(), vecList.end(),
                    newVList.begin(), 0,
                    plus<int>(), minus<int>());    //Line 9
```

يقوم هذا البيان بحساب حاصل الضرب الداخلي لكل من `vecList` و `newVList` ويتم استبدال الضرب * بالطرح - ويتم تخزين النتيجة في `sum` وهي:

```
sum = 0 + (1 - 2) + (2 - 4) + (3 - 5) + (4 - 7) + (5 - (-9))
      + (6 - 11) + (7 - 12) + (8 - 14)
      = -10
```

انظر الآن الى البيان التالي:

```
partial_sum(vecList.begin(), vecList.end(),
            newVList.begin()); //Line 10
```

هذا البيان يستخدم الدالة `partial_sum` لتوليد تتابع العناصر 1، 3، 6، 10، 15، 21، 28، 36. يتم تحديد هذه العناصر من أجل `newVList` وهي:

```
newVList = {1, 3, 6, 10, 15, 21, 28, 36}
```

انظر الآن الى البيان التالي:

```
partial_sum(vecList.begin(), vecList.end(),
            newVList.begin(), multiplies<int>()); //Line 11
```

هذا البيان يستخدم الدالة `partial_sum` لتوليد تتابع العناصر 1، 2، 6، 24، 120، 720، 5040، 40320. لاحظ أن البيان في الصف 11 يحسب الضرب الجزئي لعناصر `vecList` عن طريق استبدال علامة الزائد + بعلامة الضرب *. يتم تحديد هذه العناصر للقائمة `newVList` وهي:

```
newVList = {1, 2, 6, 24, 120, 720, 5040, 40320}
```

نترك لك كتمرين كتابة برنامج يوضح كيفية استخدام الدالات `inner_product` و `partial_sum` (انظر تمرين البرمجة 6 في نهاية هذا الفصل).

مراجعة سريعة

1. مكتبة القالب المعياري تقدم قوالب فئة تعالج القوائم، والمرصوصات، والطوابير.
2. المكونات الثلاث الرئيسية لمكتبة القالب المعياري هي: الحاويات، والمكررات، والخوارزميات.
3. يتم استخدام الخوارزميات للتحكم في العناصر الموجودة في حاوية ما.
4. الفئات الرئيسية من الحاويات هي الحاويات التتابعية، والحاويات المترابطة، ومكيفات الحاويات.
5. الفئة `pair` تسمح لك بدمج قيمتين في وحدة واحدة. يمكن للدالة أن تنتج قيمتين عن طريق استخدام الفئة `pair`. الفئة `map` والفئة `multimap` تستخدمان الفئة `pair` لإدارة عناصرهما.

6. تعريف الفئة pair موجود في الملف الرئيسي utility.
7. الدالة make_pair تسمح لك بعمل أزواج دون تحديد النوع pair بوضوح.
8. تعريف الفئة make_pair موجود في الملف الرئيسي utility.
9. العناصر الموجودة في حاوية تتابعية يتم ترتيبها تلقائياً وفقاً لمعيار ترتيب. معيار الترتيب الافتراضي هو المعامل الارتباطي > (أصغر من).
10. الحاويات التتابعية المسبقة التعريف في مكتبة القالب المعياري هي sets، و multisets، و maps، و multimap.
11. الحاويات من النوع set لا تسمح بوجود مكررات.
12. الحاويات من النوع multiset تسمح بوجود مكررات.
13. اسم الفئة التي تقوم بتعريف الحاوية set هو set.
14. اسم الفئة التي تقوم بتعريف الحاوية multiset هو multiset.
15. اسم الملف الرئيسي المحتوي على تعريفات الفئات set و multiset وتعريفات الدالات لتطبيق عمليات متنوعة على هذه الحاويات هو set.
16. يمكن استخدام العمليات insert، و erase، و clear لادخال أو حذف عناصر من المجموعات.
17. الحاويات map و multimap تدير عناصرها في صيغة مفتاح / قيمة. يتم ترتيب العناصر تلقائياً وفقاً لمعيار ترتيب يتم تطبيقه على المفتاح.
18. معيار الترتيب الافتراضي لمفتاح الحاويات map و multimap هو المعامل الارتباطي > (أصغر من). يمكن للمستخدم كذلك تحديد معيار ترتيب آخر. بالنسبة الى أنواع البيانات المعرفة بواسطة المستخدم مثل الفئات يجب أن يتم اثقال المعاملات الارتباطية بشكل صحيح.
19. الفرق الوحيد بين الحاويات map و multimap هو أن الحاوية multimap تسمح بوجود مكررات بينما لا تسمح الحاوية map بذلك.
20. اسم الفئة التي تقوم بتعريف الحاوية map هو map.
21. اسم الفئة التي تقوم بتعريف الحاوية multimap هو multimap.
22. اسم الملف الرئيسي المحتوي على تعريفات الفئات map و multimap وتعريفات الدالات لتطبيق عمليات متنوعة على هذه الحاويات هو map.
23. غالبية الخوارزميات العامة موجودة في الملف الرئيسي algorithm.
24. الفئات الرئيسية من خوارزميات مكتبة القالب المعياري هي الخوارزميات التعديلية، والغير تعديلية، والرقمية، والمكدسة.
25. الخوارزميات الغير تعديلية لا تقوم بتعديل عناصر الحاوية.
26. الخوارزميات التعديلية تقوم بتعديل عناصر الحاوية عن طريق اعادة ترتيب، وحذف، و/أو تغيير قيم العناصر.

27. الخوارزميات التعديلية التي تغير ترتيب العناصر وليس قيمها يطلق عليها كذلك خوارزميات تغير.

28. يتم تصميم الخوارزميات الرقمية للقيام بحسابات رقمية على عناصر الحاوية.

29. هدف الدالة عبارة عن قالب فئة يقوم بإثقال معامل استدعاء الدالة () operator.

30. أهداف الدالة الحسابية المسبقة التعريف هي plus، minus، multiplies، و divides، و modulus، و negate.

31. أهداف الدالة المترابطة المسبقة التعريف هي equal_to، و not_equal_to، و greater، و greater_equal، و less، و less_equal.

32. أهداف الدالة المنطقية المسبقة التعريف هي logical_not، و logical_and، و logical_or.

33. الأصول عبارة عن أنواع خاصة من أهداف الدالة التي تنتج قيم منطقية.

34. الأصول الأحادية تتحقق من خاصية محددة لمعطى واحد والأصول الثنائية تتحقق من خاصية واحدة لزوج من المعطيات – أي اثنين.

35. يتم عادةً استخدام الأصول لتحديد معيار بحث أو ترتيب.

36. في مكتبة القالب المعياري يجب أن يقوم الأصل دائماً بإنتاج نفس النتيجة لنفس القيمة.

37. الدالات التي تقوم بتعديل حالتها الداخلية لا يمكن اعتبارها أصول.

38. تقدم مكتبة القالب المعياري ثلاث مكررات – back_inserter، و front_inserter، و inserter – تسمى مكررات ادخال من أجل ادخال العناصر في المكان المقصود.

39. يقوم back_inserter باستخدام عملية push_back للحاوية بدلاً من معامل الاسناد.

40. يقوم front_inserter باستخدام عملية push_front للحاوية بدلاً من معامل الاسناد.

41. بما أن الفئة vector لا تدعم العملية push_front اذن لا يمكن استخدام هذا المكرر من أجل الحاويات vector.

42. المكرر inserter يستخدم عملية الادخال الخاصة بالحاوية بدلاً من معامل الاسناد.

43. يتم استخدام الدالة fill لملاً حاوية بعناصر ويتم استخدام الدالة fill_n لملاً العناصر n التالية.

44. يتم استخدام الدالتين generate و generate_n لتوليد عناصر وملاً التابع.

45. يتم استخدام الدالات find و find_if و find_end و find_first_of لإيجاد العناصر في نطاق معطى.

46. يتم استخدام الدالة remove لحذف عناصر معينة من تتابع ما.

47. يتم استخدام الدالة remove_if لحذف عناصر معينة من تتابع ما باستخدام بعض المعايير.

48. تقوم الدالة remove_copy بنسخ العناصر في تتابع ما داخل تتابع آخر عن طريق استبعاد عناصر معينة من التابع الأول.

49. تقوم الدالة `remove_copy_if` بنسخ العناصر في تتابع ما داخل تتابع آخر عن طريق استبعاد عناصر معينة باستخدام معيار معين من التتابع الأول.
50. يتم استخدام الدالات `swap`، `iter_swap`، و `swap_ranges` لتبادل العناصر.
51. يتم استخدام الدالات `search`، و `search_n`، و `sort`، و `binary_search` للبحث عن العناصر.
52. يتم استخدام الدالة `adjacent_find` لإيجاد أول ظهور من العناصر المتتالية التي تلي معيار معين.
53. الخوارزمية `merge` تقوم بدمج قائمتين مرتبتين.
54. يتم استخدام الخوارزمية `inplace_merge` لدمج تتابعين مرتبتين ومتتاليتين.
55. الخوارزمية `reverse` تقوم بعكس ترتيب العناصر في نطاق معطى.
56. الخوارزمية `reverse_copy` تقوم بعكس ترتيب العناصر في نطاق معين أثناء النسخ داخل النطاق المقصود. لا يتم تعديل المصدر.
57. الخوارزمية `rotate` تقوم بدوران العناصر في نطاق معطى.
58. الخوارزمية `rotate_copy` تقوم بنسخ عناصر المصدر في المكان المقصود بترتيب دائري.
59. الخوارزمية `count` تقوم باحصاء حالات ظهور قيمة محددة في نطاق معطى.
60. الخوارزمية `countt_if` تقوم باحصاء حالات ظهور قيمة محددة في نطاق معطى يلبي معيار محدد.
61. يتم استخدام الخوارزمية `max` لتحديد الحد الأقصى من بين قيمتين.
62. يتم استخدام الخوارزمية `max_element` لتحديد العنصر الأكبر في نطاق معطى.
63. يتم استخدام الخوارزمية `min` لتحديد الحد الأدنى من بين قيمتين.
64. يتم استخدام الخوارزمية `min_element` لتحديد العنصر الأصغر في نطاق معطى.
65. يتم استخدام الخوارزمية `random_shuffle` لترتيب العناصر في نطاق معطى بشكل عشوائي.
66. يتم استخدام الخوارزمية `for_each` لتناول ومعالجة كل عنصر في نطاق معطى عن طريق تطبيق دالة يتم تمريرها كمعامل.
67. الدالة `transform` تصنع تتابعاً من العناصر عن طريق تطبيق عمليات محددة على كل عنصر في نطاق معطى.
68. الخوارزمية `includes` تحدد ما إذا كانت العناصر في نطاق واحد تظهر في نطاق آخر.
69. يتم استخدام الخوارزمية `set_intersection` لإيجاد العناصر المشتركة بين نطاقين من العناصر.
70. يتم استخدام الخوارزمية `set_union` لإيجاد العناصر الموجودة في نطاقين من العناصر.

71. يتم استخدام الخوارزمية `set_difference` لإيجاد العناصر في نطاق واحد من العناصر التي لا تظهر في نطاق آخر من العناصر.

72. الخوارزمية `set_symmetric_difference` باعطائها نطاقين من العناصر تقوم بتحديد العناصر الموجودة في النطاق الأول والغير موجودة في النطاق الثاني أو العناصر الموجودة في النطاق الثاني والغير موجودة في النطاق الأول.

73. الخوارزميات `accumulate`، `adjacent_difference`، و `inner_product`، و `partial_sum` عبارة عن دالات رقمية تتحكم في البيانات الرقمية.

تمارين

1. ما هو الفرق بين حاوية مكتبة القالب المعياري وخوارزمية مكتبة القالب المعياري؟

2. افترض أن لديك البيان التالي:

```
pair<int, string> temp;
```

أ- اكتب بيان C++ يقوم بتخزين الزوج (1, Hello) داخل `temp`.

ب- اكتب بيان C++ يقوم باخراج الزوج الذي تم تخزينه في `temp` على جهاز اخراج قياسي.

3. افترض أن لديك البيان التالي:

```
pair<string, string> name;
```

ما هي المخرجات اذا وجدت من البيانات التالية؟

```
name = make_pair("Duckey", "Donald");  
cout<<name.first<<" "<<name.second<<endl;
```

4. وضح كيف تختلف الحاوية `set` عن الحاوية `map`.

5. أ. قم باعلان حاوية `Map` اسمها `stateDataMap` لتخزين أزواج بالصيغة (اسم الولاية، اسم

العاصمة) حيث يكون اسم الولاية واسم العاصمة متغيران من النوع `string`.

ب. اكتب بيانات C++ تضيف الأزواج التالية الى `stateDataMap`:

(نبراسكا، لنكوان)، (نيويورك، ألباني)، (أوهيو، كولمبوس)، (كاليفورنيا، ساكرامنتو)،

(ماساشيوسيتس، بوسطن)، و(تكساس، أوستين).

ج. اكتب كود C++ يقوم باخراج البيانات المخزنة في `stateDataMap`.

د. اكتب كود C++ يقوم بتغيير عاصمة كاليفورنيا الى لوس أنجلوس.

6. ما هو الفرق بين الحاوية `set` و `multiset`؟

7. ما هو هدف دالة مكتبة القالب المعياري؟

8. افترض أن charList حاوية vector وأن:

```
charList = {a, A, B, b, c, d, A, e, f, K}
```

افترض كذلك أن:

```
lastElem = remove_if(charList.begin(), charList.end(), islower);  
ostream_iterator<char> screen(cout, " ");
```

حيث lastElem مكرر vector داخل حاوية vector من النوع char. ما هي مخرجات البيان التالي؟

```
copy(charList.begin(), lastElem, screen);
```

9. افترض أن intList حاوية vector وأن:

```
intList = {18, 24, 24, 5, 11, 56, 27, 24, 2, 24}
```

افترض كذلك أن:

```
vector<int>::iterator lastElem;  
ostream_iterator<int> screen(cout, " ");  
vector<int> otherList(10);
```

```
lastElem = remove_copy(intList.begin(), intList.end(),  
                        otherList.begin(), 24);
```

ما هي مخرجات البيان التالي؟

```
copy(otherList.begin(), lastElem, screenOut);
```

10. افترض أن intList حاوية vector وأن:

```
intList = {2, 4, 6, 8, 10, 12, 14, 16}
```

ما هي قيمة النتيجة بعد تنفيذ البيان التالي؟

```
result = accumulate(intList.begin(), intList.end(), 0);
```

11. افترض أن intList حاوية vector وأن:

```
intList = {2, 4, 6, 8, 10, 12, 14, 16}
```

ما هي قيمة النتيجة بعد تنفيذ البيان التالي؟

```
result = accumulate(intList.begin(), intList.end(),  
                    0, multiplies<int>());
```

12. افترض أن setA و setB و setC و setD معرفة كما يلي:

```
int setA[] = {3, 4, 5, 8, 9, 12, 14};  
int setB[] = {2, 3, 4, 5, 6, 7, 8};  
int setC[] = {2, 5, 5, 9};  
int setD[] = {4, 4, 4, 6, 7, 12};
```

افترض كذلك ان:

```
int AunionB[10];
int AunionC[9];
int BunionD[10];
int AintersectB[4];
int AintersectC[2];
```

ما هي العناصر المخزنة في A اتحاد B، و A اتحاد C، و B اتحاد D، و A تقاطع B، و A تقاطع C بعد تنفيذ البيانات التالية؟

```
set_union(setA, setA + 7, setB, setB + 7, AunionB);
set_union(setA, setA + 7, setC, setC + 4, AunionC);
set_union(setB, setB + 7, setD, setD + 6, BunionD);
set_intersection(setA, setA + 7, setB, setB + 7, AintersectB);
set_intersection(setA, setA + 7, setC, setC + 4, AintersectC);
```

تمارين برمجة

1. اكتب برنامج يوضح كيفية استخدام الدالات find و find_if.
2. اكتب برنامج يوضح كيفية استخدام الدالات find_end و find_first_of.
3. اكتب برنامج يوضح كيفية استخدام الدالات replace، replace_if و replace_copy، و replace_copy_if يجب أن يقوم برنامجك باستخدام الدالة lessThanEqualTo50 كما هو موضحة في المثال 13-13.
4. اكتب برنامج يوضح كيفية استخدام الدالات adjacent_find و merge، و inplace_merge.
5. اكتب برنامج يوضح كيفية استخدام الدالات set_difference و set_symmetric_difference.
6. اكتب برنامج يوضح كيفية استخدام الدالات inner_product و partial_sum.
7. (سوق البورصة مرة أخرى) في تمرين البرمجة 8 من الفصل رقم 4 تم سؤالك أن تقوم بتصميم برنامج يقوم بتحليل أداء البورصات المدارة بواسطة شركة محلية تتاجر في البورصة وفي نهاية كل يوم ينتج البرنامج قائمة بهؤلاء البورصات مرتبة حسب رمز البورصة. سوف يحب مستثمر الشركة الآن أن يروا قائمة أخرى من البورصات المرتبة بواسطة النسبة المكتسبة من كل بورصة. الشركة تطلب منك كذلك انتاج القائمة مرتبة حسب نسبة المكسب / الخسارة. لهذا تحتاج الى ترتيب قائمة البورصة بواسطة هذا المكون. بالرغم من هذت فانك لا تقوم بترتيب القائمة مادياً بهذا المكون وهو نسبة المكسب / الخسارة وبدلاً من هذا عليك توفير ترتيب منطقي على أساس هذا المكون.

للقيام بهذا أضف عنصر بيانات لضم مؤشرات قائمة البورصة المرتبة حسب مكون نسبة المكسب / الخسارة. أطلق على هذه الحاوية indexByGain. عند طباعة القائمة مرتبة بمكون نسبة المكسب / الخسارة قم باستخدام الحاوية indexByGain لطباعة القائمة. عناصر المصفوفة indexByGain سوف تدل على مكون البورصة الذي سوف يتم طبعه بعد هذا.

8. أعد عمل تمرين البرمجة 14 من الفصل رقم 5 حتى يستخدم فئة مكتبة القالب المعياري set لمعالجة قائمة الشرائط.

9. أعد عمل تمرين البرمجة 15 من الفصل رقم 5 حتى يستخدم فئة مكتبة القالب المعياري set لمعالجة قائمة الشرائط المؤجرة بواسطة العميل وقائمة أعضاء المتجر.

10. أعد عمل تمرين البرمجة 15 من الفصل رقم 5 حتى يستخدم فئة مكتبة القالب المعياري set لمعالجة قائمة الشرائط التي يمتلكها المتجر وقائمة الشرائك المؤجرة بواسطة العملاء وقائمة أعضاء المتجر.

ملحق
(أ)
كلمات محفوظة

| | | | |
|--------------|-------------|------------|------------------|
| and | and_eq | asm | auto |
| bitand | bitor | bool | break |
| case | catch | char | class |
| compl | const | const_cast | continue |
| default | delete | do | double |
| dynamic_case | else | enum | explicit |
| export | extern | false | float |
| for | friend | goto | if |
| inline | int | long | mutable |
| namespace | new | not | not_eq |
| operator | or | or_eq | private |
| protected | public | register | reinterpret_cast |
| return | short | signed | sizeof |
| static | static_cast | struct | switch |
| template | this | throw | true |
| try | typedef | typeid | typename |
| union | unsigned | using | virtual |
| void | volatile | wchar_t | while |
| xor | xor_eq | | |

الملحق
(ب)
أسبقية العامل

الأسبقية (من الأعلى الى الأدنى)

| العامل | الارتباط |
|-------------------------------------|----------------------|
| :: (حل ثنائي المجال) | من اليسار الى اليمين |
| :: (حل أحادي المجال) | من اليمين الى اليسار |
| () | من اليسار الى اليمين |
| [] -> | من اليسار الى اليمين |
| ++ -- (كعوامل postfix) | من اليمين الى اليسار |
| typeid dynamic_cast | من اليمين الى اليسار |
| static_cast const_cast | من اليمين الى اليسار |
| reinterpret_cast | من اليمين الى اليسار |
| ++ -- (كمعامل prefix) ! + - (أحادي) | من اليمين الى اليسار |
| ~ & (عنوان) * (إزالة إشارة) | من اليمين الى اليسار |
| new delete sizeof | من اليمين الى اليسار |
| *.* -> | من اليسار الى اليمين |
| * / % | من اليسار الى اليمين |
| + - | من اليسار الى اليمين |
| << >> | من اليسار الى اليمين |
| =< < => > | من اليسار الى اليمين |
| ! == | من اليسار الى اليمين |
| & | من اليسار الى اليمين |
| ^ | من اليسار الى اليمين |
| | من اليسار الى اليمين |
| && | من اليسار الى اليمين |
| | من اليسار الى اليمين |
| ?: | من اليمين الى اليسار |
| =% /= *= -= += = | من اليمين الى اليسار |
| =^ = =& =<< =>> | من اليمين الى اليسار |
| throw | من اليمين الى اليسار |
| , (معامل التتابع) | من اليسار الى اليمين |

الملحق (ج) مجموعات الرموز

الكود المعياري الأمريكي لتبادل المعلومات

| الكود المعياري الأمريكي لتبادل المعلومات | | | | | | | | | | |
|--|-----|-----|-----|-----|-----|-----|----------|-----|-----|----|
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| ht | bs | bel | ack | enq | eot | etx | stx | soh | nul | 0 |
| dc3 | dc2 | dc1 | dle | si | so | cr | ff | vt | lf | 1 |
| gs | fs | esc | sub | em | can | etb | syn | nak | dc4 | 2 |
| ' | & | *** | \$ | # | " | ! | <u>b</u> | us | rs | 3 |
| 1 | 0 | / | . | - | , | + | * |) | (| 4 |
| ; | : | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 5 |
| E | D | C | B | A | @ | ? | > | = | < | 6 |
| O | N | M | L | K | J | I | H | G | F | 7 |
| Y | X | W | V | U | T | S | R | Q | P | 8 |
| c | b | a | ' | _ | ^ |] | / | [| z | 9 |
| m | l | k | j | i | h | g | f | e | d | 10 |
| w | v | u | t | s | r | q | p | o | n | 11 |
| | | del | - | { | | } | z | y | x | 12 |

الأعداد من 12- في العمود الأول تحدد الرقم الأيسر والأعداد 0-9 في الصف الثاني تحدد الرقم الأيمن من كل رمز في مجموعة بيانات الكود المعياري الأمريكي لتبادل المعلومات. على سبيل المثال الرمز الموجود في الصف ذو الرقم 6 (العدد في العمود الأول) والعمود ذو الرقم 5 (العدد في الصف الثاني) هو A. لهذا يكون الرمز الموجود في الموضع 65 (وهو الرمز رقم 66) هو A. فضلاً عن هذا يقوم الرمز b في الموضع 32 بتمثيل الرمز الفارغ.

الرموز الـ 32 الأولى وهي الرموز الموجودة في المواضع من 00 إلى 31 والموضع 127 عبارة عن رموز غير قابلة للطباعة. هذا الجدول يوضح اختصارات هذه الرموز ومعاني هذه الاختصارات تكون كما يلي:

| | | | | | |
|------------------|-----|-----|-----------------|-----|----------------------|
| رمز باطل | nul | vt | تبويب رأسي | syn | عطل متزامن |
| بداية العنوان | soh | ff | استرجاع الصيغة | etb | نهاية القالب المنقول |
| بداية النص | stx | cr | انتاج النقل | can | الغاء |
| نهاية النص | etx | so | تحويل للخارج | em | نهاية الوسط |
| نهاية النقل | eot | si | تحويل للداخل | sub | استبدال |
| استفسار | enq | dle | ترك وصلة بيانات | esc | ترك |
| اعتراف | ack | dc1 | تحكم الجهاز 1 | fs | فاصل الملف |
| رمز الجرس (صغير) | Bel | dc2 | تحكم الجهاز 2 | gs | فاصل المجموعة |
| زر اعادة الكتابة | bs | dc3 | تحكم الجهاز 3 | rs | فاصل المسجل |
| | ht | dc4 | تحكم الجهاز 4 | us | فاصل الوحدة |
| | lf | nak | اعتراف سلبي | del | حذف |

كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد

| كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد | | | | | | | | | | |
|--|---|----|---|---|----------|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 6 | | | | | <u>b</u> | | | | | |
| 7 | | | | | | . | < | (| + | |
| 8 | & | | . | | | | | | | |
| 9 | ! | \$ | * |) | ; | - | - | / | | |
| 10 | | | | | | | ^ | , | % | _ |
| 11 | > | ? | | | | | | | | |
| 12 | | | : | # | @ | ' | = | " | | a |
| 13 | b | c | d | e | f | g | h | i | | |
| 14 | | | | | | j | k | l | m | n |
| 15 | o | p | q | r | | | | | | |
| 16 | | - | s | t | u | v | w | x | y | z |
| 17 | | | | | | | | \ | } | { |
| 18 |] | [| | | | | | | | |
| 19 | | | | A | B | C | D | E | F | G |
| 20 | H | I | | | | | | | | J |
| 21 | K | L | M | N | O | P | Q | R | | |
| 22 | | | | | | | S | T | U | V |
| 23 | W | X | Y | Z | | | | | | |
| 24 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

الأعداد من 6- 24 في العمود الأول تحدد الرقم الأيسر والأعداد 0-9 في الصف الثاني تحدد الرقم الأيمن من كل رمز في مجموعة بيانات كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد. على سبيل المثال الرمز الموجود في الصف ذو الرقم 19 (العدد في العمود الأول) والعمود ذو الرقم 3 (العدد في الصف الثاني) هو A. لهذا يكون الرمز الموجود في الموضع 193 (وهو الرمز رقم 194) هو A. فضلاً عن هذا يقوم الرمز b في الموضع 32 بتمثيل الرمز الفارغ. الجدول لا يوضح جميع الرموز الموجودة في مجموعة رموز كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد. في الحقيقة تكون الرموز الموجودة في المواضع . - 63 و 250-255 رموز تحكم غير قابلة للطبع.

الملحق

(د)

اثقال العوامل

الجدول التالي يحدد العوامل التي يمكن اثقالها.

| العوامل التي يمكن اثقالها | | | | | | | |
|---------------------------|-----|-----|-----|----|----|----|-----|
| | & | ^ | % | / | * | - | + |
| > | <= | < | == | = | | && | ! |
| ^= | %= | /= | *= | -= | += | != | >= |
| -- | ++ | <<= | >>= | >> | << | &= | = |
| delete | new | - | () | [] | -> | , | ->* |

الجدول التالي يحدد العوامل التي لا يمكن اثقالها.

| العوامل التي لا يمكن اثقالها | | | |
|------------------------------|----|----|----|
| sizeof | ?: | :: | .* |

الملحق (هـ) الملفات الرئيسية

مكتبة C++ المعيارية تحتوي على العديد من الدالات المسبقة التعريف المسماة ثوابت وعلى أنواع بيانات محددة يقوم هذا الملحق بمناقشة بعض من أساليب المكتبة المعتادة الأكثر انتشاراً. وانها لفكرة جيدة أن تعود الى كتيب النظام الخاص بك من أجل ايضاحات اضافية ومن أجل رؤية الأساليب التقليدية الأخرى التي تقدمها المكتبة المعيارية.

الملف الرئيسي **cassert**:

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|-------------------|--|--|
| assert (تعبير) | التعبير هو أي تعبير int وهو يكون عادةً تعبير منطقي | إذا كانت قيمة التعبير غير صفرية (صحيحة) يستمر البرنامج في العمل وإذا كانت قيمة التعبير صفرية (خطأ) يتوقف تنفيذ البرنامج على الفور. يتم استعراض التعبير واسم الملف المحتوي على كود المصدر ورقم السطر في كود المصدر. |

إذا تم وضع موجه ما قبل المعالجة `#define NDEBUG` قبل الموجه `<cassert>` `#include` يتم اهمال جميع بيانات القبول `assert`.

الملف الرئيسي **cctype**:

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|-------------------|-----------------------|--|
| isalnum (ch) | ch عبارة عن قيمة char | الدالة تنتج قيمة int كما يلي: إذا كان ch حرف أو رقم أي ('A' - 'Z' أو 'a' - 'z' أو '0' - '9') تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| isctrl (ch) | ch عبارة عن قيمة char | الدالة تنتج قيمة int كما يلي: إذا كان ch رمز تحكم (في الكود المعياري الأمريكي لتبادل المعلومات قيمة رمز 0-31 أو 127) تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|-------------------|-----------------------|---|
| isdigit (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch رقم من "0" الى "9" تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| isgraph (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch رمز غير فارغ ومطبوع (في الكود المعياري الأمريكي لتبادل المعلومات "1" خلال 0-0) تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| islower (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch حرف صغير ('a' الى 'z') تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| isprint (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch رمز قابل للطبع بما فيه الفراغ (في الكود المعياري الأمريكي لتبادل المعلومات " " خلال 0~0) تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| ispunct (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch رمز تشكيل تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| isspace (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch رمز أبيض الطابع (فارغ، سطر جديد، تبويب، انتاج النقل، استرجاع الصيغة) تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| isupper (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch حرف كبير ('A' الى 'Z') تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| isxdigit (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch رقم من فئة الستة عشر ("0" - "9" أو "A" - "Z" أو "a" الى "z" تنتج قيمة غير صفرية (حقيقية) وبخلاف هذا تنتج صفر (خطأ). |
| tolower (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch حرف كبير تنتج القيمة int للحرف الصغير المكافئ لch وبخلاف هذا تنتج القيمة int للرمز ch. |

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|-------------------|-----------------------|---|
| toupper (ch) | ch عبارة عن قيمة char | تنتج الدالة قيمة int كما يلي: اذا كان ch حرف صغير تنتج القيمة int للحرف الكبير المكافئ لـ ch وبخلاف هذا تنتج القيمة int للرمز .ch |

الملف الرئيسي cmath:

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|-------------------|--|---|
| acos (x) | x تعبير ذو علامة عشرية $-1.0 \leq x \leq 1.0$ | قوس جيب تمام x قيمة بين 0.0 و π |
| asin (x) | x تعبير ذو علامة عشرية $-1.0 \leq x \leq 1.0$ | قوس جيب تمام x قيمة بين $-\pi/2$ و $2/\pi$ |
| atan (x) | x تعبير ذو علامة عشرية | قوس جيب تمام x قيمة بين $-\pi/2$ و $2/\pi$ |
| ceil (x) | x تعبير ذو علامة عشرية | العدد الكلي الأصغر $x \leq$ (الحد الأعلى من x) |
| cos (x) | x تعبير ذو علامة عشرية ويتم قياسه بالتقدير الدائري | جيب تمام مثلث الزاوية. على سبيل المثال اذا $x = 90$ فان جيب تمام x تكون صفر |
| cosh (x) | x تعبير ذو علامة عشرية | جيب التمام hyperbolic للتعبير x |
| exp (x) | x تعبير ذو علامة عشرية | يتم رفع قيمة e الى قوة x ($e = 2.718$) |
| fabs (x) | x تعبير ذو علامة عشرية | القيمة المطلقة لـ x |
| floor (x) | x تعبير ذو علامة عشرية | العدد الكلي الأكبر $x \geq$ (الحد الأدنى من x) |
| log (x) | x تعبير ذو علامة عشرية حيث $0.0 < x$ | اللوغاريتم الطبيعي (القاعدة e) لـ x |
| log10 (x) | x تعبير ذو علامة عشرية حيث $0.0 < x$ | اللوغاريتم المشترك (القاعدة 10) لـ x |

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|-------------------|--|---|
| pow (x, y) | x و y تعبير ذو علامة عشرية. اذا كان x = 0.0 فان y يجب أن تكون موجبة واذا كان $0.0 \leq x$ فان y يجب أن تكون عدد صحيح | يتم رفع x الى قوة y |
| sin (x) | x تعبير ذو علامة عشرية ويتم قياسه بالتقدير الدائري | جيب مثلث الزاوية. على سبيل المثال اذا كان $x = 90$ فان جيب زاوية x يكون 1 |
| sinh (x) | x تعبير ذو علامة عشرية | جيب الزاوية hyperbolic للتعبير x |
| sqrt (x) | x تعبير ذو علامة عشرية حيث $0.0 \leq x$ | الجذر التربيعي لـ x |
| tan (x) | x تعبير ذو علامة عشرية ويتم قياسه بالتقدير الدائري | ظل الزاوية. على سبيل المثال اذا كان $x = 45$ فان ظل الزاوية x يكون 1 |
| tanh (x) | x تعبير ذو علامة عشرية | الظل hyperbolic للتعبير x |

الملف الرئيسي :cstddef

هذا الملف الرئيسي من بين ملفات أخرى يحتوي على تعريف الثابت الرمزي: NULL : المؤشر null المعتمد على النظام (عادةً صفر).

الملف الرئيسي :cstring

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|--------------------------|---|--|
| strcat (destStr, srcStr) | char مصفوفات و destStr و srcStr مصفوفات char منتهية الالغاء ويجب أن تكون destStr كبيرة بشكل كاف لحمل الناتج | العنوان الرئيسي لكل من destStr و srcStr بما فيها رمز الالغاء "0/" متسلسلة على نهاية destStr |
| strcmp (str1, str2) | str1 و str2 مصفوفات char منتهية الالغاء | القيمة الناتجة تكون كما يلي: قيمة int > صفر اذا $str2 > str1$ قيمة int صفر اذا $str2 = str1$ قيمة int < صفر اذا $str2 < str1$ |

| اسم ومعامل الدالة | نوع المعامل | قيمة انتاج الدالة |
|--------------------------|--------------------------------------|---|
| strcpy (destStr, srcStr) | destStr و srcStr مصفوفات char منتهية | يتم انتاج العنوان الرئيسي ل destStr ويتم نسخ srcStr بداخل destStr |
| strlen (str) | str مصفوفات char منتهية الالغاء | قيمة int \leq صفر وهو طول str (باستثناء "0") |

الملف الرئيسي string:

هذا الملف الرئيسي الذي لا يتم الخلط بينه وبين الملف الرئيسي cstring يقدم نوع بيانات يحدده المبرمج يسمى string. بالارتباط مع النوع string هناك نوع بيانات يسمى : string : size_type وثابت يسمى npos : string. هذه الأنواع تعرف كما يلي:

string : size_type نوع صحيح غير محدد مرتبط بعدد الرموز الموجودة في المقطع.

string : npos القيمة القصوى من النوع string : size_type.

هناك عدة دالات مرتبطة بالنوع string ويتم اعطاء البعض منها أدناه. str و str1 و str2 متغيرات (أهداف) من النوع string الا اذا ذكر غير ذلك. مكان الرمز الأول في متغير من النوع string هو صفر والرمز الثاني هو 1 وهكذا.

| اسم ومعامل الدالة | المعطيات | الأثر وقيمة انتاج الدالة |
|---------------------------|--|--|
| str.c_str () | لا توجد | العنوان الرئيسي ل C-string (المصفوفة char المنتهية الالغاء) يتوافق مع الرموز في str. |
| getline (istreamVar, str) | istreamVar متغير تدفق مدخلات (من النوع istream أو ifstream). str هدف (متغير) من النوع string | يتم ادخال الرموز من istreamVar وحفظها في str حتى تتم مواجهة رمز السطر الجديد. (يتم استهلاك رمز السطر الجديد ولا يتم حفظه في str). القيمة الناتجة من هذه الدالة مهمة لأن المبرمجين عادةً يستدعون this كدالة void. |
| str.empty () | لا توجد | تنتج true اذا كانت str فارغة أي أن عدد الرموز في string هو صفر وبخلاف هذا تنتج false. |

| اسم ومعامل الدالة | المعطيات | الأثر وقيمة انتاج الدالة |
|-----------------------|---|---|
| str.length () | لا توجد | قيمة من النوع string::size_type معطية عدد الرموز في المقطع |
| str.size () | لا توجد | قيمة من النوع string::size_type معطية عدد الرموز في المقطع |
| str.find (strExp) | str هدف string وتكون strExp تعبير string يتم تقييمه الى مقطع. قد يكون التعبير المقطعي strExp رمز كذلك. | الدالة find تبحث str للعثور على أول ظهور للمقطع أو الرمز المحدد بواسطة strExp. اذا كان البحث ناجح تنتج الدالة الموضع في str حيث يبدأ التوافق. اذا لم يكن البحث ناجح تنتج الدالة القيمة الخاصة string::npos. |
| str.substr (pos, len) | عددان صحيحان غير محددان pos و len يمثل موضع بدء (المقطع الفرعي في str) و len يمثل طول (المقطع الفرعي). قيمة pos يجب أن تكون أقل من (str.length () | هدف string مؤقت يضم مقطع فرعي من str دءاً من pos. طول المقطع الفرعي هو على الأكثر رموز len. اذا كان len كبير للغاية فانه يعني "الى نهاية" المقطع في str. |
| str1.swap (str2); | معامل واحد من النوع string.str1 و str2 أهداف من النوع string. | يتم تبادل محتويات str1 و str2. |
| str.clear (); | لا توجد | تزيل جميع الرموز من str. |
| str.erase (); | لا توجد | تزيل جميع الرموز من str. |
| str.erase (m); | معامل واحد من النوع string::size_type | تزيل جميع الرموز من str دءاً من المؤشر m. |
| str.erase (m, n); | معاملان من النوع int. | دءاً من المؤشر m تزيل الرموز n التالية من str. اذا كانت n < طول str فانها تزيل جميع الرموز بدءاً من الرمز رقم m. |
| str.insert (m, c); | المعامل m من النوع string::size_type والمعامل c عبارة عن رمز. | تدخل الرمز c عند المؤشر m داخل str. |

| اسم ومعامل الدالة | المعطيات | الأثر وقيمة انتاج الدالة |
|---------------------------|---|--|
| str.insert (m, n, c); | المعاملان m و n من النوع string::size_type والرمز c عبارة عن رمز. | تدخل عدد n من ظهور الرمز c عند المؤشر M داخل str. |
| sr1.insert (m, str2); | المعاملان m و n من النوع string::size_type والرمز str2 عبارة عن مقطع. | تدخل جميع رموز str2 عند المؤشر m داخل str. |
| sr1.replace (m, n, str2); | المعاملان m و n من النوع string::size_type والرمز str2 عبارة عن مقطع. | دعاً من المؤشر m تستبدل الرموز n التالية من str1 بجميع رموز str2. اذا كان n < طول str1 اذن يتم استبدال جميع الرموز حتى نهاية str1. |

الملحق

(9)

موضوعات C++ اضافية

التوريث والمؤشرات والدالات الافتراضية:

الفصل 2 ناقش تعدد الأشكال عبر اثقال العامل وعبر القوالب. هذا القسم يصف نوع ثالث من تعدد الأشكال – عبر الدالات الافتراضية.

كمعامل يمكن تمرير هدف الفئة اما بالقيمة أو بالاشارة. فضلاً عن هذا من المعتاد أن أنواع العوامل الفعلية والرسمية تتوافق. بالرغم من هذا وفي حالة الفئات تسمح C++ للمستخدم بتمرير هدف فئة مستمدة الى معامل رسمي لنوع الفئة الرئيسي.

أولاً لنقم بمناقشة الحالة التي يكون فيها المعامل الرسمي اما معامل اشارة أو مؤشر. لكي نكون محددين لنقم بالنظر الى الفئات التالية:

```
class baseClass
{
public:
    void print();
    baseClass(int u = 0);

private:
    int x;
};

class derivedClass: public baseClass
{
public:
    void print();
    derivedClass(int u = 0, int v = 0);

private:
    int a;
};
```

الفئة baseClass عبارة عن فئة لها ثلاثة عناصر والفئة derivedClass عبارة عن صيغة مستمدة من الفئة baseClass وتحتوي كذلك على ثلاثة عناصر خاصة بها. كلتا الفئتين لها دالة العنصر print. افترض أن تعريفات دالات عنصر كلتا الفئتان تكون كما يلي:

```
void baseClass::print()
{
    cout<<"In baseClass x = "<<x<<endl;
}

baseClass::baseClass(int u)
{
    x = u;
}

void derivedClass::print()
{
    cout<<"In derivedClass ***: ";
    baseClass::print();
    cout<<"In derivedClass a = "<<a<<endl;
}

derivedClass::derivedClass(int u, int v)
    : baseClass(u)
{
    a = v;
}

Consider the following function:
void callPrint(baseClass& p)
{
    p.print();
}
```

الدالة callPrint لها معامل رسمي p من النوع baseClass. يمكنك استدعاء الدالة callPrint عن طريق استخدام هدف اما من النوع baseClass أو من النوع derivedClass كمعامل. فضلاً عن هذا يقوم هيكل الدالة callPrint باستدعاء دالة العنصر print. انظر الى الدالة التالية main:

```
int main()
{
    baseClass one(5); //Line 1
    derivedClass two(3, 15); //Line 2

    one.print(); //Line 3
    two.print(); //Line 4

    cout<<"*** Calling the function callPrint ***"
    <<endl; //Line 5
    callPrint(one); //Line 6
    callPrint(two); //Line 7

    return 0;
}
```

المخرجات:

في الفئة الأساسية $5 = x$

في الفئة المستمدة ***: في الفئة الأساسية $3 = x$

في الفئة المستمدة $15 = a$

*** استدعاء الدالة callPrint

في الفئة الأساسية $5 = x$

في الفئة الأساسية $3 = x$

البيانات في الصفوف من 1 الى 5 واضحة. لنقم بالنظر الى البيانات في الصفين 6 و7. البيان في الصف 6 يقوم باستدعاء الدالة callPrint ويقوم بتمرير الهدف one كمعامل يقوم بتوليد الصف الخامس من المخرجات كما يقوم البيان في الصف 7 باستدعاء الدالة callPrint أيضاً ولكنه يقوم بتمرير الهدف two كمعامل يقوم بتوليد الصف السادس من المخرجات. المخرجات الناتجة عن طريق الصفين 6 و7 توضح فقط قيمة x بالرغم من أنه تم تمرير هدف فئة مختلف كمعامل في كل مرة. (بما أنه تم تمرير الهدف two في الصف 7 كمعامل للدالة callPrint فان المخرجات الناتجة بواسطة الصف 7 يجب أن تكون هي نفسها مخرجات الصفين الثاني والثالث من المخرجات.) هذا لأن دالة العنصر print للفئة الأساسية baseClass يتم تنفيذها بالنسبة الى كلا البيانين (الصف 6 و7). هذا يرجع الى حقيقة أن ربط دالة العنصر print في هيكل الدالة callPrint حدث في وقت مجمع. بما أن المعامل p الخاص بالدالة callPrint من النوع baseClass بالنسبة الى البيان (); p.print اذن يقوم المجمع بربط الدالة print من الفئة baseClass. بشكل أكثر تحديداً يتم في ربط زمن التجميع توليد الكود اللازم لاستدعاء دالة محددة عن طريق المجمع. (يعرف ربط زمن التجميع كذلك بالربط الساكن.)

بالنسبة الى البيان في الصف 7 يكون المعامل الحقيقي من النوع derivedClass. لهذا عندما يتم تنفيذ هيكل الدالة two يجب منطقياً أن يتم تنفيذ الدالة print للهدف two وهذا ليس هو الوضع الدائم. لهذا أثناء تنفيذ البرنامج كيف تقوم C++ بتصحيح هذه المشكلة المتعلقة بعمل الاستدعاء للدالة المناسبة؟ تقوم C++ بتصحيح هذه المشكلة عن طريق توفير آلية الدالات الافتراضية. يحدث ربط الدالات الافتراضية في زمن تنفيذ البرنامج وليس في زمن التجميع. هذا النوع من الربط يطلق عليه الربط في زمن التشغيل. بشكل رسمي أكثر لا يقوم المجمع في الربط في زمن التشغيل بتوليد الكود لاستدعاء دالة محددة وبدلاً من هذا يقوم بتوليد معلومات كافية لتمكين نظام زمن التشغيل من توليد الكود الخاص باستدعاء الدالة المناسبة. كما يطلق على الربط في زمن التشغيل أيضاً الربط الحركي. في C++ يتم اعلان الدالات الافتراضية باستخدام الكلمة المحفوظة virtual. لنقم باعادة تعريف الفئات السابقة باستخدام هذه الخاصية:

```

class baseClass
{
public:
    virtual void print();           //virtual function
    baseClass(int u = 0);

private:
    int x;
};

class derivedClass: public baseClass
{
public:
    void print();
    derivedClass(int u = 0, int v = 0);

private:
    int a;
};

```

لاحظ أننا نحتاج الى اعلان دالة virtual فقط في الفئة الأساسية.
 تعريف دالة العنصر print هي كما سبق. اذا قمنا بتنفيذ البرنامج السابق بهذه التعديلات تكون
 المخرجات كما يلي.
المخرجات:

في الفئة الأساسية $5 = x$

في الفئة المستمدة ***: في الفئة الأساسية $3 = x$

في الفئة المستمدة $15 = a$

*** استدعاء الدالة callPrint

في الفئة الأساسية $5 = x$

في الفئة المستمدة ***: في الفئة الأساسية $3 = x$

في الفئة المستمدة $15 = a$

هذه المخرجات توضح أنه بالنسبة الى البيان في الصف رقم 7 يتم تنفيذ دالة print من الفئة المستمدة
 (انظر الى السطرين الأخيرين من المخرجات).

تتطبق كذلك هذه المناقشة عندما يكون المعامل الرسمي عبارة عن مؤشر الى فئة ما ويتم تمرير مؤشر
 الفئة المستمدة كمعامل حقيقي. لايضاح هذه الخاصية افترض أن لدينا الفئات السابقة. (نفترض أن
 تعريف الفئة baseClass موجود في الملف الرئيسي baseClass.h وتعريف الفئة derivedClass
 موجود في الملف الرئيسي derivedClass.h). انظر الى البرنامج التالي:

```

//Virtual Functions
#include <iostream>
#include "derivedClass.h"
using namespace std;

void callPrint(baseClass *p);

int main()
{
    baseClass *q; //Line 1
    derivedClass *r; //Line 2

    q = new baseClass(5); //Line 3
    r = new derivedClass(3, 15); //Line 4

    q->print(); //Line 5
    r->print(); //Line 6

    cout<<"*** Calling the function callPrint ***"
    <<endl; //Line 7
    callPrint(q); //Line 8
    callPrint(r); //Line 9

    return 0;
}

void callPrint(baseClass *p)
{
    p->print();
}

```

المخرجات:

في الفئة الأساسية $5 = x$

في الفئة المستمدة ***: في الفئة الأساسية $3 = x$

في الفئة المستمدة $15 = a$

*** استدعاء الدالة callPrint

في الفئة الأساسية $5 = x$

في الفئة المستمدة ***: في الفئة الأساسية $3 = x$

في الفئة المستمدة $15 = a$

ان تمرير هدف فئة مستمدة ما كعامل لمعامل رسمي لنوع الفئة الأساسية يعمل بشكل جيد فقط اذا كان المعامل الرسمي اما معامل اشارة أو مؤشر. اذا كان المعامل الرسمي معامل قيمة لا يعمل تمرير هدف فئة مستمدة ما الى معامل رسمي لنوع الفئة الأساسية بشكل جيد حتى في حالة الدالة الافتراضية. تذكر أنه اذا كان المعامل الرسمي معامل قيمة فانه يتم نسخ قيمة المعامل الحقيقي داخل المعامل الرسمي. اذا كان المعامل الرسمي من النوع class يتم نسخ عناصر بيانات الهدف الحقيقي داخل عناصر البيانات المقابلة لها من المعامل الرسمي.

افترض أن لدينا الفئات السابقة – وهي baseClass و derivedClass. انظر الى تعريف الدالة التالي:

```
void callPrint(baseClass p) //p is a value parameter
{
    p.print();
}
```

وافترض كذلك أن لدينا الاعلان التالي:

```
derivedClass two;
```

الهدف two له عنصري بيانات x و a. يتم توريث عنصر البيانات x من الفئة الأساسية. انظر الى استدعاء الدالة التالي:

```
callPrint(two);
```

في هذا البيان يتم نسخ عناصر بيانات two داخل عناصر بيانات p لأن المعامل الرسمي p عبارة عن معامل قيمة. بالرغم من هذا وبما أن p هدف من النوع baseClass فان له عنصر بيانات واحد فقط. نتيجة لهذا يتم نسخ عنصر البيانات x من two فقط داخل عنصر البيانات x من p. كما أن البيان:

```
p.print ( );
```

في هيكل الدالة يتسبب في تنفيذ دالة العنصر print من الفئة baseClass.

مخرجات البرنامج التالي توضح هذه الفكرة. (كما سبق فنفترض أن تعريف الفئة baseClass موجود في الملف الرئيسي baseClass.h وتعريف الفئة derivedClass موجود في الملف الرئيسي derivedClass.h).

```
//Virtual Functions and value parameters

#include <iostream>

#include "derivedClass.h"

using namespace std;

void callPrint(baseClass p);

int main()
{
    baseClass one(5); //Line 1
    derivedClass two(3, 15); //Line 2

    one.print(); //Line 3
    two.print(); //Line 4

    cout<<"*** Calling the function callPrint ***"
        <<endl; //Line 5
    callPrint(one); //Line 6
    callPrint(two); //Line 7

    return 0;
}

void callPrint(baseClass p) //p is a value parameter
{
    p.print();
}
```

المخرجات:

في الفئة الأساسية $5 = x$

في الفئة المستمدة ***: في الفئة الأساسية $3 = x$

في الفئة المستمدة $15 = a$

*** استدعاء الدالة callPrint ***

في الفئة الأساسية $5 = x$

في الفئة الأساسية $3 = x$

انظر عن قرب الى مخرجات البيانات في الصفين 6 و 7 (آخر سطرين من المخرجات). في الصف 7 يتم نسخ عناصر بيانات two داخل عناصر بيانات p لأن المعامل الرسمي p عبارة عن معامل قيمة. بالرغم من هذا وبما أن p هدف من النوع baseClass فان له عنصر بيانات واحد فقط. نتيجة لهذا يتم نسخ عنصر البيانات x من two فقط داخل عنصر البيانات x من p. فضلاً عن هذا فان البيان `p.print ()` يقوم في الدالة callPrint بتنفيذ الدالة print من الفئة baseClass وليس الفئة derivedClass. لهذا فان السطر الأخير من المخرجات يوضح قيمة x (عنصر بيانات two). لا يمكن تمرير هدف من نوع الفئة الأساسية الى معامل رسمي من نوع الفئة المستمدة.

الفئات والمدمرات الافتراضية:

هناك شئ يوصى به من أجل الفئات ذات عناصر بيانات المؤشر وهو أن هذه الفئات لها المدمر. يتم تنفيذ المدمر تلقائياً عندم يخرج هدف الفئة من المجال. لهذا اذا خلق الهدف أهداف حركية يمكن تحديد المدمر لازالة تخصيص مكان حفظها. اذا تم تمرير هدف فئة مستمدة الى معامل رسمي من نوع الفئة الأساسية يتم تنفيذ مدمر الفئة الأساسية بصرف النظر عما اذا تم تمرير هدف الفئة المستمدة بالاشارة أم بالقيمة. من الناحية المنطقية يجب أن يتم تنفيذ مدمر الفئة المستمدة عندما يخرج هدف الفئة المستمدة من المجال.

لتصحيح هذه المشكلة يجب أن يكون مدمر الفئة الأساسية افتراضياً. ان المدمر الافتراضي للفئة الأساسية يقوم تلقائياً بجعل مدو الفئة المستمدة افتراضياً. عندما يتم تمرير هدف فئة مستمدة الى معامل رسمي من نوع الفئة الأساسية يتم تنفيذ مدمر الفئة المستمدة عند خروج الهدف من المجال. بعد تنفيذ مدمر الفئة المستمدة يتم تلقائياً تنفيذ مدمر الفئة الأساسية. لهذا عندما يتم تدمير هدف الفئة المستمدة يتم كذلك تدمير جزء الفئة الأساسية من هدف الفئة المستمدة (أي العناصر الموروثة من الفئة الأساسية).

اذا اشتملت الفئة الأساسية على دالات افتراضية تحتاج الى جعل مدمر الفئة الأساسية افتراضياً.

عنوان العامل والفئات:

لقد قام الفصل 3 باستخدام عنوان العامل & لتخزين عنوان متغير ما في متغير مؤشر. كما يتم استخدام عنوان العامل كذلك لعمل أسماء مستعارة لهدف ما. انظر الى البيانات التالية:

```
int x;
```

```
int &y = x;
```

البيان الأول يعلن أن x متغير من النوع int والبيان الثاني يعلن أن y اسم مستعار للمتغير x. هذا يعني أن كلا من x و y يشيران الى نفس موقع الذاكرة. لهذا تكون y متغير مؤشر ثابت. البيان:

```
y = 25;
```

يحدد قيمة y (ومن هنا) قيمة x عند 25. بالمثل يقوم البيان:

```
x = 2 * x + 30;
```

بتحديث قيمة x (ومن هنا) قيمة y.

يمكن كذلك استخدام عنوان العامل من أجل انتاج عنوان عنصر بيانات خاص لفئة ما. بالرغم من هذا اذا لم تكن حذراً يمكن أن تتسبب هذه العملية في أخطاء خطيرة في البرنامج. المثال التالي يساعد على توضيح هذه الفكرة.

انظر الى تعريف الفئة التالي:

```
//Header file testadd.h

#ifndef H_testAdd
#define H_testAdd

class testAddress
{
public:
    void setX(int);
    void printX() const;
    int& addressOfX(); //this function returns the
                     //address of the private data member

private:
    int x;
};

#endif
```

تعريفات دالات تطبيق دالات العنصر تكون كما يلي:

```
//Implementation file testAdd.cpp

#include <iostream>
#include "testAdd.h"
```

```

using namespace std;

void testAddress::setX(int inX)
{
    x = inX;
}

void testAddress::printX() const
{
    cout<<x;
}

int& testAddress::addressOfX()
{
    return x;
}

```

بما أن ما تنتجه الدالة addressOfX هو عنوان موقع ذاكرة من النوع int فان البيان:

```
return x;
```

ينتج عنوان x.

بعد هذا نقم بكتابة برنامج بسيط يقوم باستخدام الفئة testAddress ويوضح ما يمكن ألا يعمل على نحو صحيح. لاحقاً سوف نوضح كيفية اصلاح هذه المشكلة.

```

//Test program

#include <iostream>
#include "testAdd.h"

using namespace std;

int main()
{
    testAddress a;
    int &y = a.addressOfX();

    a.setX(50);
    cout<<"x in class testAddress = ";
    a.printX();
    cout<<endl;

    y = 25;
    cout<<"After y = 25, x in class testAddress = ";
    a.printX();
    cout<<endl;

    return 0;
}

```

المخرجات:

x في الفئة testAddress = 50

بعد y = 25، تكون x في الفئة testAddress = 25

في البرنامج السابق وبعد تنفيذ البيان:

```
int &y = a.addressOfx();
```

تصبح y اسم مستعار لعنصر البيانات الخاص x للهدف a. لهذا يقوم البيان:

```
y = 25;
```

بتغيير قيمة x.

قال الفصل الأول أن عناصر البيانات الخاصة لا يمكن تناولها خارج الفئة. مع هذا وعن طريق انتاج عناوين هذه العناصر يمكن للمبرمج التحكم فيها. من احدى طرق حل هذه المشكلة عدم امداد مستخدم الفئة بعنوان عناصر البيانات الخاصة أبداً. بالغرم من هذا قد يكون من اللازم في بعض الأحيان تقديم عنوان عنصر البيانات الخاص. كيف يمكننا منع البرنامج من التحكم المباشر في عناصر البيانات الخاصة؟ لاصلاح هذه المشكلة نقوم باستخدام الكلمة const قبل نوع انتاج الدالة. بهذه الطريقة لا يزال يمكننا انتاج عناوين عناصر البيانات الخاصة ولكننا في نفس الوقت نمنع المبرمج من التحكم المباشر في عناصر البيانات الخاصة. لنقم باعادة كتابة الفئة testAddress باستخدام هذه الخاصية:

```
#ifndef H_testAdd
#define H_testAdd

class testAddress
{
public:
    void setX(int);
    void printX() const;
    const int& addressOfX(); //this function returns the
                           //address of the private data
                           //member
private:
    int x;
};

#endif
```

تعريف الدالة addressOfX في ملف التطبيق هو:

```
const int& testAddress::addressOfX()
{
    return x;
}
```

يقوم نفس البرنامج الآن بتوليد خطأ في زمن التجميع.

الملحق

(ز)

C++ لمبرمجي الجافا

يفترض هذا الكتاب أنك على علم بالعناصر الأساسية لـ C++ مثل أنواع البيانات، وبيانات الاسناد، والادخال / الاخراج، وبنيات التحكم، والدالات وتمرير المعاملات، وآلية الأسماء، والمصفوفات. بالرغم من هذا يقوم هذا الملحق من أجل مساعدتك بعمل مراجعة سريعة لهذه العناصر الأساسية من C++. فضلاً عن هذا اذا كنت قد تناولت الجافا كلغة برمجة أولى فان هذا الملحق يساعدك على معرفة العناصر الأساسية من C++. بالاضافة الى وصف العناصر الأساسية من C++ نقوم كذلك بمقارنة خصائص متنوعة للغة C++ مع خصائص الجافا. لمزيد من التفاصيل عن لغة C++ ارجع الى كتاب برمجة C++ : من تحليل المشكلة الى تصميم البرنامج بواسطة الكاتب والمذكور في المراجع ((7) الملحق ح).

أنواع البيانات:

تقع أنواع بيانات C++ في ثلاث فئات – أنواع بيانات بسيطة، وأنواع بيانات منظمة، ومؤشرات. الفصل الأول يصف الفئات المحددة بواسطة المستخدم والتي تقع في فئة أنواع البيانات المنظمة والفصل 3 يصف المؤشرات. هذا القسم يناقش أنواع البيانات البسيطة. فضلاً عن هذا نقوم في جزء لاحق من هذا الملحق بمناقشة المصفوفات كنوع بيانات منظم في C++ بشكل مختصر. ان نوع البيانات البسيط من C++ مماثل لنوع بيانات الجافا البدائية. من أنواع البيانات المتكاملة char، bool، short، int، و long، و unsigned int. الجدول ز-1 يقوم بتعريف نطاق القيم التي تنتمي الى بعض من أنواع البيانات هذه.

جدول ز-1: قيم وتخصيص الذاكرة لثلاثة أنواع بيانات بسيطة

| نوع البيانات | القيم | التخزين |
|--------------|--------------------------------|---------|
| int | من - 2147483648 الى 2147483647 | 4 |
| bool | true و false | 1 |
| char | من - 128 الى 127 | 1 |

استخدم هذا الجدول كدليل فقط. قد يسمح المجمعون المختلفون بوجود نطاقات مختلفة من القيم. قم بمراجعة وثائق المجمع الخاص بك.

ان نوع البيانات int في لغة C++ يعمل بنفس الطريقة التي يعمل بها نفس نوع البيانات int في لغة الجافا.

لاحظ أن نوع البيانات char في C++ عبارة عن مجموعة من 256 قيمة بينما نوع البيانات char في الجافا عبارة عن مجموعة من 65536 قيمة. بالاضافة الى التعامل مع الأرقام الصغيرة يتم استخدام نوع البيانات char لتمثيل رموز عبارة عن حروف وأرقام ورموز خاصة. من المعتاد أن تقوم لغة C++ باستخدام رموز الكود المعياري الأمريكي لتبادل المعلومات وهو عبارة عن مجموعة مكونة من 128 رمز وتم توضيحها في الملحق (ج) للتعامل مع الرموز.

نوع البيانات bool له قيمتين فقط وهما: true و false. كذلك يطلق على القيمتين true و false قيم منطقية. التعبير الذي يتم تقييمه عند true أو false يسمى **تعبير منطقي**.

للتعامل مع الأعداد العشرية تقدم لغة C++ نوع البيانات ذو العلامة العشرية. ان لغة C++ توفر ثلاثة أنواع من البيانات للتحكم في الاعداد العشرية هي: float، double، و long double. كما في حالة أنواع البيانات المتكاملة تختلف أنواع البيانات float، double، و long double في مجموعات القيم. ان نوعي البيانات float و double في C++ يعملان بنفس طريقة عملهما في الجافا.

عوامل وتعبيرات حسابية:

تعمل العوامل الحسابية الخمسة - +، -، *، / - بنفس الطريقة التي تعمل بها في الجافا. فضلاً عن هذا يتم تشكيل وتقييم التعبيرات الحسابية في لغة C++ بنفس الطريقة التي يتم بها ذلك في الجافا. بالاضافة الى هذا فان عامل الزيادة ++، وعامل النقص --، وعوامل الاسناد المركبة + =، و - =، و * =، و / =، و % = يعملون في C++ بنفس الطريقة التي يعملون بها في لغة الجافا.

عامل الطرح في C++ يتخذ الصيغة التالية:

`static_cast <dataType> expression`

كما يمكنك استخدام عامل الطرح الذي يشبه C:

`dataType expression`

الثوابت المسماة والمتغيرات وبيانات الاسناد:

يتم اعلان الثوابت المسماة في C++ باستخدام الكلمة المحفوظة const. التركيب العام لاعلان ثابت مسمى هي:

```
const dataType identifier = value;
```

على سبيل المثال يقوم البيان التالي باعلان أن conversion ثابت مسمى من النوع double ويحدد القيمة 2.54 له.

cast double conversion = 2.54

في لغة C++ يتم اعلان المتغيرات بنفس الطريقة المستخدمة في لغة الجافا ويكون تركيب بيان الاسناد واحداً في كل من اللغتين.

الترتيب العام لاعلان متغير واحد أو عدة متغيرات هو:

```
dataType identifier, identifier, . . . ;
```

على سبيل المثال تقوم البيانات التالية باعلان أن amountDue متغير من النوع double وأن counter متغير من النوع int.

```
double amountDue;
```

```
int counter;
```

تركيب بيان الاسناد هو:

```
variable = expression;
```

في بيان الاسناد يجب أن تتفق قيمة التعبير expression مع نوع بيانات المتغير. يتم تقييم التعبير الموجود على الجانب الايمن ويتم تحديد قيمته للمتغير الموجود على الجانب الأيسر. على سبيل المثال افترض أن amountDue متغير من النوع double وأن quantity متغير من النوع int. اذا كانت قيمة quantity تبلغ 20 اذن فان البيان التالي يحدد 150.00 الى amountDue:

```
amountDue = quantity * 7.50;
```

مكتبة C++: موجّهات ما قبل المعالجة:

يتم تعريف عدد صغير فقط من العمليات مثل العمليات الحسابية وعمليات الاسناد في لغة C++ ويتم تقديم العديد من الدالات والرموز اللازمة لتشغيل برنامج C++ كمجموعة من المكتبات. كل مكتبة لها اسم ويشار اليها **بالملف الرئيسي**. على سبيل المثال أوصاف الدالات اللازمة لأداء دالات رياضية للدخال / الاخراج موجودة في الملف الرئيسي iostream. بالمثل تتواجد أوصاف بعض من الدالات الرياضية المفيدة للغاية مثل power، absolute، و sine في الملف الرئيسي cmath. اذا كنت تريد استخدام دالات الادخال / الاخراج أو دالات رياضية فأنك تحتاج الى أن تخبر الحاسوب المكان الذي يمكن العثور فيه على الكود اللازم. انك تقوم باستخدام موجّهات ما قبل المعالجة وأسماء الملفات الرئيسية لاخبار الحاسوب عن مواقع الكود المتوفر في المكتبات. تتم معالجة موجّهات ما قبل المعالجة ببرنامج يسمى **المعالج الأولي**.

موجهات ما قبل المعالجة عبارة عن أوامر يتم تقديمها الى برنامج المعالج الأولي الذي يقوم بتعديل نص برنامج C++ قبل تجميعه. جميع أوامر المعالج الأولي تبدأ بالرمز # ولا توجد فواصل منقوطة في نهاية أوامر المعالج الأولي لأنها ليست أوامر C++. لاستخدام ملف رئيسي في برنامج C++ استخدم موجه المعالج الأولي include. التركيب العام لتضمين ملف رئيسي مقدم من النظام في برنامج C++ هو:

```
#include <headerFileName>
```

على سبيل المثال يقوم البيان التالي بتضمين الملف الرئيسي iostream في برنامج C++:

```
#include <iostream>
```

يتم وضع موجهات ما قبل المعالجة التي تتضمن الملفات الرئيسية كسطور أولى من البرنامج حتى يمكن استخدام المحددات المعلنه في هذه الملفات الرئيسية خلال البرنامج. (في C++ يجب أن يتم اعلان المحددات قبل امكانية استخدامها).

يتم تقديم ملفات رئيسية معينة كجزء من C++. الملحق هـ يصف بعضاً من الملفات الرئيسية الشائعة الاستخدام كما يمكن للمبرمجين أن يقوموا بعمل ملفات رئيسية خاصة بهم. في الفصل الأول تعلمت كيفية عمل ملفات رئيسية خاصة بك ثم ادخالها في برنامج C++.

برنامج C++:

كل برنامج C++ مكون من جزئين: مرجحات ما قبل المعالجة والبرنامج. موجهات ما قبل المعالجة هي أوامر تقوم بتوجيه المعالج الى تعديل برنامج C++ قبل التجميع. البرنامج يحتوي على بيانات تحقق نتائج مفيدة. موجهات ما قبل المعالجة الى جانب بيانات البرنامج تشكل **كود مصدر C++**. لكي يكون كود المصدر مفيداً يجب حفظه في ملف يكون امتداده `.cpp`.

عندما يتم تجميع البرنامج يقوم المجمع بتوليد كود الهدف الذي يتم حفظه ملف امتداده `(.obj)`. عند ربط كود الهدف بموارد النظام يتم انتاج الكود القابل للتنفيذ ويتم حفظه في ملف امتداده `(.exe)`. اسم الملف المحتوي على كود الهدف واسم الملف المحتوي على الكود القابل للتنفيذ هو نفسه اسم الملف المحتوي على كود المصدر. على سبيل المثال اذا كان كود المصدر موجوداً في ملف يسمى `firstProg.cpp` فان اسم الملف المحتوي على كود الهدف هو `firstProg.obj` واسم الملف المحتوي على الكود القابل للتنفيذ هو `firstProg.exe`.

الامتدادات المعطاة في الفقرات السابقة – وهي `.cpp`، `.obj`، و `.exe` – تعتمد على النظام. لكي تكون شديد التأكد تفقد أوراق أدوات تطوير النظام أو البرنامج.

برنامج C++ عبارة عن مجموعة من الدالات وواحدة من هذه الدالات الدالة main. لهذا يجب أن يحتوي كل برنامج C++ على الدالة main. الأجزاء الرئيسية للدالة main هي العنوان وهيكل الدالة. العنوان يكون بالصيغة التالية:

typeOfFunction main (argument list)

على سبيل المثال البيان:

int main ()

يعني أن الدالة main تنتج قيمة من نوع البيانات int وليس لها معطيات.

يجب أن يتم وضع هيكل الدالة بين أقواس مجمدة ({ و }) وهو يحتوي على نوعين من البيانات:

■ بيانات اعلان.

■ بيانات قابلة للتنفيذ.

يتم استخدام بيانات الاعلان لاعلان أشياء مثل المتغيرات. في C++ يمكن اعلان المتغيرات أو المحددات في أي مكان في البرنامج ولكن يجب اعلانها قبل أن يمكن استخدامها. البيانات القابلة للتنفيذ تقوم بأداء الحسابات، والتحكم في البيانات، وقبول المدخلات، وعمل المخرجات، وغيرها. من البيانات القابلة للتنفيذ التي مررت بها بيانات الاسناد والادخال والايخارج.

ما يلي مثال على برنامج C++.

```
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;

    num1 = 10;
    num2 = 2 * num1;

    cout<<"num1 = "<<num1<<" , and num2 = "<<num2<<endl;
    return 0;
}
```

القسم التالي يناقش الادخال والايخارج بالتفصيل.

الادخال والايخارج:

ان ادخال البيانات واخراج نتائج البرنامج شئ أساسي لأي لغة برمجة. بما أن الادخال / الاخراج يختلف بشكل كبير في C++ والجافا فان هذا القسم يصف الادخال / الاخراج في C++ بالتفصيل.

الادخال:

وضع البيانات داخل متغيرات من جهاز الادخال القياسا يتم تحقيقه عبر استخدام cin والعامل

>>. تركيب cin مع >> يكون:

```
cin>>variable>>variable. . .;
```

>> هذا يسمى بيان ادخال (قراءة). أحياناً يطلق على هذا أيضاً بيان cin. في C++ يطلق على عامل استخلاص التدفق أو ببساطة عامل الاستخلاص.

بيان الادخال (أو cin) يعمل كما يلي. افترض أن miles متغير من نوع البيانات double. البيان:

```
cin>>miles;
```

يتسبب في حصول الحاسوب على قيمة من جهاز الادخال القياسي من نوع البيانات double ووضعها في خلية الذاكرة المسماة miles.

باستخدام أكثر من متغير مع cin يمكن قراءة أكثر من قيمة واحدة في مرة واحدة. افترض أن feet وinch متغيرات من النوع int. بيان مثل:

```
cin>>feet>>inch;
```

يحصل على عددين صحيحين من لوحة المفاتيح و يضعهما في مواقع الذاكرة feet وinch على التوالي.

يتم تعريف عامل الاستخلاص >> فقط من أجل ادخال البيانات داخل متغيرات من أنواع البيانات البسيطة. لهذا يكون الجزء الأيمن من عامل الاستخلاص >> متغير من نوع البيانات البسيط.

كيف يعمل عامل الاستخلاص >>؟ عند المسح من أجل الادخال التالي يقوم >> بتجاوز جميع الرموز البيضاء. تتكون الرموز البيضاء من فراغات ورموز محددة غير قابلة للطباعة مثل التبويبات ورمز السطر الجديد. لهذا سواء كنت تقوم بفصل بيانات الادخال بفراغات أو سطور يقوم معامل الاستخلاص >> ببساطة بالعثور على بيانات الادخال التالي في تدفق المدخلات. على سبيل المثال افترض أن payRate وhoursWorked متغيرات من النوع double. انظر الى بيان الادخال التالي:

```
cin>>payRate>>hoursWorked;
```

سواء كانت المدخلات:

48.30 15.50

أو:

48.30 15.50

أو:

15.50

48.30

فسوف يقوم ببيان الادخال السابق بتخزين 15.50 في payRate وتخزين 48.3 في hoursWorked. لاحظ أن المدخلات الأولى مفصولة بفراغ والمدخلات الثانية مفصولة بتبويب والمدخلات الثالثة مفصولة بسطر.

الآن افترض أن المدخلات هي 2. كيف يقوم عامل الاستخلاص >> بالتمييز بين الرمز 2 والعدد 2؟ معامل الجانب الأيسر لعامل الاستخلاص >> يصنع هذا الفرق. اذا كان معامل الجانب الايمن متغير من النوع char تتم معاملة المدخلات 2 على أنها الرمز 2 وفي هذه الحالة يتم تخزين قيمة الكود المعياري الأمريكي لتبادل المعلومات البالغ 2. اذا كان معامل الجانب الأيمن متغير من النوع int أو double تتم معاملة المدخلات 2 على أنها العدد 2.

بعد هذا انظر الى المدخلات 25 والبيان:

cin>>a;

حيث a متغير من بعض نوع البيانات البسيط. اذا كانت a من النوع char يتم تخزين الرمز المفرد 2 فقط في a. اذا كانت a من النوع int يتم تخزين 25 في a واذا كانت من النوع double يتم تحويل المدخلات 25 الى العدد العشري 25.0. الجدول ز-2 يقوم بتلخيص هذه المناقشة عن طريق توضيح المدخلات المتاحة لمتغير من نوع البيانات البسيطة.

جدول ز-2: مدخلات متاحة لمتغير من نوع البيانات البسيطة

| نوع البيانات | المدخلات المتاحة |
|--------------|--|
| char | رمز واحد قابل للطبع فيما عدا الفراغ |
| int | عدد صحيح قد تسبقه علامة (+ أو -) |
| double | عدد عشري قد تسبقه علامة (+ أو -). اذا كانت مدخلات البيانات الفعلية عدد صحيح يتم تحويل المدخلات الى عدد عشري مع الجزء العشري الصفري |

عند قراءة البيانات داخل متغير char يقوم عامل الاستخلاص >> قبل تجاوز أي رموز بيضاء بالعثور على وحفظ الرمز التالي فقط وتتوقف القراءة بعد رمز واحد. لقراءة البيانات داخل متغير int أو double يقوم معامل الاستخلاص >> بعد تجاوز جميع الرموز البيضاء وقراءة علامة الموجب أو

السالب (إذا وجدت) بقراءة أرقام العدد بما فيها العلامة العشرية للمتغيرات ذات العلامة العشرية ويتوقف عندما يجد رمز أبيض أو رمز بخلاف رقم ما.

فشل المدخلات:

هناك عدة أشياء يمكن أن تسير على نحو خاطئ أثناء تنفيذ البرنامج. البرنامج الذي يكون صحيح من الناحية التركيبية قد ينتج نتائج غير صحيحة. على سبيل المثال افترض أنه يتم حساب راتب الموظف الذي يعمل وقت جزئي عن طريق استخدام الصيغة التالية:

```
wages = payRate * hoursWorked;
```

إذا قمت عن طريق الخطأ بكتابة + بدلاً من * سوف تكون الأجور المحسوبة غير صحيحة بالرغم من أن البيان المحتوي على العلامة + صحيح من الناحية التركيبية.

ماذا عن محاولة قراءة بيانات غير صحيحة؟ على سبيل المثال ماذا يحدث إذا حاولت ادخال حرف داخل متغير int؟ إذا لم تتفق بيانات الادخال مع المتغيرات المقابلة لها سوف يمر البرنامج بمشكلات. على سبيل المثال سوف تتسبب محاولة قراءة حرف داخل متغير int أو double في فشل المدخلات. انظر الى البيانات التالية:

```
int a, b, c;
```

```
double x;
```

إذا كانت المدخلات:

```
w 54
```

اذن البيان:

```
cin>>a>>b;
```

سوف يتسبب في فشل المدخلات لأنك تحاول ادخال الرمز "w" داخل المتغير a من النوع int. إذا كانت المتغيرات:

```
78 48 67.93 35
```

فان بيان الادخال:

```
cin>>a>>x>>b;
```

سوف يتسبب في حفظ 35 في a وحفظ 67.93 في x، وحفظ 48 في b.

انظر الآن الى بيان القراءة التالي مع المدخلات السابقة (المدخلات ذات ثلاثة قيم):

```
cin>>a>>b>>c;
```

هذا البيان يقوم بحفظ 35 في a و 67 في b وتتوقف القراءة عند . (العلامة العشرية). بما أن المتغير التالي c من نوع البيانات int يحاول الحاسوب قراءة . داخل c وهذا يعتبر خطأ. بعد هذا يدخل تدفق المدخلات في حالة تعرف بحالة الفشل.

ماذا يحدث عندما يدخل تدفق الادخال في حالة الفشل؟ بمجرد دخول تدفق المدخلات في حالة الفشل يتم اهمال جميع بيانات الادخال / الاخراج الأخرى التي تستخدم هذا التدفق. لسوء الحظ يستمر تنفيذ البرنامج بهدوء بالقيم المخزنة في المتغيرات أياً كانت ويقدم نتائج غير صحيحة.

الاجراج:

في C++ يتم تحقيق الاخراج على جهاز الاخراج القياسي عبر استخدام cout والعامل <<. تركيب cout مع << يكون:

```
cout<<expression or manipulator<<expression or manipulator...;
```

هذا يسمى بيان اخراج أحياناً يطلق على هذا أيضاً بيان cout. في C++ يطلق على << عامل ادخال التدفق أو ببساطة عامل الادخال. توليد مخرجات بالبيان cout يتبع قاعدتين:

1. يتم تقييم التعبير ويتم طباعة قيمته عند نقطة الادخال الحالية على جهاز الاخراج. (على الشاشة تكون علامة الادخال عند مكان المؤشر).
 2. يتم استخدام متحكم لتكوين المخرجات. أبسط المتحكمين هو endl (الرمز الأخير هو الحرف el) الذي يتسبب في انتقال نقطة الادخال الى بداية السطر التالي.
- المثال ز-1 يوضح كيفية عمل بيانات cout. في البيان cout يتم تقييم مقطع أو تعبير متضمن اما متغير واحد فقط أو قيمة واحدة.

مثال ز-1:

انظر الى البيانات التالية. المخرجات موضحة على يمين كل بيان.

| المخرجات | البيان | |
|--------------|----------------------------------|---|
| 7 | cout<<29 / 4<< endl; | 1 |
| Hello there. | cout<< "Hello there. "<<endl; | 2 |
| 12 | cout<<12<<endl; | 3 |
| 7 + 4 | cout<< "4 + 7" <<endl; | 4 |
| 11 | cout<< 4 + 7 <<endl; | 5 |
| A | cout<< 'A' <<endl; | 6 |
| 11 = 7 + 4 | cout<< "4 + 7 = " <<4 + 7<<endl; | 7 |
| 17 | cout<<2 + 3 * 5<<endl; | 8 |
| Hello there. | cout<< "Hello \nthere. "<<endl; | 9 |

:Setprecision

تقوم باستخدام المتحكم setprecision للتحكم في مخرجات الأعداد ذات العلامة العشرية. المخرجات الافتراضية للأعداد ذات العلامة العشرية عبارة عن ترميز علمي. هناك بعض أدوات لتطوير البرمجيات قد تستخدم حد أقصى يبلغ ستة منازل عشرية للمخرجات الافتراضية للأعداد ذات العلامة العشرية. بالرغم من هذا عندما تتم طباعة راتب موظف ما تكون المخرجات المرجوة عبارة عن حد

أقصى يبلغ منزلتين عشريتين. لطباعة مخرجات ذات علامة عشرية عند منزلتين عشريتين تقوم باستخدام المتحكم `setprecision` لتحديد الضبط عند 2. التركيب العام للمتحكم `setprecision` هو:

```
setprecision(n)
```

حيث `n` هو عدد المنزلات العشرية.

تقوم باستخدام المتحكم `setprecision` مع `cout` ومعامل الاستخلاص. على سبيل المثال يقوم البيان:

```
cout<<setprecision (2);
```

بتشكيل مخرجات الأعداد العشرية الى منزلتين عشريتين حتى يقوم بيان تالي مماثل بتغيير الضبط. لاحظ أن عدد المنزلات العشرية أو قيمة الضبط يتم تمريرها كمعطى الى `setprecision`. لاستخدام المتحكم `setprecision` يجب أن يتضمن البرنامج الملف الرئيسي `iomanip`. لهذا تكون هناك حاجة الى بيان `include` التالي:

```
#include <iomanip>
```

:Fixed

لمزيد من التحكم في مخرجات الأعداد ذات العلامة العشرية يمكنك استخدام أدوات تحكم أخرى. لإخراج الأعداد ذات العلامة العشرية بصيغة عشرية معينة تقوم باستخدام أداة التحكم `fixed`. البيان التالي يحدد مخرجات الأعداد ذات العلامة العشرية بصيغة عشرية ثابتة على جهاز الإخراج القياسي:

```
cout<<fixed;
```

بعد تنفيذ البيان السابق يتم عرض جميع الأعداد ذات العلامة العشرية بالصيغة العشرية الثابتة حتى يتم إيقاف تشغيل أداة التحكم `fixed`. يمكنك إيقاف تشغيل أداة التحكم `fixed` عن طريق استخدام دالة عنصر التدفق `unsetf`. على سبيل المثال يمكنك استخدام البيان التالي لإيقاف عمل أداة التحكم `fixed` على جهاز الإخراج القياسي:

```
cout . unsetf (ios : : fixed);
```

بعد إيقاف عمل أداة التحكم `fixed` تعود الأعداد ذات العلامة العشرية الى ضبطها الافتراضي. يتم استخدام أداة التحكم `scientific` لإخراج الأعداد ذات العلامة العشرية في صيغة علمية.

:Showpoint

افترض أن الجزء العشري من العدد العشري هو صفر. في هذه الحالة عندما تقوم بتوجيه الحاسوب الى إخراج العدد العشري في صيغة عشرية ثابتة قد لا تقوم المخرجات بتوضيح العلامة العشرية والجزء العشري. لإجبار المخرجات على توضيح العلامة العشرية والأصفار التالية تقوم باستخدام أداة

التحكم showpoint. البيان التالي يحدد مخرجات الأعداد العشرية مع العلامة العشرية والأصفار التالية على جهاز الإخراج القياسي.

```
cout<<showpoint;
```

بالطبع يقوم البيان التالي بتحديد مخرجات الأعداد ذات العلامة العشرية في صيغة عشرية معينة مع العلامة العشرية والأصفار التالية على جهاز الإخراج القياسي:

```
cout<<fixed<<showpoint;
```

:setw

يتم استخدام أداة التحكم setw لإخراج تعبير في أعمدة محددة وقد تكون قيمة التعبير اما مقطع أم عدد. البيان (n) setw يخرج قيمة التعبير التالي في عدد n من الأعمدة وتكون المخرجات على اليمين. لهذا اذا قمت بتحديد عدد الأعمدة عند 8 على سبيل المثال والمخرجات تحتاج أربعة أعمدة فقط تكون الأربعة أعمدة الأولى فارغة من اليسار. فضلاً عن هذا اذا كان عدد الأعمدة المحدد أقل من عدد الأعمدة المطلوبة للمخرجات تمتد المخرجات تلقائياً الى العدد المطلوب من الأعمدة ولا يتم بتر المخرجات. على سبيل المثال اذا كانت x متغير من النوع int يقوم البيان التالي بإخراج قيمة x في خمسة أعمدة على جهاز الإخراج القياسي:

```
cout<<setw (5) <<x<<endl;
```

لاستخدام أداة التحكم setw يجب أن يشمل البرنامج الملف الرئيسي iomanip. لهذا تكون هناك حاجة الى بيان التضمين التالي:

```
#include <iomanip>
```

setw يقوم بالتحكم في التعبير التالي فقط وهو بهذا يختلف عن setprecision الذي يتحكم في مخرجات جميع الأعداد ذات العلامة العشرية حتى إيقافه

أدوات التحكم left و right:

تذكر أنه اذا كان عدد الأعمدة المحدد بواسطة أداة التحكم setw يتجاوز عدد الأعمدة المطلوب بواسطة التعبير التالي يتم وضع المخرجات على اليمين. قد تريد في بعض الأحيان أن تكون المخرجات على اليسار. لضبط المخرجات على اليسار تستخدم أداة التحكم left. التركيب الخاص بضبط أداة التحكم left هو:

```
ostreamVar<<left;
```

حيث ostreamVar متغير تدفق المخرجات. على سبيل المثال يقوم البيان التالي بضبط المخرجات لكي تكون على اليسار على جهاز الإخراج القياسي:

```
cout<<left;
```

يمكنك إيقاف أداة التحكم left عن طريق استخدام دالة التدفق unsetf. التركيب الخاص بإيقاف أداة التحكم left هو:

```
ostreamVar.unsetf(ios::left);
```

حيث ostreamVar متغير تدفق المخرجات. إيقاف أداة التحكم left يقوم بإنتاج المخرجات في ضبط صيغة المخرجات الافتراضية. على سبيل المثال يقوم البيان التالي بإيقاف أداة التحكم left على جهاز الإخراج القياسي:

```
cout . unsetf (ios : : left);
```

تركيب ضبط أداة التحكم right هو:

```
ostreamVar<<right;
```

حيث يكون ostreamVar متغير تدفق المخرجات. على سبيل المثال يقوم البيان التالي بضبط المخرجات على اليمين على جهاز الإخراج القياسي:

```
cout<<right;
```

:flush

كل من أداة التحكم end1 وتتابع هروب السطر الجديد /n يقوم بوضع نقطة الإدخال في بداية السطر التالي على جهاز الإخراج. بالرغم من هذا يوجد استخدام آخر لأداة التحكم end1.

عندما يقوم برنامج بإرسال المخرجات إلى جهاز إخراج تذهب لمخرجات أولاً إلى الفاصل في الحاسوب. عندما يمتلئ الفاصل يتم إرسال المخرجات إلى جهاز الإخراج. بالرغم من هذا وبمجرد مواجهة أداة التحكم end1 يتم إرسال المخرجات من الفاصل إلى جهاز الإخراج على الفور حتى إذا لم يكن الفاصل ممتلئاً. لهذا تقوم أداة التحكم end1 بوضع نقطة الإدخال في بداية السطر التالي على جهاز الإخراج وتساعد على إخلاء الفاصل.

قد يكون من الممكن في بعض الأحيان أن ترى المخرجات بأكملها وهذا يرجع إلى حقيقة أنه عندما يتوقف البرنامج قد لا يكون الفاصل ممتلئاً في هذا الوقت.

في لغة C++ يمكنك استخدام أداة التحكم flush لإخلاء الفاصل حتى إذا لم يكن الفاصل ممتلئاً. على عكس أداة التحكم end1 لا تقوم أداة التحكم flush بنقل نقطة الإدخال إلى بداية السطر التالي.

تركيب استخدام أداة التحكم flush هو:

```
ostreamVar<<flush;
```

حيث ostreamVar متغير تدفق مخرجات مثل cout.

على سبيل المثال يقوم البيان التالي بإرسال المخرجات من الفاصل إلى جهاز الإخراج القياسي:

```
cout<<flush;
```

انظر إلى البيانات التالية التي يكون فيها num متغير من النوع int:

```
cout<<"Enter an integer: "; //Line 1
cin>>num; //Line 2
cout<<endl; //Line 3
```

البيان في السطر 1 يخرج النص التالي: ادخل عدد صحيح. بعد اخراج هذا السطر تظل نقطة الادخال واقعة بعد المسافة الواقعة بعد النقطتين. تذكر أن مخرجات البيان في السطر 1 تذهب الى الفاصل. اذا لم يكن الفاصل ممثلاً قد لا يتم عرض هذا السطر من النص وفي هذه الحالة قد لا يكون للمستخدم أية فكرة عما يفعله بعد هذا. يمكنك وضع أداة التحكم endl في نهاية البيان في السطر 1. بالرغم من هذا تكون نقطة الادخال واقعة عند بداية السطر التالي بعد طباعة سطر النص. بهذا يتم حث المستخدم على ادخال العدد في السطر الحالي والذي لا يكون جذاب للغاية في بعض الأحيان. على الجانب الآخر افترض أن البيان في السطر 1 يتم استبداله بالبيان التالي:

```
cout<<"Enter an integer: "<<flush; //Line 1
```

في هذه الحالة يتم عرض السطر "ادخل عدد صحيح" على جهاز الاخراج القياسي حتى اذا لم يكن الفاصل ممثلاً. فضلاً عن هذا وبعد اخراج سطر النص تبقى نقطة الادخال واقعة بعد المسافة بعد النقطتين: يقوم المستخدم بادخال العدد بعد المسافة بعد النقطتين.

ادخال / اخراج الملفات:

الأقسام التالية ناقشت كيفية الادخال من لوحة المفاتيح (جهاز الادخال القياسي) وارسال المخرجات الى الشاشة (جهاز الاخراج القياسي). هذا القسم يناقش كيفية الحصول على بيانات من أجهزة ادخال أخرى مثل الأقراص (أي المخزن التالي) وكيفية حفظ المخرجات على قرص. لغة C++ تسمح للبرنامج بالحصول على البيانات مباشرةً من وحفظ المخرجات مباشرةً على مخزون فرعي. يمكن للبرنامج استخدام ادخال / اخراج الملفات وقراءة البيانات من أو كتابة البيانات في ملف ما رسمياً يتم تعريف الملف كما يلي:

الملف: مساحة في مخزن فرعي يتم استخدامها لكي تضم المعلومات.

ملف الادخال / الاخراج الرئيسي القياسي iostream يحتوي على أنواع بيانات ومتغيرات يتم استخدامها فقط للادخال من جهاز الادخال القياسي وللإخراج الى جهاز الاخراج القياسي. بالإضافة الى هذا تقدم C++ ملف رئيسي يسمى fstream يتم استخدامه لادخال / اخراج الملف. ان الملف الرئيسي fstream يحتوي من ضمن أشياء أخرى على تعريفات نوعين من البيانات: ifstream الذي يعني تدفق ملف الادخال وهو مماثل لنوع البيانات istream والنوع الثاني ofstream الذي يعني تدفق ملف الاخراج وهو مماثل لنوع البيانات ostream.

لقد تم بالفعل تعريف المتغيرين cin وcout وربطهما بأجهزة الادخال / الاخراج القياسية. بالإضافة الى هذا يمكن استخدام >> مع cin وأدوات التحكم التي تم وصفها في القسم السابق يمكن استخدامها

مع cout. هذه العوامل نفسها متاحة لادخال / اخراج الملف ولكن الملف الرئيسي fstream لا يعلن متغيرات لاستخدامها. يجب عليك اعلان متغيرات تسمى **أهداف تدفق الملف** التي تتضمن المتغيرات ifstream للادخال ومتغيرات ofstream للاخراج. يمكنك بعد هذا استخدام هذه المتغيرات معاً ومع << و >> للادخال / الاخراج. تذكر أن C++ لا تقوم تلقائياً بتهيئة المتغيرات التي يحددها المستخدم. بمجرد أن تعلن الأهداف fstream يجب أن تربط هذه الأهداف بمصادر الادخال / الاخراج.

ادخال / اخراج ملف عملية مكونة من خمس خطوات:

1. تضمين الملف الرئيسي fstream في البرنامج.
2. اعلان أهداف تدفق الملف.
3. ربط أهداف تدفق الملف بمصادر الادخال / الاخراج.
4. استخدام أهداف تدفق الملف مع << أو >> أو دالات ادخال / اخراج أخرى.
5. اغلاق الملفات.

نقوم الآن بوصف هذه الخطوات الخمس بالتفصيل. بعد هذا يقوم برنامج تخطيطي بتوضيح كيفية ظهور الخطوات في برنامج ما. الخطوة الأولى تتطلب أن يكون الملف الرئيسي fstream موجود في البرنامج. البيان التالي يحقق هذه المهمة:

```
# include <fstream>
```

الخطوة الثانية تتطلب منك اعلان أهداف تدفق الملف. انظر الى البيانات التالية:

```
ifstream inData;
```

```
ofstream outData;
```

البيان الأول يعلن أن inData هدف ifstream والبيان الثاني يعلن أن outData هدف ofstream. الخطوة الثالثة تتطلب منك ربط أهداف تدفق الملف بمصادر الادخال / الاخراج وتسمى هذه الخطوة **فتح الملفات**. يتم استخدام دالة عنصر التدفق open لفتح الملفات. التركيب العام لفتح ملف ما هو:

```
fileStreamVariable.open(sourceName);
```

هنا fileStreamVariable عبارة عن هدف تدفق ملف ويكون sourceName هو اسم ملف الادخال / الاخراج.

افترض أنك قمت بادخال الاعلانات من الخطوة 2 في برنامج ما. افترض كذلك أن بيانات الادخال مخزنة في ملف يسمى prog.dat على قرص مرن في الجزء A وأنك تريد حفظ المخرجات في ملف يسمى prog.out على قرص مرن في الجزء A. البيانات التالية تربط inData مع prog.dat وتربط

outData مع prog.out. هذا يعني أن الملف prog.dat يتم فتحه من أجل بيانات الإدخال والملف prog.out يتم فتحه من أجل بيانات الإخراج.

```
inData.open("a:\\prog.dat"); //open the input file
outData.open("a:\\prog.out"); //open the output file
```

لاحظ أن هناك اثنان من العلامة \ بعد a: حيث أن العلامة \ هي رمز الهروب. لهذا ومن أجل إنتاج a\ ضمن مقطع ما تحتاج إلى \\. فضلاً عن هذا إذا كان برنامجك C++ وملف المدخلات موجودان في نفس الجزء فإنك إذن تحتاج إلى إدخال \\ a: قبل اسم الملف. بالمثل إذا كنت تريد أن يتم حفظ ملف المخرجات في نفس الجزء مثل الجزء الخاص ببرنامج C++ يمكنك حذف \\ a: قبل اسم الملف.

الخطوة الرابعة من المعتاد أن تعمل كما يلي. تستخدم أهداف تدفق العنصر مع << أو >> أو دالات الإدخال / الإخراج الأخرى. تركيب استخدام << أو >> مع أهداف تدفق الملف هو نفسه تركيب استخدام cin و cout. بالرغم من هذا فإنك تقوم بدلاً من استخدام cin و cout باستخدام أسماء أهداف تدفق الملف التي تم إعلانها. على سبيل المثال يقوم البيان:

```
inData >> payRate;
```

بقراءة البيانات من الملف prog.dat ويقوم بحفظها في المتغير payRate. البيان:

```
outData <<"The paycheck is: $" <<pay<<endl;
```

يقوم بحفظ المخرجات – شيك السداد هو: \$ 565.78 - في الملف prog.out. هذا البيان يفترض أنه تم حساب السداد على أنه 565.78.

بمجرد أن يكتمل الإدخال / الإخراج تتطلب الخطوة الخامسة إغلاق الملفات. غلق الملف يعني أنه يتم فك ارتباط متغيرات تدفق الملف عن منطقة التخزين ويتم تحرير أهداف تدفق الملف. بمجرد تحرير هذه المتغيرات يمكن إعادة استخدامها من أجل إدخال / إخراج ملف آخر. فضلاً عن هذا يضمن غلق ملف المخرجات أنه تم إرسال كامل المخرجات إلى الملف أي أنه تم إخلاء الفاصل. إنك تقوم بغلق الملفات باستخدام دالة التدفق close. على سبيل وبافتراض أن البرنامج يتضمن الإعلانات المذكورة في الخطوتين 2 و 3 تكون بيانات غلق الملفات كما يلي:

```
inData.close ( );
```

```
outData.close ( );
```

في الصيغة التخطيطية يكون البرنامج الذي يستخدم ادخال / اخراج ملف بالصيغة التالية:

```
#include <fstream>

//Add any additional header files that you use

using namespace std;

int main()
{
    //Declare file stream variables such as the following
    ifstream inData;
    ofstream outData;

    //Additional variable declaration

    //Open the files
    inData.open("a:\\prog.dat"); //open the input file
    outData.open("a:\\prog.out"); //open the output file

    //Code for data manipulation

    //Close the files
    inData.close();
    outData.close();

    return 0;
}
```

الخطوة الثالثة تتطلب فتح الملف من أجل ادخال / اخراج ملف. يربط فتح الملف متغير تدفق الملف الذي تم اعلانه في البرنامج بملف مادي عند المصدر مثل القرص. في حالة ملف الادخال يجب أن يتواجد الملف قبل تنفيذ البيان open. اذا لم يتواجد الملف يفشل البيان open ويدخل تدفق الادخال في حالة الفشل. ليس من اللازم أن يتواجد ملف المخرجات قبل فتحه حيث أنه اذا لم يتواجد ملف المخرجات يمكن للحاسوب اعداد ملف فارغ من أجل المخرجات. اذا كان ملف المخرجات المحدد موجود بالفعل يتم افتراضياً مسح المحتويات القديمة عند فتح الملف.

بنيات التحكم:

لغتي C++ والجافا تمتلكان نفس العوامل المترابطة الستة وهي ==، و !=، و <، و <=، و >، و >= وتعمل بنفس الطريقة في كلتا اللغتين وبنيات التحكم في كل من C++ والجافا واحدة. على سبيل المثال بنيات التحكم في الاختيار هي if و if...else و switch وبنيات التحكم في الدوران هي while و for و do...while. تركيب بنيات التحكم هذه واحد في اللغتين وبالرغم من هذا توجد بعض الاختلافات. في C++ تتم معامل أي قيمة غير صفرية على أنها صحيحة true وتتم معامل القيمة صفر على أنها خطأ false. تتم تهيئة الكلمة المحفوظة true عند 1 وتتم تهيئة الكلمة المحفوظة false عند صفر. يتم تقييم التعبيرات المنطقية في C++ عند صفر أو 1 وعلى الجانب الآخر يتم تقييم التعبيرات المنطقية في الجافا عند صحيح أو خطأ.

بالإضافة إلى هذا لا يمكن طرح كتابة نوع البيانات boolean في الجافا إلى نوع عددي ولهذا لا يمكن طرح كتابة قيمه true و false إلى قيم عددية.

في C++ يمكن أن يتسبب مزج عامل الاسناد وعامل التساوي في تعبير منطقي في مشكلات خطيرة. على سبيل المثال انظر إلى البيان if التالي:

```
if (drivingCode = 5)
```

...

في C++ يقوم البيان drivingCode = 5 بانتاج القيمة 5 وبما أن القيمة 5 غير صفرية اذن يتم تقييم التعبير على أنه صحيح true. لهذا في C++ يتم تقييم التعبير على أنه صحيح وتتغير كذلك قيمة المتغير drivingCode. على الجانب الآخر في الجافا لا يمكن طرح القيمة 5 عند true أو false لأنها ليست قيمة boolean. لهذا يتسبب البيان السابق في لغة الجافا في خطأ في المجمع بينما لا يتسبب في لغة C++ في أية أخطاء تركيبية.

الأسماء:

عندما يتم تضمين ملف رئيسي مثل iostream في برنامج ما تصبح المحددات العامة في الملف الرئيسي محدّدات عامة في البرنامج كذلك. لهذا اذا كان هناك محدّد عام في برنامج ما يمتلك نفس الاسم الخاص بواحد من المحددات العامة في الملف الرئيسي يقوم المجمع بتوليد خطأ تركيبية (مثل "محدّد معاد التعريف"). يمكن حدوث نفس المشكلة اذا كان برنامج ما يستخدم مكتبات ثالثة. للتغلب على هذه المشكلة تبدأ أطراف ثالثة أسماء محدّداتها العامة برمز محدّد. فضلاً عن هذا يبدأ المجمعون أسماء محدّداتهم العامة بعلامة (_). لهذا ومن أجل تجنب أخطاء التوصيل يجب ألا تبدأ أسماء المحدد في برنامجك بعلامة (_).

تحاول لغة C++ حل هذه المشكلة المتعلقة بتداخل أسماء المحددات العامة بآلية namespace.

التركيب العام للبيان namespace هو:

```
namespace namespaceName
{
    members
}
```

حيث يكون العنصر عادةً ثابت مسمى أو اعلان متغير أو دالة أو namespace آخر. لاحظ أن namespace عبارة عن محدّد C++.

في C++ تكون namespace كلمة محفوظة.

مثال ز-2:

البيان:

```

namespace globalType
{
    const int n = 10;
    const double rate = 7.50;

    int count = 0;
    void printResult();
}

```

يقوم بتعريف أن globalType عبارة عن namespace ذات أربعة عناصر: ثوابت مسماة n، و rate، والمتغير count، والدالة printResult.

مجال عنصر namespace محلي على namespace. يمكنك عادةً أن تتناول عنصر namespace خارج ال namespace بطريقة أو اثنتين كما هما موضحتين أدناه. التركيب العام لتناول عنصر namespace هو:

```
namespaceName::identifier
```

على سبيل المثال يكون البيان التالي ضرورياً لتناول العنصر rate من الاسم globalType:

```
globalType::printResult();
```

في C++ يطلق على : عامل حل المجال. لهذا تقوم باستخدام namespaceName يليه عامل حل المجال يليه اسم العنصر من أجل تناول عنصر من namespace. هذا يعني أنك تلحق اسم namespaceName وعامل حل المجال قبل اسم العنصر.

لتبسيط تناول عنصر namespace تقدم لغة C++ استخدام البيان using وتركيب استخدام البيان using يكون كما يلي:

(أ) لتبسيط تناول جميع عناصر namespace:

```
using namespace namespaceName;
```

(ب) لتبسيط تناول عنصر namespace محدد:

```
using namespaceName::identifier;
```

على سبيل المثال يقوم البيان:

```
using namespace globalType;
```

بتبسيط تناول جميع عناصر globalType namespace ويقوم البيان:

```
using globalType::rate;
```

بتبسيط تناول العنصر rate من namespace globalType.

في C++ تكون using كلمة محفوظة.

انك عادةً تقوم بوضع البيان using بعد اعلان namespace. بالنسبة الى globalType على سبيل المثال تقوم عادةً بكتابة الكود كما يلي:

```
namespace globalType
{
    const int n = 10;
    const double rate = 7.50;
    int count = 0;
    void printResult();
}

using namespace globalType;
```

بعد البيان using لا تحتاج الى وضع namespaceName وعامل حل المجال قبل عنصر namespace لتناول عنصر Namespace. بالرغم من هذا اذا كان عنصر namespace والمحدد العام في برنامج ما يملكان نفس الاسم يجب أن يسبق كل من namespaceName وعامل حل المجال عنصر الـ namespace من أجل تناول عنصر Namespace في البرنامج. بالمثل اذا كان عنصر namespace ومحدد ما في قالب ما يملكان نفس الاسم يجب أن يسبق كل من namespaceName وعامل حل المجال عنصر الـ namespace من أجل تناول عنصر namespace هذا في القالب. الأمثلة من ز-3 وحتى ز-6 تساعد على توضيح استخدام آلة namespace.

مثال ز-3:

انظر الى كود C++ التالي:

```
#include <iostream>

using namespace std;

.
.
.
int main()
{
    .
    .
    .
}

.
.
.
```

يمكنك في هذا المثال الاشارة الى المحددات العامة للملف الرئيسي ostream مثل cin وcout وendl دون استخدام البادئة : std قبل اسم المحدد. القيد الواضح هو أن القالب (أو الدالة) الذي يشير الى المحدد العام (الملف الرئيسي iostream) يجب ألا يحتوي على أي محدد له نفس اسم هذا المحدد العام.

مثال ز-4:

انظر الى كود C++ التالي:

```
#include <cmath>

int main()
{
    double x = 15.3;
    double y;

    y = std::pow(x, 2);
    .
    .
    .
}
```

هذا المثال يتناول الدالة pow للملف الرئيسي cmath.

مثال ز-5:

انظر الى كود C++ التالي:

```
#include <iostream>
.
.
.
int main()
{
    using namespace std;
    .
    .
    .
}
.
.
.
```

في هذا المثال يمكن أن تشير الدالة main الى المحددات العامة للملف الرئيسي iostream دون استخدام البادئة : std قبل اسم المحدد. يظهر البيان using داخل الدالة main ولهذا يجب أن تقوم دالات أخرى (إذا وجدت) باستخدام البادئة : std قبل اسم المحدد العام للملف الرئيسي iostream الا اذا كان الدالة لها بيان using مماثل.

مثال ز-6:

انظر الى كود C++ التالي:

```
#include <iostream>
using namespace std;           //Line 1

int t;                          //Line 2
double u;                       //Line 3

namespace expNspace
{
    int x;                      //Line 4
    char t;                     //Line 5
    double u;                   //Line 6
    void printResult();         //Line 7
}
using namespace expNspace;

int main()
{
    int one;                    //Line 8
    double t;                   //Line 9
    double three;               //Line 10

    *
    *
    *
}

void expNspace::printResult() //definition of the function
                             //printResult
{
    *
    *
    *
}
```

في برنامج لغة C++ هذا:

- للإشارة الى المتغير t في السطر 2 في main استخدم عامل حل المجال (أي قم بالإشارة الى t على أنها t :) لأن الدالة main لها متغير يسمى t (تم اعلانه في السطر 5).
- للإشارة الى العنصر t (المعلن في السطر 5) من namespace expNspace في main استخدم البادئة : expNspace مع t (أي قم بالإشارة الى t على أنها t : expNspace) لأن هناك متغير عام يسمى t (تم اعلانه في السطر 2) و متغير يسمى t في main.
- للإشارة الى العنصر u (المعلن في السطر 6) من namespace expNspace في main استخدم البادئة : expNspace مع u (أي قم بالإشارة الى u على أنها u : expNspace) لأن هناك متغير عام يسمى u (تم اعلانه في السطر 3).

- يمكنك الإشارة الى العنصر x (المعلن في السطر 4) من namespace exp في main اما على أنه x أو x : : expNspace لأنه لا يوجد محدد عام مسمى x والدالة main لا تحتوي على أي محدد يسمى x.
 - عادةً يتم كتابة تعريف الدالة التي تكون عنصر من namespace مثل printResult خارج الnamespace كما تم في البرنامج السابق. لكتابة تعريف الدالة printResult يمكن أن يكون اسم الدالة في عنوان الدالة اما printResult أو printResult : : expNspace (لأنه لا يوجد محدد عام آخر يسمى printResult).
- المحددات الموجودة في الملفات الرئيسية التي يوفرها النظام مثل iostream و cmath و iomanip يتم تعريفها في namespace std. لهذا السبب ومن أجل تبسيط تناول المحددات من تلك الملفات الرئيسي قمنا باستخدام البيان التالي في البرامج التي نكتبها:
- ```
using namespace std;
```

### الدالات والمعاملات:

الدالات في لغة الجافا تسمى أساليب ولغة C++ بها نوعان من الدالات – دالة منتجة لقيمة و void.

### الدالات المنتجة للقيمة:

تركيب الدالة المنتجة لقيمة ما يكون:

```
functionType functionName(formal parameter list)
{
 statements
}
```

في هذا القالب التركيبي يكون functionType (نوع الدالة) هو نوع القيمة التي تنتجها الدالة ويعرف هذا النوع كذلك بنوع بيانات الدالة المنتجة للقيمة. فضلاً عن فان البيانات الموضوعه بين أقواس مجمدة تقوم بتكوين هيكل الدالة.

### التركيب: قائمة المعامل الرسمي:

التركيب العام لقائمة المعامل الرسمي هو:

```
dataType identifier, dataType identifier,...
```

### التركيب: استدعاء الداله:

تركيب استدعاء دالة منتجة للقيمة يكون:

```
functionName(actual parameter list)
```

### التركيب: قائمة المعاملات الحقيقية:

تركيب قائمة المعامل الحقيقي هو:

```
expression or variable, expression or variable, ...
```

لهذا تقوم من اجل استدعاء دالة منتجة للقيمة باستخدام اسمها مع المعاملات الحقيقية (اذا وجدت) في جملة اعتراضية.

قد تكون قائمة المعامل الرسمي للدالة فارغة وبالرغم من هذا اذا كانت قائمة المعامل الرسمي فارغة لا يزال هناك حاجة الى الجملة الاعتراضية.

الدالة المنتجة للقيمة تنتج قيمتها عبر البيان return.

### الدالات void:

تعريف الدالة void يكون تركيبه كما يلي:

```
void functionName(formal parameter list)
{
 statements
}
```

### التركيب: قائمة المعامل الرسمي:

قد تكون قائمة المعامل الرسمي فارغة واذا لم يكن المعامل الرسمي فارغاً اذن يكون لقائمة المعامل الرسمي التركيب التالي:

```
dataType& variable, dataType& variable,
```

يجب أن تقوم بتحديد كل من نوع البيانات واسم المتغير في قائمة المعامل الرسمي. الرمز & بعد نوع البيانات له معنى خاص حيث يتم استخدامه فقط من أجل معاملات رسمية محددة وتتم مناقشته لاحقاً في هذا الملحق.

### التركيب: استدعاء الدالة:

تركيب استدعاء الدالة يكون كما يلي:

```
functionName(actual parameter list);
```

### تركيب: قائمة المعامل الحقيقي:

تركيب قائمة المعامل الحقيقي يكون كما يلي:

```
expression or variable, expression or variable, ...
```

كما يحدث في الدالات المنتجة للقيمة يكون عدد المعاملات الرسمية مع أنواع بياناتها في استدعاء الدالة متوافق مع المعاملات الرسمية بالترتيب المعطى. المعاملات الحقيقية والرسمية بها توفق مع بعضها البعض. استدعاء الدالة يتسبب في تنفيذ هيكل الدالة التي يتم استدعائها. (تتم مناقشة الدالات ذات المعاملات الافتراضية لاحقاً في هذا الملحق.)

## مثال ز-7:

```
void funexp(int a, double b, char c, int& x)
{
 ...
}
```

الدالة funexp لها أربع معاملات.

---

بوجه عام يوجد نوعان من المعاملات الرسمية: **معاملات قيمة** و **معاملات اشارة**.  
**معامل القيمة**: معامل رسمي يتلقى نسخة من محتوى المعامل الحقيقي الموافق له.  
**معامل الاشارة**: معامل رسمي يتلقى موقع (عنوان الذاكرة) المعامل الحقيقي الموافق له.  
عندما تقوم بالحاق & بعد نوع البيانات في قائمة المعامل الرسمي للدالة يصبح المتغير التالي لنوع البيانات معامل اشارة.

## مثال ز-8:

```
void expfun(int one, int& two, char three, double& four);
```

الدالة expfun لها أربع معاملات:

- one وهو معامل قيمة من النوع int.
- two وهو معامل اشارة من النوع int.
- three وهو معامل قيمة من النوع char.
- four وهو معامل اشارة من النوع double.

---

من تعريف معاملات القيمة ينتج أنه اذا كان المعامل الرسمي معامل قيمة يتم نسخ قيمة المعامل الحقيقي الموافق له بداخل المعامل الحقيقي. هذا يعني أن معامل القيمة له نسخة خاصة به من البيانات. لهذا يقوم معامل القيمة أثناء تنفيذ البرنامج بالتحكم في البيانات المخزنة في موقع الذاكرة الخاص بها. بعد نسخ البيانات لا يكون لمعامل القيمة اتصال بالمعامل الحقيقي.  
على الجانب الآخر اذا كان المعامل الرسمي معامل اشارة فانه يستقبل عنوان المعامل الحقيقي الموافق له. هذا يعني أن معامل الاشارة يقوم بحفظ عنوان المعامل الحقيقي الموافق له.  
أثناء تنفيذ البرنامج للتحكم في البيانات يقوم العنوان المخزن في معامل الاشارة بتوجيهه الى مكان ذاكرة المعامل الحقيقي الموافق له. بمعنى آخر يقوم معامل الاشارة أثناء تنفيذ البرنامج بالتحكم في البيانات المخزنة في موقع ذاكرة المعامل الحقيقي الموافق له. أي تغييرات يقوم بها معامل الاشارة على بياناته تقوم على الفور بتغيير قيمة المعامل الحقيقي الموافق له.  
القيمة الثابتة لا يمكن تمريرها الى معامل اشارة.

في لغة الجافا يتم تمرير المعاملات بالقيمة فقط وهذا يعني أن المعامل الرسمي يستقبل نسخة من بيانات المعامل الحقيقي. لهذا اذا كان هناك معامل رسمي عبارة عن متغير من نوع البيانات الأولية اذن لا

يمكن تمرير قيمته خارج الدالة. على الجهة الأخرى افترض أن المعامل الرسمي متغير اشارة. بعد هذا تقوم كل من المعاملات الرسمية والحقيقية بالاشارة الى نفس الهدف. بما أن المعامل الرسمي يحتوي على عنوان الهدف الذي يقوم بتخزين البيانات اذن يمكن أن يقوم المعامل الرسمي بتغيير قيمة الهدف الحقيقي. لهذا اذا كان المعامل الرسمي في لغة الجافا متغير اشارة فانه يعمل كمعامل اشارة في لغة C++.

### معاملات الاشارة والدالات المنتجة للقيمة:

أثناء توضيح تركيب قائمة المعامل الرسمي للدالة المنتجة للقيمة قمنا باستخدام معاملات قيمة فقط. كما يمكنك استخدام معاملات اشارة في الدالة المنتجة للقيمة بالرغم من أن هذه الطريقة لا يوصى بها. من التعريف تقوم الدالة المنتجة للقيمة بانتاج قيمة واحدة وهذه القيمة يتم انتاجها عبر البيان return. اذا احتاجت الدالة الى انتاج أكثر من قيمة واحدة عليك تغييرها الى دالة void واستخدام معاملات الاشارة المناسبة لانتاج القيم.

### الدالات ذات المعاملات الافتراضية:

عندما يتم استدعاء دالة ما يجب أن يكون عدد المعاملات الحقيقية والرسمية متماثلاً. تقوم لغة C++ بتخفيف هذا الشرط بالنسبة الى الدالات ذات المعاملات الافتراضية. انك تقوم بتحديد قيمة المعامل الافتراضي عندما يظهر اسم الدالة لأول مرة مثلما يحدث في النموذج. بوجه عام تنطبق القواعد التالية على الدالات ذات المعاملات الافتراضية:

- اذا لم تقم بتحديد قيمة المعامل الافتراضي سوف يتم استخدام القيمة الافتراضية لهذا المعامل.
- جميع المعاملات الافتراضية يجب أن تكون المعاملات اليمنى من الدالة.
- افترض أن هناك دالة لها أكثر من معامل افتراضي. في استدعاء الدالة اذا لم يتم تحديد قيمة للمعامل الافتراضي يجب عليك حذف جميع المعطيات الى يمينها.
- قد تكون المعاملات الافتراضية قيم ثابتة أو متغيرات عامة أو استدعاءات للدالات.
- المستدعي يمتلك خيار تحديد قيمة بخلاف القيمة الافتراضية لأي معامل افتراضي.
- لا يمكنك تحديد قيمة ثابتة كقيمة افتراضية لمعامل اشارة.

انظر الى نموذج الدالة التالية:

```
void funcExp(int x, int y, double t, char z = 'A', int u = 67,
 char v = 'G', double w = 78.34);
```

الدالة funcExp لها سبع معاملات. المعاملات z و u و v و w هي المعاملات الافتراضية. اذا لم يتم تحديد قيمة للمعاملات z و u و v و w في استدعاء الدالة funcExp يتم استخدام قيمها الافتراضية.

افترض أن لديك البيانات التالية:

int a, b;

char ch;

double d;

استدعاءات الدالة التالية صحيحة:

1. funcExp (a, b, d);

2. funcExp (a, 15, 34.6, 'B', 87, ch);

3. funcExp (b, a, 14.56, 'D');

في البيان 1 يتم استخدام القيم الافتراضية z و u و v و w. في البيان 2 يتم استبدال القيمة الافتراضية z بواسطة 'B' ويتم استبدال القيمة الافتراضية u بواسطة 87 واستبدال القيمة الافتراضية v بواسطة قيمة ch ويتم استخدام القيمة الافتراضية w. في البيان رقم 3 يتم استبدال القيمة الافتراضية z بواسطة 'D' ويتم استخدام القيم الافتراضية u و v و w.

استدعاءات الدالة التالية غير صحيحة:

1. funcExp (a, 15, 34.6, 46.7);

2. funcExp (b, 25, 48.76, 'D', 4567, 78.34);

في البيان 1 يجب أن يتم حذف جميع القيم الافتراضية الأخرى لأنه تم حذف قيمة z. وفي البيان 2 يجب حذف قيمة w أيضاً لأنه تم حذف قيمة v.

ما يلي نماذج دالات ذات معاملات افتراضية غير صحيحة:

1. void funcOne(int x, double z = 23.45, char ch, int u = 45);

2. int funcTwo(int length = 1, int width, int height = 1);

3. void funcThree(int x, int& y = 16, double z = 34);

بما أن المعامل الثاني z في البيان 1 هو المعامل الافتراضي اذن يجب أن تكون جميع المعاملات الأخرى بعد z معاملات افتراضية كذلك وبما أن المعامل الأول في البيان 2 هو معامل افتراضي اذن يجب أن تكون جميع المعاملات القيم الافتراضية. في البيان رقم 3 لا يمكن تحديد قيمة ثابتة ل y لأن y معامل اشارة.

### المصفوفات:

في لغة C++ مثلها مثل الجافا تكون المصفوفة عبارة عن مجموعة من عدد ثابت من المكونات حيث تكون جميع المكونات من نفس نوع البيانات. بالرغم من هذا فان المصفوفات في C++ لا تكون أهداف ولهذا لا تحتاج الى أن يتم تعريفها. بعد هذا نقوم بوصف كيفية عمل المصفوفات الأحادية البعد في C++.

**المصفوفة الأحادية البعد** عبارة عن مصفوفة يتم فيها ترتيب المكونات في صيغة قائمة. الصيغة العامة لإعلان مصفوفة أحادية البعد هو:

```
dataType arrayName (intExp);
```

حيث intExp أي تعبير يتم تقييمه بعدد صحيح موجب. كما تقوم intExp بتحديد عدد المكونات في المصفوفة.

**مثال ز-9:**

البيان:

```
int num [5];
```

يعلن مصفوفة num من 5 مكونات وكل مكون يكون من النوع int. المكونات هي num [0] و num و num [1] و num [2] و num [3] و num [4].

---

### تناول مكونات المصفوفة:

في C++ يتم تناول مكونات المصفوفة مثلما يتم في لغة الجافا. الصيغة (التركيب) العامة المستخدمة من أجل تناول مكونات المصفوفة هي:

```
arrayName[indexExp]
```

حيث indexExp التي تسمى **المؤشر** تكون عبارة عن أي تعبير قيمته عدد صحيح غير سالب. قيمة المؤشر تحدد موضع المكون في المصفوفة. في C++ يبدأ مؤشر المصفوفة عند صفر. انظر الى البيان التالي:

```
int list [10];
```

هذا البيان يعلن مصفوفة list من 10 مكونات. المكونات هي list [0]، و list [1]، و...، و list [9]. بيان الاسناد:

```
list [5] = 34;
```

يقوم بحفظ 34 في list [5] وهو المكون السادس من المصفوفة list. افترض أن i متغير من النوع int. بعد هذا يكون بيان الاسناد:

```
list [3] = 63;
```

مكافئ لبيانات الاسناد:

```
i = 3;
```

```
list [1] = 63;
```

إذا كانت i تساوي 4 اذن بيان الاسناد التالي:

```
list [2 * i - 3] = 58;
```

يقوم بحفظ 58 في `list [5]` لأن  $2 * i - 3$  يتم تقييمها عند 5 يتم تقييم تعبير المؤشر أولاً معطياً موضع المكون في المصفوفة.

بعد هذا انظر الى البيانات التالية:

`list [3] = 10;`

`list [6] = 35;`

`list [5] = list [3] + list [6];`

البيان الأول يحفظ 10 في `list [3]` والبيان الثاني يحفظ 35 في `list [6]` والبيان الثالث يجمع محتويات `list [3]` و `list [6]` ويقوم بحفظ الناتج في `list [5]`.

### مؤشر المصفوفة خارج الحدود:

لسوء الحظ في C++ لا توجد حماية ضد المؤشرات الخارجة عن الحدود ولهذا لا تتحقق C++ مما اذا كانت قيمة المؤشر واقعة ضمن النطاق أم لا - أي بين صفر وحجم المصفوفة - 1. اذا خرج المؤشر عن الحدود وحاول البرنامج تناول المكون المحدد بواسطة المؤشر فان موقع الذاكرة المشار اليه بواسطة المؤشر يتم تناوله. من الممكن أن يتسبب هذا الموقف في تغيير أو تناول بيانات موقع ذاكرة لم تقصده أبداً. نتيجة لهذا اذا خرج المؤشر أثناء التنفيذ عن الحدود هناك عدة أشياء يمكن أن تحدث. التأكد من أن المؤشر واقع ضمن النطاق هي مسؤولية المبرمج وحده.

### المصفوفات كمعاملات للدالة:

في C++ يتم تمرير المصفوفات بالاشارة فقط. بما أن المصفوفات يتم تمريرها بالاشارة فقط اذن فانك لا تستخدم الرمز `&` عند اعلان المصفوفة كمعامل رسمي. عند اعلان مصفوفة أحادية البعد كمعامل رسمي يتم عادةً اهمال حجم المصفوفة. اذا قمت بتحديد حجم المصفوفة الأحادية البعد عند اعلانها كمعامل رسمي فسوف يتم اهماله بواسطة المجمع. في لغة الجافا يكون المتغير `length` (الطول) مرتبط بكل مصفوفة وهو يحدد حجم المصفوفة. بالرغم من هذا لا يوجد متغير كهذا مرتبط بمصفوفات C++. لتمرير حجم المصفوفة لدالة ما نقوم باستخدام معامل آخر كما يحدث في الدالة التالية:

```
void initialize(int list[], int size)
{
 int count;
 for(count = 0; count < size; count++)
 list[count] = 0;
}
```

المعامل الأول للدالة `initialize` هو مصفوفة `int` من أي حجم. عندما يتم استدعاء الدالة `initialize` يتم تمرير حجم المصفوفة الفعلي كمعامل ثاني للدالة `initialize`.

عندما يكون المعامل الرسمي معامل اشارة اذن في أي وقت يتغير فيه المعامل الرسمي يتغير المعامل الحقيقي كذلك. مع هذا وبالرغم من أن المصفوفة يتم دائماً تمريرها بالاشارة فانه لا يزال يمكنك منع

الدالة من تغيير المعامل الحقيقي. انك تقوم بهذا عن طريق استخدام الكلمة المحفوظة const في اعلان المعامل الرسمي. انظر الى الدالة التالية:

```
void example(int x[], const int y[], int sizeX, int sizeY)
{
 ...
}
```

هنا يمكن للدالة example تعديل المصفوفة x وليس المصفوفة y. محاولة تغيير y تتسبب في خطأ في زمن التجميع. اذا لم تكن تريد أن تقوم الدالة بتعديل المصفوفة قم باعلان مصفوفة ثابتة كمعامل رسمي وهذه سوف يكون تدريب برمجة جيد.

## الملحق

### (ح)

## المراجع

1. جي بوتش، التحليل والتصميم القائم على الهدف، الطبعة الثانية، دراسة أديسون- ويسلي، شهادة الماجستير، 1995.
2. اي هورويتز، واس ساهني، واس راجاسيكاران، خوارزميات الحاسوب C++، صحافة علوم الحاسب الآلي، نيويورك، 1997.
3. آر جونسونباو، رياضيات منفصلة، الطبعة الخامسة، برينتس هول، ابرساديل ريفر، ان جيه، 2001.
4. ان ام جوسوتيس، مكتبة C++ المعيارية. درس تعليمي ومرجع، دراسة أديسون- ويسلي، شهادة الماجستير، 1999.
5. دي اي كنوث، فن برمجة الحاسوب، المجلدات 1-3، دراسة أديسون- ويسلي، شهادة الماجستير، 1973، 1969، 1973.
6. اس بي ليبمان، وجيه لاجوي، كتاب C++ تمهيدي، الطبعة الثالثة. دراسة أديسون- ويسلي، شهادة الماجستير، 1998.
7. دي اس ماليك، برمجة C++: من تحليل المشكلة الى تصميم البرنامج، كورس تكنولوجي، بوسطون، 2002.
8. اي ام راينجولد، ودبليو جيه هensen، بنيات البيانات في الباسكال. لتل براون وشرطاه، بوسطون، 1986.
9. آر سيدجويك، خوارزميات في C، الطبعة الثالثة. دراسة أديسون- ويسلي، شهادة الماجستير، الأجزاء 1-4، 1998، الجزء 5، 2002.

**الملحق**  
**(ط)**  
**أجوبة تمارين مختارة**

**الفصل 1:**

1. أ. صح، ب. خطأ، ت. خطأ، ث. خطأ، ج. خطأ، ح. صح، خ. خطأ، د. خطأ.  
2.

شرط مسبق: يجب ألا تكون قيمة  $x$  سالبة.  
شرط تالي: اذا لم تكن قيمة  $x$  سالبة تنتج الدالة الجذر التربيعي الموجب لـ  $x$   
وبخلاف هذا يتوقف البرنامج.

4. 12

5. أ. 43

ب.  $3 + n^4$

ج.  $O(n)$

7. `#define NDEBUB`

12. أ. 6

ب. 2

ج. 2

13. أ. (i) المقوم في السطر 1.

(ii) المقوم في السطر 3.

(iii) المقوم في السطر 4.

ب.

```
CC::CC()
{
 u = 0;
 v = 0;
}
```

ج.

```
CC::CC(int x)
{
 u = x;
 v = 0;
}
```

## الفصل 2:

1. أ. صح، ب. صح، ت. صح، ث. خطأ، ج. خطأ، ح. صح، خ. صح، د. خطأ، ذ. خطأ، ر. صح، ز. خطأ، س. صح، ش. خطأ، ص. خطأ.

4. العناصر الخاصة للفئة خاصة ولا يمكن تناولها بشكل مباشر بواسطة دالات عنصر الفئة المستمدة. العناصر المحمية للفئة الأساسية يمكن تناولها بشكل مباشر عن طريق دالة عنصر الفئة المستمدة.

5. أ. البيان:

class bClass public aClass

يجب أن يكون:

class bClass : public aClass

ب. فاصلة منقوطة غير موجودة بعد {.

7.

أ.

```
yClass::yClass()
{
 a = 0;
 b = 0;
}
```

ب.

```
xClass::xClass()
{
 z = 0;
}
```

ج.

```
void yClass::two(int u, int v)
{
 a = u;
 b = v;
}
```

11.

في القاعدة:  $7 = x$

في المستمدة:  $x = 3$ ،  $y = 8$ ،  $x + y = 11$

13. لأن المعامل الأيسر ل << عبارة عن هدف تدفق وهو ليس من النوع mystery.

14. واحد.

16.

أ. friend strange operator+ (const strange&, const strange&);

ب. friend bool operator= (const strange&, const strange&);

ج. friend strange operator++ (strange&, int);

18. في السطر 2 كلمة friend قبل الكلمة bool غير موجودة. البيان الصحيح هو:

friend bool operator<= (mystery, mystery) ; // Line 2

20. لا يوجد.

23. اثنان.

24. خطأ في السطر 4. تمثيل القالب يمكن أن يكون فقط اما لنوع مدمج أو نوع محدد بواسطة

المستخدم. الكلمة type بين أقواس الزاوية يجب استبدالها اما بنوع مدمج أو نوع يحدده المستخدم.

26. (أ) 12 (ب) Sunny Day

27. (أ) 21 (ب) OneHow

28.

```
template<class Type>
void swap(Type &x, Type &y)
{
 Type temp;
 temp = x;
 x = y;
 y = temp;
}
```

### الفصل 3:

1. أ. خطأ، ب. خطأ، ت. خطأ، ث. صح، ج. صح، ح. صح، خ. خطأ، د. خطأ.

2. أ. صحيح

ب. صحيح

ت. غير صحيح حيث أن p متغير مؤشر و x متغير int. لا يمكن تحديد قيمة x ل p.

ث. صحيح

ج. صحيح.

ح. غير صحيح حيث أن \*p متغير int و q متغير مؤشر. لا يمكن تحديد قيمة q ل \*p.

5. ب، ج

8. البيان في السطر 5 ينسخ قيمة p داخل q. بعد تنفيذ هذا البيان تشير كل من p و q الى موقع الذاكرة ذاته. البيان في السطر 7 يزيل تخصيص موقع الذاكرة المشار اليه بواسطة q والذي بدوره يجعل كل من p و q غير صحيحين. لهذا تكون القيم المطبوعة بواسطة البيان في السطر 8 غير متوقعة.

40 32 25 19 14 10 7 5 4 4.9

12. البيان في السطر 7 ينسخ قيمة p داخل q. بعد تنفيذ هذا البيان تشير كل من p و q الى نفس المصفوفة. البيان في السطر 8 يزيل تخصيص موقع الذاكرة وهو المصفوفة المشار اليها بواسطة p والذي بدوره يجعل q غير صحيحة. لهذا تكون القيم المطبوعة بواسطة البيان في السطر 10 غير متوقعة.

13. المصفوفة p: 25 17 11 7 5

المصفوفة q: 5 7 11 17 25

16. الفئات ذات عناصر بيانات المؤشر يجب أن تتضمن المدمر وتقوم بإثقال عامل الاسناد وتوفر مقوم النسخ عن طريق تضمينه في تعريف الفئة وتقديم تعريفه.

18.

أ. البيان يصنع هدف arrayListType يسمى intList حجمه 100. عناصر intList تكون من النوع int.

ب. البيان يصنع هدف arrayListType يسمى stringList حجمه 1000. عناصر stringList تكون من النوع string.

ت. البيان يصنع هدف arrayListType يسمى salesList حجمه 100. عناصر salesList تكون من النوع double.

#### الفصل 4:

3. `vector<double> doubleList(50);`

5. `ostream_iterator<int> screen(cout, " ");`

6. انها تنتج المخرجات التالية:

13 11 9 7 5

8 6 4 2 0.7

9 7 3.9

95 200 100 75 50.11

210 35 5 23 0 34 76 70.15

## الفصل 5:

1. أ. خطأ، ب. خطأ، ت. خطأ، ث. خطأ، ج. صح، ح. صح
3. أ. صح، ب. صح، ت. خطأ، ث. خطأ، ج. صح.
4. أ. صحيح
- ب. صحيح
- ت. صحيح
- ث. غير صحيح (B مؤشر بينما \*list عبارة عن struct)
- ج. صحيح
- ح. غير صحيح (B مؤشر بينما A->link->info عبارة عن int)
- خ. صحيح
- د. صحيح
- ذ. صحيح
6. هذه حلقة غير محدودة تقوم باستمرار بطباعة 18.
7. أ. هذا الكود غير صحيح. البيان `s->info = B;` غير صحيح لأن B مؤشر و `s->info` عبارة عن `int`.
- ب. هذا الكود غير صحيح. بعد تنفيذ البيان `s = s->info` تكون s تساوي NULL وبهذا لا تكون `s->info` موجودة.
9. 28 20 42 30
- 10.
- العنصر المراد حذفه غير موجود في القائمة.
- 25 45 15 2 38 18
12. `intList = {3, 23, 43, 56, 11, 23, 25}`
- 14.
- 50 35 23 45 0 34
- 138 23 45 38 0 34 76 46

## الفصل 6:

1. أ. صح، ب. صح، ت. خطأ، ث. خطأ، ج. خطأ.
4. يطلق على الدالة تكرارية مباشرة إذا كانت تستدعي نفسها.
5. الدالة التي تستدعي دالة أخرى وتتسبب في النهاية في استدعاء الدالة الأصلية يقال أنها تكرارية بشكل غير مباشر.

6. الدالة التكرارية التي يكون فيها البيان الأخير الذي يتم تنفيذه هو الاستدعاء التكراري تسمى دالة تكرارية ذيلية.

9. أ. لا تنتج أية مخرجات.

ب. لا تنتج أية مخرجات.

ت. 5 6 7 8 9

ث. لا تنتج أية مخرجات.

10. أ. 15

ب. 6

12. افترض أن low تحدد مؤشر البداية وأن high تحدد مؤشر النهاية في المصفوفة. سوف يتم عكس عناصر المصفوفة بين low و high.

```
if(low < high)
{
 a. swap(list[low], list[high])
 b. reverse elements between low + 1 and high - 1
}
```

14.  $C(5,3) = 10$ ;  $C(9,4) = 126$

### الفصل 7:

1.  $3 = x$

$9 = y$

7

13

4

7

2.  $45 = x$

$23 = x$

$5 = x$

عناصر المصفوفة: 5 14 34 10

3. أ. 26

ب. 45

ت. 8

ث. 29

4. أ.  $AB+CD+*E-$

ب.  $ABC+D*-EF / +$

ت.  $AB+CD- / E+F*G-$

ث.  $ABCD+*+EF / G*-H+$

Winter Spring Summer Fall Cold Warm Hot .6

50 40 30 20 10 .7

## الفصل 8:

1. أ. 0 = queueRear ، 50 = queueFront

ب. 99 = queueRear ، 51 = queueFront

3. أ. 76 = queueRear ، 25 = queueFront

ب. 75 = queueRear ، 26 = queueFront

51 .5

7. عناصر الطابور: 2 4 16 9 5

5 16 5 .9

10. الدالة mystery تعكس عناصر الطابور كما تضاعف قيم عناصر الطابور.

13.

```
template<class Type>
void reverseQueue(queueType<Type> &q, stackType<Type> &s)
{
 Type elem;

 while(!q.isEmptyQueue())
 {
 elem = q.front();
 q.deleteQueue();
 s.push(elem);
 }

 while(!s.isEmptyStack())
 {
 elem = s.top();
 s.pop();
 q.addQueue(elem);
 }
}
```

## الفصل 9:

1. أ. خطأ، ب. صح، ت. خطأ، ث. خطأ.

2. أ. 8، ب. 6، ت. 1، ث. 8.

3.

ب.

| المكرر | first | last | mid | list [mid] | عدد المقارنات  |
|--------|-------|------|-----|------------|----------------|
| 1      | 0     | 10   | 5   | 55         | 2              |
| 2      | 0     | 4    | 2   | 17         | 2              |
| 3      | 3     | 4    | 3   | 45         | 2              |
| 4      | 4     | 4    | 4   | 49         | 1 (وجدت صحيحة) |

تم العثور على العنصر في الموضع 4 والعدد الكلي للمقارنات هو 7.

ت.

| المكرر | first | last | mid          | list [mid] | عدد المقارنات |
|--------|-------|------|--------------|------------|---------------|
| 1      | 0     | 10   | 5            | 55         | 2             |
| 2      | 6     | 10   | 8            | 92         | 2             |
| 3      | 9     | 10   | 9            | 98         | 2             |
| 4      | 10    | 10   | 10           | 110        | 2             |
| 5      | 11    | 10   | الحلقة تتوقف |            |               |

هذا البحث غير ناجح. إجمالي عدد المقارنات 8.

4.

```
template<class elemType>
class orderedArrayListType: public arrayListType<elemType>
{
public:
 int binarySearch(const elemType& item);
 orderedArrayListType(int n = 100);
};
```

5. يوجد 30 مقدار في جدول البعثة وكل مقدار يمكنه ضم 5 عناصر.

10. أ. يتم ادخال العنصر ذو المؤشر 15 عند HT [15] والعنصر ذو المؤشر 101 عند HT[0] والعنصر ذو المؤشر 116 عند HT [16] والعنصر ذو المؤشر صفر عند HT [1] والعنصر ذو المؤشر 217 عند HT [17].

101.11

16. عامل التحميل  $\alpha = 1001 / 850 \approx 0.85$

17. عامل التحميل  $\alpha = 1001 / 500 \approx 0.5$

أ.  $\{ 1 + (1 - \alpha) \} = 1.5$  ( 1/2 )

ت.  $(1 - \alpha / 2) = 1.25$

## الفصل: 10

1. 3

3. 10، 12، 18، 21، 25، 28، 30، 32، 58، 15

6. أ. 36، 38، 32، 16، 40، 28، 48، 80، 64، 95، 54، 100، 58، 65، 55

9. أثناء المرور الأول يتم عمل 6 مقارنات. بعد مرورين لخوارزمية الترتيب بالتكدس تكون القائمة:

85، 72، 82، 47، 65، 50، 76، 30، 20، 60، 28، 25، 45، 17، 35، 14، 94،

100

10.

```
template<class elemType>
class orderedArrayListType: public arrayListType<elemType>
{
public:
 int binarySearch(const elemType& item);
 void insertOrd(const elemType&);

 void selectionSort();
 void insertionSort();
 void quickSort();

 orderedArrayListType(int size = 100);

private:
 void recQuickSort(int first, int last);
 int partition(int first, int last);
 void swap(int first, int second);
 int minLocation(int first, int last);
};
```

## الفصل 11:

1. أ. خطأ، ب. صح، ت. خطأ، ث. خطأ.

3.  $L_A = \{B, C, D, E\}$

4.  $R_A = \{F, G\}$

5.  $R_B = \{E\}$

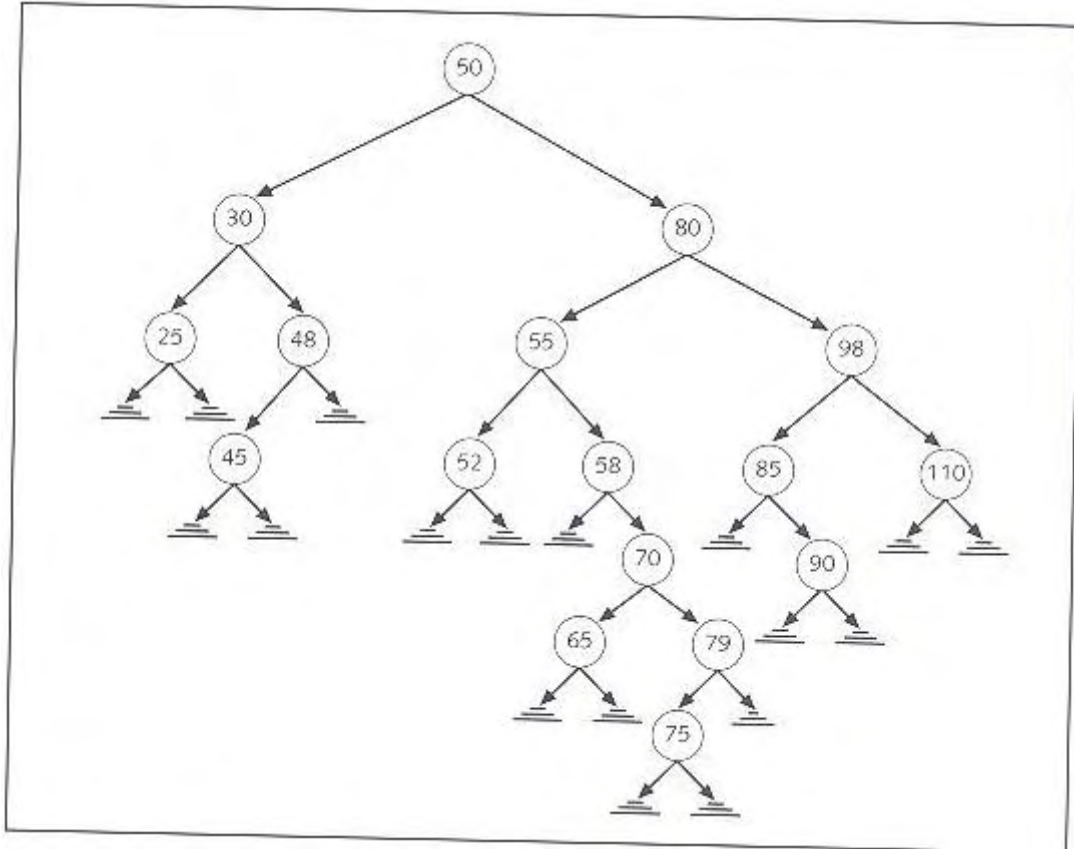
6. D C B E A F G

7. A B C D E F G

8. D C E B G F A

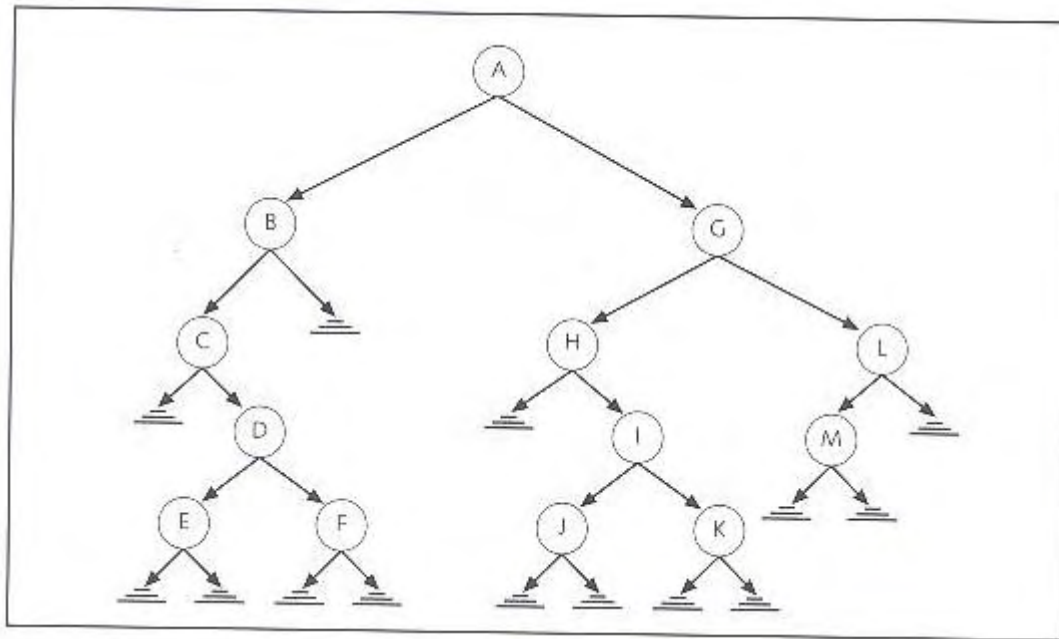
9. 79 - 70 - 58 - 55 - 80

12.



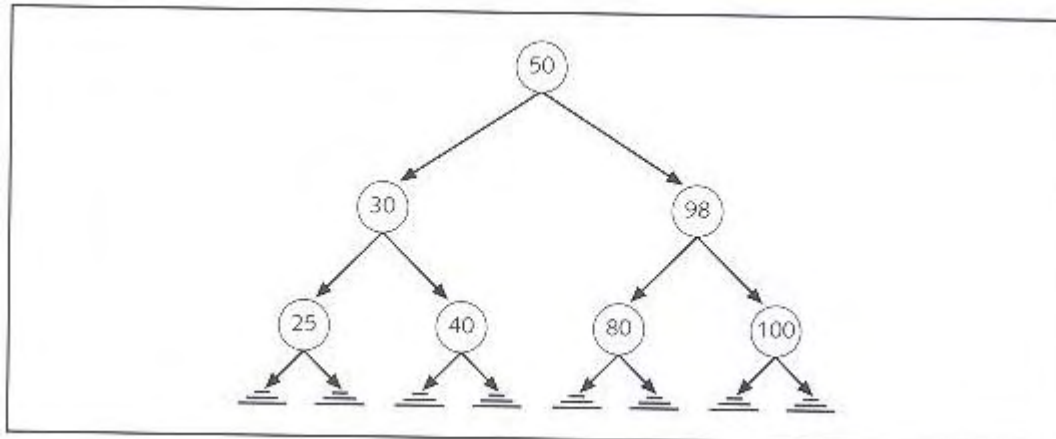
شكل ط - 1

15.



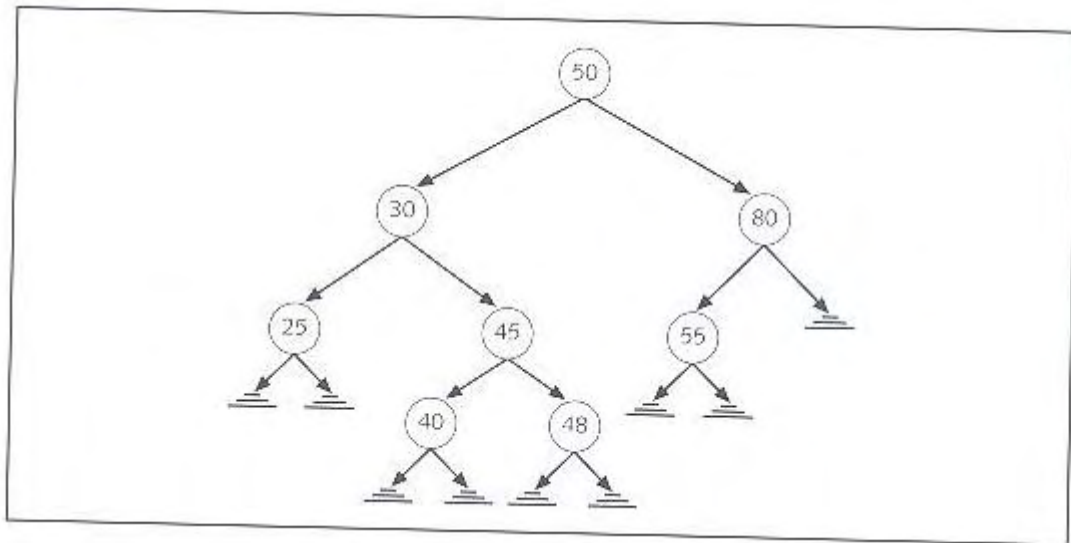
شكل ط - 2

17. عامل اتزان العقدة الجذرية هو صفر.



شكل ط - 3

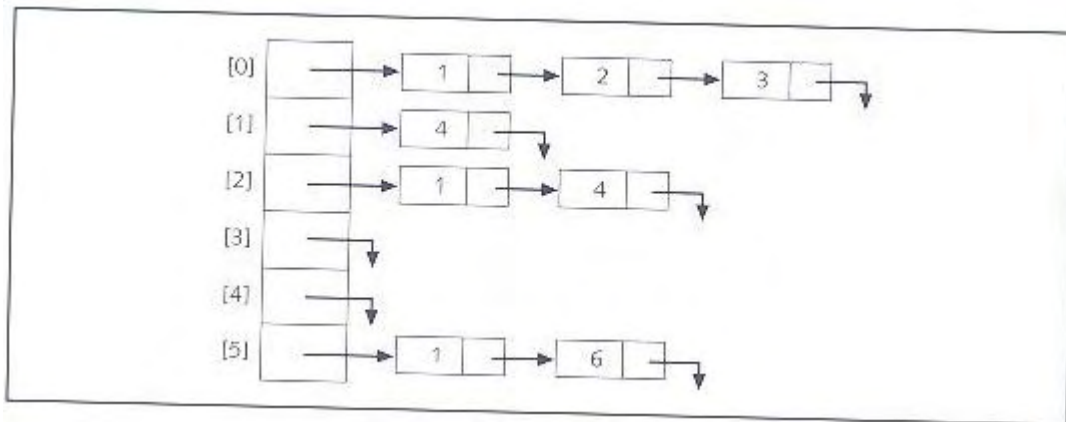
18. عامل اتزان العقدة الجذرية هو -1.



شكل ط - 4

## الفصل 12:

2.

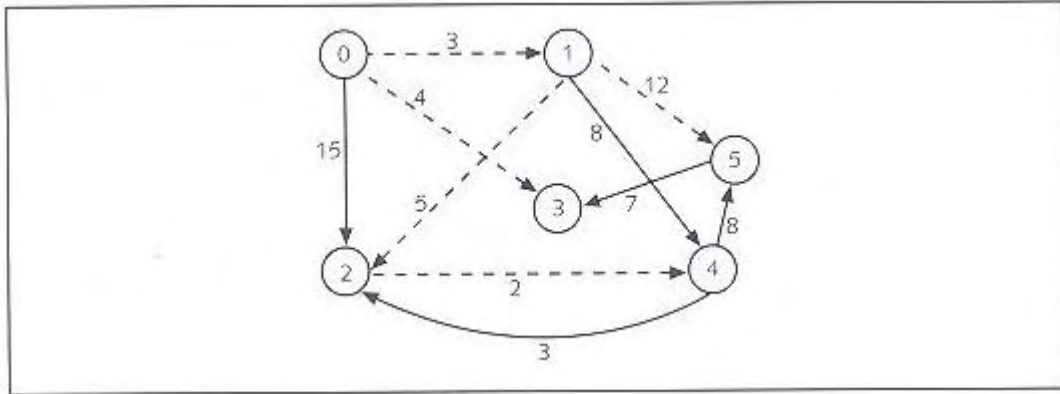


شكل ط - 5

5 3 2 4 1 0.3

5 4 3 2 1 0.4

6.



شكل ط - 6

قمة المصدر: صفر

أقصر مسافة من المصدر الى كل قمة

القمة      أقصر\_مسافة

0      0

3      1

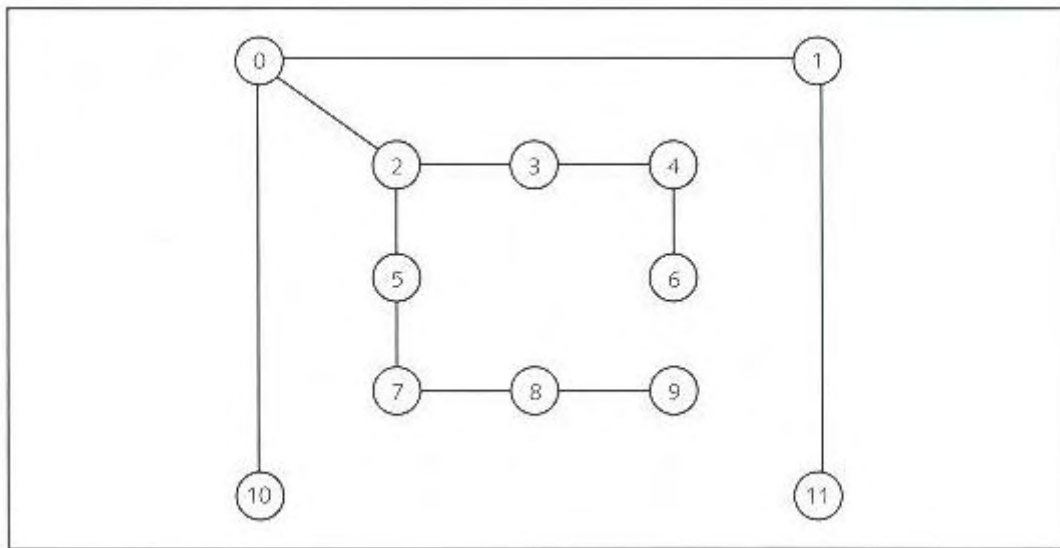
8      2

4      3

10      4

15      5

7.



شكل ط - 7

9, 3, 8, 4, 5, 6, 2, 0, 7, 1, 10.

### الفصل 13:

cout<<temp.first<<" "<<temp.second<<endl; .2 ب.

.3 Duckey Donald

map<string, string> stateDataMap; .5 أ.

.ج

```
map<string, string>::iterator mapItr;
cout<<left;
cout<<"The elements of stateDataMap:"<<endl;
for(mapItr = stateDataMap.begin();
 mapItr != stateDataMap.end(); mapItr++)
 cout<<setw(15)<<mapItr->first
 <<setw(15)<<mapItr->second<<endl;
cout<<endl;
```

A B A K .8

11. صفر

## فهرس

### ملاحظة:

- أرقام الصفحات التي تليها كلمة def تشير الى تعريفات وأوصاف مصاحبة لها.
- أرقام الصفحات التي تليها كلمة tbl تشير الى جداول.

### الرموز:

& (و) عنوان المعامل. انظر عنوان المعامل

رمز معامل الاشارة

153، 896، 897، 901

&& (علامات و): وعامل 849 tbl.

<> (أقواس زاوية)

محددات ملف النظام الرئيسي، 41، 77

انظر أيضاً عامل الاستخلاص (<<)، والعامل "أكبر من" (<)، وعامل الادخال (>>)، والعامل "أصغر من" (>).

\* (النجمة). انظر عامل ازالة الاشارة وعامل الضرب.

\* = (نجمة – علامة يساوي): عامل اسناد مركب، 849 جدول.

| (خط): عامل فاصل، 849 جدول.

|| (خطوط): عوامل أو، 849 جدول.

{ } (أقواس): كمحددات للدالة، 878

[ ] (أقواس). انظر عامل مؤشر المصفوفة

: (نقطتان): عامل محدد تناول العنصر، 17

: : (أكثر من نقطتان): عامل حل المجال، 26، 849 جدول، 891

, (فاصلة): عامل تتابع، 849 جدول.

. (نقطة). انظر عامل تناول الهنصر (عامل النقطة)

" (علامة تنصيص مزدوجة): محدد الملف الرئيسي المعرف بواسطة المستخدم، 41، 77

= (علامة يساوي) عامل اسناد. انظر عامل تعبير postfix لعامل الاسناد، 446

== (علامات يساوي). انظر عامل التساوي.

! (علامة التعجب): عامل لا، 849 جدول.

! = (علامة تعجب – علامة يساوي). انظر عامل عدم التساوي.

< (علامة "أكبر من"). انظر عامل "أكبر من"

< = (علامة "أكبر من" – علامة يساوي). انظر عامل "أكبر من أو يساوي".

<< (علامات "أكبر من"). انظر عامل الاستخلاص.

- > (واصلة – قوس زاوية أيمن): سهم عامل تناول العنصر، 144-145، 849 جدول.
- > (علامة "أصغر من"). انظر عامل "أصغر من"
- > = (علامة "أصغر من" – علامة يساوي). انظر عامل "أصغر من أو يساوي".
- >> (علامات "أصغر من"). انظر عامل الادخال.
- (علامة ناقص) عامل النقص، 849 جدول. عامل الطرح. انظر عامل الطرح.
- رمز لغة التشكيل الموحدة ، 20
- (علامات ناقص). انظر عامل الانقاص.
- # (علامة العدد) تعبير postfix رمز عددي، 446. رمز موجه قبل المعالجة 877
- رمز لغة التشكيل الموحدة 20
- ( ) (جملة اعتراضية). انظر الجمل الاعتراضية.
- % (علامة النسبة المئوية): العامل (المتبقي)، 849 جدول.
- + (علامة زائد). انظر عامل الجمع.
- عامل زائد، 849 جدول.
- رمز لغة التشكيل الموحدة 20
- ++ (علامات زائد). انظر عامل الزيادة.
- += (زائد – علامة يساوي): عامل اسناد مركب، 849 جدول.
- ?: (علامة استفهام – نقطتان): عامل شرطي 849 جدول.
- ; (فاصلة منقوطة) في تعريفات الفئة 16 ليست في موجهات قبل المعالجة 877
- / (شرطة مائلة). انظر عامل القسمة.
- / = (شرطة مائلة – علامة يساوي): عامل اسناد مركب 849 جدول.
- (مسافة) (رمز مسافة خالية): عامل استخلاص 880.
- (تلدة): بادئية اسم المدمر، 35.
- \_ (الخط التحتي): رمز المحدد، 890

#### الأعداد:

- صفر: مؤشر باطل، 147، 860.
- لغز الأربع ملكات، 392 – 394
- لغز الثماني ملكات، 391، 395-395

## A

- دالات القيمة المطلقة، 859 جدول.
- أنواع البيانات المجردة، 37 تعريف
- تعريف، 37
- تعريف الشجرات الثنائية 644-646، شجرات AVL 698
- شجرات البحث 654-655
- تعريف نماذج الفئة من أجل 112-114
- تعريف الفئات 38-39.
- الفئة customerType 497-498
- لغز العدد n من الملكات 396-396.
- فئة queueType 508.
- فئة serverListType 503-504.
- فئة serverType 499-501
- فئة videoType 339-341، 701-729
- تعريف الرسوم البيانية 728-729، 735-736.
- تعريف البعثة 547-549.
- تعريف القوائم 37-38، 38-39.
- القوائم المبنية على مصفوفات 182-185، 524-526.
- القوائم المرتبة المبنية على مصفوفات 529.
- لأنواع مختلفة من العناصر 112-114.
- القوائم المتصلة 290-293.
- القوائم المتصلة من الطرفين 319-321.
- القوائم المتصلة المرتبة 308.
- تعريف الطوابير 478-480
- الطوابير المتصلة 484-485.
- تعريف شجرات الامتداد 752
- تعريف المرصوبات كمرصوبات متصلة 429-430، 439
- المرصوبات كمصفوفات 411-413.
- القوائم المتصلة ك 290-307
- التجريد 36-37.
- تجريد البيانات 37 تعريف.

محدددي التناول. انظر محدددي تناول العنصر

تناول عناصر المصفوفة 900-901.

موارد مكتبة C++. انظر استخدام (تناول) موارد مكتبة C++

عناصر الفئة 21-22، 24، 144-145.

عناصر البيانات الخاصة بالفئة الأساسية في الفئات المستمدة 68، 70، 872-874.

عناصر struct، 144-145

الدالة accumulate (مسيقى التعريف) 835، 836-837.

الدالة (x) acos (مسبقة التعريف) 859 جدول.

قوائم المعاملات الحقيقية

الدالات المنتجة للقيمة 896

دالات void 896.

معاملات حقيقية: منع الدالة من التغير 25، 902.

المكيفات. انظر مكيفات الحاوية

اضافة عناصر:

الى جداول البعثرة 550

الى الطوابير 470، 471، كمصفوفات 471-472، 482.

طوابير متصلة 487، 508، 509.

الى مرصوصات 409-410، 411، كمصفوفات 415-418

كقوائم متصلة 433-436.

انظر كذلك ادخال عناصر

عامل الجمع (+)

الانقال 97-98، 107.

الأسبقية 849 جدول.

دالة (عملية) اضافة طابور 470، 471

للطوابير المتصلة 487، 508، 509

للطوابير كمصفوفات 471-472، 482.

عنوان العامل (&) 135، 137-138، 139.

عمل أسماء وهمية للأهداف 872

الأسبقية 849 جدول.

انتاج ناوين عناصر البيانات الخاصة 872-874.

عنوان (موقع ذاكرة) التخزين في المؤشرات 135-143.

تمثيلات الرسم البياني لقائمة التجاور 724-725، 726.

تمثيلات الرسم البياني لقالب التجاور 723-723.

قمم التجاور (في الرسم البياني) 723، 726.

الدالة adjacent\_difference (مسبقة التعريف) 835-837.

الدالة adjacent\_find (مسبقة التعريف) 792، 815-816.

أنواع البيانات المجردة. انظر خوارزميات أنواع البيانات المجردة 218

تحليل 8-15

خوارزميات التتبع الخلفي 391-398، 399.

عام. انظر خوارزميات مكتبة القالب المعياري

مقارنات المفاتيح في. انظر مقارنات المفاتيح (في الخوارزميات)

عمليات في: الاحصاء 11-12

عملية هيمنة 12

ترتيب. انظر كؤشر التركيب الزمني  $O(g(n))$

خوارزميات تكرارية 369، 372-373

الاستخدامات 218.

انظر كذلك خوارزميات البحث وخوارزميات الترتيب وخوارزميات محددة أخرى.

أسماء وهمية للأهداف: العمل 872

تخصيص الذاكرة. انظر تخصيص الذاكرة.

الكود المعياري الأمريكي لتبادل المعلومات. انظر مجموعة رموزه.

و (&) عنوان المعامل. انظر عنوان المعامل

رمز معامل الإشارة

153، 896، 897

علامات و (&&): عامل الواو 849 جدول.

تحليل (تحليل متطلبات البرنامج) 3.

العامل واو (&&):

الأسبقية 849 جدول.

أقواس زاوية (<>)

محددات ملف النظام الرئيسي، 41، 77

انظر أيضاً عامل الاستخلاص (<<)، والعامل "أكبر من" (<)، وعامل الإدخال (>>)، والعامل "أصغر من" (>).

الشجرات الثنائية الكيفية 644، 652.

دالات جيب التمام، وجيب الزاوية، وظل الزاوية 859 جدول.

حسابي:

حساب المؤشر 151، 207.

انظر كذلك عمليات حسابية.

تعبيرات حسابية 876.

تعبيرات infix 441 مقابل تعبيرات postfix 442 جدول.

الجمل الاعتراضية في: 441

الأسبقية 849 جدول.

Postfix. انظر التعبيرات postfix

انظر كذلك عوامل حسابية

عمليات حسابية

خوارزميات رقمية 788، 835-839.

على المؤشرات 151.  
انظر كذلك عوامل حسابية، دالات الاحصاء.  
عوامل حسابية 876.  
الانتقال 9699.  
في التعبيرات postfix 441، 446.  
الأسبقية 849 جدول.  
انظر كذلك العمليات الحسابية  
أهداف دالة مكتبة القالب المعياري الحسابية 789-791.  
عناصر المصفوفة (مصفوفة أحادية البعد)  
التناول 900-901.  
الايجاد 800-802  
الظهور المترابط المتعاقب 792، 815-816.  
الأكبر / الأصغر (انظر أكبر عنصر:ايجاد، واصغر عنصر: ايجاد).  
الحذف 802-806  
الاستبدال 806-808.  
عامل مؤشر المصفوفة ( [ ] )  
الانتقال 172-173، 177.  
الأسبقية 849 جدول.  
مؤشرات المصفوفة 900.  
تحديد قيم ل 900-901.  
ايجاد مؤشر أكبر عنصر 565.  
ايجاد مؤشر أصغر عنصر 560-563.  
كخروج عن الحدود 901.  
محتويات التبادل 561-563، 585  
القوائم المبنية على مصفوفات 180-198.  
شجرات ثنائية من 595-602  
التحقق مما اذا كانت فارغة / ممتلئة 185.  
دالات عنصر الفئة 185-190.  
المقوم 190.  
التحويل الى تكدسات 596-602.  
مقوم النسخ 190-191.  
النسخ 794-795.  
التعريف كنوع بيانات مجرد 182-185، 524-526.  
المدمر 190.  
ادخال عناصر في 186-188، 193-194.  
الحدود 273، 631.  
انتقال عامل الاسناد 191.  
التقسيم 580-585.  
العنصر pivot 580-582، 583، 584، 585.  
مثال برمجة 198-206.

ازالة عناصر من 188-189، 190، 194.  
 استبدال عناصر في 189  
 استرجاع عناصر من 189  
 البحث 191-193.  
 الترتيب: الترتيب بالتكديس 596-605،  
 الترتيب بالادخال 566-572.  
 الترتيب السريع 579-586،  
 الترتيب بالاختيار 560-566.  
 برنامج اختبار 195-198  
 الفئة ArrayListType  
 التعريف كنوع بيانات مجرد 182-185، 524-526.  
 دالات العنصر 185-190.  
 المصفوفات 900 تعريف، 900-902  
 الرموز. انظر مقاطع C كمقاطع دائرية 473-474.  
 مصفوفات ثابتة 902.  
 تم عملها أثناء تنفيذ البرنامج. انظر المصفوفات الحركية.  
 حركية. انظر المصفوفات الحركية.  
 العناصر. انظر عناصر المصفوفة.  
 المؤشرات. انظر مؤشرات المصفوفة.  
 التهيئة: مصفوفة أحادية البعد 901.  
 الحدود 273.  
 القوائم ك. انظر القوائم المبنية على مصفوفات 152  
 أحادية البعد. انظر القوائم الأحادية البعد.  
 التمرير كمعاملات للدالات 901-902.  
 المعالجة بالحلقة for 563، 596.  
 الطوابير. انظر الطوابير كمصفوفات  
 الحجم 901.  
 المرصوصات. انظر المرصوصات كمصفوفات  
 ساكنة مقابل حركية 151.  
 مجموعة رموز الكود المعياري الأمريكي لتبادل المعلومات 851 جدول.  
 رموز غير قابلة للطباعة 851-852 جدول.  
 الدالة asin (x) (مسبقة التعريف) 859 جدول.  
 الدالة assert (مسبقة التعريف) 857 جدول.  
 الاستخدام (التناول) 6.  
 بيانات assert 6.  
 الايقاف 857.  
 الدالات assign (مسبقة التعريف): عمليات الحاوية 234 جدول، 331 جدول.  
 تحديد قيم للمتغيرات 876، 877.  
 عامل الاسناد (=) 22  
 عوامل اسناد مركبة (عملية =) 876.  
 الأسبقية 849 جدول.  
 عمليات الحاوية 228 جدول.

والنسخ العميق 156-157  
مقابل عامل التساوي (=) 890  
الانثال 162-166، للقوائم المبنية على المصفوفات 191  
للشجرات الثنائية 651.  
للمقاطع C 176-177. للقوائم المتصلة 293.  
للطوابير المتصلة 488، للمرصوصات المتصلة 439.  
للمرصوصات كمصفوفات 422.  
الأسبقية 849 جدول.  
والنسخ السطحي 154-156، 159-161.  
بيانات الاسناد 876، 877  
لأهداف الفئة 22-24  
الحاويات المترابطة 774-785.  
اعلان وتهيئة:  
الحاويات map / multimap 780-782،  
الحاويات set / multiset 775-776.  
الملفات الرئيسية 785 جدول.  
كمفات مطبقة 774.  
ادخال وحذف عنصر: الحاويات map / multimap 782،  
الحاويات set / multiset 776-777.  
دعم المكررات 785 جدول.  
حاويات مسبقة التعريف 774.  
مراجعة سريعة 842.  
معيار الترتيب 774، 775.  
الاستخدام (التناول) 775  
انظر كذلك ارتباط الحاويات 849 جدول.  
النجمة (\*). انظر عامل ازالة الاشارة: عامل الضرب  
نجمة - علامة يساوي (=): عامل اسناد مركب 849 جدول.  
تقارب 14 تعريف.  
الدالة at (مسبقة التعريف): عملية حاوية 221 جدول، 234 جدول.  
الدالة atan (x) (مسبقة التعريف) 859 جدول.  
أهداف فئة آلية 24  
شجرات AVL (شجرات بحث ثنائية متزنة الارتفاع) 675-700، 676 تعريف.  
العمليات الأساسية 678.  
البناء 694-698.  
التعريف كنوع بيانات مجرد 698.  
حذف عناصر من 678، 699  
رسوم بيانية 677.  
ادخال عناصر في 678، 679-698.  
العقد. انظر العقد (الخاصة بأشجار AVL).  
تحليل الأداء 700.  
مراجعة سريعة 711.  
الدوران (اعادة التكوين) 685-692، أمثلة 681-682، 683، 685، 695، 696، 697-698،  
الدالات 690-692، الأنواع 685-690.

أوصاف النوع 678.

B

- الدالة back (قوائم متصلة) 296، 309.
- قوائم متصلة من الطرفين 324.
- الدالة back (مسبقة التعريف)
- عملية الحاوية 221 جدول، 234 جدول، 331 جدول.
- عمليات الطابور 492 جدول.
- الدالة back (الطوابير) 470، 471.
- الطوابير المتصلة 487، 488.
- الطوابير كمصفوفات 482.
- المكرر back\_inserter 795.
- خوارزميات التتبع الخلفي 391-398، 399.
- عامل اتزان (عقد شجرة AVL) 678 تعريف.
- الدالة balanceFromLeft 691-692.
- الدالة balanceFromRight 691، 692.
- الخط (|): عامل فاصل 849 جدول.
- الخطوط (||): العامل أو 849 جدول.
- الحالة الأساسية (للتعريفات التكرارية) 368.
- تعريفات مقومات الفئة الأساسية 72.
- الاستدعاء 71، 74-75.
- استدعاء دالات عنصر الفئة الأساسية 70.
- إعادة التعريف في الفئات المستمدة 69-70.
- عناصر الفئة الأساسية
- تناول عناصر البيانات الخاصة في الفئات المستمدة 68، 70، 872-874.
- الدالات. انظر دالات عنصر الفئة الأساسية.
- معاملات الفئة الأساسية: تمرير أهداف الفئة المستمدة الى 865-871.
- الفئة الأساسية 66 تعريف.
- المقومات: تعريفات 72.
- الاستدعاء 71، 74-75.
- التعريف 71-72.
- المدمرات 871.
- قواعد التوريث 67-68، 71.
- العناصر. انظر عناصر الفئة الأساسية.
- الفئة baseClass 69، 71-72.
- باوهر اي 391.
- الدالة begin (مسبقة التعريف):
- عملية الحاوية 224، 228 جدول.
- الدالة bfTopOrder 760-761.
- مكررات ثنائية الاتجاه 239.
- دالات Big-O 14-15، 15 جدول.
- أعداد ثنائية: تحويل الأعداد العشرية الى 387-389.
- انقال عوامل ثنائية 96-103.

أصول ثنائية (أهداف الدالة) 794.  
شجرات بحث ثنائية 651-667، 653-654 تعريف.  
الحاويات المترابطة كمطبقة مع 774  
AVL. انظر شجرات AVL.  
العمليات الأساسية 654.  
التعريف كنوع بيانات مجرد 654-655.  
حذف عناصر من 644، 658-666.  
توازن الارتفاع. انظر شجرات AVL.  
ادخال عناصر في 644، 656-658.  
طولي 666-667.  
أشجار شديدة التوازن 675-676 تعريف.  
تحليل الأداء 666-667.  
مراجعة سريعة 710-711.  
البحث 655-656.  
التركيب 652-653.  
بحث ثنائي (للقوائم المرتبة) 529-535.  
الخوارزميات 529-533، 535.  
الدالات 531، 812-813، 814.  
مقارنات المفاتيح 531، 532-533، 534-535، 537 جدول.  
الحد الأدنى (الترتيب) 537، 537 جدول.  
تحليل الأداء 533-535.  
مراجعة سريعة 552-553.  
انظر كذلك أشجار البحث الثنائية.  
اجتياز الأشجار الثنائية 639-640.  
اثقال الدالة 672-673.  
مخرجات تتابع العقدة 642.  
خوارزميات / دالات غير تكرارية 668-672.  
مراجعة سريعة 710.  
خوارزميات / دالات تكرارية 640-643.  
تحديث البيانات أثناء 672-675.  
الأشجار الثنائية 631-711، 632 تعريف.  
أشجار كيفية 644.  
من قوائم مبنية على مصفوفات 595-602.  
عمليات أساسية 643-644.  
الفروع 632.  
التحقق مما اذا كانت فارغة 647.  
مقوم النسخ 650.  
النسخ 638-639، 649.  
المقوم الافتراضي 647.  
التعريف كنوع بيانات مجرد 644-646.  
التدمير 650، 650-651.  
المدمر 650-651.  
الرسوم البيانية 632-637.

الرسم 632  
 العناصر 632-633، 637  
 الارتفاع 637-638  
 العقد. انظر عقد (الأشجار الثنائية)  
 المسارات 637  
 مراجعة سريعة 710-711  
 أشجار البحث. انظر أشجار البحث الثنائية  
 الأشجار الفرعية 632-633، 654  
 الاجتياز 639-643. انظر كذلك اجتياز الشجرة الثنائية  
 الدالة binary\_search (مسبقة التعريف) 811-815  
 الدالة binarySearch 5531  
 متضمنة في:  
 الفئة 537 orderedArrayListType  
 تعريف الفئة binaryTreeType كنوع بيانات مجرد 644-646  
 دالات العنصر 646-651  
 ربط (دالات العنصر) 867  
 اختبار الصندوق الأسود 7-8  
 الفراغ (رمز المسافة الخالية) ( ): عامل الاستخلاص و 880  
 نوع البيانات bool 876  
 تعبيرات منطقية 876، 889  
 عوامل منطقية: الأسبقية 849 جدول  
 قيم منطقية 876، 889، 890  
 عملية bOp 825  
 قيم الحدود 8  
 الأقواس ( { } ): كمحددات للدالة 878  
 الأقواس ( المربعة ) ( [ ] ). انظر عامل مؤشر المصفوفة.  
 أقواس زاوية (<>) محددات الملف الرئيسي للنظام 41، 77  
 انظر كذلك عامل الاستخلاص(<<) عامل "أكبر من"(<)  
 عامل الإدخال(>>) عامل "أصغر من"(>)  
 الأقواس المجددة. انظر الأقواس  
 فروع (الشجرة الثنائية) 632  
 ترتيب القمم الأول بالعرض 734  
 الاجتياز الأول بالعرض (للسوم البيانية) 734-736  
 الترتيب الطوبوغرافي (للقمم) 754-761  
 الدالة breadthFirstTraversal 734-735  
 تعريف الفئة bSearchTreeType كنوع بيانات مجرد 564-655  
 والفئة videoBinaryTree 703  
 الفواصل: تفرغ فاصل تدفق المخرجات 885-886  
 بناء خوارزمية التكدس 596-602  
 الدالة buildHeap 602  
 بناء أشجار AVL 694-698  
 بناء قوائم متصلة من الخلف 284، 288-289، 290  
 بناء قوائم متصلة من الأمام 284، 285-288، 290

الدالة buildListBackward (قوائم متصلة) 289، 290  
الدالة buildListForward (قوائم متصلة) 288، 290

## C

الدالة c\_str (مسبقة التعريف): الدالة str.c\_str 861 جدول  
دالات C-string: ملف رئيسي 860-861 جدول  
مقاطع C 174  
المقارنة 172، 860 جدول  
التسلسل 860 جدول  
النسخ 172، 861 جدول  
دالات التحكم 174-180  
تحديد الأطوال 861 جدول  
دالات العامل من أجل المعالجة 174-180  
خصائص C++ مقابل خصائص الجافا 875-902  
مكتبة C++. انظر المكتبات (مجموعة مكتبة C++)

ملفات برنامج C++: أسماء وامتداد الملف 878  
تصميم برامج C++ 3-4. انظر كذلك التصميم القائم على الهدف  
كود قابل للتنفيذ 878  
نماذج الدالة. انظر نماذج الدالة  
كود الهدف (برنامج الهدف) 878  
دموجات ما قبل المعالجة. انظر موجهات ما قبل المعالجة  
كود المصدر (برنامج المصدر) 878  
التركيب 878-879  
التركيب. انظر التركيب  
الاختبار. انظر برامج الاختبار  
انظر كذلك برامج لغة المستوى الرفيع  
الدالة callPrint 866-871  
الاستدعاءات. انظر استدعاءات الدالة  
مثال برمجة ماكينة الحلوى 45-55  
الدالة capacity (مسبقة التعريف): عملية الحاوية vector 225 جدول  
الحالة: تحويل الرموز الى كبير / صغير 858-859 جدول  
الملف الرئيسي cassert 6، 857 جدول  
تعريف الفئة cAssignmentOprOverLoad 164  
دالات العنصر 164-166

عامل الطرح 876

أسبقية static\_cast 849 جدول

الملف الرئيسي ctype 857-859 جدول

الدالة (x) ceil (مسبقة التعريف) 859 جدول

الهدف 5 cerr

التسلسل (البعثة المفتوحة) 540 تعريف، 551-594

تحليل الأداء 554

نوع البيانات char 875، 876

انظر كذلك المتغيرات char

نوع بيانات المتغيرات char 875، 876

ادخال (قراءة) البيانات بداخل 880-881

اخراج القيم 883 جدول

مخرجات صحيحة 881 جدول

انظر كذلك المتغيرات

مصفوفات الرموز. انظر مقاطع C

دالات الرمز (الملف الرئيسي ctype) 857-859 جدول.

مجموعات الرموز (برامج الترميز)

مجموعة الكود المعياري الأمريكي لتبادل المعلومات 854 جدول رموز غير قابلة للطباعة 851-852

جدول

مجموعة كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد 853 جدول

مصفوفات. انظر مقاطع C

التحقق من قيم الرمز 857-859 جدول

التحويل الى الوضع الصغير / الكبير 858-859 جدول

الادخال (القراءة) 880-881

الادخال في مقاطع 862-863 جدول

الاخراج 883 جدول

الاستبدال في المقاطع 863 جدول

كمقاطع. انظر مقاطع C (النوع string)

الدالات المنتجة للقيمة 857-859 جدول

انظر كذلك المتغيرات char

مجموعات الرموز

التحقق مما اذا كانت الاشجار الثنائية فارغة 647

التحقق مما اذا كانت القوائم فارغة / ممتلئة

القوائم المبنية على مصفوفات 185

القوائم المتصلة 293، القوائم المتصلة من الطرفين 321

التحقق مما اذا كانت الطوابير فارغة / ممتلئة 470

الطوابير المتصلة 486  
الطوابير كمصفوفات 481  
التحقق مما اذا كانت المرصوصات ممتلئة / فارغة 410  
مرصوصات متصلة 430، 433  
المرصوصات كمصفوفات 415  
عقد ثمار (الأشجار الثنائية) 632-633، 636  
بيانات cin 879-881، 887  
فشل الادخال 881-882  
التركيب 879  
قوائم متصلة دائرية 336-337  
طوابير دائرية كمصفوفات 473-474  
الفئة (كلمة محفوظة) 16  
عمل تعريفات الفئة 16-20، 25-26، 31-33، 36-37، 43-44  
نماذج الدالة في 16، 25-26  
الملفات الرئيسية 40-41، 114، 307  
تضمين مقوم النسخ في 169-172  
تضمين المدمرات في 159  
دالات العامل 88، 95-96، 103  
الوضع 40، 114  
بيانات عنصر الفئة. انظر عناصر بيانات الفئة  
دالات عنصر الفئة. انظر دالات عنصر (عناصر دالة) الفئة  
عناصر الفئة 16  
فئات التناول 16-17، 19، 67، 79-80  
التناول 21-22، 24 عبر المؤشرات 144-145  
عناصر البيانات. انظر عناصر بيانات (الفئات)  
الاعلان 16-17، 19  
عناصر الدالة. انظر دالات العنصر (عناصر دالة) (الفئات)  
قواعد التوريث 67-68، 71  
عمل عام أو خاص 17  
تخصيص ذاكرة 21  
عناصر الهدف. انظر عناصر هدف الفئات  
خاص. انظر عناصر الفئة الخاصة.  
محمي. انظر عناصر الفئة المحمية.  
عام. انظر عناصر الفئة العامة  
المجال 24  
الأنواع 16، 80  
عناصر المتغير. انظر عناصر بيانات (الفئات)  
أهداف الفئة (متغيرات الفئة) 15، 20  
الأسماء الوهمية 872  
عمليات مدمجة على 22  
نسخ القيم 22-24، 159-162، 166-172  
البيانات من أجل 4 العمليات على (انظر العمليات (على البيانات)).  
الاعلان 20-21، 24، 34، 166

أهداف الفئة المستمدة 71  
التهيئة الافتراضية للعنصر 166  
في الدالات 24  
مؤشر خفي الى. انظر المؤشر this  
التحديد 4، 15، 56-55  
اخفاء المعلومات 26، 40-44  
التهيئة عند الاعلان 166  
ادخال البيانات الى 55  
العوامل المطبقة على 22، 86، 87  
انقال عوامل من أجل. انظر انقال العامل  
أزواج. انظر الأزواج (عناصر مزدوجة)  
التمرير كمعاملات للدالات 24-25، 167-168  
الاشارة الى الكل 88-89  
انتاج قيمة 89-90  
المجال 24  
تفاصيل التحديد: الاخفاء 40-44  
مجال الفئة 24  
قوالب الفئة 110 تعريف، 112-114  
التعريف 112-113  
عناصر الدالة. انظر دالات العنصر (عناصر الدالات) (لقوالب الفئة)  
عناصر الدالة كقوالب للدالة 114  
أهداف الدالة 788، 789-794  
وملفات التطبيق 114  
تمثيل 114، 727-728  
مراجعة سريعة 116-117  
قوالب مكتبة القالب المعياري 217  
التركيب 112  
انظر كذلك الحاويات والمكررات  
متغيرات الفئة. انظر أهداف الفئة  
الفئات 4، 15-20، 16 تعريف  
عنوان العامل (و) 872-874  
الأساس. انظر الفئة الأساسية  
فئات مركبة 80-85  
العمل من فئات متواجدة. انظر التوريث  
العمل من قوالب 112-114  
التعريف 16-20، 25-26، 31-33، 36-37، 43-44 كنوع بيانات مجرد 38-39 (انظر كذلك انواع  
البيانات قبل التجريد)  
تعريفات. انظر تعريفات الفئة  
المستمدة. انظر الفئات المستمدة  
الدالات الصديقة 92-95  
الملفات الرئيسية 40-41  
التحديد 55-56  
أمثلة. انظر أهداف الفئة

عقد القائمة المتصلة 275  
كقوائم متصلة. انظر القوائم المتصلة  
عناصر. انظر عناصر الفئة  
انقال العامل. انظر انقال العامل  
بعناصر بيانات المؤشر: الحدود (الخصوصيات) 157-172، 646  
مؤشرات الى 144-147  
مثال برمجة 45-55  
كطوابير. انظر الطوابير  
مراجعة سريعة 57-58  
العلاقات بين. انظر التكوين: التوريث  
كمرصوعات. انظر المرصوعات مقابل البنيات 144  
التركيب 16  
القولب. انظر قولب الفئة  
المتغيرات. انظر أهداف الفئة  
الفئة classIllusFriend 92-95  
دالات clear (مسبقة التعريف)  
عمليات الحاوية 222 جدول، 228 جدول، 229 جدول، 777 جدول، 782 جدول  
الدالة str.clear 862 جدول  
الدالة clearGraph 730  
تفريغ الرسوم 730  
فاصل تدفق المخرجات 885-886  
المقاطع 862 جدول  
الدالة clearList 190  
مقومات الفئة clockType 31-33، 35  
التعريف 17-20، 25-26، 31-33، 36-37  
التعزيزات 86  
الملف الرئيسي 41  
ملف التطبيق 42  
الدالة close (مسبقة التعريف) 888  
البثرة المغلقة. انظر المعالجة المفتوحة  
ملفات الاغلاق 888  
التجميع (في جداول البعثرة)  
التجميع الأولي 541-542  
التحقيق 543، 545  
التجميع الثانوي 545  
الملف الرئيسي cmath 859-860 جدول، 877  
حل التعارض 540-551  
التسلسل (البعثرة المفتوحة) 540 تعريف، 549-551  
المعالجة المفتوحة (البعثرة المغلق) 540 تعريف، 540-549، أساليب التطبيق 540-545، حذف  
العنصر 545-547  
مراجعة سريعة 553-555  
التعارضات (في جداول البعثرة) 539  
الحل. انظر حل التعارض

النقطتان ( : ) : عامل محدد تناول العنصر 17  
عدة نقاط ( : ) : عامل حل المجال 26، 849 جدول، 891  
ضبط الأعمدة (أعمدة البيانات) 885  
إخراج البيانات في 884-885  
الفاصلة ( , ) : عامل تتابعي 849 جدول  
دالة اللوغاريتم المشترك 859 جدول  
مقارنة:  
مقاطع C 172، 860 جدول  
مؤشرات 150  
عوامل المقارنة. انظر عوامل الارتباط  
شجرة المقارنة لترتيب ثلاثة عناصر 578-579  
خوارزميات البحث المبني على المقارنة  
الحد الأدنى (الترتيب) 537-538 ، 537 جدول  
انظر كذلك خوارزميات البحث  
خوارزميات الترتيب المبني على المقارنة  
الحد الأدنى (الترتيب) 578-579، 578 جدول  
انظر كذلك خوارزميات الترتيب  
مقارنات المفاتيح. انظر مقارنات المفاتيح (في الخوارزميات)  
الربط في زمن التجميع 867  
أعداد مركبة: معالجة 103-108  
مثال برمجة الأعداد المركبة 103-108  
تركيب الدالات / الخوارزميات. انظر مؤشر التركيب الزمني  $O(g(n))$   
التكوين 80-85  
مثال برمجة 243-264  
عوامل اسناد مركبة (عامل =) 876  
الأسبقية 849 جدول  
محاكاة الحاسوب 494-496  
انظر كذلك محاكاة خدمة السينما  
خطط ترميز الحاسبات الآلية. انظر مجموعات الرموز  
البرامج. انظر البرامج (برامج الحاسوب)  
تسلسل المقاطع:  
مقاطع C 860 جدول  
العامل الشرطي ( ؟ : ) :  
الأسبقية 849 جدول  
بيانات شرطية. انظر بيانات if / بيانات if...else  
رسوم بيانية متصلة 723  
قمم متصلة (في رسوم بيانية) 723  
عناصر مترابطة متتالية: الأيجاد 792، 815-816  
Const (كلمة محفوظة)  
للمصفوفات 902  
لدالات العنصر 19، 25  
للثوابت المسماة 876  
لأنواع الانتاج 874

أسبقية العامل const\_cast 849 جدول  
 تعريف نوع const\_reference 242 جدول  
 مصفوفات ثابتة 902  
 مؤشرات ثابتة 152  
 حدود متعددة ثابتة 198 تعريف  
 معاملات اشارة ثابتة 19، 25، 902 مقابل معاملات قيمة 19، 25  
 الثوابت. انظر ثوابت مسماة  
 المقومات 30-35  
 للقوائم المبنية على مصفوفات 190  
 مقومات الفئة الأساسي: التعريفات 72، الاستدعاء 71، 74-75  
 مقوم التحويل 178  
 النسخ. انظر مقوم النسخ لفئة العميل 498-499  
 الافتراضي. انظر تعريف المقومات الافتراضية 31-33  
 مقومات الفئة المستمدة 68، 71، تعريفات 71، 74-75  
 تنفيذ 31، 34، 84  
 كدالات 31  
 الاستدعاء 34، مقومات الفئة الأساسية 71، 74-75، مقومات هدف العنصر 84  
 للقوائم المتصلة 293-294، قوائم متصلة من الطرفين 321  
 للمرصوصات المتصلة 432  
 مقومات هدف العنصر 84  
 الأسماء 31  
 مع معاملات 31، 227 جدول  
 معاملات افتراضية 35 التعريف 33، 74-75، 77 الاستدعاء 34، 84  
 الخصائص 31  
 للطوابير كمصفوفات 483  
 للمرصوصات كمصفوفات 421-422  
 لطوابير العملاء المنتظرين 508  
 الملفات الرئيسية لمكيفات الحاوية 785 جدول  
 دعم المكررات 785 جدول  
 الطوابير ك 492-493  
 المرصوصات ك 460-462  
 الحاويات 218-237، 774-785  
 مترابط. انظر الحاويات المترابطة  
 الدالة begin 224  
 الفئات 218، 769  
 عناصر الاحصاء 821، 823  
 اعلان مكررات داخل 223-224، 241-242  
 deque. انظر الحاويات deque  
 الدالة end 224  
 الملاء 796-798، 798-800  
 ايجاد العناصر 800-802  
 ظهور مترابط مكرر 792، 815-816، أكبر عنصر 821-822، 824،  
 أصغر عنصر 822، 824

توليد العناصر 798-800  
الملفات الرئيسية 785، 785 جدول  
دعم المكررات 785، 785 جدول  
القائمة. انظر حاويات القائمة  
دالات عنصر مشتركة 224، 227-228  
اخراج عناصر 229-233  
مراجعات سريعة 265-266، 357، 841، 842  
استبدال العناصر 806-808  
انتاج أماكن العنصر الأول / الأخير 224  
عكس العناصر 818-821  
دوران العناصر 818-821  
التتابع. انظر الحاويات المتتابعة وكذلك الحاويات deque وحاويات القائمة والحاويات vector  
تبادل العناصر 808-811  
تعريفات الأنواع المشتركة للجميع 242 جدول  
Vector. انظر الحاويات vector  
بنيات التحكم 889-890  
مقومات التحويل 178  
تحويل:  
القوائم المبنية على المصفوفات الى تكديسات 596-602  
الرموز للوضع الصغير / الكبير 858-859 جدول  
الأعداد العشرية الى أعداد ثنائية 387-389  
مقومات النسخ 166-172، 209  
للقوائم المتصلة على المصفوفات 190-191  
للأشجار الثنائية 650  
موجودة في تعريفات الفئة 169-172  
للقوائم المتصلة 305-306  
للطوابير المتصلة 488  
للمرصوعات المتصلة 439  
والنسخ السطحي 166-168  
للمرصوعات كمصفوفات 422، مرصوعات متصلة 439  
الدالة copy: اخراج عناصر الحاويات 229-233  
نسخ:  
القوائم المبنية على مصفوفات 794-795  
الأشجار الثنائية 638-639، 649  
مقاطع C 172، 861 جدول  
قيم هدف الفئة 22-24، 159-162، 166-172  
متغيرات حركية 154-157  
قوائم متصلة 304-305، 305-306  
المؤشرات 149-150  
سطحي. انظر النسخ السطحي (البيانات)  
المرصوعات 413، 421  
الدالة copyList(قوائم متصلة) 304-305  
الدالة copyStack 413، 421

الدالة copyTree 638-639، 649  
 الدالة cos (x) (مسبقة التعريف) 859 جدول  
 الدالة cosh (x) (مسبقة التعريف) 859 جدول  
 الدالة count (مسبقة التعريف) 821، 823  
 دالات الاحصاء : دالات مسبقة التعريف 821، 823  
 الدالة count\_if (مسبقة التعريف) 821، 823  
 حلقات for المعدودة. انظر الحلقات for  
 عد الناصر 821، 823  
 الهدف 5 cout  
 بيانات cout 882-889، 887  
 التركيب 882  
 الامتداد .cpp 40، 878  
 الدالة createGraph 729، 730  
 الدالة createSpanningGraph 752  
 الملف الرئيسي cstdint 860  
 الملف الرئيسي cstring 860-861 جدول  
 الدالة ct.begin 228 جدول  
 الدالة ct.clear 228 جدول، 777 جدول، 782 جدول  
 الدالة ct.empty 227 جدول  
 الدالة ct.end 228 جدول  
 الدالة ct.erase 228 جدول، 777 جدول، 782 جدول  
 الدالة ct.find 777 جدول  
 الدالة ct.insert 228 جدول، 777 جدول، 782 جدول  
 الدالة max\_size 227 جدول  
 الدالة ct.rbegin 228 جدول، 231  
 الدالة ct.rend 228 جدول  
 الدالة ct.size 227 جدول  
 الدالة ctl.swap 228 جدول  
 البيانات ct <elementType> 775-776 جدول، 780-781 جدول  
 البيانات ct <elementType, sortOp> 775-776 جدول، 780-781 جدول  
 أقواس مجمعة. انظر الأقواس  
 المؤشر: الانتقال الى بداية السطر التالي 882  
 أهداف العميل (محاكاة خدمة السينما) 495 تعريف  
 الفئة. انظر الفئة customerType  
 الحصول على وضبط الزمن بين حالات الوصول 510  
 عمليات على 497  
 زمن الخدمة. انظر زمن التعامل  
 طوابير الانتظار. انظر أهداف طابور العميل المنتظر  
 أهداف العميل (مثال برمجة السينما) 351  
 زمن خدمة العميل. انظر زمن التعامل  
 الفئة customerType (محاكاة خدمة السينما) 496-499  
 عناصر البيانات 496  
 التعريف كنوع بيانات مجرد 497-498

دالات العنصر 498-499  
الفئة customerType (محاكاة خدمة السينما) 351  
الدوائر (في الرسوم البيانية) 723

## D

البيانات:

- في عقد شجرة ثنائية: تحديث 672-675
- ادخال. انظر ادخال البيانات
- بيانات غير صحيحة 881-882
- التحكم بالمؤشرات 139-143
- للأهداف 4، العمليات على (انظر العمليات (على البيانات))
- الاخراج. انظر اخراج البيانات
- نسخ سطحي. انظر النسخ السطحي
- انظر كذلك البيانات char والبيانات int والقيم
- تجريد البيانات 37 تعريف
- فئات تناول عناصر البيانات (للفئات) 16-17، 19، 67، 79-80
- التناول 21-22، 24
- الاعلان 16-17، 19
- تناول عنصر الدالة ل 16، 19
- قواعد التوريث 67-68، 71
- التهيئة 26، 30-35
- عمل عام أو خاص 17
- تخصيص الذاكرة 21
- المؤشرات كحدود (خصوصيات) 157-172، 646
- المجال 24
- عمليات البيانات. انظر العمليات (على البيانات)
- بنيات البيانات. انظر أنواع البيانات المجردة، المصفوفات، الفئات، القوائم، الطوابير، المرصوصات، البنيات
- أنواع البيانات:
- مجردة. انظر أنواع البيانات المجردة
- الفئات الأساسية 875. انظر كذلك المؤشرات، أنواع البيانات البسيطة، أنواع البيانات المنظمة.
- أنواع بيانات تدفق الملف 887
- العلامة العشرية. انظر أنواع البيانات ذات العلامة العشرية
- متكامل. انظر أنواع البيانات المتكاملة
- أنواع ذات معاملات (من الفئات) 112
- الخصائص 37
- أنواع بيانات بسيطة 875-876
- معرف بواسطة المستخدم (معرف بواسطة المبرمج) انظر أنواع بيانات العد، أنواع البيانات المنظمة
- الفئة dateType 81-83
- ازالة تخصيص الذاكرة:
- للأهداف الحركية 158-159، 871
- لعقد القوائم المتصلة 294، 305، 322

برامج التصحيح 7  
ضبط الصيغة العشرية الثابتة 884  
الأعداد العشرية: التحويل الى أعداد ثنائية 389-387  
المنزلات العشرية: مخرجات الضبط 883  
النقطة العشرية: توضيح بالأصفار 884  
بيانات الاعلان 879-878  
أهداف الفئة 20  
ثوابت مسماة 876  
مصفوفات أحادية البعد 900  
متغيرات 877  
اعلان:  
مصفوفات: انظر اعلان:  
عناصر فئة مصفوفات أحادية البعد 16-17، 19  
أهداف الفئة 20-21، 24، 34، 166  
الحاويات: حاويات مترابطة 775-776، 780-782، حاويات deque 233-234،  
حاويات قائمة 330، حاويات vector 219-220  
دالات صديقة 92  
محددات 16، 879  
مكررات داخل حاويات 223-224، 241-242  
ثوابت مسماة 876  
مصفوفات أحادية البعد 900، كمعاملات رسمية 901، 902، مع المؤشرات 148، 151-153  
زوج من الأهداف 770  
مؤشرات 134-135، 142، كمعاملات للدالات 153  
أهداف المرصوصة 439  
المتغيرات 16، 876-877، 879  
أهداف الفئة المستمدة 71، أهداف تدفق الملف 887-888،  
التهيئة عندما 16، متغيرات البنية 144  
دالات افتراضية 867-868  
انظر كذلك أسبقية بيانات الاعلان 849 جدول  
تقليل المؤشرات 150-151  
النسخ العميق (للبيانات) 156-157، 209  
المقومات الافتراضية 31، 35، 227 جدول  
للأشجار الثنائية 647  
التعريف 33، 74-75  
للسوم البيانية 731  
الاستدعاء 34  
للقوائم المتصلة 293-294، القوائم المتصلة من الطرفين 321  
للطوابير المتصلة 488  
للمرصوعات المتصلة 432  
محدد تناول العنصر الافتراضي 16، 67  
تهيئة العنصر الافتراضي (لأهداف الفئة) 166  
معاملات افتراضية  
مقومات 35

دالات ذات 898-899  
 تعريفات. انظر تعريفات الفئة وتعريفات الدالة  
 العامل delete  
 تدمير الأشجار الثنائية 650  
 تدمير المتغيرات الحركية 149  
 الأسبقية 849 جدول  
 التركيب 149  
 الدالة deleteFromTree 663-665  
 الدالة deleteNode (أشجار البحث الثنائية) 665-666  
 الدالة deleteNode (قوائم متصلة من الطرفين) 328-329  
 الدالة deleteNode (قوائم متصلة) 299-303  
 الدالة deleteNode (قوائم متصلة مرتبة) 315-316  
 الدالة deletQueue (عملية):  
 للطوابير المتصلة 487، 488، 508، 509  
 للطوابير كمصفوفات 471، 472-473، 482-483  
 حذف (إزالة) العناصر 802-806  
 من القوائم المبنية على مصفوفات 188-189، 190، 194  
 من الحاويات المترابطة:  
 الحاويات map/multimap 782، الحاويات set/multiset 776-777  
 من أشجار البحث الثنائية 644، 658-666، أشجار AVL 678، 699  
 من جداول البعثة 545-547، من القوائم المتصلة 282-284، 299-303  
 القوائم المتصلة من الطرفين 327-329، القوائم المتصلة المرتبة 315-316  
 من الطوابير 470، 47—1 كمصفوفات 471، 472-473، 482-483  
 طوابير متصلة لـ 487، 488، 508، 509، طوابير الأسبقية 605  
 من الحاويات التتابعية 222 جدول، 223، 229 جدول  
 من المرصوصات 409، 410، 411، 461 جدول، المرصوصات المتصلة 437-439،  
 المرصوصات كمصفوفات 418-420  
 المحددات. انظر الأقواس: علامة تنصيب مزدوجة: عامل الفصل وعامل التتابع  
 الترتيب الأول للعرض (للقمم) 732  
 اجتياز (الرسوم البيانية) الأول للعمق 731، 732-734  
 الترتيب الطوبوغرافي (للقمم) 754  
 الدالة depthFirstTraversal 733  
 البيان [index] deq 234 جدول  
 الدالات deq.assign 234 جدول  
 الدالة deq.at 234 جدول  
 الدالة deq.back 234 جدول  
 الدالة deq.front 234 جدول  
 الدالة deq.pop\_front 234 جدول  
 الدالة deq.push\_front 234 جدول  
 الفئة deque 233  
 الحاويات deque 233-237  
 الاعلان والتهيئة 233-234  
 الملف الرئيسي 785 جدول

ادخال عناصر في 233  
دعم المكررات 785 جدول  
عمليات على 234  
مراجعة سريعة 265-266  
انظر كذلك الحاويات، الحاويات التتابعية  
عامل ازالة الاشارة (\*) 135-136، 139  
عملية المكررات 237  
وضع 134-135  
التوظيف برمز معامل الاشارة 153  
الأسبقية 144، 849 جدول  
دالات عنصر الفئة المستمدة 71  
التعريف 69-70، 76-77  
اعلان أهداف الفئة المستمدة 71  
التمرير الى معاملات الفئة الأساسية 865-871  
الفئات المستمدة 66 تعريف  
المقومات 68، 71، التعريفات 71، 74-75  
التعريف 69-70، 73-75، 77  
استمداد طوابير متصلة من قوائم متصلة 488-490  
استمداد مرصوعات متصلة من قوائم متصلة 439-441  
اثقال دالة في 69  
الملفات الرئيسية 77  
قواعد التوريث 67-68، 71  
دالات العنصر. انظر دالات عنصر الفئة المستمدة  
الاهداف. انظر أهداف الفئة المستمدة  
اعادة تعريف دالات عنصر الفئة الأساسية 69-70  
التركيب 67  
الفئة derivedClass 70، 73-75  
التصميم. انظر تصميم البرنامج  
الدالة destroy 650، 651-650  
تدمير:  
الأشجار الثنائية 650، 651-650  
القوائم المتصلة 294، القوائم المتصلة من الطرفين 322  
الطوابير 470، كمصفوفات 477، 481، طوابير متصلة 486، 487  
مرصوعات 410 كمصفوفات 415، مرصوعات متصلة 432  
الدالة (العملية) destroyList:  
للقوائم المتصلة 294، للقوائم المتصلة من الطرفين 322  
الدالة (العملية) destroyQueue 470  
للطوابير المتصلة 486، 487  
للطوابير كمصفوفات 477، 481  
الدالة destroyStack 410  
للمرصوعات المتصلة 432  
للمرصوعات كمصفوفات 415  
الدالة destroyTree 650

المدمرات 35، 227 جدول  
للقوائم المبنية على مصفوفات 190  
للأشجار الثنائية 651-650  
ازالة تخصيص الذاكرة 158-159، 305، 871  
كدالات 35  
للمسوم البيانىة 731  
التضمين في تعريفات الفئة 159  
للقوائم المتصلة 305

للطوابير المتصلة 488  
للمرصصات المتصلة 439  
بادئة الاسم 35  
للطوابير: كمصفوفات 483-484، طوابير متصلة 488  
لقوائم مقدمى الخدمة 505  
للمرصصات: كمصفوفات 421، 422، القوائم المتصلة 439  
المدمرات الافتراضية 871  
لطوابير العملاى المنتظرين 508  
الدالة dft 732-733  
الدالة dftAtVertex 733-734  
تعريف النوع difference\_type 242 جدول  
الفروق بين المجموعات: دالات مسبقة التعريف 827، 832-835  
الرسوم البيانىة: انظر الرسوم البيانىة المباشرة  
التكرار المباشر 371  
الحواف المباشرة (للأشجار الثنائية) 632  
رسوم بيانىة مباشرة 721، 722، 723، 726  
القمم: ترتيب طوبوغرافى 754-761  
ايقاف بيانات assert 857  
الدالة divideList 590-591  
هدف الدالة <Type>divides، 789 جدول  
تقسيم القوائم المتصلة 589-591  
دالة بعثرة التقسيم 539  
عامل القسمة ( / ): الأسبقية 849 جدول  
حلقات do...while 889  
خاصية نوع بيانات المجال 37  
عملية السيطرة: فى الخوارزميات 12  
النقطة (عامل النقطة): انظر عامل تناول العنصر  
نوع البيانات double، 876  
انظر كذلك المتغيرات double  
بعثرة مزدوجة 545  
علامة تنصيص مزدوجة ( " ): محدد الملف الرئيسى المحدد بواسطة المستخدم 41، 77  
دوران مزدوج (للأشجار AVL) 687-689، 690، 696، 697-698  
الدالات من أ 690—692

نوع بيانات المتغيرات 876 double  
ادخال (قراءة) بيانات 881-880  
مدخلات غير صحيحة 882-881  
مدخلات صحيحة 881 جدول  
انظر كذلك المتغيرات  
الطوابير المزدوجة النهاية. انظر الحاويات deq  
القوائم المتصلة من الطرفين 329-318  
العمليات الاساسية 319-318  
التحقق مما اذا كانت فارغة 321  
دالات عنصر الفئة 329-321  
المقوم الافتراضي 321  
التعريف كنوع بيانات مجرد 321-319  
حذف عناصر (عقد) من 329-327  
تدمير 322  
تهيئة 322  
ادخال عناصر (عقد) في 327-324  
الأطوال: تحديد 322  
العقد. انظر العقد التحتية (للقوائم المتصلة)  
الطباعة بترتيب عكسي 323  
الطباعة بترتيب عكسي 323  
مراجعة سريعة 357  
البحث 324-323  
الاجتياز 318، 323-322  
انظر كذلك حاويات القائمة  
مقوم الفئة doublyLinkedList 321  
التعريف كنوع بيانات مجرد 321-319  
انظر كذلك قوائم متصلة من الطرفين  
أسلوب العمل الوهمي (مشكلة طابور الانتظار) 509  
مصفوفات حركية 151 تعريف  
العمل 148، 151-153  
ازالة تخصيص الذاكرة 159-158  
مثال البرمجة 206-198  
مقابل المصفوفات الساكنة 151  
الحاويات vector، 219  
اظر كذلك الحاويات deque والحاويات vector  
الربط الحركي 867  
بنيات البيانات الحركية. انظر الأشجار الثنائية، المصفوفات الحركية، القوائم المتصلة، الطوابير  
المتصلة، المرصوبات المتصلة  
أهداف حركية: ازالة تخصيص الذاكرة 871  
متغيرات حركية 147-149، 148 تعريف  
النسخ 154-157  
العمل 148-149  
التدمير 149

عامل dynamic\_cast: الأسبقية 849 جدول

E

مجموعة رموز كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد 853 جدول  
الحواف (في الرسوم البيانية) 721، 723  
وزن الحافة 737  
مثال برمجة نتائج الانتخابات 606-626  
الدالات empty (مسبقة التعريف)  
عملية الحاوية 225 جدول، 227 جدول  
عمليات الطابور 492 جدول  
عملية المرصوصة 461 جدول  
الدالة str.empty 861 جدول  
جمل اعتراضية فارغة  
في اعلانات هدف الفئة 34  
في قوائم المعامل الرسمي 896  
أشجار فرعية فارغة (من الأشجار الثنائية) 632  
التغليف 4 تعريف، 85  
خطط الترميز. انظر مجموعات الرموز  
الدالة end (مسبقة التعريف): عملية حاوية 224، 228 جدول  
أداة التحكم end 882  
اخلاء فاصل تدفق المخرجات 885-886  
اعادة وضع المؤشر 882  
أنواع بيانات العد 875  
علامة يساوي (=) عامل الاسناد. انظر عامل الاسناد  
عامل تعبير postfix 446  
علامات يساوي (=) . انظر عامل التساوي  
عامل "يساوي":  
في تعبير infix. انظر عامل التساوي (=)  
في تعبير postfix (=) 446  
هدف الدالة equal\_to<Type>، 791 جدول  
عامل التساوي (=) مقابل عامل الاسناد (=) 890  
عملية الحاوية 228 جدول  
اتقال 97-98، 108، 177  
تعريف الفئة pair 772 جدول  
الأسبقية 849 جدول  
الدالة equalTime 28-30  
فئات التساوي (حالات الاختبار) 7  
الدالات erase (مسبقة التعريف)  
عمليات الحاوية 222 جدول، 228 جدول، 229 جدول، 777 جدول، 782 جدول  
الدالات str.erase، 862 جدول  
مسح المقاطع 862 جدول  
الأخطاء:  
في التعبيرات postfix: معالجة برنامج الآلة الحاسبة 446، 448

تدفق الخطأ القياسي 5  
 ايولر ليونهارد: على مشكلة كوبري كونيسبرج 720  
 علامة التعجب (!): عامل لا 849 جدول  
 علامة تعجب – علامة يساوي (! =). انظر عامل عدم التساوي  
 الامتداد .exe ، 878  
 كود قابل للتنفيذ 40، 478  
 ملفات الكود القابل للتنفيذ: الامتداد 878  
 بيانات قابلة للتنفيذ 878-879  
 تنفيذ المقومات 31، 34، 84  
 الدالة  $\exp(x)$  (مسبقة التعريف) 859 جدول  
 دالات أسية 859 جدول، 860 جدول  
 تعبيرات: فئات أساسية. انظر تعبيرات حسابية، تعبيرات منطقية  
 كود التبادل العشري الذي يتم ترميزه بكود ثنائي ممتد. انظر مجموعة رموز كود التبادل العشري الذي  
 يتم ترميزه بكود ثنائي ممتد  
 الامتدادات: لملفات البرنامج 878  
 عامل الاستخلاص (<<) 879، 880  
 في البيان cin 879-882، 887  
 الانتقال 87، 99-100، 100-101، 108، 178  
 الأسبقية 849 جدول  
 استخدام أهداف تدفق الملف مع 888

## F

الدالة  $\text{fabs}(x)$  (مسبقة التعريف) 859 جدول  
 عوامل (الأعداد الصحيحة) 368  
 دالة الحساب 369-370  
 حالة الفشل (تدفق المدخلات) 882  
 انظر كذلك فشل الادخال  
 أعداد فيبوناكي في أشجار AVL، 700  
 حساب: الطريقة التكرارية 379-383  
 بنية بيانات الداخل أولاً يخرج أولاً. انظر الطوابير  
 ادخال / اخراج ملف 887-889  
 الملف الرئيسي 887  
 خطوات المعالجة 887-888، 889  
 أسماء وامتدادات الملف لملفات البرنامج 878  
 أنواع بيانات تدفق الملف 887  
 أهداف تدفق الملف  
 الاعلان 887-888  
 الاستخدام مع << و >> ، 888  
 الملفات 887 تعريف  
 الاغلاق 888  
 ادخال / اخراج بيانات من / الى. انظر ادخال / اخراج ملف  
 الفتح 888، 889

أسماء وامتدادات ملف البرنامج 878  
 انظر كذلك الملفات الرئيسي، وملفات التطبيق، وملفات المدخلات،  
 وملفات المخرجات، وأسماء ملفات محددة  
 الدالة fill (مسبقة التعريف) 796-798  
 الدالة fill\_n function (مسبقة التعريف) 796-798  
 ملأ الحاويات 796-798، 800-798  
 الدالات find (مسبقة التعريف) 800-801  
 عمليات الحاوية 777 جدول، 782 جدول  
 الدالة str.find، 862 جدول  
 الدالة find\_end (مسبقة التعريف)، 801-802  
 الدالة find\_first\_of (سبقة التعريف) 801-802  
 الدالة find\_if (مسبقة التعريف) 800-801  
 ايجاد العناصر 800-802  
 حالات ظهور مترابطة ومتعاقبة 792، 815-816  
 أكبر عنصر: المنهج التكراري 565، دالات مسبقة التعريف 821-822، 824،  
 المنهج التكراري 372-375  
 أصغر عنصر: المنهج التكراري 560-563، دالات مسبقة التعريف 822-824  
 مقاطع فرعية 862 جدول  
 انظر كذلك البحث  
 First. انظر المؤشر head (للقوائم المتصلة)  
 العنصر الأول (من الطوابير). انظر العنصر الأمامي (من الطوابير)  
 بنية بيانات الداخل أولاً يخرج أولاً. انظر الطوابير  
 ضبط صيغة عشرية ثابتة 884  
 أداة التحكم fixed 884  
 نوع البيانات float، 876  
 أنواع البيانات ذات العلامة العشرية، 875، 876  
 أنواع بيانات الأعداد ذات العلامة العشرية 875، 876  
 المخرجات الافتراضية 883  
 تشكيل المخرجات: التحكم في الدقة 883، ترميز علمي 883، 884  
 الدالة floor (x) (مسبقة التعريف) 859 جدول  
 أداة التحكم flush 885-886  
 التركيب 886  
 نتي دالة البعثة 539  
 حلقات loop، 889  
 معالجة المصفوفات 563، 569  
 الدالة for\_each (مسبقة التعريف) 825-827  
 قوائم المعاملات الرسمية  
 الدالات المنتجة للقيمة 895  
 دالات void، 896  
 اعلان المعاملات الرسمية كمؤشرات اشارة 153  
 اعلان مصفوفات أحادية البعد 901، 902  
 تمرير أهداف الفئة المستمدة الى معاملات الفئة الأساسية 865-871  
 تمرير الدالات الى دالات أخرى 672-675، 825-827

أنواع. انظر معاملات الاشارة ومعاملات القيمة.  
تشكيل المخرجات في أعمدة 884-885  
ضبط البيانات 885  
انظر كذلك أدوات التحكم (في المخرجات)  
مكررات أمامية 239  
أشجار (رسوم بيانية) حرة 746  
Friend (دالة محفوظة) 92  
الدالات الصديقة (دالات لغير الأعضاء) 92-95  
الاعلان 92  
التعريف 92  
دالات العامل 95-96، 103  
اثقال عوامل ثنائية 98-99  
مراجعة سريعة 116  
العنصر الأمامي (من الطوابير) 470، 471  
الطوابير المتصلة 484، انتاج 487، 488، 508، 509  
الطوابير كمصفوفات 471، الانتاج 481-482  
الانتاج 470  
الدالة front (قوائم متصلة) 295-296  
القوائم المتصلة من الطرفين 324  
الدالة front (مسبقة التعريف)، عملية حاوية 221 جدول، 234 جدول، 331 جدول  
عملية طابور 492 جدول  
الدالة front (طوابير) 470  
طوابير متصلة 487، 488، 508، 509  
الطوابير كمصفوفات 481-482  
المؤشر front. انظر المؤشر queueFront  
المكرر front\_inserter 795  
الملف الرئيسي fstream 887  
عامل استدعاء الدالة: الاثقال 789  
استدعاءات الدالة:  
الى مقومات الفئة الأساسية 71، 74-75  
الى عنصر الفئة الأساسية 70  
استدعاءات تكرارية 369، 390  
الى دالات منتجة للقيمة 895  
الى دالات void، 896، 897  
انظر كذلك استدعاءات دالات خاصة  
تعريفات الدالة:  
ملف من أجل، انظر ملف التطبيق  
دالات صديقة 92  
دالات عنصر (الفئات) 16، 26-28، 82-83، 85  
اثقال عامل الاسناد 162-163  
اثقال عوامل ثنائية 97، 99  
اثقال عامل الاستخلاص 101  
اثقال عامل الادخال 100

الوضع 40، 114  
 الدالات المنتجة للقيمة 895  
 الدالات void، 896  
 انظر كذلك تعريفات دالات خاصة تحت الانشاء  
 عناصر الدالة. انظر دالات العنصر  
 أسماء الدالات:  
 الانتقال. انظر ائقال الدالة  
 دون جمل اعتراضية 672  
 عناصر الدالة (لخوارزميات مكتبة القالب المعياري) 788، 789-794  
 أهداف دالة حسابية 789-791  
 الملف الرئيسي 789  
 أهداف دالة منطقية 794 جدول  
 الأصول 794  
 مراجعة سريعة 842  
 أهداف دالة مترابطة 791-794، 791-792 جدول  
 ائقال الدالة 85، 109:  
 في الفئات المستمدة 69  
 مع القوالب 110-114  
 دالات الاجتياز 672-673  
 نماذج الدالة  
 في تعريفات الفئة 16، 25-26  
 الدالة copy 229-230  
 الدالات الصديقة 92  
 ائقال عامل الاسناد 162  
 ائقال عوامل ثنائية 97، 99  
 ائقال عامل الاستخلاص 101  
 ائقال عامل الادخال 100  
 شرط مسبق / تالي في 6-7 من خوارزميات مكتبة القالب المعياري 796  
 انظر كذلك نماذج خاصة تحت التركيب، وأوصاف خوارزميات خاصة لمكتبة القالب المعياري  
 قوالب الدالة 85، 110-114  
 عناصر دالة قالب الفئة 114  
 التركيب 110  
 الدالات 5  
 مقارب 14 تعريف  
 دالات Big-O 14-15، 15 جدول كعناصر فئة. انظر دالات العنصر (عناصر الدالة)  
 أهداف الفئة في، 24  
 التركيب. انظر مؤشر التركيب الزمني لـ  $O(g(n))$  ذات المعاملات الافتراضية 898-899  
 تعريفات. انظر تعريفات الدالة  
 دالات صديقة. انظر الدالات الصديقة (دالات لغير العنصر)  
 معدل النمو. انظر تطبيق مؤشر التركيب الزمني لـ  $O(g(n))$ ، 5-7، 25-30  
 لتطبيق عناصر بيانات الفئة. انظر مكتبات المقومات. انظر الملفات الرئيسية main. انظر الدالة  
 main التي ليس لها نوع. انظر ترتيب دالات العامل. انظر ائقال مؤشر التركيب الزمني لـ  $O(g(n))$ .  
 انظر معامل ائال الدالة. انظر معاملات (معطيات) الدالات

جمل اعتراضية في، 896  
التمرير كمعاملات الى دالات أخرى 672-675، 825-827  
تمرير المعاملات الى 114، مصفوفات 901، أهداف فئة 24-25، أهداف الفئة المستمدة  
الى معاملات الفئة الأساسية 865-871، دالات 672-675، 825-827، مؤشرات 153، أهداف  
المرصوصة 422  
مؤشرات الى 672  
شروط تالية 5-7  
شروط مسبقة 5-7  
مسبق التعريف. انظر الدالات المسبقة التعريف  
المنع من تغيير المعاملات الحقيقية 25، 902  
التكراري. انظر الدالات التكرارية  
انتاج قيم. انظر انتاج القيم (من الدالات المنتجة للقيمة)  
لبحث القوائم. انظر دالات البحث.  
لترتيب القوائم. انظر دالات الترتيب.  
خوارزميات مكتبة القالب المعياري 786-788، 796-840  
دالات المقطع (النوع string) 861، 861-863 جدول  
القوالب. انظر قوالب الدالة  
معرف بواسطة المستخدم. انظر الدالات المنتجة للقيمة، والدالات void، ودالات خاصة.  
منتج للقيمة. انظر دالات منتجة للقيمة  
الدالات الافتراضية 867-871  
Void. انظر الدالات void وانظر كذلك دالات خاصة

## G

جاوس سي اف 391  
الدالة generate (مسبقة التعريف) 798-800  
الدالة generate\_n (مسبقة التعريف) 798-800  
توليد عناصر الحاوية 798-800  
خوارزميات عامة. انظر خوارزميات مكتبة القالب المعياري  
الدالة getAdjacentVertices، 727  
الدالة getFreeServerId، 505  
الدالة getline (مسبقة التعريف) 861 جدول  
الدالة getNumberOfBusyServers، 505  
محددات عامة: مشكلة تداخل الأسماء 890  
جولومب اس، 391  
مثال برمجة تقرير الدرجات، 243-264  
اجتيازات الرسم البياني، 731-736  
الاجتياز الأول للعرض 734-736  
الاجتياز الأول للعمق 731، 732-734  
الرسوم البيانية 719-763، 721 تعريف  
عمليات أساسية 726. انظر كذلك اجتيازات الرسم البياني  
الاخلاء، 730  
رسوم بيانية متصلة، 723  
عمل 729-730

دوائر في، 723  
 مقوم افتراضي، 731  
 التعريف كنوع بيانات مجرد، 728-729، 735-736  
 المدمر، 731  
 رسوم بيانية مباشرة، 721، 722، 723، 726  
 الحواف، 721، 723. انظر كذلك وزن الحافة  
 خوارزمية شجرة الامتداد الأدنى، 744-754  
 الطباعة 731  
 مراجعة سريعة، 762-763  
 تمثيل 723-725، 726  
 خوارزمية أقصر مسار 737-738، 743-738  
 رسوم بيانية بسيطة، 723  
 رسوم بيانية فرعية، 721  
 مصطلحات، 721، 723، 737، 746  
 الاجتياز، 731. انظر كذلك اجتياز الرسم البياني  
 كأشجار، 746 وانظر كذلك أشجار الامتداد  
 الرسوم البيانية الغير مباشرة، 721، 722  
 الاستخدامات 721، 737  
 القمم. انظر القمم  
 رسوم بيانية موزونة، 737. انظر كذلك الأشجار الموزونة  
 تعريف الفئة graphType كنوع بيانات مجرد، 728-729، 735-736  
 دالات العنصر، 729-731  
 امتداد فئة الترتيب الطوبوغرافي topologicalOrder، 754-755  
 ائقال عامل "أكبر من" (<)، 177  
 تعريفات الفئة pair، 772 جدول  
 الأسبقية 849 جدول  
 ائقال عامل "أكبر من أو يساوي" (<=)، 177  
 تعريف الفئة pair، 772 جدول  
 الأسبقية 849 جدول  
 علامات "أكبر من" (<<). انظر عامل الاستخلاص  
 هدف الدالة <Type>greater، 791 جدول  
 هدف الدالة <Type>greater\_equal، 792 جدول  
 خوارزمية طمع (للرسوم البيانية)، 737-738، 743-738  
 معدل نمو الدالات. انظر مؤشر التركيب الزمني لـ  $O(g(n))$

## H

الامتداد h، 40  
 علاقات "has-a". انظر التوريث  
 دالات البعثرة، 538-539  
 دالات أمثلة، 539  
 جداول البعثرة، 538  
 اضافة عناصر الى، 550  
 التجميع في. انظر الى التجميع (انظر حل التعارض)

تنظيم البيانات في، 538-539  
حذف العناصر من، 545-547، 550  
البحث، 550  
البعثة، 538-552  
مغلق. انظر المعالجة المفتوحة  
التعريف كنوع بيانات مجرد، 547-549  
بعثة مزدوجة، 545  
دالات، 538-539  
مفتوح. انظر التسلسل  
تحليل الأداء، 551-552  
مراجعة سريعة 553-555  
اعادة البعثة، 542  
جداول. انظر جداول البعثة.  
المؤشر head (للقوائم المتصلة)، 274، 276، 278-279  
الاعلان، 275  
ملفات رئيسية، 875-863، 877-878  
لتعريفات الفئة، 40-41، 114، 307  
للفئات المستمدة، 77  
امتداد، 40  
محددات اسم الملف (علامات ارفاق)، 41  
لأهداف الدالة، 789  
التضمين، 40-41، 77، 877-878، ادراجات متعددة: تجنب، 78-79  
تحويلات التسمية، 41  
للقوائم المتصلة المرتبة، 316-318  
للطوابير، 494  
للمرصوعات، 423، 439، 460  
انظر كذلك ملفات رئيسية محددة  
عقد رئيسية (في القوائم المتصلة)، 335-336، 357-358  
خوارزميات التكدس  
خوارزميات مكتبة القالب المعياري، 788  
انظر كذلك خوارزميات تحت التكدس  
الترتيب بالتكدس، 494، 595-605  
بناء خوارزمية تكدس، 596-602  
تحليل الأداء، 604-605  
مقابل الترتيب السريع، 605  
خوارزمية الترتيب، 602-604  
الدالة heapify، 602، 605  
التكدسات، 595 تعريف  
تحويل القوائم المبنية على مصفوفات الى، 596-602  
تطبيق طوابير الأسبقية، 605-606  
الترتيب، 602-604  
ارتفاع الأشجار الثنائية، 637-638  
الدالة hieght (أشجار ثنائية)، 638

أشجار متوازنة الارتفاع. انظر أشجار AVL  
اخفاء تفاصيل عمليات البيانات، 26، 44-40  
برامج لغة عالية المستوى C++. انظر أنواع كود برامج C++، 878  
التصميم، 3-4. انظر كذلك التصميم القائم على الهدف  
مثال البرمجة الخاص بأعلى معدل درجات، 425-428  
جداول البعثرة. انظر جداول البعثرة  
دالات جيب التمام، وجيب الزاوية، وظل الزاوية، 859 جدول، 860 جدول  
شرطة - قوس زاوية أيمن (->): سهم عامل تناول العنصر، 144-145، 849 جدول

## I

الادخال / الاخراج (عمليات الادخال / الاخراج)

ادخال / اخراج ملف، 887-889

الملف الرئيسي iostream، 877، 887

انظر كذلك ادخال بيانات (قراءة البيانات)، اخراج البيانات

المحددات:

التعريف، 16، 879

التسمية، 890

الاشارة الى محددات دالة عنصر في تعريفات دالة، 26

استخدام (تناول) عناصر الأسماء، 891-892

انظر كذلك الثوابت المسماة

تحديد الفئات، 55-56

تحديد الأهداف، 4، 15، 55-56

تحديد العمليات، 4، 55-56

بيانات if، 889

بيانات if ... else، 889

نوع البيانات ifstream، 887

تفاصيل تطبيق أهداف الفئة: اخفاء، 26، 44-40

ملفات التطبيق، 40، 42، 92

قوالب الفئة، 114

الامتداد، 40

موجه ما قبل المعالجة #، 877-878

الدالة includes (مسبقة التعريف)، 827-829

تضمين ملفات رئيسية، 40-41، 77، 877-878

ادراج متعددة: تجنب، 78-79

عامل الزيادة (+ +)، 86، 879

عملية المكرر، 237

الأسبقية، 849 جدول

زيادة المؤشرات، 150-151

الحلقات for ذات المؤشر. انظر الحلقات for

المؤشرات. انظر مؤشرات المصفوفة.

التكرار الغير مباشر، 371

عامل ازالة التوجيه. انظر عامل ازالة الاشارة

عامل عد التساوي (! =)

عملية الحاوية، 228 جدول  
 الانتقال 178-177  
 تعريف الفئة pair، 772 جدول  
 الأسبقية 849 جدول  
 تكرار غير محدود، 371  
 تعبيرات infix، 441  
 مقابل تعبيرات postfix، 442 جدول  
 ترميز infix، 441  
 إخفاء المعلومات، 26، 40-44  
 التوريث، 4 تعريف، 66-80  
 توريث متعدد، 66  
 توريث خاص، 67، 80  
 مثال برمجة، 243-264  
 التوريث المحمي، 80  
 التوريث العام، 79-80  
 مراجعة سريعة، 115  
 القواعد للفئات الأساسية والمستمدة، 67-68، 71  
 توريث أحادي، 66  
 تركيب (تسلسل)، 66-67  
 انظر كذلك الفئات الأساسية: الفئات المستمدة  
 الدالة (عملية) initializeList (تهيئة القائمة)  
 للقوائم المتصلة، 294-295، القوائم المتصلة من الطرفين، 322  
 دالة (عملية) initializeQueue (تهيئة الطابور)، 470  
 للطوابير المتصلة، 487-488  
 للطوابير كمصفوفات، 477، 480-481  
 دالة (عملية) initializeStack (تهيئة المرصوعة)، 410  
 للمرصوعات المتصلة، 432  
 للمرصوعات كمصفوفات، 414  
 التهيئة:  
 المصفوفات، والمصفوفات أحادية البعد، 901، الطوابير كمصفوفات، 480-481، المرصوعات  
 كمصفوفات، 414  
 الحاويات المترابطة: الحاويات map / multimap، 780-782، الحاويات set / multiset، 775-  
 776  
 أهداف الفئة، 166  
 الحاويات: الحاويات deque 233-234، حاويات القائمة 330، حاويات vector 219-220  
 عناصر بيانات (الفئات)، 26، 30-35  
 قوائم متصلة، 293-294، 294-295،  
 قوائم متصلة من الطرفين 322  
 مؤشرات، 147  
 الطوابير، 470، كمصفوفات 477، 480-481، الطوابير المتصلة 487-488  
 المرصوعات 410، كمصفوفات 414، مرصوعات متصلة 432  
 متغيرات: عند الإعلان 16  
 الدالة inner\_product (مسبقة التعريف)، 835، 838، 839

الدالة inorder (اجتياز شجرة ثنائية)، 642-643  
 الاجتياز في الترتيب (للأشجار الثنائية)، 639، 640-643  
 ائصال الدالة، 672-673  
 تتابع مخرجات العقدة، 642  
 خوارزمية / دالة غير تكرارية، 668-669  
 خوارزمية / دالة تكرارية، 640-643  
 الدالة inplace\_merge (مسبقة التعريف)، 817  
 المدخلات:  
 الاثبات، 6  
 انظر كذلك ادخال البيانات (قراءة البيانات)  
 فشل الادخال، 881-882  
 ملفات المدخلات  
 عمل صيغة قوائم متصلة، 351-352  
 الفتح، 888  
 مكررات الادخال، 238  
 بيانات الادخال. انظر بيانات cin  
 تدفق المدخلات: حالة الفشل، 882  
 ادخال البيانات (قراءة البيانات)، 879-881  
 بيانات cin، 879-881  
 لأهداف الفئة، 55  
 من الملفات، 887-889  
 بيانات غير صحيحة. انظر فشل الادخال  
 المقاطع، 861 جدول  
 الدالة insert (بعثرة)، 549  
 الدالة insert (قوائم)، 193-194  
 الدالات insert (اشجار البحث الثنائية)، 656، 657-658  
 أشجار AVL، 698  
 الدالات insert (مسبقة التعريف)  
 عمليات الحاوية، 222 جدول، 228 جدول، 229 جدول، 777 جدول، 782 جدول، 795  
 الدالات str.insert، 862-863 جدول  
 مكررات الادخال، 794-796  
 الدالة insertAt (القوائم)، 186-188  
 الدالة insertEnd (قوائم)، 188  
 المكرر inserter، 795  
 الدالة insertFirst (القوائم المتصلة)، 290، 297-298، 299  
 ادخال رموز في مقاطع، 862-863 جدول  
 ادخال عناصر:  
 في قوائم مبنية على مصفوفات 186-188، 193-194، في قوائم مرتبة 535-537  
 في الحاويات المترابطة:  
 الحاويات set / multiset، 776-777، map / multimap، 782  
 في أشجار AVL، 678، 679-698  
 في أشجار البحث الثنائية، 644، 656-658  
 في الحاويات deque، 233

في القوائم المتصلة، 279-282، 297-299، 335-336،  
 القوائم المتصلة من الطرفين، 324-327، القوائم المرتبة، 309، 31-314، 335-336  
 في طوابير الأسبقية، 605  
 في الحاويات vector، 219، 221، 222 جدول، 223، في النهاية 222-223  
 انظر كذلك اضافة العناصر  
 الدالة insertIntoAVL، 692-694  
 عامل الادخال (>>)، 882  
 في البيانات cout، 882-889، 887  
 الانتقال، 86، 87، 99-100، 105-106، 178، 295، 322-323  
 الأسبقية 849 جدول  
 استخدام أهداف تدفق الملف مع، 888  
 الترتيب بالادخال:  
 القوائم المبنية على مصفوفات، 566-572  
 القوائم المتصلة، 573-577  
 تحليل الأداء، 578  
 الدالة insertionSort، 572  
 الدالة insertLast (قوائم متصلة)، 290، 298-299  
 الدالة insertNode:  
 قوائم متصلة من الطرفين، 325-327  
 قوائم متصلة مرتبة، 313-314  
 الدالات insertOrd، 536-537  
 التضمين في:  
 الفئة orderedArrayListType، 537  
 تمثيل قوالب الفئة، 114  
 نوع البيانات int، 875، 876  
 انظر كذلك المتغيرات int  
 المتغيرات int  
 نوع البيانات، 875، 876  
 ادخال (قراءة) البيانات داخل، 880-881  
 مدخلات غير صحيحة، 881-882  
 مدخلات صحيحة، 881 جدول  
 انظر كذلك المتغيرات  
 الاعداد الصحيحة:  
 أنواع البيانات، 875، 876  
 العوامل، 368، 369-370  
 انظر كذلك البيانات char والبيانات int  
 أنواع البيانات المتكاملة، 875، 876  
 ملفات تمهيدية. انظر الملفات الرئيسية  
 تقاطع المجموعات، 721  
 دالة مسبقة التعريف، 827، 829-832  
 بيانات غير صحيحة، 881-882  
 استدعاء المقومات، 34  
 مقومات الفئة الأساسية، 71، 74-75

مقومات هدف العنصر، 84  
 الادخال / الاخراج (عمليات الادخال / الاخراج)  
 ادخال / اخراج ملف، 887-889  
 الملف الرئيسي iostream، 877، 887  
 انظر كذلك ادخال البيانات (قراءة البيانات) و اخراج البيانات  
 الملف الرئيسي iostream، 877، 887  
 العلاقات "is – a". انظر التكوين  
 دالة (ch) isalnum (مسبقة التعريف)، 857 جدول  
 دالة (ch) iscntrl (مسبقة التعريف)، 857 جدول  
 دالة (ch) isdigit (مسبقة التعريف)، 858 جدول  
 الدالة isEmpty (القوائم المبنية على المصفوفات)، 185-186  
 الدالة isEmpty (اشجار ثنائية)، 647  
 الدالة isEmpty (رسوم بيانية)، 729  
 دالة (عملية) isEmptyList :  
 للقوائم المتصلة، 293  
 للقوائم المتصلة من الطرفين، 321  
 دالة (عملية) isEmptyQueue ، 470:  
 للطوابير المتصلة، 486  
 للطوابير كمصفوفات، 481  
 دالة (عملية) isEmptyStack ، 410:  
 للمرصوصات المتصلة، 433  
 للمرصوصات كمصفوفات، 415  
 الدالة isFull : للقوائم المبنية على المصفوفات، 185-186  
 دالة (عملية) isFullQueue ، 470:  
 للطوابير المتصلة، 486  
 للطوابير كمصفوفات، 481  
 دالة (عملية) isFullStack ، 410:  
 للمرصوصات المتصلة، 430، 433  
 للمرصوصات كمصفوفات، 415  
 الدالة (ch) isgraph (مسبقة التعريف)، 858 جدول  
 الدالة (ch) islower (مسبقة التعريف)، 858 جدول  
 الدالة (ch) isprint (مسبقة التعريف)، 858 جدول  
 الدالة (ch) ispunct (مسبقة التعريف)، 858 جدول  
 الدالة (ch) isspace (مسبقة التعريف)، 858 جدول  
 المكررات istream، 242-243  
 الدالة (ch) isupper (مسبقة التعريف)، 858 جدول  
 الدالة (ch) isxdigit (مسبقة التعريف)، 858 جدول  
 الدالة iter\_swap (مسبقة التعريف)، 808-811  
 التكرار. انظر الاعداد  
 المكررات، 218، 223، 237-243  
 مكررات ثنائية الاتجاه، 239  
 للحاويات، 785 جدول  
 الاعلان داخل الحاويات، 223-224، 241-242

مكررات أمامية، 239  
التسلسل، 241  
مكررات ادخال، 238  
مكررات الادخال، 796-794  
المكررات istream، 243-242  
عمليات على، 237، 238 جداول، 239 جداول، 240 جدول  
المكررات ostream، 243 والدالة copy، 233-231  
مكررات الاخراج، 238  
مراجعة سريعة، 265، 266، 842  
مكررات التناول العشوائي، 240  
مكررات التدفق، 243-242  
المكررات typedef، 223، 242-241  
أنواع، 237

## J

خصائص الجافا مقابل خصائص C++، 875-902  
ضبط المخرجات، 885

## K

مقارنات المفتاح، (في الخوارزميات)، 524  
أشجار البحث الثنائي، 667  
أبحاث ثنائية، 531، 532-533، 534-535، 537 جداول  
الترتيب بالتكديس، 604  
الترتيب بالادخال، 578  
الترتيب بالدمج، 595  
الترتيب السريع، 586 جدول  
الترتيب بالاختيار، 566، 578  
الأبحاث المتتابعة، 527-528، 535، 537 جدول  
مفاتيح (عقد شجرة البحث الثنائية)، 654  
مفاتيح (عناصر مجموعة البيانات):  
لجداول البعثة، 538، 539  
للقوائم، 524  
انظر كذلك مقارنات المفاتيح (في الخوارزميات)  
الكلمات الرئيسية. انظر الكلمات المحفوظة  
مشكلة برج كونيبرج، 720

## L

اللغات:  
الرسوم الخاصة بلغة التشكيل الموحدة، 20  
انظر كذلك لغات البرمجة  
اثقال الدالة larger، 109  
القالب، 110-112  
أكبر عنصر: ايجاد منهج تكراري، 565

دالات مسبقية التعريف، 821-822، 824  
 منهج تكراري، 372-375  
 الدالة largest، 373-375  
 دالة أكبر عدد صحيح، 859 جدول  
 العنصر الأخير (من الطابور). انظر العنصر الخلفي (من الطوابير)  
 بنية بيانات الداخل أخيراً يخرج أولاً. انظر المرصوصات  
 العقد الورقسة (للأشجار الثنائية)، 637  
 قوس زاوية أيسر (>). انظر العامل "أصغر من"  
 قوس زاوية أيسر – علامة يساوي (>=). انظر العامل "أصغر من أو يساوي"  
 أقواس زاوية يسرى (>>) انظر عامل الادخال  
 أداة التحكم left، 885  
 دوران أيسر (لأشجار AVL)، 685-686، 689، 696، 698  
 دالات من أجل، 690، 691-692  
 مخرجات مضبوطة على اليسار، 885  
 ليمر دي اتش، 391  
 الدالة length (القوائم المتصلة من الطرفين)، 322  
 الدالة length (القوائم المتصلة)، 295  
 الدالة length (مسبقية التعريف):  
 الدالة str.length، 862 جدول  
 أطوال:  
 المقاطع C: التحديد، 861 جدول  
 القوائم المتصلة: التحديد، 295  
 القوائم، 180  
 العامل "أصغر من" (>)  
 معيار ترتيب الحاوية التتابعية، 774، 775  
 الاثقال، 177  
 تعريف الفئة pair، 772 جدول  
 الأسبقية، 849 جدول  
 عامل أسبقية عنصر الطابور، 494  
 اتقال عامل "أصغر من أو يساوي" (>=)، 177  
 تعريف الفئة pair، 772 جدول  
 الأسبقية، 849 جدول  
 علامات "أصغر من" (>>). انظر عامل الادخال  
 هدف الدالة less<Type>، 792 جدول  
 هدف الدالة less\_equal<Type>، 792 جدول  
 مكتبات (مجموعة مكتبة C++)، 877  
 اشارات الى. انظر الى الملفات الرئيسية  
 بنية بيانات الداخل أخيراً يخرج أولاً. انظر المرصوصات  
 أشجار البحث الثنائية الطولية، 666-667  
 التحقيق الطولي (في جداول البعثة)، 540-542  
 تحليل الأداء، 551  
 طوابير طولية. انظر الطوابير كمصفوفات  
 الأبحاث الطولية. انظر البحث التتابعي

المرصوصات الطولية. انظر المرصوصات كمصفوفات  
القوائم المتصلة، 357-273  
تمثيلات الرسم البياني لقائمة التجاور، 724-725، 726  
تعريف كنوع بيانات مجرد، 290-307، 290-293  
المميزات، 631  
عمليات أساسية، 278، 290، 336، 337  
مؤشرات التركيب الزمني، 307 جدول  
البناء من الخلف، 284، 288-289، 290  
البناء من الأمام، 284، 285-288، 290  
التحقق مما اذا كانت فارغة / ممتلئة، 293  
قوائم متصلة دائرية، 336-337  
دالات عنصر الفئة، 293-307  
مقوم النسخ، 305-306  
النسخ، 304-305، 305-306  
العمل من ملفات الادخال، 351-352  
المقوم الافتراضي، 293-294  
التعريف كنوع بيانات مجرد، 290-293  
حذف عناصر (عقد) من، 282-284، 299-303  
استمداد طوابير متصلة من، 488-490  
استمداد مرصوصات متصلة من، 439-441  
التدمير، 294  
المدمر، 305  
القسم، 589-591  
متصلة من الطرفين. انظر القوائم المتصلة من الطرفين  
العيوب، 631  
Head، 274، 275، 276، 278-279  
ذات عقد header و trailer، 335-336، 357-358  
التهيئة، 293-294، 294-295  
ادخال عناصر (عقد) في، 279-282، 297-299، 335-336  
تحديد: الأطوال، 295  
عقد. انظر العقد (للقوائم المتصلة)  
مرتّب. انظر القوائم المتصلة المرتبة  
اخراج البيانات، 295  
اثقال عامل الاسناد من أجل، 293، 306  
المؤشرات، 276-278، 278-279، 285، 289، 453-457،  
head، 274، 275، 276، 278-279  
مثال برمجة، 337-357  
خصائص، 275-278  
مراجعة سريعة، 357-358  
البحث، 296-297، 302، 303، 345-348  
الترتيب: الترتيب بالادخال، 573-577،  
الترتيب بالدمج، 587-595، الترتيب بالاختيار، 565  
الاجتياز، 278-279

الاستخدامات، 218  
 طوابير متصلة، 488-484  
 اضافة عناصر الى، 487، 508، 509  
 التحقق مما اذا كانت فارغة، 486  
 التحقق مما اذا كانت ممثلة، 486  
 التعريف كنوع بيانات مجرد، 485-484  
 حذف عناصر من، 487، 488، 508، 509  
 استمداد من قوائم متصلة، 490-488  
 تدمير، 486، 487  
 عنصر أمامي، 484، الانتاج، 487، 488، 508، 509  
 التهيئة، 488-487  
 تعريفات دالة العنصر، 488-486  
 مؤشرات من أجل، 484، 486، 487  
 العنصر الخلفي، 484، الانتاج، 487، 488  
 برنامج اختبار، 491  
 المرصوبات المتصلة، 441-428  
 اضافة عناصر الى، 436-433  
 ائقال عامل الاسناد، 439  
 التحقق مما اذا كانت فارغة 433  
 التحقق مما اذا كانت ممثلة، 430، 433  
 مقوم النسخ، 439  
 اعلان الأهداف، 439  
 المقوم الافتراضي، 432  
 التعريف كنوع بيانات مجرد، 439، 430-429  
 استمداد من قوائم متصلة، 441-439  
 تدمير، 432  
 مدمر، 439  
 كفارغ / غير فارغ، 431  
 ملفات رئيسية، 439  
 تهيئة، 432  
 ازالة عناصر من، 439-437  
 العنصر العلوي، 429، الازالة، 439-437، الانتاج 437  
 الدالة linkedInsertionSort، 577  
 تعريف الفئة linkedListGraph، 726  
 الفئة linkedListType، 726  
 مقوم النسخ، 306-305  
 المقوم الافتراضي، 294-293  
 التعريف كنوع بيانات مجرد، 293-290  
 المدمر، 305  
 والفئة linkedListGraph، 726  
 والفئة linkedQueueType، 490-488  
 والفئة linkedStackType، 441-439  
 دالات العنصر، 307-293

والفئة videoListType، 343، 345  
انظر كذلك القوائم المتصلة  
الفئة linkedQueueType،  
التعريف كنوع بيانات مجرد، 484-485  
الفئة linkedListType، 488-490  
انظر كذلك الطوابير المتصلة  
فئة linkedStackType  
المقوم الافتراضي، 432  
التعريف كنوع بيانات مجرد، 429-430  
فئة linkedListType و، 439-441  
انظر كذلك المرصوعات المتصلة  
وصلات (العقد في القوائم المتصلة)، 274  
آخر وصلة كـ NULL، 274، 276  
الفئة list، 330  
الحاويات القائمة، 330-335  
اعلان وتهيئة، 330  
ملف رئيسي، 785 جدول  
دعم المكررات، 785 جدول  
عمليات على ، 331-335، 332-331 جدول  
مراجعة سريعة، 357  
الاستخدام (التناول)، 330  
انظر كذلك الحاويات: الحاويات المتتابعة  
عمليات القائمة: مؤشرات التركيب الزمني، 195 جدول  
المؤشر list (للمرصوعات كمصفوفات)، 414  
معالجة القائمة مع مصفوفات أحادية البعد، 180-198  
انظر كذلك القوائم والطوابير والمرصوعات المتصلة  
بيانات listCont<elementType>list، 330 جدول  
دالات listCont.assign، 331 جدول  
دالات listCont.back، 331 جدول  
دالات listCont.front، 331 جدول  
دالات listCont.merge، 332 جدول  
دالة listCont.pop\_front، 331 جدول  
دالة listCont.push\_front، 331 جدول  
دالة listCont.remove، 331 جدول  
دالة listCont.remove\_if، 331 جدول  
دالة listCont.reverse، 332 جدول

دالة listCont.sort ، 331 - 332 جدول

دالة listCont.splice ، 331 جدول

دالة listCont.unique ، 331 جدول

القوائم، 180 تعريف

مبنية على مصفوفة. انظر القوائم المبنية على مصفوفات

عمليات اساسية، 180-181، 209

التعريف كنوع بيانات مجرد، 37-38، 38-39 لأنواع العناصر المختلفة 112-114

محدد العنصر. انظر عامل التتابع

الطول، 180

متصل. انظر القوائم المتصلة وكذلك القوائم المتصلة من الطرفين والقوائم المتصلة المرتبة

العمليات: مؤشرات التركيب الزمني، 195 جدول

المعالجة. انظر معالجة القائمة

البحث. انظر قوائم البحث

الترتيب. انظر خوارزميات الترتيب

انظر كذلك المصفوفات الأحادية البعد

تعريف الفئة listType كنوع بيانات مجرد، 38-39

انظر كذلك الفئة arrayListType

قالب فئة listType، 112-113

الدالة  $\log(x)$  (مسبقة التعريف)، 859 جدول

الدالة  $\log_{10}(x)$  (مسبقة التعريف)، 859 جدول

تعبيرات منطقية (تعبيرات Boolean)، 876، 889

عوامل منطقية (عوامل Boolean): الاسبقية 849 جدول

أهداف دالة مكتبة القالب المعياري المنطقية، 794 جدول

قيم منطقية (قيم Boolean)، 876، 889، 890

هدف الدالة  $\text{logical\_and} \langle \text{Type} \rangle$ ، 794 جدول

هدف الدالة  $\text{logical\_not} \langle \text{Type} \rangle$ ، 794 جدول

هدف الدالة  $\text{logical\_or} \langle \text{Type} \rangle$ ، 794 جدول

نوع البيانات long، 875

نوع البيانات long double، 876

الحلقات، 889

لوكاسيفيتس (عالم رياضيات بولندي)، 441

## M

- الدالة main، 40
- الصيغة، 879-878
- الذاكرة الرئيسية. انظر ازالة تخصيص الذاكرة: تخصيص الذاكرة
- الدالة make\_pair، 774-772
- أدوات التحكم (المخرجات)، 882
- أداة التحكم end1، 882، 886-885
- أداة التحكم fixed، 884
- أداة التحكم flush، 886-885
- أداة التحكم left، 885
- أداة التحكم right، 885
- أداة التحكم scientific، 884
- أداة التحكم setprecision، 883
- أداة التحكم setw، 885-884
- أداة التحكم showpoint، 884
- الفئة map، 770
- الحاويات map، 785-780
- الاعلان والتهيئة، 782-780
- الملف الرئيسي، 785 جدول
- ادخال وحذف عنصر، 782
- دعم المكررات، 785 جدول
- معيان الترتيب، 780، 781
- الاستخدام (التناول)، 780
- دالات حسابية (الملف الرئيسي cmath)، 860-859 جدول
- مصفوفات: تمثيلات رسم مصفوفة التجاور، 724-723
- الدالة max (مسبقة التعريف)، 821
- الدالة max\_element (مسبقة التعريف)، 822، 824
- الدالة max\_size (مسبقة التعريف): عملية الحاوية، 225 جدول، 227 جدول
- عامل تناول العنصر (عامل النقطة) (.)، 21، 22
- الأسبقية، 144، 849 جدول
- سهم عامل تناول العنصر (->)، 145-144

الأسبقية، 849 جدول

عامل محدد تناول العنصر ( : )، 17

محدد تناول العنصر، 16، 19، 80-79

محدد افتراضي، 16، 67

بيانات العنصر. انظر عناصر البيانات (الفئات)

دالات العنصر (عناصر الدالة) (لقوالب الفئة) كقوالب دالة، 114

الدالات المشتركة بين جميع الحاويات، 224، 228-227

الدالات المشتركة بين الحاويات التابعة، 228-229

فئات تناول دالات العنصر (عناصر الدالة) (الفئات) كقوالب دالة، 16-17، 19، 67، 80-79

تناول عناصر الفئة الأخرى، 16، 19

التناول، 21-22، 24، 28-30

دالات الفئة arrayListType، 185-190

دالات الفئة الأساسية. انظر دالات عنصر الفئة الأساسية

دالات الفئة binaryTreeType، 646-651

ربط، 867

دالات الفئة cAssignmentOprOverLoad، 164-166

المحدد const، 19، 25

دالات الفئة customerType، 498-499

التعريف، 16، 26-28، 82-83، 85

تعريفات: الوضع: 40، 114

دالات الفئة المستمدة، انظر دالات عنصر الفئة المستمدة

دالات الفئة graphType، 729-731

التطبيق، 25-30

قواعد التوريث، 67-68، 71

دالات الفئة linkedListType، 293-307

عمل عام أو خاص، 17

تخصيص ذاكرة، 21

دالات عامل، 95-96، 103

انقال عوامل ثنائية، 96-98

دالات الفئة pointerDataClass، 170-172

المؤشرات في، 145-147

دالات خاصة، 19، 293  
المجال، 24  
دالات الفئة serverListType، 504-507  
دالات الفئة serverType، 501-502  
دالات الفئة videoType، 341-343، 702-703  
دالات افتراضية، 867-871  
دالات الفئة waitingCustomerQueueType، 508-509  
مقومات هدف العنصر:  
استدعاء / تمرير المعطيات الى، 84  
أهداف عنصر الفئات، 80-85  
المقومات، 84  
متغيرات العنصر. انظر عناصر البيانات (الفئات)  
الذاكرة (الذاكرة الرئيسية). انظر ازالة تخصيص الذاكرة وتخصيص الذاكرة  
عناصر فئة تخصيص الذاكرة، 21  
دالات تكرارية، 390  
خلايا الذاكرة: العناوين. انظر العناوين  
الدالة merge (مسبقة التعريف)، 816-817  
عمليات حاوية القائئمة، 332 جدول  
الترتيب بالدمج  
القوائم المتصلة، 587-595  
تحليل الأداء، 595  
الدالة mergeList، 593-594  
الدالة mergeSort، 594  
دمج القوائم / القوائم الفرعية المرتبة، 591-594، 816-817  
دالة البعثرة المتوسطة المربع، 539  
الدالة min (مسبقة التعريف)، 822  
الدالة min\_element (مسبقة التعريف)، 822، 824  
خوارزمية شجرة الامتداد الأدنى، 744-754  
الدالة من أجل، 752-753  
أشجار الامتداد الأدنى، 749  
الدالة minimalSpanning، 752-753

الدالة minLocation، 563  
عامل الطرح ( - ) :  
الأسبقية، 849 جدول  
علامة ناقص ( - )  
عامل الناقص، 849 جدول  
عامل الطرح. انظر عامل الطرح  
رمز لغة التشكيل الموحدة، 20  
علامات ناقص ( - - ). انظر عامل النقص  
هدف الدالة <Type>minus، 789 جدول  
خوارزميات تعديلية (مكتبة القالب المعياري)، 787-788، 787 جدول  
برمجة وحدات، 4  
عامل المعامل (%): الأسبقية، 849 جدول  
هدف الدالة <Type>modulus، 789 جدول  
محاكاة خدمة السينما، 517-494  
تحديد الفئة، 509-496  
دالة للتشغيل، 512-511  
المعلومات المطلوبة، 511، 510  
البرنامج الرئيسي، 512-510  
تحديد الهدف، 495  
أهداف. انظر أهداف العميل: هدف قائمة مقدمي الخدمة، أهداف مقدم الخدمة، أهداف طابور العملاء  
المنتظرين  
نتائج اختبار تشغيل العينة، 517-512  
بدء خوارزمية التعامل، 511-510  
كمشتق زمني، 496  
الفئة msTreeType: تعريف كنوع بيانات مجرد، 752  
الفئة multimap، 770  
الحاويات multimap، 785-780  
الاعلان والتهيئة، 782-780  
الملف الرئيسي، 785 جدول  
ادخال وحذف عنصر، 782

دعم المكرر، 785 جدول  
معيان الترتيب، 780، 781  
الاستخدام (التناول)، 780  
توريث متعدد، 66  
انقال عامل الضرب ( \* )، 107  
الأسبقية، 849 جدول  
هدف الدالة <Type> multiplies، 789 جدول  
الحاويات multiset، 775-780  
الاعلان والتهيئة، 775-776  
الملف الرئيسي، 785 جدول  
ادخال وحذف عنصر، 776-777  
دعم المكرر، 785 جدول  
معيان الترتيب، 775، 776  
الاستخدام (التناول)، 775  
خوارزميات التحول (مكتبة القالب المعياري)، 787-788

## N

لغز العدد n من الملكات، 392-398  
اعلان ثوابت مسماة، 876  
NULL / صفر (المؤشر NULL)، 147، 860  
ثابت نوع البيانات string : : size\_type، 861  
أسماء:  
المصفوفات، 152  
المقومات، 31  
المدمرات، 35  
الدالات. انظر أسماء الدالة  
الملف الرئيسي، 41  
دالات العامل، 87  
آلية namespace، 890-895  
عنصر namespace: استخدام (تناول)، 891-892  
بيانات namespace، 891-891

تسمية المحددات، 890  
دالة اللوغاريتم العادي، 859 جدول  
موجه ما قبل المعالجة NDEBUB، 6، 857  
هدف الدالة negate<Type>، 789 جدول  
العامل new  
عمل مصفوفات حركية، 148، 151-153  
عمل متغيرات حركية، 147-149  
الأسبقية، 849 جدول  
التركيب، 148  
رمز السطر الجديد (n \): عامل الاستخلاص و، 880  
مثال البرمجة الخاص بالفئة newString، 172، 174-180  
عامل اتزان العقد (لأشجار AVL)، 678 تعريف  
تعريف struct، 678  
عقد (الأشجار الثنائية) الأشجار AVL. انظر عقد (أشجار AVL)  
المفاتيح: في أشجار البحث الثنائية، 654  
المستويات، 637  
مؤشر / الى، 636، 642  
جذر. انظر العقدة الجذرية.  
تعريف struct، 636  
تتابعات مخرجات الاجتياز، 642  
الأنواع، 632-633، 637  
تحديث البيانات في، 672-675  
عقد (القوائم المتصلة)، 274-278  
المكونات، 274، 276  
العمل، 279-280  
ازالة تخصيص الذاكرة من أجل، 294، 305، 322  
التعريف، 275، 279  
الحذف، 282-284، 299-303:  
من قوائم متصلة من الطرفين، 327-329  
من قوائم متصلة مرتبة، 315-316  
الادخال: في قوائم متصلة من الطرفين، 324-327،

العقدة الأولى، 290، 297-298  
 العقدة الأخيرة، 290، 298-299،  
 في قوائم متصلة مرتبة، 309، 310-314  
 اخراج البيانات، 295: من قوائم مرتبة بترتيب عكسي، 308  
 مؤشرات الى: التقديم، 277، 278-279  
 انتاج بيانات أول عقدة، 295-296:  
 من قوائم متصلة من الطرفين، 324  
 انتاج بيانات آخر عقدة، 296  
 من قوائم متصلة من الطرفين، 324  
 عقد (مرصوصات متصلة) : ادخال، 434-435  
 دالات لغير العنصر. انظر الدالات الصديقة  
 خوارزميات غير تعديلية (مكتبة القالب المعياري)، 786-787، 787 جدول  
 رموز غير قابلة للطبع (مجموعة رموز الكود المعياري الأمريكي لتبادل المعلومات) ،  
 851-852 جدول  
 دالات غير تكرارية: طباعة قوائم متصلة مرتبة بترتيب عكسي، 452-460  
 الدالة nonRecursiveInTraversal، 669  
 الدالة nonRecursivePostTraversal، 672  
 الدالة nonRecursivePreTraversal، 670  
 عامل "لا يساوي". انظر عامل عدم التساوي  
 العامل لا ( ! ) : الأسبقية 849 جدول  
 هدف الدالة <Type> not\_equal\_to، 791 جدول  
 Npos. انظر الثابت المسمى npos : : string  
 فئة nQueensPuzzle، 396-398  
 الثابت المسمى NULL (المؤشر صفر)، 147، 860  
 المؤشر الصفري (./ NULL)، 147، 860  
 علامة الرقم ( # )  
 رمز رقم التعبير postfix، 446  
 رمز موجه ما قبل المعالجة، 877  
 رمز لغة التشكيل الموحدة، 20  
 الأعداد:  
 معالجة عدد مركب، 103-108

أنواع البيانات. انظر أنواع البيانات ذات العلامة العشرية، أنواع البيانات المتكاملة حسابات عدد فيبوناكي، 379-383  
انظر كذلك البيانات char والبيانات int  
خوارزميات عددية (مكتبة القالب المعياري)، 788، 835-839  
أنواع بيانات رقمية. انظر أنواع البيانات ذات العلامة العشرية، أنواع البيانات المتكاملة

## O

مؤشر التركيب الزمني ( $O(g(n))$  للدالات / الخوارزميات)، 14-15، 15 جدول  
لخوارزميات البحث القائمة على المقارنة، 537-538، 537 جدول  
لخوارزميات الترتيب القائمة على المقارنة، 578-579، 578 جدول، 586 جدول، 595  
لعمليات القائمة، 195 جدول، للقوائم المتصلة، 307 جدول  
لعمليات المرصوعات كمصفوفات، 423 جدول  
الامتداد .obj، 878  
كود الهدف (برنامج الهدف)، 40، 878  
ملفات كود الهدف: الامتداد، 878  
التصاميم القائمة على الهدف، 4  
المبادئ الأساسية، 4، التغليف، 4 تعريف، 85. انظر كذلك التوريث وتعدد الأشكال  
أنواع بيانات. انظر أصعب خطوات الفئات، 55  
انظر كذلك التكوين  
برمجة قائمة على الهدف. انظر التصميم القائم على الهدف  
لغات البرمجة القائمة على الهدف، 4  
اهداف. انظر أهداف الفئة  
نوع البيانات ofstream، 887  
مصفوفات أحادية البعد، 900  
مصفوفات ثابتة، 902  
الاعلان، 900، كمعاملات رسمية، 901، 902،  
مع المؤشرات، 148، 151-153  
عناصر. انظر عناصر المصفوفة (مصفوفات أحادية البعد)  
مؤشرات. انظر مؤشرات المصفوفة  
التهيئة، 901

معالجة القائمة بواسطة، 180-198  
التمرير كمعاملات للدالات، 901-902  
انظر كذلك القوائم  
اختيار في اتجاه واحد: بيانات if، 889  
OOD. انظر التصميم القائم على الهدف  
OOP. لغات (البرمجة القائمة على الهدف)، 4  
انظر كذلك التصميم القائم على الهدف  
op= (عوامل اسناد مركب): الأسبقية، 849 جدول  
المعالجة المفتوحة (البعثة المغلقة)، 540 تعريف، 540-549  
أساليب التطبيق، 540-545  
حذف عنصر، 545-547  
مراجعة سريعة، 553-554  
الدالة open (مسبقة التعريف)، 888  
بعثة مفتوحة، انظر التسلسل  
فتح الملفات، 888، 889  
عمليات (على البيانات)، 4  
عملية مسيطرة في الخوارزميات، 12  
اخفاء التفاصيل، 26، 40-44  
التحديد، 4، 55-56  
خاصية نوع بيانات العمليات، 37  
دالات العامل، 87 تعريف:  
لمعالجة مقاطع C، 174-180  
في تعريفات الفئة، 88، 95-96، 103  
كدالات عنصر، 96-98، 103  
أسماء، 87  
كدالات لغير العنصر، 98-99، 103  
التركيب، 87  
انظر كذلك الدالة \*operator: ودالات عامل محدد آخر  
انقال عامل، 86-103، 172-173  
عوامل حسابية، 96-99  
عامل مؤشر المصفوفة، 172-173

عامل الاسناد، 162-166:

للقوائم المبنية على مصفوفات، 191

للأشجار الثنائية، 651، للمقاطع C، 176-177، للقوائم المتصلة، 293، 306

للتوابير المتصلة، 488، للمرصوصات المتصلة، 439

للمرصوصات كمصفوفات، 422

عوامل ثنائية، 96-103

عامل الاستخلاص، 99-100، 100-101، 108، 178

دالات من أجل. انظر دالات العامل

عامل ادخال، 99-100، 105-106، 178، 295، 322-323

حاجة الى، 86-87:

عوامل يمكن ائصالها، 855 جدول

عوامل لا يمكن ائصالها، 88، 855 جدول

أمثلة برمجة، 103-108، 174-180، 606-626، 702-703

مراجعة سريعة، 115-116

عوامل مترابطة، 96-99، 177-178، 702-703

قواعد (قيود)، 88، 95-96

عوامل أحادية، 103

الدالة \*operator، 107

الدالة [ ] operator، 172-173، 177

الدالة = operator، 162-166، 176-177، 293، 306، 422، 439، 488، 651

الدالة == operator، 97-98، 108، 177

الدالة != operator، 177-178

دالة > operator، 177

دالة >= operator، 177

دالة >> operator، 99-100، 100-101، 108، 178

دالة < operator، 177

دالة <= operator، 177

دالة << operator، 99-100، 105-106، 178، 295، 322-323

انقال ( ) operator، 789

دالة + operator، 97-98، 109

العوامل:

عنوان العامل. انظر عنوان العامل.  
حسابي. انظر عوامل حسابية  
مؤشر مصفوفة. انظر عامل مؤشر المصفوفة  
اسناد. انظر عامل الاسناد  
الترابط، 849 جدول  
ثنائي. انظر عوامل ثنائية  
عوامل فئة مطابقة للهدف، 22، 86، 87  
عوامل اسناد ملاكبة، 849 جدول، 876  
نقص. انظر عامل النقص  
العامل delete، 149  
ازالة الاشارة. انظر عامل ازالة الاشارة  
استخلاص. انظر عامل الاستخلاص  
زيادة. انظر عامل الزيادة  
ادخال. انظر عامل الادخال  
منطقي. انظر العوامل المنطقية  
تناول عنصر. انظر عامل تناول العنصر (عامل النقطة)  
سهم عامل تناول العنصر، 144-145، 849 جدول  
عامل محدد تناول العنصر، 17  
العامل new، 147-149، 151-153  
اثقال. انظر اثقال العامل  
أسبقيّة. انظر أسبقيّة (العوامل)  
ترابطي. انظر العوامل الترابطية  
عامل حل المجال، 26، 849 جدول، 891  
أحادي. انظر العوامل الأحادية  
الفئة OpOverClass، 95، 96  
اثقال عوامل من أجل 101-103. انظر كذلك اثقال العامل  
أو العامل (||): الأسبقيّة، 849 جدول  
ترتيب (الدالات / الخوارزميات). انظر مؤشر التركيب الزمني  $O(g(n))$   
ترتيب الأسبقيّة. انظر أسبقيّة (العوامل)  
تعريف القوائم المبنية على الهدف كنوع بيانات مجرد، 529  
ادخال عناصر في، 535-537

قوائم متصلة مرتبة، 318-307  
عمليات أساسية، 308-307، 336  
دالات عنصر الفئة، 308، 316-309  
التعريف كنوع بيانات مجرد، 308  
حذف عناصر (عقد) من، 316-315  
عمل ملفات رئيسية، 318-316  
ادخال عناصر (عقد) في، 309، 310-314، 336-335  
الدمج، 594-591، 817-816  
عقد. انظر العقد السفلية (للقوائم المتصلة)  
اخراج البيانات بترتيب عكسي، 308  
الطباعة بترتيب عكسي، 376-379، 460-452  
البحث، 310-309  
تعريف الفئة `orderedArrayListType` كنوع بيانات مجرد، 529  
تضمين دالات في، 537، 564  
تعريف الفئة `orderedLinkedListType` كنوع بيانات مجرد، 308  
ملف رئيسي، 318-316  
انظر كذلك القوائم المتصلة المرتبة  
المكررات `ostream`، 243  
ودالة النسخ `copy`، 233-231  
مؤشرات خارج الحدود (المصفوفات)، 901  
المخرجات:  
التشكيل. انظر تشكيل المخرجات  
توليد. انظر توليد البيانات  
ملفات المخرجات: فتح، 888  
مكررات المخرجات، 238  
أدوات التحكم في المخرجات. انظر أدوات التحكم  
بيانات المخرجات. انظر بيانات `cout`  
تدفق المخرجات: اخلاء الفاصل، 886-885  
اخراج البيانات  
اخلاء فاصل تدفق المخرجات، 886-885  
في أعمدة، 885-884

بيانات cout، 889-882  
الى ملفات، 889-887  
من قوائم متصلة، 295، قوائم متصلة من الطرفين بترتيب عكسي، 323،  
قوائم متصلة مرتبة بترتيب عكسي، 308  
وضع زيادة (مقادير جدول البعثة)، 539، 550  
وضع زيادة (المرصوعات كمصفوفات) التحقق منها، 417-418  
الاثقال:

عامل استدعاء الدالة، 789  
الدالات. انظر اثقال دالة  
عوامل. انظر اثقال العامل  
سيطرة دالات عنصر الفئة الأساسية في فئات مستمدة، 69-70

## P

الفئة pair، 769، 774-770  
عوامل ترابطة من أجل، 771، 772 جدول  
الاستخدام (التناول)، 770  
النوع pair، 772  
أزواج (أهداف زوجية)  
مقارنة، 771-772  
عمل، 772-774  
اعلان، 770  
حواف متوازية (في رسوم بيانية)، 723  
قوائم معاملات. انظر قوائم المعاملات، وقوائم المعاملات الرسمية  
تمرير معامل. انظر تمرير المعاملات: الى دالات  
أنواع ذات معاملات (من الفئات): قوالب الفئة 112 كمعاملات (معطيات) للدالات  
حقيقي. انظر المعاملات الحقيقية  
دالات ذات: معاملات افتراضية، 898-899  
رسمي. انظر معاملات رسمية  
تمرير. انظر تمرير المعاملات الى الدالات  
معاملات المحاكاة، 510  
العقد الأصلية (للأشجار الثنائية)، 632-637

جملة اعتراضية (( ))  
في تعبيرات حسابية، 441  
الأسبقية، 849 جدول  
في اعلانات هدف الفئة، 34  
في قوائم المعاملات الرسمية، 896  
الدالة partial\_sum (مسبقة التعريف)، 835، 838-839  
الدالة partition، 585  
الترتيب بالتقسيم. انظر الترتيب بالدمج والترتيب السريع  
تقسيم القوائم المبنية على مصفوفات، 580-585  
الفئة partTimeEmployee، 75-77  
تمرير المعاملات:  
الى قوالب فئة، 114، 727-728  
الى دالات، 114، مصفوفات، 901-902،  
أهداف فئة، 24-25، 167-168، أهداف فئة مستمدة الى معاملات فئة اساسية،  
865-871، دالات، 672-675، 825-827، في الجافا، 898، المؤشرات، 153، أهداف المرصوصة  
422  
الى مقومات هدف العنصر، 84  
مسارات (في أشجار ثنائية) 637  
مسارات (في رسوم بيانية) 723  
خوارزمية أقصر مسار، 737-738، 743-738  
مسارات بسيطة، 723  
وزن المسار، 737  
علامة مئوية (%) : عامل الباقي، 849 جدول  
أشجار بحث ثنائية فائقة الاتزان، 675-676 تعريف  
مدة (عامل النقطة). انظر عامل تناول العنصر  
الفئة personalInfoType، 83-85  
الفئة personType والفئة customerType، 351  
تعريفات، 43-44، 89-90  
تعريفات عنصر الدالة، 90-91  
وفئة partTimeEmployee، 75  
العنصر pivot (قوائم مبنية على مصفوفات)، 580-582، 583، 584، 585

عامل الزائد (+): الأسبقية، 849 جدول  
 علامة زائد (+)  
 عامل الجمع. انظر عامل الجمع  
 عامل الزيادة، 849 جدول  
 رمز لغة التشكيل الموحدة، 20  
 علامات زائد (+ +). انظر عامل الزيادة  
 هدف الدالة <Type>plus، 789 جدول  
 علامة زائد – يساوي (=+): عامل اسناد مركب، 849 جدول  
 مؤشر حسابي، 151، 207  
 عناصر بيانات مؤشر (الفئات):  
 الحدود (الخصوصيات)، 157-172، 646  
 نوع بيانات المؤشر، 133، 134  
 نوع بيانات pointer، 242 جدول  
 متغيرات المؤشر. انظر المؤشرات  
 الفئة pointerDataClass، 157-158  
 مقوم النسخ، 166-167، 169-170، 172  
 المدمر، 158-159  
 دالات العنصر، 170-172  
 مؤشرات، 134-151، 134 تعريف  
 تناول عناصر الفئة / البنية عبر 144-145 عمليات حسابية  
 على 151  
 عمليات أساسية، 135-143 من / الى عقد شجرة ثنائية 636، 642  
 في دالات عنصر الفئة 145-147  
 مقارنة 150  
 مؤشرات ثابتة 152  
 نسخ 149-150  
 كنوع بيانات (الفئات): الحدود (الخصوصيات) 157-172، 646  
 الاعلان 134-135، 142  
 اعلان كمعاملات للدالات 153  
 اعلان مصفوفات أحادية البعد 148، 151-153  
 التقليل 150-151

والنسخ العميق 156-157، 209  
الى مصفوفات حركية 148، 151-153، 158-159  
الى متغيرات حركية 147-149  
الى دالات 672  
الزيادة 150-151  
التهيئة 147  
للقوائم المتصلة 276-278، 278-279، 285، 289، 453-457،  
تقديم 277، 278-279، head 274، 276، 278-279  
للطوابير المتصلة 484، 486، 487  
التحكم في البيانات 139-143  
عمليات على 149-151  
تمرير كمعاملات للدالات 153  
تمرير أهداف فئة مستمدة الى معاملات مؤشر فئة أساسية 868-869  
للطوابير 471  
للطوابير كمصفوفات 471، 473، 477، التقديم 475  
مراجعة سريعة 207-209  
كانتاج قيمة 154  
ونسخ سطحي 154-156، 638  
للمرصوعات كمصفوفات 414  
تخزين عناوين في 135-143  
المؤشر this 88-89  
غير مهياً 159  
انظر كذلك مؤشرات محددة للمكررات  
توزيع بويرون 511  
ترميز بولندي 441  
تعدد الأشكال 4 تعريف  
انظر كذلك اثقال الدالة، اثقال العامل، والدالات الافتراضية  
عمليات متعددة الحدود: أمثلة برمجة 198-206  
الدالات pop (مسبقة التعريف) عملية الطابور 492 جدول  
عمليات المرصوعة 409، 410، 411، 461 جدول  
للمرصوعات المتصلة 437-439، للمرصوعات كمصفوفات 418-420

الدالة pop\_back (مسبقة التعريف): عملية حاوية 222 جدول، 229 جدول  
الدالة pop\_front (مسبقة التعريف): عملية حاوية 234 جدول، 331 جدول  
شروط تالية (للدالات) 5-6

تحديد 6-7

تعبيرات postfix 441-446، 462

عوامل حسابية في 441، 446

التقييم باستخدام مرصوصات 442-446 وانظر كذلك

خوارزمية تقييم برنامج حاسب التعبيرات postfix 442

في مقابل التعبيرات infix 442 جدول

برنامج حاسب التعبيرات postfix 446-452

معالجة الخطأ 446، 448

خوارزمية أساسية 447-448

مخرجات 446

تحديد برنامج 449-452

ادخال / اخراج عينة 452

الحفاظ على المرصوصة 446

ترميز postfix 441، 462

انظر كذلك التعبيرات postfix

عوامل ما بعد الزيادة / النقص: الأسبقية 849 جدول

الدالة postorder (اجتياز شجرة ثنائية) 643

اجتياز ما بعد الترتيب (لشجرة ثنائية) 640

تتابع مخرجات العقدة 642

خوارزمية / دالة غير تكرارية 671-672

دالة تكرارية 643

علامة الجنيه (#)

رمز عدد التعبير postfix 446

رمز موجه ما قبل المعالجة 877

الدالة pow (x, y) (مسبقة التعريف) 860 جدول

أسبقية (العوامل) 849 جدول

عامل ازالة الاشارة 144، 849 جدول

عامل النقطة 144

دقة (قيم):

الاعداد ذات العلامة العشرية 883

الشروط المسبقة (للدالات) 5-6

التحديد 6-7

المصفوفة predCount 755، 756، 757، 758، 759، 760

دالات مسبقة التعريف مذكورة 857 جدول، 857-859 جدول، 859-860 جدول

الاستخدام (التناول) 877-878

أصول (أهداف الدالة) 794

ترميز prefix 441

عوامل ما قبل الزيادة / النقص: الأسبقية 849 جدول

الدالة (preorder اجتياز الشجرة الثنائية) 640

اجتياز ما قبل الترتيب (لشجرة ثنائية) 640

تتابع مخرجات العقدة 642

خوارزمية / دالة غير تكرارية 670

دالة تكرارية 643

موجهات ما قبل المعالجة 877-878، 878

في الملفات الرئيسية للفئة المستمدة 77

857 NDEBUG

التركيب 877

خوارزمية بریم. انظر خوارزمية شجرة الامتداد الادنى

تجميع أولي (في جداول البعثة) 541-542

التحقيق و، 543، 545

الدالة print (قوائم مبنية على مصفوفات) 186

الدالات print (baseClass/ derivedClass) 865-871

الدالة printGraph 731

طباعة: قوائم مبنية على مصفوفات 186

قوائم متصلة من الطرفين بترتيب عكسي 323

رسوم بيانية 731

قوائم متصلة 295، قوائم متصلة من الطرفين بترتيب عكسي 323، قوائم متصلة مرتبة بترتيب

عكسي 376-379، 452-460

أشجار موزونة 753-754

الدالة print:istReverse: قوائم متصلة مرتبة، 378-379

الدالة printShortestDistance، 743

الدالة printTreeAndWeight

طوابير الأسبقية، 493-494، 605-606

انظر كذلك الطوابير

فئة priority\_queue (مكتبة القلب المعياري)، 494

عناصر الفئة الخاصة، 16، 17، 79-80

عناصر البيانات. انظر عناصر البيانات الخاصة

الاعلان 19

عناصر الدالة 19، 293

قواعد التوريث 67-68، 71

رمز لغة التشكيل الموحدة 20

عناصر البيانات الخاصة 17

تناول عناصر الفئة الأساسية في فئات مستمدة 68، 70، 872-874

التناول مع الدالات الصديقة 92-95

الاعلان 19

منع المبرمجين من التلاعب 874

عناصر الدالة الخاصة 19

التوريث الخاص 67، 80

امكانية حدوث عدد y من الأحداث في زمن محدد 511

تتابع التحقيق (في جداول البعثة)

مع البعثة المزدوجة 545

مع التحقيق الطولي 540

مع التحقيق التربيعي 542-543، 543-544

مع التحقيق العشوائي 542

حل المشكلة:

بالتكرار 372-389

التكرار في مقابل المكررات 390

تصميم البرنامج 3-4

انظر كذلك التصميم المبني على الهدف

بنية البرنامج 878-879

انظر كذلك تصميم البرنامج

أنواع البيانات المعرفة بواسطة المبرمج

انظر أنواع بيانات العد

البرمجة:

برمجة وحدات 4

نصوص الموارد 903

أمثلة البرمجة:

القوائم المبنية على مصفوفات 198-206

ماكينة الحلوى 45-55

الفئات 45-55

الأعداد المركبة 103-108

التكوين 243-264

مصفوفات حركية 198-206

نتائج الانتخابات 606-626

تقرير الدرجات 243-264

أعلى درجات 425-428

التوريث 243-264

قوائم متصلة 337-357

الفئة newString 174-180

انتقال العامل 103-108، 174-180، 606-626، 702-703

عمليات متعددة الحدود 198-206

خوارزميات الترتيب 606-626

المرصوعات 425-428

الحاويات vector 243-264

متجر الفيديو 337-357، 701-709

لغات البرمجة:

مستوى رفيع. انظر برامج لغة المستوى الرفيع

لغات قائمة على الهدف 4

برامج (برامج حاسوب) C++. انظر برامج C++ التصحيح 7

التصميم 3-4. انظر كذلك التصميم القائم على الهدف  
مراحل التطوير 2، 3-8  
لغة المستوى الرفيع. انظر برامج لغة المستوى الرفيع  
التطبيق 5-7  
دورة الحياة 2  
عملية الصيانة 2  
تحليل المتطلبات 3  
البنية 878-879  
الاختبار. انظر برامج الاختبار  
عناصر الفئة المحمية 79-80، 185، 293  
رمز لغة التشكيل الموحدة 20  
التوريث المحمي 80  
عناصر الفئة العامة 16-17، 17، 79-80  
التناول 24  
الاعلان 19  
قواعد التوريث 67-68  
رمز لغة التشكيل الموحدة 20  
التوريث العام 79-80  
دالات push (مسبقة التعريف):  
عملية طابور 492 جدول  
عمليات مرصوفة 409-410، 411، 461 جدول  
للمرصوفات المتصلة 433-436  
للمرصوفات كمصفوفات 415-418  
الدالة push\_back (مسبقة التعريف): عملية حاوية  
222-223، 222 جدول، 229 جدول، 795  
الدالة push\_front (مسبقة التعريف): عملية الحاوية، 234 جدول، 331 جدول، 795

## Q

تحقيق تربيعي (في جداول البعثة) 542-545  
تطبيق البعثة مع، 547-549  
تحليل الأداء 552

والتجميع الأولي 543-545  
علامة استفهام – نقطتان (؟): عامل شرطي 849 جدول  
الفئة queue (متبة القالب المعياري) (مكيف حاوية) 492-493  
العمليات المدعمة 492 جدول  
الملف الرئيسي للطاير 494  
المؤشر queueFront 471  
للتواير المتصلة 484، 486، 487  
للتواير كمصفوفات 471، 473، 477 التقديم 475  
المؤشر queueRear 471  
للتواير المتصلة 484، 487  
للتواير كمصفوفات 471، 473، 477 التقديم 475  
التواير 469-517، تعريف 470  
اضافة عناصر الى 470، 471  
كمصفوفات. انظر التواير كمصفوفات  
عمليات أساسية 470-471  
التحقق مما اذا كان فارغ / ممتلئ 470  
في محاكاة الحاسوب 495  
انظر كذلك محاكاة السينما  
كمكيفات للحاوية 492-493  
تدمير 470  
مزوجة النهاية. انظر الحاويات deque  
حركي. انظر التواير المتصلة  
العنصر الأمامي 470، 471  
الانتاج 470  
ملف رئيسي (مكتبة القالب المعياري) 494، 785 جدول  
تهيئة 470  
ادخال عناصر في تواير الأسبقية 605  
طولي. انظر التواير كمصفوفات  
متصل. انظر التواير المتصلة  
مؤشرات من أجل 471  
تواير الأسبقية 493-494، 605-606

مراجعة سريعة 517  
عنصر خلفي 470، 471  
الانتاج 471  
ازالة عناصر من 470، 470  
طوابير مؤقتة 509  
استخدامات 218، 469  
الطوابير كمصفوفات 471-484  
اضافة عناصر الى 471-472، 482  
مشكلة الظهور الممتلئ 473-475  
التحقق مما اذا كان ممتلئ / فارغ 481  
كدائري 473-474  
مقوم 483  
عد العناصر 477  
التعريف كنوع بيانات مجرد 478-480  
حذف عناصر من 471، 472-473، 482-483  
تدمير 477، 481  
مدمرات 483-484  
التمييز بين الطوابير الفارغة والممتلئة 475-478  
العنصر الأمامي 471  
الانتاج 481-482  
تهيئة 480، 477-481  
مؤشرات من أجل 471، 473، 477  
تقديم 475  
عنصر خلفي 471  
الانتاج 482  
تعريف الفئة queueType كنوع بيانات مجرد 478-480  
و  
الفئة waitingCustomerQueueType 507-508  
أنظمة عمل الطابور 495 تعريف  
التصميم 495-496  
انظر كذلك محاكاة السينما

مراجعات سريعة:  
حاويات مترابطة 842  
أشجار ثنائية 711-710  
قوالب الفئة 117-116  
الفئات 58-57  
الحاويات 842، 841، 357، 266-265  
الحاويات deque 266-265  
دالات صديقة 116  
أهداف دالة 842  
رسوم بيانية 763-762  
توريث 115  
مكررات 842، 266، 265  
قوائم متصلة 358-357  
حاويات القائمة 357  
انقال عامل 116-115  
مؤشرات 209-207  
طوابير 517  
تكرار 399-398  
خوارزميات بحث 555-552  
حاويات متتابعة 357، 266-265  
مبادئ هندسة البرمجيات 57-56  
خوارزميات الترتيب 626، 555-552  
مرصصات 462  
خوارزميات مكتبة القالب المعياري 843-842، 841  
قوالب 116  
حاويات vector 265  
الترتيب السريع:  
قوائم مبنية على مصفوفات 586-579  
مقابل الترتيب بالتكدس 605  
تحليل الأداء 586  
الدالة QuickSort 586

علامات تنصيب. انظر علامة التنصيب المزدوجة

## R

RAM (ذاكرة التناول العشوائي)

انظر ازالة تخصيص الذاكرة وتخصيص الذاكرة

مكررات التناول العشوائي 240

تحقيق عشوائي (في جداول البعثة) 542

والتجميع الأولي 545

الدالة random\_shuffle (مسبقة التعريف) 822-823، 824

الدالة rbegin (مسبقة التعريف): ملية حاوية 228 جدول، 231

بيانات قراءة. انظر البيانات cin

قراءة بيانات. انظر ادخال البيانات

العنصر الخلفي (من الطوابير) 470، 471

طوابير متصلة 484، انتاج 487، 488

الطوابير كمصفوفات 471

الانتاج 482

المؤشر rear. انظر المؤشر queueRear

الدالة recMergeSort 594

اعادة بناء أشجار AVL. انظر دوران أشجار AVL

سجلات. انظر structs

الدالة recQuickSort 586

التكرار 369-377، 368 تعريف

خوارزميات. انظر خوارزميات تكرارية

تعريفات 368-369

تكرار مباشر / غير مباشر 371

دالات. انظر دالات تكرارية

تكرار غير محدود 371

مقابل المكررات 390

حل المشكلة مع 372-389

مراجعة سريعة 398-399

خوارزميات تكرارية 369، 372-373

خوارزميات التتبع الخلفي 391-398، 399  
تعريفات تكرارية 368-369  
استدعاءات دالة تكرارية 369، 390  
دالات تكرارية 369-390  
دالة height للشجرة الثنائية 638  
استدعاءات الى 369، 390  
تصميم 371  
حسابات عدد فيبوناكي 379-383  
بحث أكبر عنصر 372-375  
تخصيص ذاكرة من أجل 390  
الترتيب بالدمج 594  
طباعة قوائم متصلة مرتبة بترتيب عكسي 376-379  
حل المشكلة مع 372-389  
الترتيب السريع 586  
حل مشكلة برج هانوي 383-387  
اعادة تعريف دالات عنصر الفئة الأساسية في الفئات المستمدة 69-70  
رمز معامل الاشارة (&) 153، 896، 897، 901  
الوضع مع عامل ازالة الاشارة 153  
معاملات اشارة 897 تعريف، 897-898  
ثابت. انظر معاملات اشارة ثابتة  
اعلان معاملات رسمية ك 153  
تمرير المصفوفات ك 901-902  
تمرير أهداف الفئة ك 25  
تمرير أهداف فئة مستمدة الى معاملات فئة أساسية 865-868  
تمرير مؤشرات ك 153  
رمز 153، 896، 897، 901  
في الدالات المنتجة لقيمة 898  
نوع البيانات reference 242 جدول  
اعادة البعثرة 542  
العامل reinterpret\_cast: الأسبقية 849 جدول  
عوامل ترابطية 889

اثقال 96-99، 177-178، 702-703

لأهداف الفئة pair 771، 772 جدول

الأسبقية 849 جدول

أهداف فئة مكتبة القالب المعياري الترابطية 791-794، 791-792 جدول

عامل الباقي (%)

الأسبقية 849 جدول

الدالة remove (قوائم) 194

الدالة remove (مسبقة التعريف) 802-806

عملية حاوية 331 جدول

الدالة remove\_copy (مسبقة التعريف) 802-806

الدالة remove\_copy\_if (مسبقة التعريف) 802-806

الدالة remove\_if (مسبقة التعريف) 802-806

عملية حاوية 331 جدول

الدالة removeAt (قوائم) 188-189

ازالة عناصر. انظر حذف (ازالة) عناصر

الدالة rend (مسبقة التعريف) عملية حاوية 228 جدول

حلقات اعادة (مكرر) 889

مقابل التكرار 390

الدالة replace (مسبقة التعريف) 806-808

الدالة strl.replace 863 جدول

الدالة replace\_copy (مسبقة التعريف) 806-808

الدالة replace\_copy\_if (مسبقة التعريف) 806-808

الدالة replace\_if (مسبقة التعريف) 806-808

الدالة replaceAt (قوائم) 189

استبدال عناصر 806-808

فيقوائم مبنية على مصفوفات 189

رموز في مقاطع 863 جدول

تحليل المتطلبات (للبرامج) 3

كلمات محفوظة: مذكورة 847

الدالات resize (مسبقة التعريف):

عمليات حاوية vector 222 جدول، 229 جدول

الدالة retrieveAt (قوائم) 189  
استرجاع عناصر:  
من قوائم مبنية على مصفوفات 189  
انظر كذلك الانتاج  
انتاج قيم (من دالات منتجة للقيمة):  
محدد نوع الانتاج const 874  
مؤشرات ك 154  
انتاج أكثر من واحد 770، 898  
انتاج:  
قيم هدف الفئة 89-90  
بيانات العقدة الأولى (من القوائم المتصلة) 295-296  
القوائم المتصلة من الطرفين 324  
العنصر الأمامي (من الطوابير) 470  
الطوابير المتصلة 487، 488، 508، 509  
الطوابير كمصفوفات 481-482  
دالة تنتج قيم. انظر انتاج قيم (من دالات منتجة للقيمة):  
بيانات العقدة الأخيرة (من القوائم المتصلة) 296  
القوائم المتصلة من الطرفين 324  
عناوين عنصر بيانات خاص 872-874  
العنصر الخلفي (من الطوابير) 471:  
الطوابير المتصلة 487، 488  
الطوابير كمصفوفات 482  
مقاطع فرعية 862 جدول  
العنصر العلوي (من المرصوصات) 409، 410، 411، 437، 461 جدول  
المرصوصات المتصلة 437  
الدالة reverse (مسبقة التعريف) 818-821  
عملية حاوية القائمة 332 جدول  
ترميز بولندي عكسي. انظر ترميز postfix  
الدالة reverse\_copy (مسبقة التعريف) 818-821  
الدالة reversePrint :  
قوائم متصلة من الطرفين 323

قوائم متصلة مرتبة 378-376

عكس العناصر 821-818

قوس زاوية أيمن (<). انظر عامل "أكبر من"

قوس زاوية أيمن – علامة يساوي (<=). انظر عامل "أكبر من أو يساوي"

أقواس زاوية يمني (<<). انظر عامل الاستخلاص

أداة التحكم 885 right

دوران الى اليمين (لأشجار AVL) 685-686، 689، 695، 696، 698

دالات من أجل 690-691، 692

العقدة الجذرية (للأشجار الثنائية) 632

المفتاح: في أشجار البحث الثنائية 654

مؤشر الى 636

أشجار ذات جذور (رسوم بيانية) 746

الدالة rotate (مسبقة التعريف) 821-818

الدالة rotate\_copy (مسبقة التعريف) 821-818

الدالة rotateToLeft 690

الدالة rotateToRight 690، 691

دوران أشجار AVL 685-692

أمثلة 681-682، 683، 685، 695، 696، 697-698

دالات 690-692

أنواع 685-690

دوران العناصر 821-818

الربط في زمن التشغيل 867

الدالة runSimulation: خوارزمية 511-512

## S

أداة التحكم 884 scientific

ترميز (تشكيل) علمي:

اخراج أعداد ذات علامة عشرية في 883، 884

المجال: عناصر الفئة 24

عامل حل المجال ( : ) 26، 891

الأسبقية 849 جدول

خوارزميات البحث 523-555:  
أشجار البحث الثنائية 655-656  
بحث ثنائي 529-533، 535  
عملية مسيطرة 12-13  
مقارنات مفاتيح في. انظر مقارنات المفتاح (في الخوارزميات)  
ترتيب (الحد الأدنى):  
خوارزميات مبنية على المقارنة 537، 537 جدول، 538  
خوارزمية الترتيب 1 (انظر البعثرة)  
الأداء 523  
تحليل الأداء 12، 13، 527-528، 533-535  
مراجعة سريعة 552-555  
بحث تنابعي 191-192، 296-297، 302، 348، 526-527  
انظر كذلك البعثرة: دالات البحث  
الدالة search (أشجار بحث ثنائية) 655، 656  
الدالة search (قوائم متصلة) 296-297  
الدالة search (مسبقة التعريف) 811-815  
دالات بحث:  
أشجار البحث الثنائية 655-656  
بحث ثنائي 531، 812-813، 814  
دالات مسبقة التعريف 811-815  
بحث تنابعي 193، 296-297، 303، 348  
الدالة search\_n (مسبقة التعريف) 811-815  
بحث. انظر البحث الثنائي والبحث التتابعي  
بحث:  
أشجار البحث الثنائية 655-656  
جداول البعثرة 550  
قوائم. انظر قوائم البحث  
انظر كذلك العثور على عناصر  
بحث القوائم: خوارزميات من أجل. انظر خوارزميات البحث  
قوائم مبنية على المصفوفات 191-193  
دالات من أجل. انظر دالات البحث

قوائم متصلة 296-297، 302، 303، 345-348

قوائم متصلة مرتبة 309-310

تجميع ثانوي (في جداول البعثة) 545

مخزون فرعي: ادخال / اخراج ملف من / الى 887-889

اختيار: بنيات التحكم 889

الترتيب بالاختيار:

القوائم المبنية على مصفوفات 560-566

القوائم المتصلة 565

تحليل الأداء 566، 578

برنامج اختبار 564-565

الدالة selectionSort 564

التضمين في الفئة orderedArrayListType 564

فاصلة منقوطة ( , ):

في تعريفات الفئة 16

مثل لا في موجهات ما قبل المعالجة 877

عامل الفصل ( | ) : الأسبقية 849 جدول

الدالة seqCont.clear 229 جدول

الدالة seqCont.erase 229 جدول

الدالة seqCont.insert 229 جدول

الدالة seqCont.pop\_back 229 جدول

الدالة seqCont.push\_back 229 جدول

الدالات seqCont.resize 229 جدول

الدالة seqSearch 139، 194

حاويات تتابعية 218-227، 233-237، 330-335

ملفات رئيسية 785 جدول

دعم المكرر 785 جدول

دالات عنصر مشتركة 228-229

مراجعات سريعة 265-266، 357

انواع 218-219. انظر كذلك الحاويات deque وحاويات القائمة وحاويات vector

انظر كذلك الحاويات

عامل التتابع ( , ) : الأسبقية 849 جدول

قوائم تتابعية. انظر القوائم

خوارزميات البحث التتبعي 191-192، 296، 302، 348، 526-527

على القوائم المبنية على مصفوفات 191-193

الدالات 193، 296-297، 303، 348

مقارنات مفتاح في 527-528، 535، 537 جدول

على القوائم المتصلة 296-297، 302، 303

تحليل الأداء 527-528، 535

مراجعة سريعة 552

هدف قائمة مقدم الخدمة (محاكاة السينما) 495، 502-503

فئة. انظر الفئة serverListType

عمليات على 503

أهداف مقدم الخدمة (محاكاة السينما) 495 تعريف

فئة. انظر الفئة serverType

زمن خدمة العميل. انظر زمن التعامل

الحصول على وضبط العدد 510

قائمة. انظر هدف قائمة مقدم الخدمة

عمليات على 499

الفئة serverListType 502-507:

مقوم 504

عناصر بيانات 502

تعريف كنوع بيانات مجرد 503-504

مدمر 505

دالات العنصر 504-507

الفئة serverType 499-502:

عناصر بيانات 499

تعريف كنوع بيانات مجرد 499-501

دالات العنصر 501-502

الحاويات set 775-780

الاعلان والتهيئة 775-776

ملف رئيسي 785 جدول

ادخال وحذف عنصر 776-777

دعم المكرر 785 جدول  
معيار الترتيب 775، 776  
الاستخدام (التناول) 775  
عمليات نظرية الوضع: دالات مسبقة التعريف 827-835  
مصطلحات نظرية الوضع 721  
الدالة set\_difference (مسبقة التعريف) 827، 832-835  
الدالة set\_intersection (مسبقة التعريف) 827، 829-832  
الدالة set\_symmetric\_difference (مسبقة التعريف) 827، 833-835  
الدالة set\_union (مسبقة التعريف) 827، 830-832  
الدالة setCustomerInfo 498  
أداة التحكم setprecision 883  
أداة التحكم setw مقابل 885  
الدالة setServerBusy 505-506  
الدالة setSimulationParameters 510  
أداة التحكم setw 884-885  
النسخ السطحي (للبينات) 154-156، 159-161، 166-168، 208، 209، 638  
التجنب 156-157، 161، 168-170، 638-639، 650  
نوع البيانات short 875  
أقصر مسار (في الرسوم البيانية) 737:  
خوارزمية 737-738، 743-742  
دالة 742-743  
الدالة shortestPath 742-743  
أداة التحكم showpoint 884  
خلط العناصر بشكل عشوائي 822-823، 824  
عامل اسناد بسيط. انظر عامل الاسناد  
بيانات اسناد بسيطة. انظر بيانات الاسناد  
أنواع البيانات البسيطة 875-876  
معرف بواسطة المستخدم (معرف بواسطة المبرمج). انظر أنواع بيانات العد  
متغيرات: مدخلات صحيحة من أجل 880-881  
انظر كذلك المتغيرات char والمتغيرات double  
والمتغيرات int والمتغيرات string وكذلك المتغيرات

رسوم بيانية بسيطة 723  
مسارات بسيطة (في الرسوم البيانية) 723  
محاكاة (الأنظمة) 494 تعريف  
انظر كذلك محاكاة الحاسب الآلي  
معاملات المحاكاة: الحث\صول على الوالضبط 510  
الدالة  $\sin(x)$  (مسبقة التعريف) 860 جدول  
توريث أحادي 66  
الدالة  $\sinh(x)$  (مسبقة التعريف) 860 جدول  
الدالة  $\text{size}$  (مسبقة التعريف) عملية حاوية 225 جدول، 227 جدول  
عملية طابور 492 جدول  
عملية مرصوصة 461 جدول  
الدالة  $\text{str.size}$  862 جدول  
نوع بيانات مقطعي  $\text{size\_type}$  . انظر نوع البيانات  $\text{size\_type} : \text{string}$   
نوع البيانات  $\text{size\_type}$  242 جدول  
العامل  $\text{sizeof}$  الأسبقية 849 جدول  
شرطة مائلة (/). انظر عامل القسمة  
شرطة مائلة – علامة يساوي (= /): عامل اسناد مركب 849 جدول  
ايجاد أصغر عنصر:  
منهج تكراري 560-563  
دالات مسبقة التعريف 822، 824  
أصغر عدد كلي:  
دالة 859 جدول  
برامج 1-2:  
انظر كذلك البرامج  
هندسة البرمجيات 2  
المبادئ 2-15، مراجعة سريعة 56-57  
خوارزميات الترتيب 559-626:  
شجرة مقارنة 578-579  
الترتيب بالتكدس 595-604  
الترتيب بالادخال 566-578

الترتيب بالدمج 587-595  
ترتيب (الحد الأدنى):  
خوارزميات مبنية على المقارنة 578-579، 578 جدول  
تحليل الأداء 566، 578، 586، 595، 604-605  
مثال برمجة 606-626  
مراجعة سريعة 552-555، 626  
ترتيب سريع 579-586  
الترتيب بالاختيار 560-566  
الدالة sort (مسبقة التعريف) 811-815  
عمليات حاوية القائمة 331-332 جدول  
دالات الترتيب:  
الترتيب بالتكدس 604  
الترتيب بالادخال 572، 577  
الترتيب بالدمج 593-594  
دالة مسبقة التعريف 811-815:  
عمليات حاوية القائمة 331-332 جدول  
الترتيب السريع 585-586  
الترتيب بالاختيار 563-564  
ترتيب القوائم:  
خوارزميات. انظر خوارزميات الترتيب  
القوائم المبنية على مصفوفات: الترتيب بالتكدس 595-605  
الترتيب بالادخال 566-572، الترتيب السريع 579-586  
الترتيب بالاختيار 560-566  
دالات. انظر دالات الترتيب  
قوائم متصلة: الترتيب بالادخال 573-577  
الترتيب بالدمج 587-595، الترتيب بالاختيار 565  
كود المصدر (برنامج المصدر) 878  
ملفات كود المصدر: الامتداد 878  
قمة المصدر 737  
مسافة (رمز مسافة فارغة) ( ) : عامل استخلاص و880  
أشجار امتداد (رسوم بيانية) 745، 746

التعريف كنوع بيانات مجرد 752  
خوارزمية شجرة الامتداد الأدنى 744-754  
رموز خاصة. انظر قسم الرموز في أول هذا الفهرس  
اخفاء تفاصيل تحديد أهداف الفئة 40-44  
ملفات التحديد. انظر الملفات الرئيسية  
الدالة splice (مسبقة التعريف):  
عمليات حاوية القائمة 331 جدول  
الدالة sqrt (x) (مسبقة التعريف): 860 جدول  
أقواس مربعة ([ ]). انظر عامل مؤشر المصفوفة  
دالة الجذر التربيعي 860 جدول  
الفئة stack (مكتبة القالب المعياري) (مكيف حاوية) 460-462  
عمليات مدعمة 461 جدول  
ملفات المرصوصة الرئيسية 423، 439، 460  
اعلان أهداف المرصوصة 439  
التمرير كمعاملات الى دالات 422  
المرصوصات 407-462، 409 تعريف  
اضافة عناصر الى 409-410، 411  
كمصفوفات. انظر المرصوصات كمصفوفات  
عمليات أساسية 410  
مؤشرات التركيب الزمني 423 جدول  
التحقق مما اذا كانت فارغة / ممتلئة 410  
كمكيفات حاوية 460-462  
النسخ 413، 421  
اعلان أهداف المرصوصة 439  
تدمير 410  
حركي. انظر مرصوصات متصلة  
تقييم التعبيرات postfix 442-446  
ملفات رئيسية 423، 439، ملف مكتبة القالب المعياري 460، 785 جدول  
التهيئة 410  
طولي. انظر المرصوصات كمصفوفات  
متصل. انظر المرصوصات المتصلة

تقييم التعبيرات postfix:  
برنامج الحاسب 452-446  
مثال برمجة 428-425  
مراجعة سريعة 462  
ازالة عناصر من 409، 410، 411  
عنصر علوي 408-409، ازالة 409، 410، 411،  
انتاج 409، 410، 411، 461 جدول  
الاستخدامات 218، 408، 442  
المرصوعات كمصفوفات 411-424  
اضافة عناصر الى 415-418  
اثقال عامل الاسناد 422  
عمليات أساسية: مؤشرات التركيب الزمني 423 جدول  
التحقق مما اذا كانت فارغة 415  
التحقق مما اذا كانت ممثلة 415  
مقوم 421-422  
مقوم النسخ 422  
التعريف كنوع بيانات مجرد 411-413  
تدمير 415  
مدمر 421، 422  
ملفات رئيسية 423  
تهيئة 414  
التحقق من حالة الزيادة 417—418  
مؤشر من أجل 414  
مثال برمجة 425-428  
ازالة عناصر من 418-420  
برنامج اختبار 424  
عنصر علوي 411، 413، 429:  
ازالة 418-420  
التحقق من حالة النقص 420  
المتغير stackTop 411، 413، 429  
الفئة stackType:

التعريف كنوع بيانات مجرد 411-413

الملف الرئيسي 423

انظر كذلك المرصوصات كمصفوفات

تدفق الخطأ القياسي 5

دالات قياسية. انظر دالات مسبقة التعريف

ادخال بيانات من جهاز الادخال القياسي 879

اخراج بيانات الى جهاز الاخراج القياسي 882

STL انظر مكتبة القالب المعياري

البيانات:

بيانات assert 857:

اسناد. انظر بيانات الاسناد

اعلان: انظر بيانات العلان

بيانات قابلة للتنفيذ 878-879

بيانات ادخال. انظر بيانات cin

بيانات namespace 890-891

بيانات الاخراج. انظر بيانات cout

المصفوفات الساكنة في مقابل المصفوفات الحركية 151

ربط ساكن 867

أهداف فئة ساكنة 24

عامل الطرح static\_cast 876

الأسبقية 849 جدول

الاسم std 895

البادئة : : std ، 892 ، 893

تحسين تدريجي 4

STL (مكتبة القالب المعياري) 217-264 ، 769-844

مكونات أساسية 218-219 ، 769

قوالب الفئة 217

انظر كذلك الحاويات والمكررات وخوارزميات مكتبة القالب المعياري

خوارزميات مكتبة القالب المعياري (خوارزميات عامة) 789-840

فئات 786

أوصاف 796-840

## أشكال 789

أهداف الدالة من أجل التطبيق 788، 789-794

نماذج الدالة 796

خوارزميات التكدس 788

خوارزميات تعديلية 787-788

خوارزميات التغيير 787-788

خوارزميات غير تعديلية 786-787، 787 جدول

خوارزميات رقمية 788، 835-839

مراجعة سريعة 841، 842-843

أهداف دالة مكتبة القالب المعياري. انظر أهداف الدالة (لخوارزميات مكتبة القالب المعياري)

الدالة str.c\_str 861 جدول

الدالة str.clear 862 جدول

الدالة str.empty 861 جدول

الدالات str.erase 862 جدول

الدالة str.find 862 جدول

الدالات str.insert 862 جدول

الدالة str.length 862 جدول

الدالة str.size 862 جدول

الدالة str.substr 862 جدول

الدالة str1.swap (str2) 862 جدول

الدالة strcat (مسبقة التعريف) (مقاطع C) 860 جدول

الدالة strcmp (مسبقة التعريف) (مقاطع C) 860 جدول

الدالة strcpy (مسبقة التعريف) (مقاطع C) 861 جدول

أنواع بيانات التدفق: أنواع بيانات تدفق الملف 887

عامل استخلاص التدفق. انظر عامل الاستخلاص (<<)

عامل ادخال التدفق. انظر عامل الدخال (>>)

مكررات الادخال 242-243

انظر كذلك المكررات ostream

متغيرات التدفق:

Cin: انظر البيانات cin

Cout: انظر البيانات cout

أهداف تدفق الملف 887-888

الثابت المسمى npos : : string 861

نوع البيانات size\_type : : string 861

دالات المقطع (النوع string) 861، 863-861 جدول

انظر كذلك دالات المقطع C

الملف الرئيسي string 861، 863-861 جدول

متغيرات string:

اخراج القيم 883 جدول

تبادل المحتويات 862 جدول

انظر كذلك متغيرات المقاطع

المقاطع (النوع string):

التحقق مما اذا كان فارغ 861 جدول

الاخلاء 862 جدول

المسح 862 جدول

دالات للتحكم 861، 863-861 جدول

ادخال (قراءة) 861 جدول

ادخال رموز في 862-863 جدول

الاخراج 883 جدول

استبدال الرموز في 863 جدول

ايجاد مقاطع فرعية في 862 جدول

انظر كذلك المقاطع C والمتغيرات string

الدالة strlen (مسبقة التعريف) 861 جدول

رسوم بيانية قوية الارتباط 723

أنواع البيانات struct. انظر structs

عناصر struct: التناول عبر المؤشرات 144-145

اعلان متغيرات struct 144

انظر كذلك structs

تعريف عقدة الشجرة AVL 678

تعريف عقدة الشجرة الثنائية 636

مقابل الفئات 144

المكونات. انظر عناصر struct

عقد القائمة المتصلة 275  
مؤشرات الى 144-147  
انظر كذلك متغيرات struct  
أنواع البيانات المنظمة. انظر أنواع البيانات المجردة، والمصفوفات،  
والفئات، والقوائم، والطوابير، والمرصوصات، والبنيات structs  
تصميم منظم 4  
برمجة منظمة 4  
بنيات. انظر structs  
رسوم بيانية فرعية 721  
برامج فرعية. انظر الدالات  
الدالة substr (مسبقة التعريف): الدالة str.substr 862 جدول  
مقاطع فرعية:  
ايجاد 862 جدول  
انتاج 862 جدول  
عامل الطرح ( - ): الأسبقية 849 جدول  
أشجار فرعية (للأشجار الثنائية) 632، 654  
ترميز اللاحقة. انظر ترميز postfix  
الدالة swap (مسبقة التعريف) 808-811  
عملية حاوية 228 جدول  
الدالة str1.swap (str2) 862 جدول  
الدالة swap (خوارزميات الترتيب) 563، 585  
الدالة swap\_ranges (مسبقة التعريف) 808-811  
تبادل العناصر 808-811  
محتويات مؤشر المصفوفة 561-563، 585  
محتويات المتغير string 862 جدول  
البيانات switch 889  
رموز. انظر قسم الرموز في أول هذه الفهرس  
فروق متماثلة بين المجموعات: دالة مسبقة التعريف 827، 833-835  
تركيب:  
قوائم المعامل الحقيقي والدالات المنتجة لقيمة 896  
والدالات void 896

بيانات الاسناد 877  
بيانات cin 879  
تناول عنصر الفئة 21  
قوالب الفئة 112  
الفئات 16  
مقوم النسخ في تعريف الفئة 170  
البيانات cout 882  
استدعاء المقوم الافتراضي 34  
العامل delete 149  
الفئات المستمدة 67  
أداة التحكم flush 886  
قوائم المعامل الرسمي: والدالات المنتجة لقيمة 895  
والدالات void 896  
استدعاءات الدالة : الدالات المنتجة للقيمة 895 والدالات void 896  
قوالب الدالة 110  
مكررات istream 242  
أداة التحكم left 885  
اعلانات ثابت محدد 876  
بيانات namespace 890  
العامل new 148  
اعلانات مصفوفة أحادية البعد 900  
الدالة open 888  
دالات العامل 87  
المكررات ostream 243  
انقال عامل الاسناد  
نموذج / تعريف الدالة 162-163  
انقال عوامل ثنائية  
نماذج / تعريفات الدالة 97، 99  
انقال عامل الاستخلاص  
نموذج / تعريف الدالة 101  
انقال عامل الادخال

نموذج / تعريف الدالة 100  
اعلانات المؤشر 134  
موجهات ما قبل المعالجة 877  
أداة التحكم right 885  
أداة التحكم setprecision 883  
قوالب 110  
الدالة unsetf 885  
البيانات namespace / using namespace 891  
دالات منتجة لقيمة 895-896  
اعلانات المتغير 877  
دالات void 896-897  
ملفات النظام الرئيسية:  
المحددات (<>) 41  
التضمين 41، 77، 877-878  
محاكاة النظام. انظر محاكاة (الأنظمة)  
**T**  
رمز التبويب، وعامل الاستخلاص 880  
دالات تكرارية ذيلية 371  
الدالة  $\tan(x)$  (مسبقة التعريف) 860 جدول  
الدالة  $\tanh(x)$  (مسبقة التعريف) 860 جدول  
تمثيلات القالب 114  
قوالب 85، 110  
اثقال دالة مع 110-114  
مراجعة سريعة 116  
التركيب 110  
انظر كذلك قوالب الفئة وقوالب الدالة  
طوابير مؤقتة 509  
حالات اختبار 7  
برامج اختبار 7-8:  
بيانات assert 6  
اختبار الصندوق الأسود 7-8

اختبار الصندوق الأبيض 7، 8

محاكاة خدمة السينما. انظر محاكاة خدمة السينما

المؤشر this 88-89

العامل throw: الأسبقية 849 جدول

تلدة ( -): بادئة اسم المدمر 35

التركيب الزمني. انظر مؤشر التركيب الزمني  $O(g(n))$

محاكاة مستمدة من الزمن 496

رموز. انظر المحددات

الدالة tolower (ch) (مسبقة التعريف) 858 جدول

العنصر العلوي (من المرصوصات) 408-409

المرصوصات المتصلة 429 الازالة 437-439 الانتاج 437

الازالة 409، 410، 411

الانتاج 409، 410، 411، 461 جدول

المرصوصات كمصفوفات 411، 413، 429

الازالة 418-420

الدالة top (مسبقة التعريف) عملية مرصوصة 409، 410، 411، 437، 461 جدول

تصميم من أعلى الى أسفل 4

الترتيب الطبوغرافي (للقمم) 754-761

الاجتياز الأول للعرض 754-761

الاجتياز الأول للعمق 754

مصفوفة topologicalOrder 754، 756، 757، 758، 759، 760

الفئة topologicalorderT 754-755

الدالة toupper (ch) (مسبقة التعريف) 859 جدول

مشكلة برج هانوي 383-387، 390

عقد trailer (في قوائم متصلة) 335-336، 357-358

أصفار المنزل: توضيح العلامة العشرية 884

زمن التعامل (زمن خدمة العميل) 495 تعريف، 503، 505-506

الحصول على وضبط 510

الدالة transform (مسبقة التعريف) 825-827

الاجتيازات. انظر اجتيازات الشجرة الثنائية واجتيازات الرسم البياني

اجتياز:

أشجار ثنائية 639-643. انظر اجتيازات الشجرة الثنائية  
رسوم بيانية 731. انظر كذلك اجتيازات الرسوم البيانية  
القوائم المتصلة 278-279، قوائم متصلة من الطرفين 318، 322-323  
أشجار:  
ثنائية. انظر رسوم الأشجار الثنائية 745، 746. انظر كذلك أشجار الامتداد  
دالات مثلثية 859 جدول، 860 جدول  
اختيارات ثنائية الاتجاه: بيانات if...else 889  
خاصية نوع بيانات اسم النوع 37  
241-242 Typedef const\_iterator  
242 Typedef const\_reverse\_iterator  
241، 223 Typedef iterator  
:Typedef  
242 Reverse\_iterator  
241-242، 223 typedef مكررات بيانات  
تعريفات النوع المشتركة بين جميع الحاويات 242 جدول  
العامل typeid الأسبقية 849 جدول

## U

رسم لغة التشكيل الموحدة 20  
عوامل أحادية:  
الانقال 103  
الأسبقية 849 جدول  
أصول أحادية (أهداف الدالة) 794  
التحقق من حالة النقص (للمرصوعات كمصفوفات) 420  
الخط التحتي ( \_ )، رمز محدد 890  
رسوم بيانية غير مباشرة 721، 722  
رسوم لغة التشكيل الموحدة 20  
اتحادات المجموعات 721  
دالة مسبقة التعريف 827، 830-832  
الدالة unique (مسبقة التعريف):  
عمليات حاوية قائمة 331 جدول

الدالة unsetf (مسبقة التعريف) 884

التركيب 885

الدالة updateServers 506-507

الدالة updateWaiting Queue -508، 509

تحديث البيانات في عقد شجرة ثنائية 672-675

ملفات المستخدم. انظر ملفات التطبيق

أنواع البيانات المحددة بواسطة المستخدم. انظر أنواع بيانات العد

وأنواع البيانات المنظمة

دالات معرفة بواسطة المستخدم. انظر الدالات المنتجة للقيمة والدالات void

ودالات خاصة

ملفات رئيسية يحددها المستخدم:

التضمين 40-41، 77

استخدام (تناول) موارد مكتبة C++

عناصر الأسماء 891-892

دالات مسبقة التعريف 877-878

البيانات using namespace / namespace name 891-892، 895

التركيب 891

الملف الرئيسي utility 770، 772

## V

تصحيح المدخلات 6

معاملات القيمة 897 تعريف، 897

مقابل معاملات الاشارة الثابتة 19، 25

اعلان المؤشرات ك 153

تمرير أهداف الفئة ك 24-25، 167-168

تمرير أهداف فئة مستمدة الى معاملات فئة أساسية 869-871

تمرير المؤشرات ك 153

قوائم المعاملات الحقيقية للدالات المنتجة للقيمة 896

الدالة assert 6، 857 جدول

استدعاءات الى 895

دالات رموز 857-859 جدول

قوائم المعامل الرسمي 895  
دالات رياضية 859-860 جدول  
معاملات اشارة في 898  
انتاج قيم: محدد نوع الانتاج const 874  
المؤشر ك 154، انتاج أكثر من واحد 770، 898  
دالات مقاطع: دالات مقاطع C 860-861 جدول  
النوع string 861، 863-861 جدول  
التركيب 895-896  
الدالات void كدالات منتجة لقيم متعددة 898  
تعريف النوع value\_type 242 جدول  
القيم:  
قيم الحدود 8  
منطقي. انظر قيم منطقية  
انتاج. انظر انتاج قيم (من دالات منتجة للقيمة)  
متغيرات. انظر متغيرات under  
انظر كذلك البيانات والأعداد  
متغيرات:  
تحديد قيم لها 876، 877  
كعناصر فئة. انظر عناصر البيانات (الفئات)  
الاعلان 16، 876-877، 879  
أهداف تدفق الملف 887-888  
التهيئة عند 16  
متغيرات struct 144  
متغيرات حركية 147-149  
أهداف تدفق الملف 887-888  
التهيئة عند الاعلان 16  
ادخال قيم. انظر ادخال البيانات  
لمعالجة القائمة 181  
اخراج قيم. انظر اخراج البيانات  
تخزين عناوين في مؤشرات 135-143  
تدفق. انظر متغيرات التدفق

انظر كذلك المتغيرات char والمتغيرات double،  
والمتغيرات int والمتغيرات string وكذلك معاملات (معطيات) الدالات

العملية [index] vecCont 221 جدول

الدالة vecCont.at 221 جدول

الدالة vecCont.back 221 جدول

الدالة vecCont.capacity 225 جدول

الدالة vecCont.clear 222 جدول

الدالة vecCont.empty 225 جدول

الدالة vecCont.erase 222 جدول

الدالة vecCont.front 221 جدول

الدالة vecCont.insert 222 جدول

الدالة vecCont.max\_size 225 جدول

الدالة vecCont.pop\_back 222 جدول

الدالة vecCont.push\_back 222 جدول

الدالة vecCont.resize 222 جدول

الدالة vecCont.size 225 جدول

الفئة vector 219

الحاويات vector 219-227:

تناول عناصرها 219، 221

عملياتها الأساسية 221-223

الاعلان والتهيئة 219-220

اعلان مكررات بداخلها 223-224، 241-242

حذف عناصر منها 221، 222 جدول، 223

الملف الرئيسي 785 جدول

ادخال عنصر فيها 219، 221، 222 جدول، 223

في النهاية 222-223

دعم المكررات 785 جدول

اخراج العناصر 229-233

مثال برمجة 243-264

مراجعة سريعة 265

دالات الحجم 225 جدول

الانتقال عبر العناصر 222  
انظر كذلك الحاويات: الحاويات المتتابعة  
الأهداف vector. انظر الحاويات vector  
القمم (في الرسوم البيانية) 721  
تجاوز 723، 726  
الترتيب الأول للعرض 734  
متصلة 723  
ترتيب أول للعمق 732  
تتبع القمم التي تتم زيارتها 731  
الترقيم 729  
المصدر 737  
ترتيب طوبوغرافي 754-761  
قائمة الفيديو (مثال برمجة متجر الفيديو) 343-350  
أهداف الفيديو (مثال برمجة متجر الفيديو) 338-343، 701-703  
أمثلة برمجة متجر الفيديو 337-357، 701-709  
الفئة videoBinaryTree 703-706  
الفئة bSearchTreeType 703  
التعريف 703-704  
دالات العنصر 704-706  
الفئة videoListType (قائمة متصلة) 343-350  
الفئة linkedListType 343، 345  
الفئة videoType:  
تعريفها كنوع بيانات مجرد 339-341، 701-702  
دالات العنصر 341-343، 702-703  
مدمرات افتراضية 871  
دالات افتراضية:  
الربط 867  
الاعلان 867-868  
تمرير أهداف الفئة المستمدة الى معاملات الفئة الأساسية 867-871  
الدالات void:  
قوائم المعامل الحقيقي 896

الاستدعاءات اليها 896، 897

قوائم المعامل الرسمي 896

كدالات منتجة للقيمة المتعددة 898

تمرير المعامل. انظر تمرير المعاملات الى دالات

مع المعاملات 896-898

التركيب 896-896

## W

أهداف طابور العميل المنتظر (محاكاة خدمة السينما) 495، 507-508

تناول العناصر 508

فئة. انظر الفئة waitingCustomerQueueType

تحديد أن جميع العناصر تمت معالجتها 509

الفئة waitingCustomerQueueType 507-509

المقوم 508

التعريف كنوع بيانات مجرد 508

مدمر 508

دالات عنصر 508-509

الفئة queueType 507-508

والكر آر جيه 391

وزن الحافة 737

وزن المسار 737

خوارزمية أقصر مسار 737-738، 738-743

رسوم بيانية موزونة 737

انظر كذلك الأشجار الموزونة

أشجار (رسوم بيانية) موزونة 746

الطباعة 753-754

انظر كذلك أشجار الامتداد

تعريف الفئة weightedGraphType 737-738

المصفوفة bool weightFound 738

الحلقات while 889

اختبار الصندوق الأبيض 7، 8

رمز المسافة: عامل الاستخلاص 880  
رموز الكلمة. انظر الكلمات المحفوظة