



البرمجة بلغة C++



منشورات جامعة دمشق
كلية الهندسة الميكانيكية والكهربائية

البرمجة (1)

البرمجة غرضية التوجه بلغة C++
OOP IN C++

الدكتور
أكرم مذكور
قسم هندسة الحواسيب والأتمتة

الدكتور المهندس
مأمون يونس
قسم هندسة الحواسيب والأتمتة

1435 – 1434
2014 – 2013 : جامعة دمشق



الفهرس

الفصل الأول

مدخل إلى البرمجة غرضية بلغة C++

٢١	١-١- مقدمة
٢٥	١-٢- البرمجة غرضية التوجه
٢٧	١-٣- سمات البرمجة غرضية التوجه
٢٧	١-٣-١- التغليف
٢٨	١-٣-٢- الوراثة
٢٩	١-٣-٣- تعدد الأشكال
٣٠	١-٤- البنى
٣٠	١-٤-١- مقدمة
٣٠	١-٤-٢- تعريف البنية
٣٢	١-٤-٣- الوصول إلى حقول البنية

الفصل الثاني

الصفوف في C++

٤٣	١-٢- مقدمة
----	------------------

٤٥	٢-٢- بناء نمط المعطيات في الصف
٥٣	٢-٣- كيفية الوصول إلى أعضاء الصف
٥٥	٢-٤- تجزئة البرنامج
٦٣	٢-٥- تعريف المتحولات كأعضاء عامة
٦٤	٢-٦- التابع الباني
٦٥	٢-٧- التابع المدمر
٦٩	٢-٨- تعدد البواني
٧٠	٢-٩- الباني الافتراضي
٧٥	٢-١٠- استدعاء التوابع البنية والمدمرة
٨٣	٢-١١- بناء المكس
٨٧	٢-١٢- بناء الصف date
٩٠	٢-١٣- بناء الصف Time
٩٢	٢-١٤- بناء الصف Point
١٠٠	٢-١٥- الأغراض الثابتة والتوابع الثابتة
١٠٥	٢-١٦- إسناد الأغراض
١٠٦	٢-١٧- تمرير الأغراض إلى التوابع
١١٢	٢-١٨- التركيب
١٢٢	٢-١٩- التوابع الصديقة والصفوف الصديقة
١٢٣	٢-١٩-١- التوابع الصديقة
١٣٤	٢-٢٠- مصفوفات الأغراض
١٣٧	٢-٢١- استخدام مؤشرات الأغراض
١٣٩	٢-٢٢- المؤشر this
١٤٧	٢-٢٣- استخدام العمليتين new و delete

١٥١	٢٤-٢- أعضاء الصفوف الساكنة
١٥٥	٢٥-٢- توابع الصفوف الساكنة
١٦٠	٢٦-٢- المراجع
١٦١	٢٥-٢-١- تمرير المراجع إلى التوابع والأغراض
١٦٤	٢٥-٢-٢- إعادة المراجع
١٦٨	٢٥-٢-٣- المراجع المستقلة
١٧٠	٢٦-٢- مسائل عامة محلولة
٢٠١	٢٧-٢- مسائل عامة

الفصل الثالث

التحميل الزائد للعوامل

٢١١	٣-١- مقدمة
٢١١	٣-٢- مبادئ التحميل الزائد
٢١٢	٣-٣- التحميل الزائد للعوامل الثنائية
٢١٣	٣-٣-١- التحميل الزائد لعملية الجمع
٢١٥	٣-٣-٢- التحميل الزائد لعملية الطرح
٢١٧	٣-٣-٣- التحميل الزائد لعملية الإسناد
٢٢١	٣-٣-٤- المعامل الأيمن للتحميل الزائد من النوع int
٢٢٦	٣-٤- التحميل الزائد لعمليات المقارنة والمنطقية
٢٣٣	٣-٥- التحميل الزائد للعوامل الأحادية
٢٣٣	٣-٥-١- التحميل الزائد للعملية ++ و --
٢٣٦	٣-٥-٢- التحميل الزائد لعملية إشارة السالب
٢٣٨	٣-٦- استخدام التوابع الصديقة

٢٣٩	٣-٦-١- التحميل الزائد للعوامل الثنائية كتاب صديق
٢٤٤	٣-٦-٢- التحميل الزائد للعوامل الأحادية كتاب صديق
٢٥٥	تمارين عامة
٢٥٧	٣-٧-١- التحميل الزائد لمجري الدخل والخرج
٢٥٧	٣-٧-١- التحميل الزائد لعملية مجري الخرج
٢٦٤	٣-٧-٢- التحميل الزائد لعملية مجري الدخل
٢٧٠	٣-٧-٣- قاعدة مجرى dec,oct,hex, showbase
٢٧١	٣-٨- أمثلة
٢٨٣	مسائل عامة محلولة
٢٩٥	مسائل عامة

الفصل الرابع الوراثة

٣٠٧	٤-١- مقدمة
٣٠٩	٤-٢- الصفوف الأساسية والصفوف المشتقة
٣٠٩	٤-٢-١- التحكم بالوصول إلى الصف المشتق
٣١٥	٤-٢-٢- استخدام الأعضاء المحمية
٣١٨	٤-٣- البواني والمدمرات
٣٤٧	تمارين
٣٤٨	٤-٤- الوراثة المتعددة
٣٤٦	٤-٤-١- الصف المشتق صفاً أساسياً لصف مشتق آخر
٣٥٦	٤-٤-٢- الصف المشتق يرث أكثر من صف أساسي
٣٦٦	٤-٤-٢- الصفوف الأساسية الافتراضية

٣٧٦	مسائل عامة محلولة
٤٢٦	مسائل عامة

الفصل الخامس

تعدد الأشكال

٤٣٥	١-٥ مقدمة
٤٣٦	٢-٥ المؤشرات إلى الصفوف المشتقة
٤٤٠	٣-٥ التوابع الافتراضية
٤٤٧	٤-٥ الصفوف المجردة
٤٥٧	٥-٥ الصفوف المجردة النقية
٤٦٨	٦-٥ تطبيقات تعدد الأشكال
٣٧٥	أسئلة
٣٧٦	مسائل عامة محلولة
٣٨٧	مسائل عامة

الفصل السادس

القوالب ومعالجة الاستثناءات

٤٩٣	١-٦ مقدمة
٤٩٣	٢-٦ قوالب التوابع
٥٠٨	مسائل عامة محلولة
٥١٢	٣-٦ قوالب الصفوف
٥١٧	١-٣-٦ قالب الصف stack
٥٢٤	٢-٣-٦ قالب الصف لعدد من أنواع لمعطيات

٥٢٩	٤-٦- معالجة الاستثناءات
٥٣٩	٦-٤-١- معالج الاستثناء
٥٤٨	٦-٤-٢- معالجة الاستثناءات التي يرميها new
٥٥٢	مسائل عامة محلولة

الفصل السابع

معالجة الملفات

٥٧٣	٧-١- مقدمة
٥٧٣	٧-٢- تعريف الملف
٥٧٣	٧-٣- أسماء الملفات
٥٧٤	٧-٤- عملية استخدام الملفات
٥٧٦	٧-٥- مكتبة الإدخال و الإخراج في لغة C++
٥٧٩	٧-٦- فتح ملف
٥٧٩	٧-٦-١- تابع فتح الملف (open)
٥٨١	٧-٦-٢- حالات استخدام التابع (open)
٥٨٦	٧-٧- إغلاق الملف
٥٨٨	٧-٨- استخدام المعامل (<<) من أجل كتابة في الملف
٥٩٢	٧-٩- شكل خرج الملف
٥٩٦	٧-١٠- استخدام المعامل >> من أجل القراءة من ملف
٥٩٨	٧-١١- تحديد نهاية الملف
٦٠١	٧-١٢- التوابيع الأعضاء لكتابة وقراءة الملفات
٦٠٨	٧-١٣- التابع العضو getline

- ٦١٠١٤-٧- التابعان get () و put ()
- ٦١١١-٤-٦- التابع العضو get ()
- ٦١٢٢-٤-٦- التابع العضو put ()
- ٦١٦١٥-٧- ملفات الوصول العشوائي
- ٦٢٠١٦-٧- إنشاء السجلات باستخدام البنى
- ٦٢٥١٧-٧- فتح ملف لعمليتي الإدخال والإخراج
- ٦٣٣المراجع
- ٧٣٧المصطلحات العلمية



مقدمة

إن التقدم التكنولوجي الذي يشهده عصرنا الحالي ، والتطورات السريعة في مجال تقنية الدارات المتكاملة أسهمت إسهاماً مباشراً في تطوير مختلف العلوم الهندسية النظرية منها والتطبيقية . وهذه المساهمة ناتجة عن توافر الوسائل المادية المختلفة لتحقيق مختلف النظريات الهندسية بشكل عام والنظريات العائدة لعلوم هندسة الحاسوب بشكل خاص ، ومن أهم هذه الوسائل الحواسيب والشبكات الحاسوبية والبرمجيات .

إن الحواسيب أصبحت وسيلة أساسية في تطوير مختلف العلوم الهندسية، التي أثرت تأثيراً كبيراً على مختلف مجالات الحياة . لذلك فإن تأهيل الكوادر الفنية المتخصصة في مجال هندسة الحواسيب والبرمجيات عمل لا مفر منه لتسهم في سد احتياجات مختلف القطاعات الإنتاجية والخدمية وفي مجال تطوير البحث العلمي التقني .

لقد تطورت لغات البرمجة جنباً إلى جنب مع تطور هندسة الحاسوب ، وظهرت عدة لغات برمجة واختفت بعض منها خلال عملية التطور ، حيث فشلت بعض لغات البرمجة مثل PL/I ونجح بعضها الآخر ، وصمد لفترة طويلة مثل FORTRAN و PASCAL . لكن لغة واحدة أثبتت أنها اللغة الأقوى والأفضل حتى الآن وهي لغة C/C++ .

لقد ظهرت لغة C مع ظهور نظام تشغيل الشبكات UNIX في الستينيات من القرن الماضي ، وانتشرت معه وتطورت حتى أصبحت معياراً ليس في نظام تشغيل UNIX فحسب بل في جميع أنظمة التشغيل التي صممت على غرارته مثل Linux ، أو التي نافسته مثل نظام تشغيل Windows .

وعندما ظهرت فكرة البرمجة غرضية التوجه OOP في السبعينات من القرن الماضي كان لا بد من تطوير لغات برمجة تدعم هذه التقنية ، وقد اعتمدت معظم اللغات الجديدة على إضافة تقنيات البرمجة غرضية التوجه OOP إلى لغات إجرائية موجودة مسبقاً ، وبما أن لغة C كانت الأشهر من بينها فظهرت لغة C++ التي أضافت مزايا وأفكارها البرمجة الغرضية التوجه OOP إلى لغة C وأصبحت بعد ذلك لغة البرمجة الأولى في العالم .

وتطورت لغة C++ مع الزمن ، وأضيف إليها العديد من المزايا ، وأصبح لها معيار خاص من ANSI ، وتدعى لغة C++ المطابقة لهذا المعيار بـ ANSIC++ . وأصبح تعلم هذه اللغة ضرورة أساسية لكل مبرمج محترف .

يمر عالم تطوير البرمجيات حالياً بثورة هدفها بناء برمجيات سريعة وصحيحة واقتصادية ، لكن بقي هذا الهدف حتى الآن هدفاً بعيد المنال ، في الوقت الذي يتزايد الطلب فيه على برمجيات جديدة وأكثر فعالية وقوة . تعد الأغراض Object من المكونات البرمجية القابلة للاستخدام أكثر من مرة ، وهي قادرة على تمثيل عناصر العالم الحقيقي المحيط بنا . كما أن المبرمجين يكتشفون حالياً أن استخدام أسلوب البرمجة الذي يعتمد على تجزئة المسألة المدروسة ، وعلى استخدام أساليب التصميم والبرمجة غرضية التوجه ، يجعل من مجموعات تطوير البرمجيات أكثر إنتاجية مما كانت عليه عند استخدام الأساليب البرمجية السابقة مثل أسلوب البرمجة الهيكلية .

وتم تطوير العديد من لغات البرمجة غرضية التوجه الأخرى مثل لغة SmallTalk من قبل PARC التي تُعد لغة غرضية التوجه فقط ، في حين تعتبر لغة C++ لغة هجينة وبالتالي فهي لغة هيكلية و غرضية التوجه .

وتتألف برامج C++ من مكونات تسمى الصفوف Classes والتتابع Functions ، ومن أجل برمجة أحد هذه المكونات ربما نحتاج إلى كتابة برنامج

كامل بلغة C++ ، لكن المبرمجين بلغة C++ يستفيدون من ميزات المجموعة الغنية من الصفوف والتوابع المعرفة مسبقاً ضمن مكتبات C++ المعيارية . لذلك يمكن فعلياً تقسيم عملية تعلم لغة C++ إلى قسمين : يعتمد الأول منها على تعلم لغة C++ نفسها في حين يسمح الثاني بتعلم كيفية استخدام الصفوف الملحقه بـ C++ ، واستخدام التوابع الموجودة ضمن المكتبة المعيارية لـ C++ .

ويصف هذا الكتاب البرمجة غرضية التوجه OOP بلغة C++ ، ويهدف إلى تعلم البرمجة غرضية التوجه بلغة C++ بشكل احترافي ، حيث يتدرج من المفاهيم البسيطة وينتهي بشرح معمق لأكثر مواضيع اللغة تعقيداً ، ويدعم الشرح بالأمثلة الواضحة والقوية في آن واحد . كما يلي كل فصل مجموعة من المسائل العامة التي تدرب الطالب على حل المسائل بنفسه وتعمق فهمه للمواضيع وتطبيقاتها العملية . والهدف الأساسي لهذا الكتاب هو تغطية منهاج مقرر " البرمجة (1) " في قسم هندسة الحواسيب والأتمتة لطلاب السنة الأولى. ولكن تم وضع مادة هذا الكتاب بحيث تمكن مختلف القراء من الاستفادة منه في مختلف التطبيقات . ويتكون من ستة فصول وملحق وسرد للمراجع الأجنبية والعربية .

ويتضمن الفصل الأول مدخل إلى البرمجة غرضية التوجه بلغة C++ ، والهدف من هذا الفصل التعرف على عدد من الميزات الأكثر أهمية في C++ ، وكما هو معروف فإن مكونات أي لغة حاسوبية لا تعمل بشكل مستقل منفصلة عن بعضها بعضاً ، بل تعمل معاً كنظام متكامل .

أما الفصل الثاني فيتناول الميزة الأولى في البرمجة غرضية التوجه ، وهي الصفوف ، وباستخدام وسيلة الصف فإن البرامج الكبيرة ستكون واضحة المعالم، حيث تكون جميع البرامج ذات العلاقة ، في مواقع محددة لها ، وهذا الأسلوب يؤدي إلى مفهوم الوحدات المتكاملة ، و يؤدي بالتأكيد إلى الصيانة الكفوة وتسهيلها . ويبحث في موضوع بواني الصف والمدمرات وتعدد البواني وتمير

الأغراض إلى التوابع والتوابع الصديقة والصفوف الصديقة والأعضاء الساكنة والتوابع الثابتة .

ويغطي الفصل الثالث التحميل الزائد للعوامل الثنائية والأحادية كتوابع أعضاء عامة ، والتحميل الزائد لعوامل المقارنة والعوامل المنطقية والتحميل الزائد للعوامل الثنائية والأحادية كتوابع صديقة ، والتحميل الزائد لمجرى الدخل والخرج .

ويغطي الفصل الرابع الميزة الثانية للبرمجة غرضية التوجه وهي الوراثة والاستفادة من الخصائص المعرفة في برامج أخرى سابقة ، ويطلق مصطلح الوراثة على العملية التي يكتسب غرض ما من خلالها على خصائص غرض آخر، ويتناول الصفوف المشتقة من الصف الأساسي ، ويواني ومدمرات الصفوف المشتقة، والوراثة المتعددة والصفوف الافتراضية .

والفصل الخامس خُصص لموضوع تعدد الأشكال في البرمجة غرضية التوجه حيث يتم استخدام اسم التابع نفسه لأغراض متعددة ، ويبحث في المؤشرات إلى الصفوف المشتقة ، والتوابع الافتراضية ، والصفوف المجردة والصفوف المجردة النقية وتطبيقاتها .

والفصل السادس يغطي القوالب ومعالجة الاستثناءات ، ويتناول قوالب التوابع والصفوف ومعالجة الاستثناءات ومعالجة الاستثناءات الذي يرميها المعامل new .

والفصل السابع والأخير يغطي معالجة الملفات ويتناول توابع التعامل مع الملفات وكيفية تخزين المعطيات (كتابة المعطيات) واسترجاعها (قراءة المعطيات) من الملفات من خلال توابع وصفوف لغة C++ .

ونأمل أن نكون قد وضعنا للقارئ المبادئ الأساسية للانطلاق في مجال
البرمجة غرضية التوجه ، وأسهمنا بتقديم المادة العلمية المتواضعة لكوارثنا
الهندسية لتساعدكم في تطوير معلوماتهم ، بحيث تتماشى مع التطور الكمي
والنوعي في مجال هندسة الحواسيب والأتمتة .
ولا يفوتنا أن ننقدم بالشكر والعرفان إلى جميع الزملاء الذين أسهموا في إبراز
هذا العمل إلى حيز الوجود .

والله ولي التوفيق

المؤلفان

دمشق / حزيران / 2013



الفصل الأول

مدخل إلى البرمجة الغرضية التوجه

في لغة C++

OBJECT ORIENTED PROGRAMMING
IN C++





البرمجة الغرضية التوجه

Object Oriented Programming

١-١ - مقدمة

صُممت لغة CPL (Combined Programming Language) في عام 1960 ، تم ذلك في جامعة لندن ، وقد استخدمت فيها بعض تعليمات لغة ALGOL-60، وفي عام 1967 انبثقت لغة BCPL من لغة CPL على يد مارتين ريتشاردز، ثم قام تومبسون بتطوير لغة BCPL وسماها لغة B وتعد هذه اللغات لغات من البرمجة متدنية المستوى (Low Level Programming Languages). وفي عام 1972 قامت مخابر شركة AT&T الأمريكية باستنباط لغة جديدة من لغة B أخذ منها أفضل التعليمات وأضاف إليها أوامراً جديدة وأنواعاً جديدة للمعطيات وكثيراً من التوابع التي تفيد المبرمج وسُميت هذه اللغة بلغة C . ومنذ ذلك التاريخ أخذت لغة C شهرة واسعة ، لأنها أصبحت تنتمي إلى اللغات البرمجية العالية المستوى LLL من حيث سهولة الاستخدام من ناحية ، ومن ناحية ثانية تنتمي إلى اللغات البرمجية المتدنية المستوى من حيث قدرة اللغة على مخاطبة مكونات الحاسوب العتادية Hardware.

ومع نجاح لغة C وانتشارها انتشرت لهجات كثيرة لمستخدمي لغة C وكاد يحدث معها ما حدث للغة BASIC من وجود BASIC لكل نوع من أنواع الأجهزة (الحواسيب) سواء كانت أجهزة IBM أو Apel أو Singlar لولا أن قام معهد القياسات الأمريكية بعملية توحيد لهذه اللهجات المختلفة فأصدر في عام 1983 اللغة القياسية (ANSI C)، والجدير بالذكر أن معظم مترجمات لغة C تتوافق مع ANSI C مثل : Borland C, Microsoft C, Lattice C.

وتتميز لغة C بمجموعة من المزايا مثل:

- **C لغة عامة:** هذا يعني أنها تصلح لتنفيذ برامج قواعد المعطيات والرسومات والحسابات ونظم التشغيل وغيرها.
- **C لغة بنيوية Structured Language :** هذا يعني أن البرنامج المكتوب بلغة C هو عبارة عن تابع رئيسي يمكن أن يستدعي مجموعة من التتابع الأخرى وكل تابع هو عبارة عن مجموعة من التعليمات.
- **C لغة تتعامل على مستوى " البت " Bit Manipulation:** حيث نستطيع أن نقرأ ونكتب ونغير ونقوم بعمليات على مستوى الـ Bit وكما هو معروف فإن الـ Bit هو أصغر وحدة لقياس المعلومات داخل الحاسوب وهو خانة من ثمانية خانات تعادل في مجموعها محرف واحد Byte. وهذه الميزة جعلت من لغة C لغة متخصصة في بعض مجالات التحكم الآلي والروبوت وبرامج معالجة الصور وضغط الملفات واكتشاف الأعطال.
- **C لغة متنقلة Portable:** هذا يعني أنه يمكن للبرنامج المكتوب بلغة C أن يعمل مع أكثر من جهاز مثل IBM أو Apple أو الأجهزة المتوسطة والكبيرة مع بعض التعديلات الطفيفة.
- **C لغة سريعة :** لأن أدوات اللغة تتعامل بشكل مباشر مع الآلة (الحاسوب) ما يختصر وقت التنفيذ.
- **C لغة قياسية:** هذا يعني أن معظم مترجمات اللغة تتوافق مع اللغة القياسية ANSI C .
- **C لغة نظام التشغيل UNIX:** هذا يعني أن نظام التشغيل الشهير UNIX مكتوب بهذا اللغة ، وهذا يدل على إثراء هذه اللغة ومرونتها.
- تعتبر لغة C++ لغة متطورة عن لغة C ، هذه اللغة تتمتع بجميع ميزات لغة C بالإضافة إلى أنها تدعم البرمجة غرضية التوجه (Object Oriented Programming) ، وتعد لغة البرمجة C++ مجموعة أشمل من لغة C .

وتعد لغة C++ من اللغات البرمجة العالية المستوى وفي الوقت نفسه قريبة من لغة التجميع ذات المستوى المحدود كما أنها تعد لغة برمجة إجرائية (يمكن كتابة برنامج يحتوي على إجراءات و توابع فقط) وغرضية التوجه. إن لغة C++ هي لغة متعددة الاستخدامات إلا أنها مناسبة لبرامج أنظمة التشغيل. وتكتب معظم أجزاء أنظمة التشغيل والبرامج المساعدة لها باستخدام لغة C++ .

يتجنب الكثيرون من مبرمجي لغة C++ و لغة C نظراً لشهرتهما ك لغات برمجة معقدة. وقد أدى ذلك إلى استخدام الكثيرين للغات مثل Visual BASIC و Delphi ثم Java و C# والأخيرتين هما مجرد تسهيل (و ليس تطوير) للغة C++ وبدأ في وقت معين أن لغات مثل Java سوف تستولي على سوق البرمجيات التجارية من لغة C++ مع هجرة عدد كبير من مبرمجي C++ إلى Java و C#، إلا أنه سرعان ما أدرك المبرمجون أن لغة C++ هي السبيل الوحيد لإنتاج برامج تجارية قوية وسريعة وتؤدي المطلوب منها بكفاءة وبأقل عدد من الأخطاء.

تدعم لغة C++ لغة C بشكل كامل وعمليا يمكن لمعظم برامج C أن تترجم بواسطة مترجمات لغة C++ ولهذا تسمى هذه المترجمات بمترجمات لغة C/C++ إلا أن الأمر لا يخلو من بعض المشاكل غير المتوقعة، مثلاً قد يحدث أن نسمي متغيراً باسم new وهي كلمة محجوزة في لغة C++ وليست محجوزة في لغة C ، إذا كان للتابع وسيط void فإنه يأخذ عدد غير محدد من الوسائط ، أما في لغة C++ فيعني أن التابع من غير وسائط. الإضافة الأهم التي أتت بها لغة C++ عن لغة C هي البرمجة عن طريق الأغراض. حيث تعتمد لغة C على البرمجة الإجرائية والتي كانت كافية في وقتها. إلا أنه بظهور أنظمة التشغيل ذات الواجهة الرسومية انتقل العديد من المبرمجين إلى البرمجة الغرضية التوجه . إلا

أن لغة C مازالت تستخدم في برمجة نظام التشغيل Windows و UNIX، وأبرز هذه الإضافات هي:

- الصفوف والأغراض.
- التحميل الزائد للعمليات: ويُعدها الكثير أهم إضافة في لغة C++، لأنها جعلت من لغة C++ لغة قابلة للتوسع، هناك أكثر من 30 عملية يمكن التحميل الزائد عليها.
- القوالب.
- تعدد الأشكال.
- الوراثة: وهي إمكانية تطوير صف جديد يمكنه أن يرث جميع خصائص صف آخر، وفي هذه الحالة، يسمى الصف بصف مشتق.

ربما تعود أسباب صعوبة لغتي C و C++ لأسباب تاريخية أكثر منها واقعية وهو ما يردده المبرمجون المعتادون على استخدام لغة C++، وهذا يعود بنا إلى أصل لغة C وهي لغة CPL والتي اشتهرت بشدة تعقيدها ما تسبب في ابتعاد المبرمجين عنها فتم تطويرها وتبسيطها إلى لغة سميت BCPL، ولم تلق الكثير من النجاح ولكنها تطورت إلى لغة B، وعندما أرادوا تطويرها سموها لغة C (وهذا هو أصل تسمية C)، وبالطبع ورثت لغة C++ خصائص لغة C. إلا أنه لا يمكن إنكار أن مفاهيم مثل المؤشرات pointers، والمؤشرات هي مفاهيم مربكة للمبرمج المبتدئ، بالإضافة إلى مفاهيم الوراثة Inheritance وتعدد الأشكال polymorphism والقوالب Templates التي تترك حتى المبرمجين المحترفين. وكذلك لا تقدم لغة C++ الحماية الكافية للمبرمج كي لا يقع في أخطاء التشغيل التي قد تتسبب في توقف الحاسوب عن العمل.

إن الهدف من هذا الفصل هو التعرف على عدد من الميزات الأكثر أهمية في C++ . وكما هو معروف فإن مكونات أي لغة حاسوبية لا تعمل بشكل مستقل منفصلة عن بعضها البعض ، بل تعمل معاً كنظام متكامل .

وصممت لغة C++ لتدعم البرمجة الغرضية التوجه ، لذلك سيبدأ هذا الفصل بوصف للغة الغرضية التوجه OOP ، وسنجد من خلال هذا الوصف أن العديد من مزايا لغة البرمجة C++ تتعلق بطريقة أو بأخرى بالبرمجة غرضية التوجه، وفي الواقع تسيطر نظرية البرمجة الغرضية التوجه OOP على معظم نواحي لغة C++ . ويجب معرفة أنه يمكن أن نستخدم لغة C++ لكتابة برامج ليست غرضية التوجه ، أي أن طريقة استخدام لغة C++ تتعلق بالمبرمج بالدرجة الأولى ، إضافة إلى تقديم العديد من مزايا C++ الهامة ، ويناقش هذا الفصل بعض الفوارق بين أساليب البرمجة في لغة C و C++ والعديد من مظاهر المرونة في كتابة البرامج . مع أن بعض هذه الميزات ليس لها علاقة بالبرمجة الغرضية التوجه لكنها مستخدمة في معظم برامج لغة C++ .

تُعد لغة C++ واحدة من لغات البرمجة المنتشرة عالمياً والمستخدمه لتطوير برامج صناعية قوية وذات أداء عالي.

١-٢- البرمجة الغرضية التوجه

تُعد البرمجة الغرضية التوجه OPP طريقة قوية للتعامل مع المسائل البرمجية ، فقد ظهرت في كل نقطة من مراحل تطورها طريقة جديدة تساعد المبرمجين على التعامل مع البرامج المتزايدة التعقيد . ورغم أن البرمجة البنوية كانت تعطي نتائج ممتازة عندما تطبق على برامج متوسطة التعقيد ، إلا أنها كانت تفشل عند بعض النقاط حين يصل البرنامج إلى حد معين ، ولكتابة برامج أكثر

تعتقداً كانت الحاجة إلى طريقة جديدة في العمل البرمجي ، وتلبية لهذه الحاجة فقد ظهرت البرمجة غرضية التوجه واختصاراً " OOP " .
ويختلف البرنامج المكتوب بأسلوب OOP عن البرنامج المكتوب باللغة التقليدية بالأمور التالية :

- **وحدة بناء البرنامج:** البرنامج التقليدي عبارة عن تابع رئيسي ، ومجموعة توابع فرعية، ويتم استدعاء التوابع الفرعية من داخل التابع الرئيسي حسب تسلسل البرنامج، كما في لغة C التقليدية. أما في البرنامج المكتوب بأسلوب OOP فقد أصبحت وحدة البرنامج هي الصف class ، الذي يتكون من المعطيات والتوابع التي تعمل على هذه المعطيات ، ويتم استدعاء التوابع كأعضاء للصفوف على المعطيات ، حسب فكرة البرنامج ، وهذا ما سيتم توضيحه فيما بعد.

- **النظرة إلى المعطيات:** في البرنامج التقليدي كان مجهود المبرمج ينصب على كود البرنامج أي سطور البرنامج التي تتولى سير العمليات بينما كانت النظرة إلى المعطيات نظرة ثانوية ولكن في أسلوب البرمجة OOP فتعتبر المعطيات جزءاً مهماً من البرنامج. وبالتالي أصبحت لدينا مكتبة مكونة ليس فقط من التوابع بل من الصفوف التي تحتوي على المعطيات والتوابع التي تتعامل معها ووفرت هذه المكتبة الكثير من المجهود الذي يبذله المبرمج ، حيث كان يقوم بتعريف المعطيات ثم التوابع التي تتعامل معها كل مرة. وتظهر فائدة مكتبة الصفوف في برمجة النوافذ ، حيث يوجد صف للنافذة الرئيسية للبرنامج يتضمن جميع المعطيات التي تخص النافذة وكذلك التوابع التي تعمل معها، فأصبحت وحدة بناء البرنامج كبيرة وليس على المبرمج كتابة كل شيء من البداية. وسوف نرى كيف يقل عدد سطور البرنامج

المكتوب (من ناحية البرمجة) باستخدام مكتبة الصفوف عن البرنامج المكتوب باستخدام مكتبة التوابع. تأخذ البرمجة الغرضية التوجه OOP أفضل ميزات البرمجة البنيوية وتضم إليها مبادئ قوية وجديدة تسمح بتنظيم البرامج بفعالية أكبر . وتعتمد البرمجة الغرضية التوجه OOP على تحليل المشكلة إلى أجزائها الأساسية بحيث يصبح كل عنصر أو جزء عبارة عن كائن محتوى ذاتياً يضم تعليماته الخاصة والمعطيات العائدة إليه، ويستطيع المبرمج يستطيع إدارة برامج أكبر .

١-٣-٣ - سمات البرمجة غرضية التوجه

تتشترك جميع لغات البرمجة الغرضية التوجه OOP بما فيها C++ بثلاث سمات مميزة مشتركة وهي :

١. التغليف (Encapsulation(Classes) .
٢. الوراثة Inheritance
٣. تعدد الأشكال Polymorphism .

١-٣-١ - التغليف

Encapsulation(Classes)

يوجد عدد غير محدود من التعريفات والتفسيرات لمصطلح البرمجة غرضية التوجه ، ولكن يمكن وصف البرمجة غرضية التوجه بأنها البرمجة المتعلقة بالمعطيات والوسائل (التوابع) التي تستخدم تلك المعطيات، حيث يتم تسمية المعطيات والوسائل باسم معين هو الغرض Object ويكون هذا الغرض مكتفياً ذاتياً ، إذ يشكل وحدة برمجية متكاملة .

وحسب هذا التعريف فإن كثيراً من البرامج التي تقوم بمهمة معينة وتحتوي على المعطيات التي تحتاجها لأداء عملها تسمى الغرض Object . فمثلاً قيامنا برسم شكل أو صورة على شاشة الحاسوب فهذا غرض ، وبالتالي نستطيع وصف تلك الصورة أو الشكل باللون والحجم وكثافة التظليل ومواصفات أخرى .
والسؤال الذي يمكن طرحه الآن هو كيف تستطيع لغة ++C توفير الوسيلة التي تؤدي إلى الغرض ؟

والجواب هو أن لغة ++C توفر وسيلة تسمى وسيلة التصنيف أو الصفوف ، وهذه الوسيلة تشبه عملية بنى المعطيات Data Structures ولكنها تتصف بصفات أخرى لا تتصف بها عملية بنى المعطيات حيث تحتوي هذه الوسيلة على برامج بحد ذاتها ، وتعتمد هذه البرامج على بنى المعطيات .
إن تحديد حالة الصف class بحد ذاتها تسمى غرضاً Object، وتوفر وسيلة الصفوف مجموعة من الميزات في أداء البرمجة حيث يمكن للمبرمج تعريف متحولات تابعة للصف ويطلق عليها متحولات خاصة لا يمكن الوصول إليها من خارج الصف نفسه . وهذا الأسلوب يساعد على حماية المعلومات من البرامج غير المسموح لها بالاستفادة من معطيات معينة .
وباستخدام وسيلة الصف فإن البرامج الكبيرة ستكون واضحة المعالم حيث تكون جميع البرامج ذات العلاقة في مواقع محددة لها ، وهذا الأسلوب يؤدي إلى مفهوم الوحدات المتكاملة و بالتأكيد يؤدي إلى الصيانة الكفاءة وتسهيلها .

١-٣-٢ - الوراثة

Inheritance

إن الوسيلة الثانية التي نتصف بها البرمجة غرضية التوجه هي الوراثة Inheritance والاستفادة من الخصائص المعرفة في برامج أخرى سابقة ، ويطلق

مصطلح الوراثة على العملية التي يكتسب غرض ما من خلالها على خصائص غرض آخر . وهذه الوسيلة تمكّن المبرمج من الحصول على صفوف جديدة وذلك بالاستفادة من التعريفات والبرامج المكتوبة في صفوف أخرى سابقة بالإضافة إلى الخصائص الجديدة التي يمكن إضافتها ، أي يستطيع غرض ما أن يرث مجموعة عامة من الخصائص ، ثم يضيف إليها الميزات الخاصة به وحده ، وهذه الوسيلة توفر الوقت والجهد للمبرمج .

وفي الوراثة يستطيع صف ما أن يرث صفاً أساسياً Base Class وذلك بتحديد الصف الأساسي الذي ينتمي إليه بالإضافة إلى الصفات الخاصة التي تجعله مميزاً عن غيره . وقد تساعد طريقة الوراثة على الحصول على معلومات من أكثر من صف وهذه الطريقة تسمى بالوراثة المتعددة Multiple Inheritance. وباستخدام الوراثة يستطيع المبرمج بناء مجموعة كبيرة من الصفوف التي ستكون بمثابة العمود الفقري لأي برنامج .

١-٣-٣- تعدد الأشكال

Polymorphism

الوسيلة الثالثة التي تتصف بها البرمجة الغرضية التوجه OOP هي الاستعمال المتعدد الأغراض ، حيث يتم استخدام اسم التابع نفسه لأغراض متعددة .

و تمثل تعدد الأشكال الميزة التي تسمح لاسم واحد أن يُستخدم لهدفين مترابطين أو أكثر لكنهما مختلفين تقنياً . وتدعم لغة ++C تعدد الأشكال ، ويمكن استدعاء هذه التوابع بالاسم نفسه ، ويحدد نوع المعطيات المستخدم لاستدعاء التابع النسخة المحددة من التابع الواجب تنفيذها ، وتكمن فائدة تعدد الأشكال في أنها تساعد على تخفيف التعقيد بالسماح لواجهة واحدة أن تعرف

بمجموعة عامة من الأفعال ، وتسمح طريقة تعدد الأشكال بالتعامل مع تعقيد أكبر من خلال إنشاء واجهة قياسية تشمل جميع الفعاليات المترابطة .

١-٤-١ - البنى Structures

١-٤-١ - مقدمة

من المعلوم أن من أهم التطبيقات في عالم البرامج تطبيقات قواعد المعطيات ، فمثلاً قاعدة معطيات موظفين ، تمثل معطيات الموظفين في شكل سجلات وكل سجل يتكون من مجموعة حقول، ومن خلال برامج قواعد المعطيات نعلم أن الملف ينقسم إلى سجلات والسجل ينقسم إلى حقول ، ودائماً نحتاج للتعامل مع السجل كوحدة وكذلك مع الحقول كوحدة. وتستخدم لغة C/C++ الكلمة المحجوزة Structure بالمفهوم نفسه الذي نستخدمه لغات البرمجة الأخرى لكلمة Record .

١-٤-٢ - تعريف البنية

البنى في لغة C++ هي نمط من أنماط المعطيات وهي عبارة عن تجمع من المتغيرات بينها علاقة تآلف ، ويمثلها اسم واحد يسمى كل منها عنصراً element أو عضواً member .

وتعد البنى مفيدة لتنظيم المعطيات المعقدة، خاصة في البرامج الكبيرة، فهي تسمح بالتعامل مع عدد من المتحولات المترابطة كما لو أنها كيان واحد وليس مجموعة كيانات منفصلة.

هناك العديد من الأمثلة التقليدية التي نتجلى من خلالها أهمية وفائدة البنى، منها مثلاً البنية المستخدمة لمثيل المعلومات الخاصة بموظف في برنامج لإدارة ذاتية ورواتب موظفين.

لنأخذ البنية التالية Time :

```
struct time {  
    int hour , minute , second ;  
} ; // End struct
```

١. تبدأ البنية بالتعليمة struct .
 ٢. ثم اسم البنية Time .
 ٣. ثم القوسين " {...} " اللذان يعبران عن جسم البنية .
 ٤. أما المتغيرات الموجودة ضمن القوسين " {...} " فهي تعبر عن مجموعة الحقول المرتبطة بالبنية . لكل حقل ضمن البنية اسم يختلف عن الآخر .
 ٥. يجب ان ينتهي تعريف البنية بفاصلة منقوطة " ; " .
 ٦. يتضمن تعريف البنية Time ثلاثة حقول من النمط int ويمكن أن تكون من أي نوع آخر (double , char) .
- لا يمكن التصريح عن أي متغير يحمل نفس اسم البنية . نسمي البنية التي تتضمن حقلاً على شكل مؤشر على سجل من نفس نمطه " بنية التأثير الذاتي " ، ويتم استخدامه من أجل ربط المعطيات مع بعضها البعض من أجل ربط المعطيات مع بعضها البعض .
- لا يقوم التعريف السابق بحجز أي مكان في الذاكرة ، فهو ليس إلا مجرد تعريف لنمط جديد من المعطيات يمكن أن يُستخدم للتصريح عن متحولات بنفس الطريقة المستخدمة للتصريح عن متحولات من أنماط أخرى مثلاً الجزء البرمجي التالي :

```

Time obj ;           // object of Class Time
Time array[ 20 ] ;   // array of Time Object
Time *P = & obj ;    // Pointer to a Time Object
Time & t = obj ;     // Reverence to a Time Object

```

يتم التصريح عن المتحول obj من النمط Time وعن array [20] من النمط Time وعن المؤشر P من النمط Time مع إعطائه عنوان obj كقيمة ابتدائية وعن المتحول t الذي يحمل عنوان الغرض .

الصيغة العامة

```

struct name {
    Type Variable1 ;
    .....
    Type Variable 2 ;
} ; //End struct

```

١-٤-٣- الوصول إلى حقول البنية

يتم الوصول إلى حقول البنية باستخدام عمليات الوصول إلى الأعضاء هي :

١. العملية " نقطة " (.) Dot Operator

٢. العملية " سهم " (->) Arrow Operator

تفيد العملية (.) في الوصول إلى حقول البنية أو إلى الأعضاء باستخدام اسم المتحول الذي يعبر عن تلك البنية . مثلاً الجزء البرمجي التالي :

```
cout << obj . hour ;
```

```
Cout << obj . minute ;
```

```
Cout << obj . second ;
```

أي يتم الوصول إلى أي عنصر من عناصر البنية second , minute , hour وذلك باستخدام اسم المتحول متبوعاً بنقطة ثم متبوعاً بنوع المتحول المراد انتقاؤه. لطباعة قيمة الحقل hour من الغرض المؤشر عليه بواسطة t نستخدم التعليمة التالية :

```
Cout << obj . hour ;
```

تفيد اشارة السهم (->) في الوصول إلى حقول البنية باستخدام مؤشر على تلك البنية . مثلاً الجزء البرمجي التالي :

```
Cout << P -> hour ;
```

```
Cout << P -> minute ;
```

```
Cout << P -> second ;
```

مثال (1-1)

البرنامج التالي يقوم بطباعة درجات الطلاب Mohamed , Ali , Omar لمقررات البرمجة (1) ، والبرمجة (2) ، وبنية الحاسوب .

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

```
struct studentRate {
```

```
    int prog1 ;
```

```
    int prog2 ;
```

```

        int comp ;
    }; // End struct

    void main ( )
    {
        studentRate Mohamed , Ali , Omar ;
        Mohamed.prog1=70;
        Mohamed.prog2=75;
        Mohamed.comp =66;

        // studentRate Muhamed
        cout<<" ----- Muhamed -----"<< endl;
        cout<<"prog1 = "<<Mohamed.prog1<<endl;
        cout<<"prog2 = "<<Mohamed.prog2<<endl;
        cout<<"comp = "<<Mohamed.comp<<endl;

        // studentRate Ali
        cout<<" ----- Ali -----"<< endl;
        Ali.prog1=60;
        Ali.prog2=70;
        Ali.comp =86;
        cout<<"prog1 = "<<Ali.prog1<<endl;
        cout<<"prog2 = "<<Ali.prog2<<endl;
        cout<<"comp = "<<Ali.comp<<endl;
    }
}

```



```

// studentRate Omar
cout<<" ----- Omar-----"<< endl;
    Omar.prog1=60;
    Omar.prog2=80;
    Omar.comp =80;
cout<<"prog1 = "<<Omar.prog1<<endl;
cout<<"prog2 = "<<Omar.prog2<<endl;
cout<<"comp = "<<Omar.comp<<endl;

// studentRate Omar
cout<<" Omar = "<< ( Omar.prog1+ Omar.prog2+
                    Omar.comp) / 3<<endl;

// studentRate Ali
cout<<" Ali = "<< ( Ali.prog1+ Ali.prog2 +
                    Ali.comp)/3<<endl;

// studentRate Mohamed
cout<<" Mohamed Rate = "<< (Mohamed.prog1 +
Mohamed.prog2+Mohamed.comp) /3<<endl;

} // End main

```

RESULT

----- Mohamed -----

prog1 = 70

prog2 = 75

comp = 66

----- Ali -----

prog1 = 60

prog2 = 70

comp = 86

----- Omar -----

prog1 = 60

prog2 = 80

comp = 80

Omar Rate = 73.333

Ali Rate = 72

Mohamed Rate = 70.333

Press any key to continue

مثال (1- 2)

اكتب برنامجاً بلغة C++ عن حالة الطقس **weather** ، ويقوم هذا البرنامج بإدخال وطباعة المتحولات التالية : درجة الحرارة **temp** ، والضغط **press** ، وسرعة الرياح **wind** . استخدم طريقتين للوصول إلى المعطيات .

١. الوصول إلى المتحولات عن طريق العملية (.) .

```
#include <iostream.h>

struct weather {
    int    temp ;
    float  press ;
    float  wind ;
}; //End struct

void main ( ) {
    weather today ;
    today.temp=25;
    today.press=755.5;
    today.wind=66.5;
    cout<<"temp= "<<today.temp<<endl;
    cout<<"press= "<<today.press<<endl;
    cout<<"wind= "<<today.wind<<endl;
} //End main
```

RESULT

temp= 25
press= 755.5
wind= 66.5

Press any key to continue

٢. الوصول إلى المتحولات باستخدام العملية (->) .

```
#include <iostream.h>

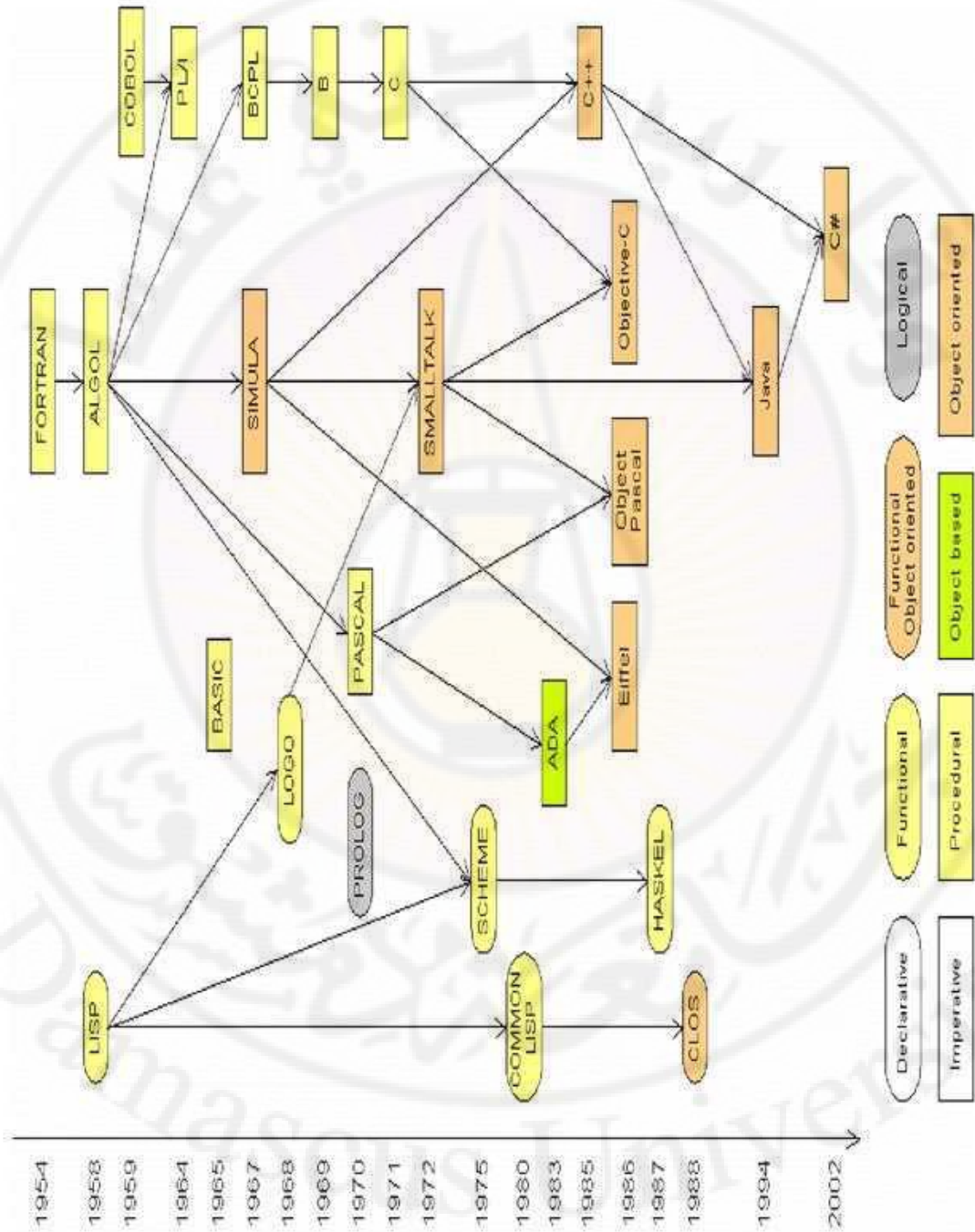
struct weather {
    int temp ;
    float press , wind ;
}; //End struct

void main ( ) {
    weather today1 ;
    weather *today ;
    today = &today1;
    today->temp=25;
    today->press=755.5;
    today->wind=66.5;
    cout<<"temp= "<<today -> temp<<endl;
    cout<<"press= "<<today -> press<<endl;
    cout<<"wind= "<<today -> wind<<endl;
} // End main
```

RESULT

```
temp= 25
press= 755.5
wind= 66.5
Press any key to continue
```

يبين الشكل التالي تطور لغات البرمجة .





الفصل الثاني
الصفوف في C++
CLASSES IN C++





الصفوف في C++

٢-١ - مقدمة

تُعد الصفوف من أهم ميزات C++ ، فهي الآلية التي تستخدم لإنشاء الأغراض Objects . لذلك تعتمد عليها الكثير من ميزات لغة C++ الأخرى. في البرمجة الغرضية التوجه نفكر في تقسيم المشكلة إلى أغراض (Objects) ، فعلى سبيل المثال :

1. برنامج عن أفراد الكلية

Humans : Students , Teachers , Dean

2 . برنامج عن الرسم البياني :

Graphics : Points , Lines , Circles

3. برنامج العمليات الرياضية :

Mathematics : Complex , Numbers , Matrixes

4 . برنامج الزمن أو الوقت :

Time : Hours , Minutes , Seconds

5 . برنامج عن الطلاب :

Student : Name , student Number , Credit

ويصرح عن صف باستخدام الكلمة المحجوزة **class** ، ويشبه التصريح عن الصف بالتصريح عن البنية structure، والبنية العامة للصف لها الشكل التالي :

1. **class** *class_name* {

2. **private** :

// private function and Variable . متحولات وتوابع خاصة

(*data members*)

3. public :

// public function and Variable . متحولات وتوابع عامة

(*member functions*)

4. } ; //object_list

- يبدأ الصف باسم الصف *class_name* (السطر 1) ثم القوسين " { } " الذي يدل على بداية جسم الصف ونهايته .
- ثم المتحولات والتوابع الخاصة المصريح عنها داخل الصف وتدعى بأعضاء الصف *class members* ، وتكون جميع المتحولات والتوابع الخاصة المصريح عنها داخل الصف خاصة بهذا الصف ، ويكون الوصول إليها فقط من قبل الأعضاء الأخرى في هذا الصف .
- للتصريح عن الأعضاء الخاصة في الصف نستخدم الكلمة المحجوزة **private** متبوعة بنقطتين (السطر 2) .
- للتصريح عن الأعضاء العامة في الصف نستخدم الكلمة المحجوزة **public** متبوعة بنقطتين (السطر 3) ، ويمكن الوصول إلى الأعضاء العامة بعد الكلمة **public** من قبل أي جزء آخر من البرنامج الذي يحتوي الصف ، أما الأعضاء الخاصة لا يمكن الوصول إليها إلا من قبل الأعضاء العامة للصف نفسه . فيما يلي مثال بسيط للتصريح عن صف اسمه : student

```
class student {
```

```
private :
```



```
int      student_id ;  
double   credit ;  
chare     student_name [ 30 ] ;
```

public :

```
void setstudent ( int , char * , double ) ;  
void print_stud ( ) ;  
} ; // End class
```

- يدعى التابع الذي يعرف ضمن الصف تابِعاً عضواً (member function) بعد التعليمة public ، وتدعى المعطيات المعرفة ضمن الصف بأعضاء المعطيات (Data members) بعد التعليمة (الكلمة المحجوزة) private .
- إن الصف class هو عبارة عن بنية تحتوي على الأسماء التعريفية والمعطيات بأنواعها int , char ، وأيضاً float , double ، وتحتوي أيضاً على تعاريف التوابع العامة بأنواعها .

٢-٢- بناء نمط المعطيات في الصف

تساعد الصفوف المبرمج على نمذجة الأغراض التي لها عدة صفات ، كما يمكنها أن تقوم بتنفيذ عدة أفعال أو عمليات ، يبين البرنامج في المثال (1-2) بناء المعطيات باستخدام مفهوم الصف student :

مثال (1-2)

```
# include <iostream.h >  
# include <string.h >
```

```

class student { // Class Declaration
private :
    int student_id ;
    char student_name[30] ;
    float credit ;
public :
    void setstudent ( int , char* , float ) ;
    void print_stud ( ) ;
}; // End class

void student :: setstudent ( int k , char *m , float d ) {
    student_id = k ;
    strcpy ( student_name , m ) ;
    credit = d ;
} // body of the function

void student :: print_stud ( ) {
    cout << student_id<<student_name<<" ";
    cout << credit << endl ;
} // body of the function

void main( )
{
    student samer , mariam ;

```

```
samer.setstudent( 965654 , " Samer  Ali  " , 80.0 ) ;
mariam.setstudent ( 123456 , " Mariam  Imran" , 90.0 );
samer . print_stud ( ) ;
mariam . print_stud ( ) ;
} // End main
```

Result

965654 Samer Ali 80

123456 Mariam Imran 90

Press any key to continue

ندرس بناء المعطيات في الصف student

- يبدأ الصف بالتعليمة class ثم اسم الصف student ، ثم المعطيات الخاصة private والتوابع العامة public ، وذلك ضمن القوسين { } والذي يسمى بجسم الصف .
- ينتهي دوماً أي صف بفاصلة منقوطة (;) ، وهذا يجعل الصف وحدة متكاملة .
- تدعى المحددات private و public بمحددات طريقة الوصول إلى الأعضاء Member Access Specifies . ويفيد التصريح عن المعطيات الأعضاء والتوابع الأعضاء بعد التعليمة public في جعل هذه المعطيات أو التوابع متاحة للاستخدام من أي مكان ضمن البرنامج وتسمى بالمعطيات أو

التوابع العامة أو واجهة الصف Interface of class ، حيث يمكن الوصول إلى أي غرض من الصف student .

- يمكن استدعاء التابعين () setstudent و () print_stud من أي جزء من البرنامج لأنهما أعضاء عامة في الصف student .

- أما المعطيات الأعضاء والتوابع الأعضاء المصرح عنها بعد التعليمات private فهي متاحة فقط للتوابع الأعضاء العامة المرتبطة بالصف وتسمى بالمعطيات الخاصة .

- أما المعطيات الخاصة في الصف student فهي : student_id , student_name , credit ، لا يمكن الوصول إليها من خارج الصف student . مع أن التصريح عن التوابع () setstudent و () print_stud قد تم دون تعريف لجسم التابع ، وسنشرح ذلك فيما بعد .

- تنتهي محددات طريقة الوصول إلى الأعضاء دائماً بنقطتين أي : private ويمكن أن تظهر عدة مرات ضمن الصف ولا يوجد أي ترتيب لورودها .

- نسمي التابع العضو الذي يحمل اسم الصف بالتابع الباني Constructor المرتبط بذلك الصف ، و يقوم التابع الباني بإعطاء قيم ابتدائية وتميرير القيم إلى المعطيات الأعضاء لكل غرض من أغراض الصف . حيث يتم استدعائه أولاً عند إنشاء الغرض ، وسوف نرى فيما بعد عدة توابع بانية لصف واحد ويتم ذلك من خلال التحميل الزائد للتابع كما سنرى لاحقاً .

- أما تعريفات المعطيات بعد التعليمات private وهي تدل على أنه لا يمكن الوصول إلى المعطيات الأعضاء الخاصة إلا من خلال التوابع الأعضاء العامة والتوابع الصديقة كما سنرى لاحقاً .

- عادة تُذكر المعطيات الأعضاء الخاصة ضمن المحدد الخاص private وتذكر التوابع الأعضاء العامة بعد المحدد العام public . ولكن يمكن أن

- يكون لدينا توابع أعضاء خاصة ضمن الجزء `private` ومعطيات أعضاء عامة ضمن الجزء `public` ولكن هذا يدل على ضعف في البرمجة .
- يمكن تعريف التوابع الأعضاء ضمن الصف ، ولكن يفضل أن تكون خارج تعريف الصف .
- العملية الثنائية (::) تستخدم لتحديد مجال الرؤية `Binary Scope Resolution Operator` .

ويمكن تعريف أي تابع من توابع الأعضاء داخل جسم الصف ، أو يمكن ذكر ذلك التعريف بعد انتهاء الصف المرتبط به . عند تعريف تابع عضو بعد ذكر تعريف الصف يجب أن يسبق اسم ذلك التابع اسم الصف متبوعاً بالعملية الثنائية لتحديد مجال الرؤية (::) وعلى اعتبار أنه يمكن أن يكون لعدة أعضاء من صفوف مختلفة بالاسم نفسه .

وتقوم عملية تحديد مجال الرؤية بالربط بين اسم العضو واسم الصف بحيث نحصل على تحديد وحيد للتوابع الأعضاء المرتبطة بصف محدد ، حتى ولو تم تعريف التابع خارج تعريف الصف فإنه يبقى خاضعاً لقواعد مجال رؤية الصف ، أي يبقى معروفاً من قبل بقية أعضاء الصف .

إن المعطيات الخاصة بعدد محدد الوصول `private` هي معطيات خاصة بالصف ولا يمكن الوصول إليها من خارجه ، إلا من خلال التوابع الصديقة `friends` كما سنرى لاحقاً ، وتهدف إلى حجب المعطيات الخاصة عن المستخدمين له ، فهي تعد بنية مخفية عن زبائنه ، وهي تساعد على زيادة القدرة على تعديل البرنامج وعلى تبسيط تعامل الزبائن مع الصف .

لنعيد كتابة المثال (2-1) باعتبار يوجد تابع بانّي في الصف، وسنلاحظ ما هي الاختلافات بين البرنامج الأول الذي يحتوي على تابع تعديل معطيات والبرنامج الثاني الذي يحتوي على تابع بانّي عادي .المثال (2-2) يوضح ذلك .

ويبين البرنامج التالي الصف student :

مثال (2-2)

```
# include <iostream.h>
# include <string.h>
class student { // Class Declaration
private :
    int student_id ;
    char student_name[30] ;
    float credit ;
public :
    student ( int , char* , float ) ;
    void print_stud ( ) ;
} ; // End class

student :: student ( int k , char *m ,float d )
{ student_id = k ;
  strcpy ( student_name , m ) ;
  credit = d ;
} // body of the function

void student :: print_stud ( )
{
```

```

        cout << student_id<<student_name<< " ";
        cout << credit << endl ;
    }    // body of the function

void  main( )
{
    student  samer ( 965654 , " Samer  Ali  " , 80.0 ) ;
    student  mariam (123456 , " Mariam Imran" , 90.0 ) ;
    samer . print_stud ( ) ;
    mariam . print_stud ( ) ;
} // End main

```

Result

965654 Samer Ali 80

123456 Mariam Imran 90

Press any key to continue

ويقوم التابع الباني بإسناد قيم إلى معطيات الأعضاء ، أي تمرير المعطيات ، ويسمى بتابع تمرير المعطيات أو تابع تعديل المعطيات ، ويمكن استخدام تابع غير التابع الباني في تمرير أو تعديل المعطيات للقيام بذلك .

إن التوابيع الأعضاء مثل (print_stud) تعرف بشكل ضمني وأنه يقوم بطباعة قيم المعطيات الأعضاء للغرض الذي يرتبط بالصف student . ما

يساعد على جعل التابع العضو أكثر ارتباطاً بالمعطيات ، وأن طريقة استخدام الصفوف تشبه إلى حد كبير مفهوم السجلات (البنى Structures) .
إذا رغبتنا في تعريف غرض من الصف student ولنفرض الغرض Samer نكتب ما يلي :

student Samer , Marina , Mariam ;

وللوصول إلى المعطيات نكتب :

student Samer (965654 , " sami Hamza " , 80.0) ;

student Mariam (123456 , " Mariam Imran" , 90.0);

إن التابع () student الذي يحمل اسم الصف هو عبارة عن تابع بانى والذي يستدعى عند تكوين الصف student ، ويتم استدعاء التابع البانى عند إنشاء أي غرض object من الصف بشكل آلي . لا يمكن إعطاء قيم ابتدائية للمعطيات الأعضاء ضمن تعريف الصف وإنما يتم بواسطة التابع البانى المرتبط بالصف .

مثلاً :

student Samer (965654 , " Sameri Ali " , 80.0) ;

فهي تعرف الغرض Samer على أنه غرض من الصف student ، وبعد ذلك يتم تحديد القيم الابتدائية كرقم الطالب واسمه ومعدله ، أي :

credit = 0 P ;

student_name = "Samer Ali" ;

student_id = 9656 ;

ويمكن الوصول إلى أي عنصر ينتمي إلى الصف باستخدام الأداة (.) dot أو السهم (->) كمؤشر على غرض وذلك كما يلي :

Samer . Print_stud () ;

أما السهم :

```
student *P ;  
P=&Samer ;  
P-> Print_stud ( ) ;
```

فإن المعطيات ستظهر بعد استدعائه بكلى الحالتين كما يلي :

965654 Samer Ali 80.0

لذلك يُعد Samer غرض object معرف حسب تعريف student على عناصر الصف ، وأن المتغيرات التي تقع ضمن private هي متغيرات خاصة بالمجموعة نفسها ولا يمكن الإشارة إليها خارج نطاق المجموعة ذاتها فمثلاً :

Samer . Credit = 60.5 ;

فهي متغيرات غير صحيحة لأن credit متغير خاص .

أما التابع ضمن الجزء public فيمكن الإشارة إليه خارج حدود الصف وهذا الأسلوب يساعد على حصر المعلومات الخاصة عن المعلومات العامة. وهذا يؤدي إلى جمع المعطيات والتابع تحت اسم واحد ، ويساعد على سهولة تداولها والإشارة إليها عند الحاجة وعدم إضاعة الوقت في إعادة تعريف التابع والمتغيرات لكل برنامج وإنما الإشارة إلى الاسم الرئيسي لها فقط .

٢-٣- كيفية الوصول إلى أعضاء الصف

يمكن الوصول إلى أعضاء معطيات الصف باستخدام العملية dot (.) أو العملية سهم (->) ، تفيد العملية نقطة (.) في الوصول إلى أعضاء الصف باستخدام اسم المتحول الذي يعبر عن ذلك الصف أما إشارة السهم (->) فهي

تساعد في الوصول إلى أعضاء الصف باستخدام مؤشر على ذلك الصف ،
يوضح المثال (2-3) كيفية الوصول إلى الأعضاء :

مثال (2-3)

```
#include <iostream.h>

class count {
public :
    count( int x1 ){ x = x1 ; }    // default constructor
    void print( )
    {
        cout << x << endl ;
    }
private :
    int x;
}; // end class Cuont

void main( )
{
    count c1( 10 ); // إنشاء غرض
    count *p;       // إنشاء مؤشر
    p = &c1 ;
    count &ref = c1 ; // إنشاء مرجع أو عنوان
```



```

c1.print ( ) ;
p -> print( ) ;
ref.print( ) ;

} // End main

```



لدينا في البرنامج ثلاث متحولات أو أغراض من النمط count وهي c1 و ref وهو عنوان غرض تابع للصف count و p مؤشر .

٢-٤- تجزئة البرنامج

إن عملية تجزئة البرنامج بلغة C++ تعني الفصل بين الواجهات والنصوص البرمجية ، والتي تعد أحد أهم المبادئ الأساسية في صياغة البرامج بصورة جيدة ، وهذا الأمر يساعد على تسهيل عملية تعديل البرامج وتطويرها ، وبالتالي يمكن تطوير وتعديل طريقة عمل الصف مع الحفاظ على واجهة ثابتة للتعامل معه. ولهذه الغاية يجب وضع التصريح عن الصف (تعريف الصف)، ضمن ملف رأسي يحمل اللاحقة (.h) ، وذلك من أجل إدراج هذا الملف ضمن أي ملف خاص بزيون يقوم بالتعامل مع هذا الصف. وأيضاً يجب وضع كافة تعريفات التوابع الأعضاء الخاصة بالصف ، بالإضافة إلى التابع الرئيسي main ضمن ملف آخر يحمل اللاحقة (.cpp) ، والذي يُشكل بدوره ملف النصوص البرمجية

للتوابع المرتبطة بالصف. ومن المعلوم ، أنه في غالبية الأحيان لا تحتاج زبائن صف للوصول إلى النصوص البرمجية للتوابع المرتبطة بالصف من أجل التعامل معه ، وإنما تحتاج إلى امتلاك قدرة الربط مع النصوص البرمجية الخاصة بغرض مشتق من الصف المعني. وبالتالي فإن استخدام مفهوم تجزئة البرنامج يساعد المبرمجين على صياغة مكتبات من الصفوف للبيع، حيث يقومون بإعطاء المستخدم الملفات الرأسية والملفات الناتجة عن ترجمة ملفات نصوص التوابع الخاصة بالصف (التي تحمل اللاحقة cpp)، دون إعطاء معلومات إضافية لها علاقة بالنصوص البرمجية الفعلية للتوابع.

وفيما يلي سنوضح كيف نقوم بتجزئة البرنامج الوارد في المثال (2- 2)

والمتعلق بالصف **student** إلى ثلاثة ملفات :

الملف الأول : نضع في هذا الملف مواصفات الصف (تعريف الصف) والذي ندعوه بالملف الرأسي ، وفي حالة الصف **student** يأخذ الشكل التالي:
" student . h " . عندئذ ، فإن الشكل العام لهذا الملف يصبح كما يلي:

```
#ifndef STUDENT_H
```

```
#define STUDENT_H
```

```
<class definition> \student
```

```
#endif
```

حيث إن الموجهان **#ifndef** و **#endif** يساعدان على عدم إدراج النص البرمجي المحصور بينهما (تعريف الصف) إذا كان الاسم **STUDENT_H** (الاختياري) معرف مسبقاً. فإذا لم يتم إدراج الملف الرأسي قبل ذلك ، فسيتم تعريف الاسم **STUDENT_H** باستخدام الموجه **#define** ومن ثم يتم إدراج محتويات الملف الرأسي الأخرى. أما إذا تم إدراج الملف الرأسي قبل ذلك، عندها

سيكون الاسم STUDENT_H معرّفاً مسبقاً ولا يمكن بالتالي إدراج محتوياته مرة ثانية. وباختصار فإن استخدام الموجهات `#endif` و `#define` و `#endif` الخاصة بمرحلة ما قبل الترجمة، يمنع إدراج الملفات الرأسية أكثر من مرة واحدة ضمن نص البرنامج.

الملف الثاني: نضع فيه كافة النصوص البرمجية للتتابع الأعضاء المرتبطة بالصف `student` ، ويمكن أن يأخذ هذا الملف الاسم :
" `student_members . cpp` " حيث أن `cpp` هي اللاحقة المستخدمة للدلالة على برنامج `C++` . وفي هذا الملف يجب إدراج الملفات التوجيهية التالية، قبل البدء بكتابة النصوص البرمجية الخاصة بالتتابع الأعضاء الخاصة بالصف:

```
#include <iostream.h > // ملف التوجيه الخاص بالإدخال والإخراج
#include <string.h > // ملف التوجيه الخاص بالسلاسل المحرفية
#include "student.h" // الملف الرأسي الحاوي على تعريف الصف
```

الملف الثالث: نضع فيه التابع الرئيسي (`main`) والذي يتضمن التصريح عن أغراض من الصف `student` ، بالإضافة إلى الاستدعاءات اللازمة لتنفيذ التتابع الأعضاء الخاصة بالصف `student` . ويمكن أن يأخذ هذا الملف الاسم " `student_main . cpp` " ، ولكن ضمن هذا الملف يجب أن نقوم بإدراج الملفين التوجيهيين التاليين:

```
# include "student.h"
# include " student_members . cpp "
```

وفيما يلي سنستعرض محتويات الملفات الثلاثة الناتجة عن تجزئة البرنامج الواردة في المثال (2 - 2) والمتعلق بالصف `student` :

اسم الملف : " student . h "

الملف الرئيسي الأول

الذي نجد ضمنه تعريفاً

للصف student

```
#ifndef STUDENT_H
#define STUDENT_H
class student // Class Declaration
{
    private :
        int    student_id ;
        char   student_name[30] ;
        float  credit ;
    public :
        student ( int , char* , float ) ;
        void print_stud ( ) ;
} ; // End class
#endif
```

اسم الملف : student_members.cpp

الملف الرئيسي الثاني

الذي نجد ضمنه

النصوص البرمجية

للتوابع العامة للصف

student

```
# include < iostream .h >
# include < srting .h >
# include " student .h "
```



```
student :: student ( int k , char *m ,float d )
```

```
{  
    student_id=k ;  
    strcpy ( student_name , m ) ;  
    credit = d ;  
} // body of the function
```

```
void student :: print_stud ( ) {  
    cout << student_id<< " " <<student_name<<" ";  
    cout << credit << endl ;  
} // body of the function
```

اسم الملف : student_main.cpp

```
# include < iostream .h >  
# include < string .h >  
# include "student.h"  
# include " student_members . cpp "
```

```
void main( ) {  
    student  samer ( 965654 , " Samer  Ali  ", 80.0 ) ;  
    student  mariam (123456 , " Mariam Imran" , 50.0 ) ;  
    samer . print_stud ( ) ;  
    mariam . print_stud ( ) ;  
} // End main
```

الملف الراسي الثالث

الذي نجد ضمنه البرنامج الرئيسي

student للصف main ()

ونلاحظ أن الملف الرئيسي الثاني والثالث يتضمنان الملف الرئيسي "student.h" ضمن علامتي اقتباس مزدوجة " " ، بدلاً من إشارتي <.....> . ويجري عادة وضع الملفات الرأسية المعرفة من قبل المبرمج ضمن نفس الدليل (المجلد) المتضمن للبرنامج الرئيسي وبالتالي فإن البحث عن هذا الملف الرئيسي والمكتوب ضمن علامتي اقتباس مزدوجة " " ، يتم في نفس الدليل (المجلد) الذي تتواجد فيه الملفات الرأسية ، وإذا لم يتم العثور على هذا الملف فإن عملية البحث تتم في مكتبة C++ ، أما إذا تم وضع الملف الرئيسي ضمن العلامتين <.....> ، أي بالشكل <student.h> ، فإن عملية البحث تتم حصراً ضمن مكتبة C++ لأن هذا الاستخدام يعني بشكل مفترض أن هذا الملف تابع للمكتبة المعيارية الخاصة بلغة C++ ولا تتم عملية البحث ضمن الدليل المحلي المرتبط بالبرنامج . ومن أجل تنفيذ البرنامج الخاص بالصف student يجب إتباع الخطوات التالية:

- التأكد بشكل نظري من عدم ارتكاب أي خطأ برمجي أو إملائي عند إعداد الملف الأول student.h (المتضمن تعريف الصف) .
- ترجمة البرنامج المحتوى في الملف الثاني student_members.cpp ومن ثم ربطه مع مكتبة اللغة.
- ترجمة البرنامج المحتوى في الملف الثالث student_main.cpp ومن ثم ربطه مع مكتبة اللغة ، بعد ذلك ينفذ هذا البرنامج . وبعد القيام بهذه الخطوات الثلاثة ، فسنحصل على الخرج التالي:

Result

```
965654 Samer Ali 80
123456 Mariam Imran 50
Press any key to continue
```

ويمكن تجزئة البرنامج الوارد في المثال (2-2) والمتعلق بالصف student إلى ملفين وذلك لتلافي استخدام الموجهات في الطريقة السابقة :

الملف الأول : نضع في هذا الملف مواصفات الصف (تعريف الصف) والذي ندعوه بالملف الرأسي ، وأيضاً نضع فيه كافة النصوص البرمجية للتوابع الأعضاء المرتبطة بالصف ، وفي حالة الصف student يأخذ هذا الملف الاسم " student . h " .

الملف الثاني : نضع فيه التابع الرئيسي (main) والذي يتضمن التصريح عن أغراض من الصف student ، بالإضافة إلى الاستدعاءات اللازمة لتنفيذ التوابع الأعضاء الخاصة بالصف student . ويمكن أن يأخذ هذا الملف الاسم " student_main . cpp " ، ولكن ضمن هذا الملف يجب أن نقوم بإدراج الملف التوجيهي student . h .

" student . h " : اسم الملف

الملف الرأسي الأول

الذي نجد ضمنه تعريفاً
للصف student
والنصوص البرمجية
للتوابع العامة للصف

```
#include < string.h >
```

```
class student { // Class Declaration
```

```
private :
```

```
int student_id ;
```

```
char student_name[30] ;
```

```
float credit ;
```

```

public :
    student ( int , char* , float ) ;
    void print_stud ( ) ;
} ; // End class

student :: student ( int k , char *m , float d ) {
    student_id=k ;
    strcpy ( student_name , m ) ;
    credit = d ;
} // body of the function

void student :: print_stud ( ) {
    cout << student_id<< " " <<student_name<<" ";
    cout << credit << endl ;
} // body of the function

```

اسم الملف : student_main.cpp

```

# include < iostream .h >
# include < string .h >
# include "student.h"
# include " student_members . cpp "

void main( ) {
    student  samer ( 965654 , " Samer  Ali  " , 80.0 ) ;
    student  mariam (123456 , " Mariam Imran" , 50.0 ) ;

```

الملف الرئيسي الثاني

الذي نجد ضمنه البرنامج الرئيسي

main () للصف **student**

```
samer . print_stud ( );
mariam . print_stud ( ) ;
} // End main
```

٢-٥- تعريف المتحولات كأعضاء عامة

يوضح البرنامج في المثال (4) كيف يمكن تعريف المتغيرات كأعضاء عامة . حيث تم التصريح عن المتحول a بعد التعليمة public وبذلك يمكن الوصول إلى المتحول a من أي جزء من البرنامج :

مثال (4-2)

```
#include <iostream.h>

class myclass {
public :
    // now a is public
    int a;
    // and there is no need for set_a() or get_a()
};

int main( )
{
    myclass ob1, ob2;
    // here, a is accessed directly
```

```

ob1.a = 10;
ob2.a = 99;
cout << ob1.a << "\n";
cout << ob2.a << "\n";
return 0;
} // End main .

```

RESULT

```
ob1.a = 10;
```

```
ob2.a = 99;
```

Press any key to continue

بما أن المتحول `a` معرف كعضو عام للصف `myclass` ، يستطيع التابع `main ( )` الوصول إليه مباشرة ، واستعملنا النقطة (`.`) للوصول إلى العضو `a`.
 عموماً ، لاستدعاء تابع عضو ما أو للوصول إلى متحول عضو ما من خارج الصف نحتاج إلى كتابة اسم الغرض متبوعاً بعامل النقطة ثم اسم العضو لتحديد عضو الغرض المشار إليه .

٢-٦- التابع الباني constructor

إن التابع `student ( )` الذي يحمل اسم الصف هو عبارة عن التابع الباني والذي يستدعى عند تكوين الصف `student` ولا يعيد أي نوع ويمكن أن يكون بدون وسطاء أو يأخذ وسيط أو أكثر . ويتم استدعاء التابع الباني عند إنشاء

أي غرض object من الصف بشكل آلي (تلقائي). لذلك توضع جميع عمليات التهيئة التي يحتاجها الغرض في هذا التابع .
ويجب أن نعرف أن عبارة التصريح عن متحول في C++ هي عبارة تنفيذية بينما كانت تعتبر في C مجرد تصريح عن متحولات . أما في C++ باعتبار أن الغرض قد يمتلك بانياً فإن عبارة التصريح عن متحول قد تسبب حدوث مجموعة من الأفعال . وبالنسبة إلى الأغراض العامة global يستدعى التابع الباني مرة واحدة عند بداية تنفيذ البرنامج . أما بالنسبة للأغراض المحلية local فإن التابع الباني يستدعى في كل مرة تنفذ فيها عبارة التصريح عن الغرض .

٢-٧- التابع المدمر Destructor

التابع المدمر هو تابع عضو خاص في الصف ، يحمل هذا التابع اسم الصف نفسه مسبقاً بالإشارة (~) . وإن دور هذا التابع هو عبارة عن دور متمم لدور التابع الباني . ويتم استدعائه عند تدمير الغرض . لا يقوم التابع المدمر بتدمير الغرض نفسه وإنما يقوم بإنهاء عملية الاحتفاظ بفضاء الذاكرة المخصصة للغرض قبل أن يبدأ النظام بالمطالبة به واستخدامه من أجل الاحتفاظ بأغراض جديدة (تحرير الذاكرة) . ولا يأخذ التابع المدمر أي قيمة لوسيط ولا يعيد أي قيمة ويكون لكل صف تابع مدمر وحيد فقط وهو لا يقبل عمليات التحميل الزائد عليه .

مثال : الصف student

public :

```
student ( int , char * , float ) ; // constructor
```

```
~ student ( ) { cout << " This is   Destructor Function \n " }
```

```
// Destructor
```

```
void print _stud ( ) ;
```

```
}; // End Class .
```

يستدعى التابع المدمر عند تدمير الغرض حيث تدمر الأغراض المحلية عندما تخرج من مجال الرؤية وتدمر الأغراض العامة في نهاية تنفيذ البرنامج .
يبين البرنامج في المثال (2- 5) صف يحتوي على تابع مدمر .

مثال (2 - 5)

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

```
class myclass {
```

```
private :
```

```
int a ;
```

```
public:
```

```
myclass( ); // constructor
```

```
~myclass( ); // destructor
```

```
void show( );
```

```
}; // End Class
```

```
myclass :: myclass( )
```

```
{
```

```
cout << " In constructor \n" ;
```

```
a = 10;
```

```
}
```

```
myclass : : ~myclass( ) {
```

```
    cout << " Destructing...\n";  
}  
void myclass :: show( )  
{  
    cout << a << "\n ";  
}  
  
int main( ) {  
    myclass ob;  
    ob.show( );  
    return 0;  
  
}// End main
```

Result

In constructor

10

Destructing...

Press any key to continue

ننشئ الأغراض ob,ob1,ob2 في الصف myclass في التابع main()
فنحصل على النتيجة التالية :

```
int main( )  
{  
    myclass ob,ob1,ob2;  
  
    ob.show( );  
    ob1.show( );  
    ob2.show( );  
  
    return 0;  
}  
//End main
```

Result

```
Constructor ob  
Constructor ob1  
Constructor ob2  
10  
10  
10  
Destructing... ob2  
Destructing... ob1  
Destructing... ob
```

Press any key to continue

ونلاحظ أن الباني يستدعى حسب ترتيب إنشاء الأغراض ، بينما المدمر يستدعى عكس ذلك ، أي يستدعى الباني عند إنشاء الغرض ob ثم عند إنشاء الغرض ob1 وعند إنشاء الغرض ob2، بينما يستدعى المدمر لتدمير الغرض ob ثم الغرض ob1 وأخيراً الغرض ob2 .

٢-٨- تعدد البواني (التحميل الزائد للبواني)

إن تمرير المتحولات إلى التابع الباني عملية سهلة وبسيطة وذلك بإضافة الوسيط المناسب إلى تصريح تعريف التابع الباني ، ثم تعيين قيمة الوسيط عند التصريح عن الغرض من الصف . فتعدد البواني هو إعادة استخدام اسم التابع الباني نفسه ضمن سياقات مختلفة أحدهما بدون وسيط أو بوسيط واحد أو اثنين أو ثلاثة... الخ .

لنأخذ الصف student على سبيل المثال

```
class student {
public :
    student ( ) ; // تابع باني افتراضي بدون وسيط
    student ( int k ) ; // تابع باني بوسيط واحد
    student ( int k , char * m ) ; // تابع باني بوسيطين
    student ( int k ; char * m , double d ) ; // تابع باني بثلاث
    وسطاء
} ; // End Class .
```


فإن المترجم يعرف من خلال السياق أيًا من البواني سيطبق . بالنسبة للصف student من الضروري فقط وجود بانين أحدهما افتراضي بدون وسيط والثاني يأخذ ثلاثة وسطاء (في مثالنا) ، ولا يمكن أن تحدد التوابع البانية أية أنماط أو قيم معادة .

٢-٩- الباني الافتراضي

Default Constructor

هو باني يحمل اسم الصف ، وهو بدون وسطاء . إذا لم نكتب أي بان فإن المترجم يقوم تلقائياً بتقديم بان افتراضي ، أما إذا كان هناك أي بان فإن المترجم لا يقدم أي بان افتراضي ، وعند إنشاء غرض يمكن لأعضائه أن تأخذ قيمة ابتدائية بواسطة الباني الافتراضي ، ويمكن تعديل هذه القيم لاحقاً بعد إنشاء الغرض ، مثلاً إذا عرفنا الصف student بدون بان كما يلي :

```
class student { // Class Declaration
private :
    int    student_id ;
    char    student_name[30] ;
    float   credit ;
public :
    set_student ( int , char* , double ) ;
    void print_stud ( ) ;
}; // End class
```

// class body

نظراً لأننا كتبنا الصف student بدون باني فإن المترجم يقوم بتقديم باني افتراضي تلقائياً وبدون وسيط . وبسبب وجود الباني الافتراضي يمكننا استخدام الصف للتصريح عن أغراض . أي :

student obj1 , obj2 , obj3 ;

إذا لم نكتب أي باني فإن المترجم يقوم بتزويدنا ببيان افتراضي بحيث يمكن لمستخدم الصف أن يصرح عن أغراض . أما إذا احتوى الصف على باني لا يمكن التصريح عن أغراض بدون وسطاء إلا إذا احتوى الصف على باني افتراضي بالإضافة إلى الباني العادي .

والأفضل أن نكتب باني افتراضي خاص بنا بدلاً من الاعتماد على المترجم . ويمكن أن يكون الباني الافتراضي بسيطاً بالقدر الذي نريده ، ويمكن أن يكون خالياً من التعليمات .

إن سلوك الباني الافتراضي المزود من قبل المترجم هو إسناد جميع أعضاء المعطيات إلى أصفار ويتم إسناد المؤشرات إلى null لضمان قيام الصف بالعمل الصحيح . ويجري استدعاء الباني الافتراضي عند التصريح عن غرض بدون قائمة وسطاء .

مثال (2-6)

في هذا المثال سنرى كيف يمكن استخدام المتحولات كوسائط للتابع الباني حيث يقوم هذا البرنامج بطلب قيمة من المستخدم ثم تمريرها إلى التابع الباني .

#include <iostream.h>

```

class myclass {
private :
    int  a1, b1 ;
public:
    myclass(int a , int b) ;
    void show( ) ;

} ; //End Class

myclass :: myclass( int a, int b)
{
    a1 = a ;
    b1 = b ;
}

void myclass :: show( )
{
    cout << a1 << " " << b1 << endl ;
}

int main()
{
    int x, y ;
    cout << "Enter two integers:  ";

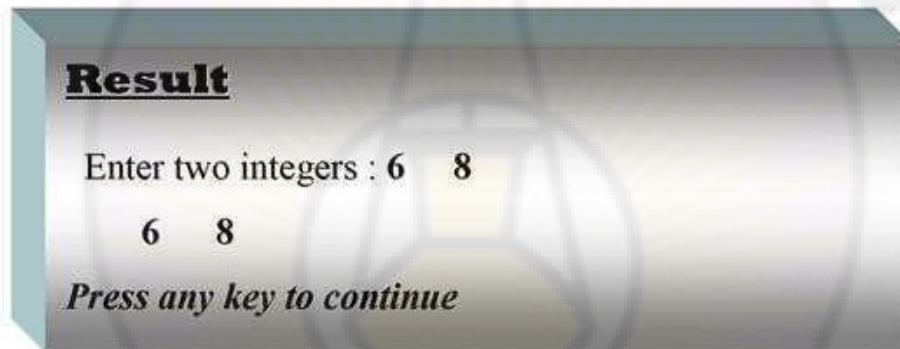
```

```

cin >> x >> y;
// use variables to construct ob
myclass ob(x, y);
ob.show ();
return 0 ;

} //End main

```



يبين هذا البرنامج نقطة هامة في الصفوف وهي إمكانية إعطاء الأغراض الشكل المطلوب عند إنشائها .

تمرين (2- 1)

أنشئ صفاً يدعى BOX بحيث يمرر إلى الباني ثلاثة قيم من النوع double يمثل كل منهما أحد جوانب الصندوق ويقوم الصف BOX بحساب حجم الصندوق ويخزن النتيجة في متغير من النوع double ، ويحتوي الصف على

بان افتراضي وبان عادي ، ثم أضف تابع طباعة كعضو يدعى (VOL ،
ويظهر حجم كل غرض من أغراض الصف .

مثال (2- 7)

اكتب برنامجاً بلغة C++ للصف name حيث يقوم الصف name بإدخال
الاسم الثلاثي first, mid, last كثلاث متحولات من النوع char وطباعتها ،
ويحتوي الصف على بان افتراضي وتابع تعديل وتابع طباعة .

البرنامج :

```
#include <iostream.h>
#include <string.h>
class name {
public:
    // default constructor
    name() {
        first[0] = mid[0] = last[0] = '\0';
        cout << " Default constructor name \n" ;
    }
    void set_name(char*,char*,char*);
    char* show( );
    ~ name(){ cout << " Destructing name \n" ; }
private:
```



```

    char first[20] ,mid[20], last[20] ;
} ; // end class

void name :: set_name( char* f,char* m,char* l)
{
    strcpy (first ,f);
    strcpy (mid ,m );
    strcpy (last ,l);
} // End function

char* name :: show( )
{
    char* temp=new char[200];
    strcpy (temp , first );
    strcat (temp ," " );
    strcat (temp ,mid );
    strcat (temp ," " );
    strcat (temp ,last );

    return temp ;
} // end function show

void main( )
{
    name ob ;
    cout<< ob.show( ) ;

```

```

cout<<" name \n " ;
ob.set_name ( " Saleem " , " Ahmed " , " Ali " ) ;
cout<< " *** "<<ob.show ( )<<" \n " ;

} // End main

```

Result

Default constructor name

name

***Saleem Ahmed Ali

Destructing name

Press any key to continue

٢-١٠ - استدعاء التوابع البانية والمدمرة

يتم استدعاء التوابع البانية والمدمرة بشكل ضمني ويتعلق ترتيب استدعاء هذه التوابع بترتيب تنفيذ عمليات دخول وخروج البرنامج لمجالات رؤية الأغراض المشتقة من الصفوف ، ويجري عادة استدعاء المدمرات حسب الترتيب العكسي لعمليات استدعاء التوابع البانية ، ويتم استدعاء التوابع البانية من أجل الأغراض المصرح عنها ضمن مجال الرؤية العام وذلك قبل تنفيذ أية عملية استدعاء آخر . ويجري استدعاء التوابع المدمرة لحظة انتهاء التابع () main من عملية استدعاء التابع exit . ويتم استدعاء التوابع البانية والمدمرة أوتوماتيكيا في كل مرة يدخل أو يخرج فيها البرنامج ضمن المجال المصرح فيه عن الأغراض المعينة

بذلك . ويجري استدعاء التوابع البانية أيضا من أجل الأغراض المحلية الساكنة static ويتم استدعاء التوابع المدمرة لها عند الخروج من التابع () main وعند استدعاء التابع exit ، ويجري تدمير الأغراض الساكنة والعامة وفقاً للترتيب العكسي لإنشائها . والمثال (2- 8) يوضح استخدام التابعين الباني والمدمر .

مثال (2- 8)

يبين هذا المثال طريقة جيدة لاستخدام التابع الباني والمدمر ، ويستخدم هذا البرنامج غرضاً من الصف timer لقياس الزمن المنقضي بين لحظة إنشاء غرض من الصف timer ولحظة تدميره ، ويقوم التابع المدمر بإظهار الزمن المنقضي على الشاشة . ونستطيع استخدام غرض لقياس فترة تشغيل برنامج ما أو الزمن الذي يقضيه تابع ما ضمن كتلة برمجية . ونؤكد من مجال الرؤية عندما نريد إيقاف التوقيت .

البرنامج

```
#include <iostream.h>
#include <time.h>
class timer {
pravste ;
clock_t start ;
public:
timer( ) ; // constructor
~timer( ) ; // destructor
```

```

}; //End class

timer :: timer( )
{
    start = clock( ) ;
}

timer :: ~timer( )
{
    clock_t end ;
    end = clock( ) ;
    cout << "Elapsed time: "
        << (end-start) / CLOCKS_PER_SEC << "\n";
}

int main( )
{
    timer ob ;
    char c ;
    // delay ...
    cout << "Press a key followed by ENTER: ";
    cin >> c ;
    return 0;
}

```

}// End main

RESULT

Press a key followed by ENTER : go

Elapsed time : 17

Press any key to continue

ويستخدم البرنامج التابع القياسي clock() من مكتبة C++ الذي يعيد عدد دورات الساعة التي مضت منذ لحظة تشغيل البرنامج ، ويتقسيم هذه القيمة على CLOCKS_PER_SEC تتحول هذه القيمة إلى ثوان .

مثال (2- 9)

اكتب برنامجاً بلغة C++ للصف student حيث يقوم الصف student بإدخال الاسم الثلاثي first, mid, last كثلاث متحولات من النوع char وطباعتها ، وإدخال ثلاث مواد دراسية course1, course2 , course3 من النوع int وحساب المعدل الوسطي للطالب ، ثم أعد كتابة البرنامج لـ 30 طالباً (على الطالب كتابة هذا البرنامج لـ 30 طالباً) . يحتوي الصف على باني عادي وتابع طباعة .

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

```
#include <string.h>
```

```
class student {
```


private:

char first[20] , mid[20], last[20] ;

int course1, course2 , course3 ;

public:

student(**char***,**char***,**char***,**int** ,**int** ,**int**) ;

void print() ;

} ; *// End class*

//constructor

student :: student(**char** *f,**char***m,**char***l,**int** c1,**int** c2,
int c3) {

strcpy(first ,f);

strcpy(mid ,m);

strcpy(last ,l);

course1 = c1 ;

course2 = c2 ;

course3 = c3 ;

} *//End constructor*

void student :: print()

{

cout<< first <<" " <<mid<<" " <<last<<" " <<endl ;

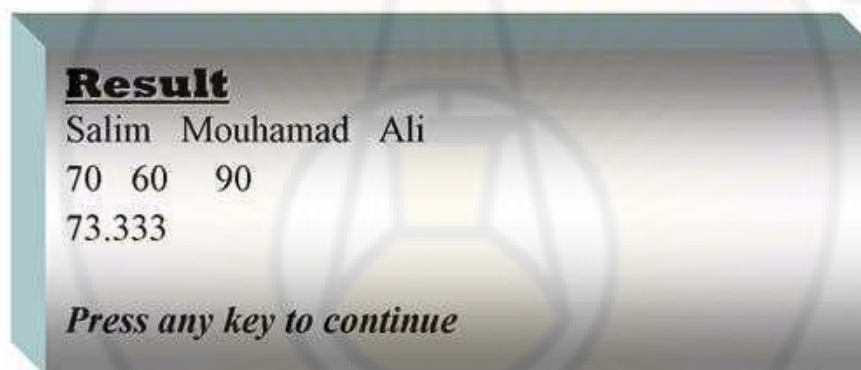
cout <<course1<<" " <<course2<<" << course3 <<endl;

doble avr_course = (course1+course2+course3) / 3 ;

```

    cout << avr_course ;
} //End function print
void main()
{
    student name1( "Salim ", "Mouhamad", "Ali", 70, 60, 90);
    s1.print( );
} //End main

```



تمرين (2- 2)

اكتب برنامجاً بلغة C++ لإنشاء صفراً يدعى card يحتفظ بإدخال بطاقة مكتبة . بحيث يخزن الصف عنوان الكتاب title واسم المؤلف author وعدد النسخ number . بحيث أن المتحولات author و title هي خاصة ومن نوع char ، أما المتحول number هو خاص ومن نوع int . يحتوي الصف على :

- ❖ تابع تمرير معطيات اسمه (store)
- ❖ وعلى تابع طباعة اسمه (show) .

أنشأ في التابع () main أغراضاً لاستدعاء التوابيع اللازمة وطباعة النتيجة .

=====

ملاحظة : حالة خاصة في التابع الباني

عند التصريح عن الغرض ob في التابع () main في الصف samp كما يلي :

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

```
class samp {
```

```
    int al;
```

```
public :
```

```
    samp( int a ) { al= a ; }
```

```
    void show( ) { cout<< al ; }
```

```
}; //end class
```

```
int main( ) {
```

```
    samp ob= 10 ;
```

```
    ob.show ( ) ;
```

```
    return 0 ;
```

```
} //End main
```

ويمكن التصريح بالشكل ; ob = 10 يحول هذا الشكل من التهيئة تلقائياً إلى استدعاء الباني samp مع القيمة 10 كوسيط . والعبارة تُعامل من قبل المترجم كما لو أنها كتبت بالشكل : ob(10) .

عموماً عند وجود بانى يأخذ وسيطاً واحداً فقط ، نستطيع استعمال أيّاً من الشكلين
لتهيئة غرض ما .

٢-١١ - بناء المكس Stack

المثال (2 - 10) التالي يظهر قوة الأغراض ، حيث ينشئ هذا البرنامج
صفاً يدعى stack ، وهو يمثل لتخزين الرموز ، وهذا الصف يحتوي على
متحولين خاصين هما stck و tos ، وتقوم المصفوفة stck بالاحتفاظ بالرموز
المدفوعة إلى المكس ويحتوي المتحول tos على دليل قمة المكس . هذا
الصف يحتوي على ثلاثة توابع عامة هي :

- التابع () init لتهيئة المكس .
- التابع () push لدفع قيمة إلى المكس .
- التابع () pop لسحب قيمة من المكس .

مثال (2 - 10)

```
# include <iostream.h>
// #define SIZE 10
const int SIZE=10;
//Declare a stack class for characters
class stack {
private :
    char stck [ SIZE ] ; // holds the stack
    int tos; // index of top-of-stack
```

public:

void init() ; *// initialize stack*

void push(char ch) ; *// push character on stack*

char pop() ; *// pop character from stack*

} ; *// End of class*

//Initialize the stack

void stack :: init()

{

tos = 0;

}

//Push a character.

void stack :: push(**char** ch)

{

if (tos == SIZE) {

cout << "Stack is full";

return;

}

stck[tos] = ch ;

tos++;

}

// Pop a character.

char stack :: pop() {


```

    if (tos == 0) {
        cout << "Stack is empty";
        return 0 ; // return null on empty stack
    }
    tos--;
    return stck [ tos ] ;
}

int main()
{
    stack s1, s2 ; // create two stacks
    // initialize the stacks
    s1.init( ) ;
    s2.init( ) ;

    s1.push('a');
    s2.push('x');

    s1.push('b');
    s2.push('y');
    s1.push('c');
    s2.push('z');

    int i ;
    for(i=0; i<3; i++) cout << "Pop s1: " << s1.pop( ) << endl;

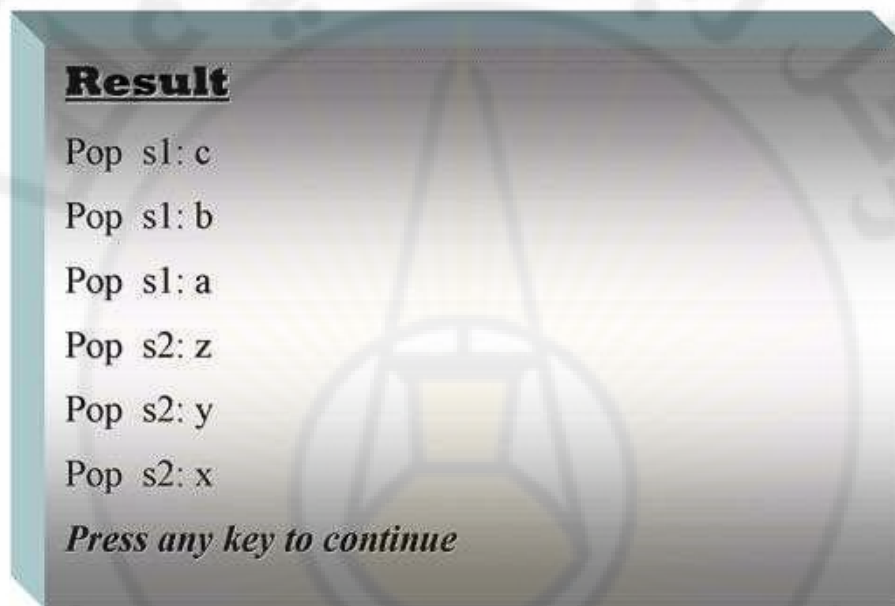
```

```

for(i=0; i<3; i++) cout << "Pop s2: " << s2.pop( ) << endl;

return 0;
} //End main

```



يقوم التابع () main بإنشاء مكديسين S1 و S2 ، ثم تدفع ثلاثة رموز إلى المكديس . إن المكديسين S1 و S2 منفصلين عن بعضهما بحيث لا تؤثر القيم المدفوعة إلى S1 على القيم المدفوعة إلى S2 ، أي أن كلاً من الغرضين يملك نسخة خاصة من المتحولين stck و tos . يُعد هذا المبدأ أساساً لفهم الأغراض ، وعلى الرغم من أن جميع أغراض هذا الصف تشترك بالتتابع الأعضاء نفسها إلا أن كل غرض ينشئ ويحتفظ بأعضاء معطياته الخاصة .

٢-١٢ - بناء الصف date

يتم استخدام عملية الإسناد (=) لإسناد غرض إلى غرض آخر من نفس النوع ، ويتم عادة تنفيذ عملية الإسناد باستخدام نسخ الأعضاء ، بحيث يتم نسخ كل عضو من أعضاء الغرض الأول ضمن نفس العضو من الغرض الثاني. يمكن تمرير الأغراض كوسطاء لتابع ويمكن أيضاً إعادتها كقيم للتوابع . ويوضح المثال (2- 11) الإسناد الافتراضي للأعضاء .

مثال (2- 11)

- اكتب برنامجاً للصف date الذي يقوم بما يلي :
- يدخل ويطبّع التاريخ (day , month , year) بالشكل " 15 may 2006 " .
 - بحيث يتم تعيين قيم ابتدائية تعتبر (01 January 2006) ، وعند الإدخال يجب أن يطبع (0) إذا كانت day أصغر من الواحد أو أكبر من (31) ، وأن يطبع (0) إذا كانت year أصغر من 2006 .
 - حيث يتم تعيين المتحولات كما يلي :
 - day يعبر عن اليوم وهو من نمط int .
 - year يعبر عن السنة وهو من نمط int
 - month يعبر عن الشهر وهو سلسلة محرفية من النمط char .
- يحتوي الصف على بائي افتراضي وتابع تعديل وتابع طباعة وتابع مدمر .

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

```
#include<string.h>
```

```

class date {
private:
    int    year , day ;
    char   month [ 20 ]
public:
    date( );
    void   setdate( int , char* , int ) ;
    void   printd( );
    ~date( ){ // delete [ ] month ;
        cout << " delete { } month "<<endl ; }
};//End class date

date :: date( ) {
    year = 2006 ;
    strcpy(month,"january" ) ;
    day = 01;
}

void date :: setdate( int y, char* m, int d )
{
    year =(y >=2006 )?y:0;
    strcpy ( month , m ) ;
    day = (d >0 && d< 32) ? d : 0 ;
}

void date :: printd( ) {

```

```

        cout<<day<<" "<<month<<" "<<year<<endl ;
    }

    void main( ) {
        date d1 ;
        d1.printd( ) ; // 01 January 2006
        cout<< endl ;
        d1.setdate(2006,"may",15);
        d1.printd( ) ;
        cout<<endl;
    } //End main

```

Result

10 January 2006

15 May 2006

delete { } month

Press any key to continue

٢-١٣ - بناء الصف Time

اكتب برنامجاً بلغة C++ للصف time يدخل ويطبع الوقت (hour , minute , second) بالشكل القياسي standard والعسكري . بحيث يتم تعيين قيم ابتدائية تعتبر (12 : 0 : 0 AM) للشكل القياسي و (0 : 0 : 0) للشكل العسكري ، وعند الإدخال يجب أن يطبع (0) إذا كانت hour و minute و second قيم غير صحيحة . يحتوي الصف على باني افتراضي وتابع تعديل معطيات وتابع طباعة للشكل القياسي وتابع طباعة آخر للشكل العسكري ، كما هو موضح في المثال (2 - 12) .

مثال (2 - 12)

```
#include<iostream.h>
class time {
private:
    int hour;
    int minute;
    int second;
public:
    time( );
    void settime(int ,int ,int );
    void printu( );
    void prints ( );
}; // End class time
```

```

Time :: time( ){ hour = minute = second = 0 ;}

void time :: settime (int h , int m , int s)
{
    Hour = ( h>=0 && h<24 ) ? h : 0 ;
    minute = ( m>=0 && m<60 ) ? m : 0 ;
    second = ( s>=0 && s<60 ) ? s : 0 ;
}

void time :: printu( )
{
    cout<<hour<<":"<<minute<<":"<<second<<endl;
}

void time :: prints( )
{
    cout<<((hour==0 || hour==12) ? 12:hour%12)
        <<":"<<minute<<":"<<second <<
        (hour<12?"AM":"PM") <<endl;
}

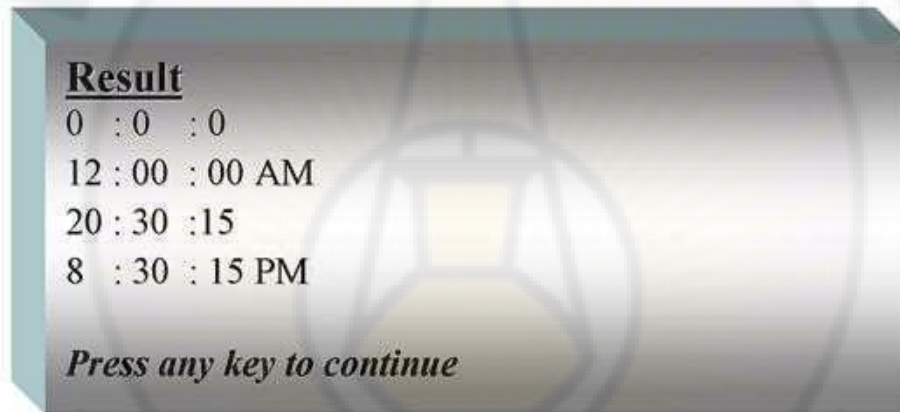
void main( )
{
    time t ;
    t.printu( ) ; // 00:00:00
    cout<<endl ;
    t.prints( ) ; //12:00:00
}

```

```

t.settime(20,30,15);
t.printu( );
cout<<endl;
    t.prints( );
cout<<endl;
} //End main

```



٢-١٤ - بناء الصف Point

اكتب برنامجاً بلغة C++ يعرف الصف point ويعرف النقطة (x, y) . ويقوم البرنامج بتعريف نقطة موجبة دوماً مهما كانت قيمة x و y . ويحتوي البرنامج على بيان افتراضي و بيان عادي لتزوير المعطيات وعلى توابيع الوصول لـ x, y (أي أن الصف لا يحتوي على تابع طباعة). ويحتوي البرنامج على تابع مدمر بحيث يطبع رسالة ما عند استدعائه، كما هو موضح في المثال (2-13).

مثال (2- 13)

```
#include <iostream.h>
class point {
private :
    int x , y ;
public :
    point ( ) { x=0 ; y=0 ;}
    point(int new_x , int new_y )
    {   if ( new_x <0 )
        new_x *= -1;
        if ( new_y <0 )
            new_y *= -1;
        x =new_x ;
        y =new_y ;
    }
    ~ point ( ) {
        cout << " destructor \n " ;
    }
    int get_x() { return x ; }
    int get_y() { return y ; }
}; // End Class
```

```

void main( )
{
    point obj1(-1,-20), obj2(10,20);
    cout << "obj1=" << obj1.get_x() <<" , " ;
    cout<< obj1.get_y() <<endl ;

    cout<<" obj2 = " << obj2.get_x() <<" , " ;
    cout<<obj2.get_y() <<endl;

} // End main

```

Result

obj1=1,20

obj2 = 10,20

destructor

Press any key to continue

مثال (2- 14)

اكتب برنامجاً بلغة C++ يعرف الصف point ويقوم البرنامج بنمذجة النقطة

(x , y) ، بحيث يكون لدينا :

$$x = x + dx$$

$$y = y + dy$$

يحتوي البرنامج على بيان افتراضي وبيان عادي لتمرير المعطيات وعلى تابع طباعة يطبع إحداثيات النقطة المنمذجة . ويحتوي البرنامج على تابع مدمر بحيث يطبع رسالة ما عند استدعائه ، كما هو موضح في البرنامج التالي .

```
#include<iostream.h>
class point {
private :
    int x , y ;
public :
    point(int new_x , int new_y ) ;
    int get_x( ) ;
    int get_y( ) ;
    void print( ) ;
    void move ( int dx ,int dy ) ;
    ~point( ) {
        cout << " Destructor " ;
    }
} ; // End Class .

point :: point ( int new_x , int new_y )
{
    x = new_x ;
    y = new_y ;
}
```

```

int point :: get_x( )
{
    return x ;
}
int point :: get_y( )
{
    return y ;
}
void point :: print( )
{
    cout <<x<<","<<y<<endl ;
}
void point :: move (int dx ,int dy )
{
    x = x + dx ;
    y = y + dy ;
}

void main( )
{
    point pt1(5,4) ;
    point pt2(1,20) ;
    point pt3(10,2) ;

```

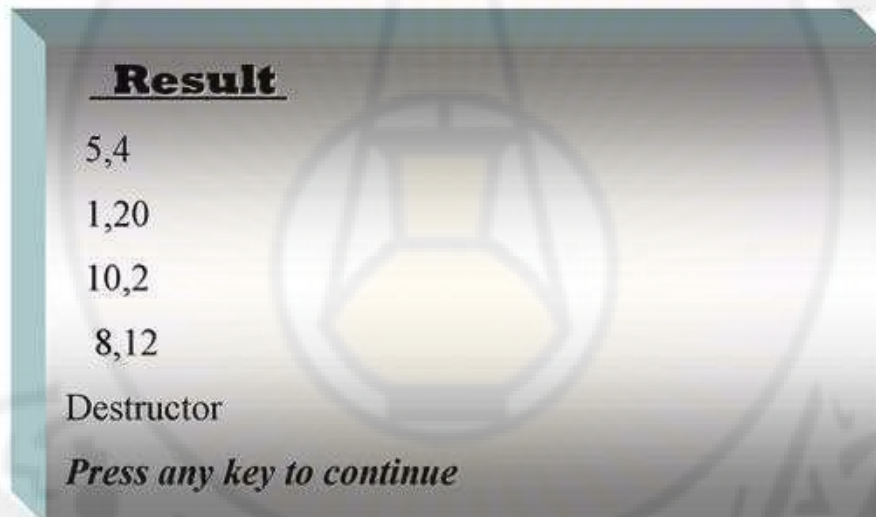
```

pt1.print( ) ;
pt2.print( ) ;
pt3.print( ) ;

pt1.move(3,8) ;
pt1.print( ) ;

} //End main

```



مثال (2- 15)

اكتب برنامجاً بلغة ++c يحتوي على صفئاً اسمه myclass يحتوي على وسطين خاصين sumEven ، sumOdd من النمط int ، ويحتوي الصف على الأعضاء العامة التالية :

- بأن افتراضي يسند القيمة صفر للمعطيات الخاصة .
- بأن عادي وسيطه مصفوفة ويقوم بحساب مجموع الأعداد الفردية والزوجية لمصفوفة من الأعداد الصحيحة مكونة من 10 عناصر، ثم يسند النتائج للمعطيات الخاصة sumEven ، sumOdd على الترتيب.
- تابع عضو لطباعة قيم المعطيات الخاصة اسمه print .
- أدخل في التابع main عشرة أعداد صحيحة و خزنها في مصفوفة x[10] ، ثم صرح عن غرض ob من الصف myclass لاستدعاء التتابع اللازمة ولحساب وطباعة النتائج .

```
#include <iostream.h>

class myclass {
    int sumEven , sumOdd ;
public:
    myclass( ){ sumEven = sumOdd=0 ; } ;
    myclass(int [ ] ); // constructor
    ~myclass( ){cout <<" destructor \n" ; } // destructor
    void print( ){
        cout <<" sumEven = "<< sumEven
            <<" sumOdd= " <<sumOdd<<endl;
    }
}; //End Class
```

```

myclass::myclass( int x[ ] ) {
    int sumEven1=0 , sumOdd1 =0;
    for(int i=0; i<10 ;i++ )
    {
        if( x[i] % 2 == 0 )
            sumEven1 = sumEven1 +x[i];
        else
            sumOdd1= sumOdd1+x[i];
    }
    sumEven =sumEven1 ;
    sumOdd = sumOdd1 ;
} //End constructor

void main( ) {
    int x[10] ;
    cout<<" Enter The Numbers "<<endl;
    for(int i=0; i<10 ;i++ )
    {
        cin>>x[i];
    }
    myclass ob( x );
    ob.print( );
} //End main

```


Result

Enter The Numbers

1 2 3 4 5 6 7 8 9 0

sumEven = 20 sumOdd= 25

destructor

Press any key to continue

٢-١٥ - الأغراض الثابتة والتوابع الثابتة

تحتاج بعض الأغراض أن تكون ذات طبيعة قابلة للتغيير ولا تحتاج لبعضها إلى ذلك ، ويمكن للمبرمج أن يستخدم التعليمات **const** لتحديد الأغراض التي لا يمكن تغييرها ، حيث تؤدي أي محاولة لفعل ذلك إلى توليد خطأ فعلي سبيل المثال :

const point pt (0 , 0) ;

في هذا التصريح عن الغرض الثابت **pt** المشتق من الصف **point** يأخذ القيمة (0 , 0) كقيمة ابتدائية للنقطة .

ويمكن للمبرمج أن يصرح عن توابع أعضاء ثابت (**constant member function**) عند ذلك لا يستطيع ذلك التابع تغيير قيمة الغرض الذي يستدعيه . ولا تستطيع الأغراض الثابتة استدعاء توابع أعضاء غير ثابتة أما التابع العضو الثابت فيمكن أن يُستدعى من قبل إما أغراض ثابتة أو غير ثابتة . لا يمكن للتوابع الأعضاء الثابتة أن تقوم بتعديل الغرض لأن المترجم يمنع القيام بذلك .

ويمكن تعريف تابع على أنه ثابت وذلك باستخدام التعليمات **const** بعد قائمة الوسائط .

ولا نحتاج إلى استخدام التعليمات **const** مع التوابع البانية والمدمرة للإغراض الثابتة . ويجب السماح للتوابع البانية بتعديل الغرض حتى يأخذ قيمة ابتدائية بشكل صحيح ، ويجب أن تكون المدمرات قادرة على إنهاء عملية الاحتفاظ بفضاء الذاكرة المخصصة للغرض قبل تدميره .

وقد نرغب أحياناً في تغيير أعضاء الصف ضمن تابع ثابت دون السماح للتابع بتغيير باقي أعضاء الثابت . ويمكن القيام بذلك من خلال التعليمات **mutable** والتي تتجاوز صفة الثبات ، أي أن العضو المحدد على أنه **mutable** يمكن أن يعدل من قبل أي تابع عضو ثابت .

مثال (2- 16)

إن الهدف من التصريح عن تابع عضو على أنه ثابت هو منعه من تعديل الغرض الذي يستدعيه ، والبرنامج التالي يوضح التوابع الثابتة .

```
// Demonstrate const member functions.
```

```
// This program won't compile.
```

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

```
class Demo {
```

```
Private :
```

```
    int i;
```

```
public:
```

```
    int geti() const {
```

```

    return i;
} // ok
void seti( int x) const
{
    i = x; // error!
}
};//End class

void main( )
{
    Demo ob;
    ob.seti(1900) ;
    cout << ob.geti( ) <<endl ;
};//End main

```

RESULT

compiling...

cons1.cpp

d:\cons1.cpp(16) : error C2166: l-value specifies

const object

Error executing cl.exe.

cons1.exe - 1 error(s), 0 warning(s)

Press any key to continue

يحاول البرنامج في المثال (2-17) التالي إنشاء مؤقت تنازلي بسيط يقرع جرساً عند انتهاء المدة الزمنية . نستطيع تحديد المدة الزمنية والزيادة عند إنشاء غرض من النوع Countdown .

مثال (2- 17)

```
// This program contains an error.
```

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

```
class Countdown {
```

```
privat :
```

```
int incr;
```

```
int target;
```

```
int current;
```

```
public:
```

```
CountDown(int delay, int i=1)
```

```
{
```

```
target = delay;
```

```
incr = i;
```

```
current = 0;
```

```
}
```

```
bool counting() const
```

```
{
```

```
current += incr;
```

```
if(current >= target) {
```

```
        cout << "\a";  
        return false;  
    }  
    cout << current << " ";  
    return true;  
}  
}; // End class  
  
int main() {  
    Countdown ob(100, 2);  
    while(ob.counting());  
    return 0;  
} // End class
```

RESULT

1900

Press any key to continue

٢-١٦ - إسناد الأغراض

Assigning Objects

يمكن إسناد غرض إلى آخر بشرط أن يكون كلا الغرضين من النوع نفسه .
افتراضياً ، عندما يسند غرض إلى آخر ، تنسخ جميع المعطيات الأعضاء على مستوى البتات . مثلاً عندما يسند الغرض ob1 إلى الغرض ob2 فإن جميع محتويات أعضاء الغرض ob1 تنسخ إلى الأعضاء المقابلة لها من الغرض ob2، والبرنامج التالي في المثال (2- 18) يوضح إسناد الأغراض.

مثال (2 - 18)

// An example of object assignment.

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

```
class point {
```

```
Private:
```

```
    int x, y;
```

```
public:
```

```
    void set( int x1, int y1) { x = x1; y = y1; }
```

```
    void show( ) { cout <<" ( "<< x <<" , "<< y <<)"<< "\n " ;
```

```
}
```

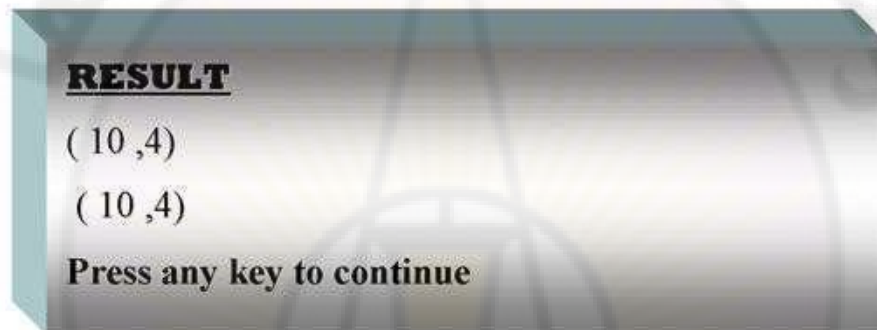
```
}; // End Class
```

```
int main( ) {
```

```
    point ob1, ob2 ;
```

```
    ob1.set(10, 4) ;
```

```
// assign ob1 to ob2
ob2 = ob1;
ob1.show( ) ;
ob2.show( ) ;
return 0 ;
} // End main
```



ويحتوي الغرض ob1 على متغيرين هما y , x أسندت لهما القيم 10 , 4 ،
ثم أسند الغرض ob1 إلى الغرض ob2 وبالتالي أسندت القيم الحالية للمتغيرين
ob1.x و ob1.y إلى المتغيرين ob2.x و ob2.y ، ويوضح الشكل السابق
النتائج عند تنفيذ البرنامج .

٢-١٧- تمرير الأغراض إلى التتابع

Passing Object To Function

يمكن تمرير الأغراض إلى التتابع بنفس أسلوب تمرير باقي أنواع المعطيات
الأخرى . علينا أن نصرح عن وسائط التابع على أنها من نوع صف، ثم استخدام
أغراض ذلك الصف كمتحولات عند استدعاء التابع ، وتمرر الأغراض إلى التتابع

بالقيمة . والبرنامج في المثال (2- 19) التالي يوضح تمرير الأغراض إلى
التتابع .

مثال (2- 19)

```
#include <iostream.h>
class samp {
private :
    int i ;
public :
    samp ( int n ) { i =n ;}
    int get_i ( ) { return i ;}
}; // End Class

// Return squar of x.i
int sqr_it ( samp x ) {
    return x.get_i() * x.get_i() ;
}

void main ( ) {
    samp a( 10) , b(20);
    cout << sqr_it(a) << endl;
    cout << sqr_it(b) << endl;
} // End main
```

Result

100

400

Press any key to continue

ينشئ هذا البرنامج صفاً اسمه samp يحتوي على متغير واحد هو (i) في حين يأخذ التابع () sqr_it من النوع samp كوسيط ويعيد مربع القيمة . يُعد التمرير بالقيمة الطريقة الافتراضية لتمرير الوسائط إلى التتابع بما فيها الأغراض ، وهذا يعني أن التابع يستخدم نسخة من المتحول وليس المتحول الأصلي . أي أن التغيرات التي تطرأ على الغرض داخل التابع لن تؤثر على الغرض الذي استدعى هذا التابع . لو أضفنا إلى البرنامج السابق السطر التالي في التابع الأساسي () main :

```
cout << a.get_i() << endl;
```

ونحصل في النتيجة على أن قيمة a تبقى كما هي 10 دون تغيير . ويمكن تمرير عنوان الغرض إلى التابع بحيث يمكن للتابع تعديل المتحول الوسيط في الاستدعاء . ويعدل البرنامج التالي في المثال (2 - 20) قيمة الغرض الذي استخدم عنوانه في استدعاء التابع () sqr_it .

مثال (2 - 20)

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

```
class samp {
```

private :

int i;

public :

samp(int n) { i= n ; }

void set_i(int n) { i= n ; }

int get_i () { return i ; }

};*// End Class*

// Set x.i to its squar .This affects calling argument.

void sqr_it(samp * x)

{

x -> set_i (x -> get_i () * x -> get_i ()) ;

cout<<"copy of a has i value of " << x -> get_i () ;

cout<< endl ;

}

void main () {

samp a(10) ;

sqr_it(&a) ;

// pass a's address to sqr_it .

cout <<" Now , a in main () has been unchanged : " ;

cout <<a.get_i () <<endl ;

// displays 100

} *// End main*

Result

copy of a has i value of 100

Now , a in main() has been unchanged : 100

Press any key to continue

إن نسخ الغرض أثناء تمريره إلى تابع ما ، يعني إنشاء غرض جديد، وكذلك عندما ينتهي التابع يدمر نسخة الغرض الممرر . لا يُستدعى التابع الباني عند عمل نسخة من غرض وتمريره إلى التابع . وسبب ذلك هو أن التابع الباني يقوم بتهيئة الغرض ، ولا يجب استدعاء هذا التابع عند عمل نسخة من الغرض الموجود أصلاً لتمريره إلى التابع لأن ذلك يؤدي إلى تغيير محتويات هذا الغرض. إن عمل نسخة من غرض نتيجة استدعاء تابع معين لا يستدعي التابع الباني ، بينما تدمير تلك النسخة يؤدي إلى استدعاء التابع المدمر ، كما يوضح البرنامج في المثال (21 - 2) .

مثال (21 - 2)

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

```
class samp {
```

```
private :
```

```
int i ;
```

```
public :
```

```
samp( int n ) {
```

```
i =n ;
```

```

cout<< " Constructing \n ";
}

~samp () { cout <<" Destructing \n ";}

int get_i () { return i ; }

}; // End Class

// Return squar of x.i

int sqr_it ( samp x )
{
    return x.get_i() * x.get_i();
}

void main ( ) {
    samp a(10);
    cout << sqr_it ( a ) << endl ;
} // End main

```

Result

Constructing

Destructing

100

Destructing

Press any key to continue

ونلاحظ من نتائج البرنامج أن التابع الباني استدعي مرة واحدة ، وذلك لإنشاء الغرض a ، أما التابع المدمر استدعي مرتين : الأولى من أجل تدمير نسخة الغرض التي مررت إلى التابع (sqr_it) ، والثانية من أجل تدمير الغرض a نفسه .

٢-١٨ - التركيب

Composition

إن أحد أهم السمات في لغة C++ هي إمكانية إعادة استخدام الشيفرة المكتوبة سابقاً ، ولكن حتى تكون عملية إعادة الاستخدام هذه في أعلى مستوياتها، يجب أن تكون قادرة على فعل أشياء أكثر من الاكتفاء بنسخ الشيفرة وإجراء بعض التعديلات عليها. والحل في لغة C++ يدور حول فكرة الصف، حيث يمكننا إعادة استخدام الشيفرة عن طريق إنشاء صفوف جديدة، ولكن بدلاً من أن نعيد بناء هذه الصفوف من الصفر بإمكاننا استخدام صفوف موجودة سابقاً، ثم بنائها واختبارها.

التركيب هو استخدام الصفوف بدون أن نعدل أو نسيء إلى الشيفرة الموجودة أصلاً، ولهذه الغاية يوجد طريقتان هما:

١- طريقة التركيب.

٢- طريقة الوراثة.

وطريقة التركيب تكمن في إنشاء غرض من صف موجود أصلاً ويكون هذا الغرض داخل الصف الجديد وحيث يقوم الصف الجديد بتركيب أغراض داخله من صفوف موجودة مسبقاً.

مثال (2- 22)

لنأخذ الصف التالي:

```
class MyClassA
{
    int x ;
    public:
    MyClassA ( ) { x = 0;}
    void set( int x1){ x = x1; }
    int read( ) { return x ; }
    int permute( ){ return x*47 ; }

};//End classA
```

وحيث إن المعطيات الأعضاء في هذا الصف هي معطيات خاصة، لذلك سيكون تضمينك لغرض من النمط MyClassA كغرض عام في صف جديد آمناً تماماً، فهو لن يستطيع أن يُعد بالأعضاء الخاصة للصف مباشرة، وسيكون لطريقة تضمينك هذا الغرض الشكل العام التالي:

```
class MyClassB
{
    int y;
    public:
```

```

MyclassA obA ;
MyclassB ( ){ y = 0 ;}
void f( int y1){ y = y1; }
int g( ){return y;}
};//End classB

```

والتابع الرئيسي main يأخذ الشكل البرمجي التالي:

```

void main( )
{
    MyclassB obB;
    obB.f(47);
    obB.obA.set(47);
    cout<< obB.obA.read( );
}
};//End classB

```

وإن الوصول إلى التوابع الأعضاء للغرض الذي تم تركيبه (يطلق عليه عادة غرض فرعي) ، ويحتاج إلى وجود غرض من الصف الجديد للوصول إلى الغرض الفرعي (والذي هو عضو من الصف الجديد) ، وإن العملية الأكثر شيوعاً هي أن تجعل الأغراض المركبة لها محدد الوصول private ، بحيث تصبح جزءاً من التحقيق المخفي (هذا يعني أننا نستطيع تغيير التحقيق دون أن يؤثر ذلك على واجهة الصف). عند ذلك سنتولى التوابع العامة في الصف الجديد تضمين واستخدام الأغراض المركبة، ولكن دون أن يفرض عليها وجود واجهات تعامل تشبه تلك الموجودة في الغرض، ويتم ذلك كما في الصف التالي:


```

class MyClassB
{
    int y;
    MyClass obA;
    public:
    MyClassB ( ){y=0;}
    void f( int y1){
        y=y1;
        obA.set(y1);
    }
    int g(){return y*obA.read( ); }
    void permute( ){ obA.permute( ); }

};//End ClassB

```

والتابع الرئيسي يأخذ الشكل البرمجي التالي:

```

void main( )
{
    MyClassB obB;
    obB.f(47);
    obB.permute( );
    cout<< obB .g( )<<endl;
} //End main

```

وفيما يلي سنقدم البرنامج بشكله الكامل:

```

#include<iostream.h>

class MyclassA
{
    int x ;
    public:
    MyclassA ( ) { x = 0;}
    void set( int x1){ x = x1; }
    int read( ) { return x ; }
    int permute( ) { return x*47 ; }
};//End classA

class MyclassB {
    int y;
    Myclass obA;
    public:
    MyclassB ( ){y=0;}
    void f(int y1){
        y=y1;
        obA.set(y1);
    }
    int g(){return y*obA.read( );}
    void permute( ){obA.permute( );}
};//End ClassB

```

```

void main( ) {
    MyclassB obB;
    obB.f(47);
    obB.permute( );
    cout<< obB .g( ) <<endl;
} //End main

```

من خلال هذا البرنامج نلاحظ أن التابع (permute) يقوم بنقل الواجهة في الصف MyclassA إلى الصف MyclassB حسب تعريفه في الصف MyclassB، بينما بقية التوابع الموجودة في الصف MyclassA تستخدم داخل الصف MyclassB وليس لها واجهات مطابقة في الصف MyclassB.

مثال (2 - 23)

- السبب يرجع إلى أن تعريف صف **date** يعبر عن التاريخ مؤلف من اليوم والشهر و السنة كمتحولات **private** ، ويحتوي على التوابع التالية:
- بأن افتراضي (لا يحتاج إلى أي وسيط) يقوم بإسناد قيم التاريخ 2013/1/ 1 إلى متحولات الصف.
 - بأن عادي يقبل ثلاثة وسطاء تمثل قيم التاريخ و يسندها إلى متحولات الصف.
 - تابع عضو يدعى **setdate** يأخذ ثلاثة وسطاء تمثل قيم التاريخ و يسندها إلى متحولات الصف.
 - عرف صف آخر **person** يعبر عن الشخص ويحتوي على اسم الشخص (من 30 حرف)، وتاريخ ميلاده من نوع الصف السابق (صف التاريخ) وكلية **public** ،

ومتحول صحيح ساكن (static) كمحول private يدعى count (يمثل عدد الأغراض التي يتم إنشاؤها من هذا الصف) ، ويحتوي على التوابع التالية:

- بان افتراضي يقوم بإسناد فراغ للاسم .

- تابع عضو يدعى printp يقوم بطباعة معلومات عن الشخص (الاسم، تاريخ الميلاد).

- تابع عضو يدعى getcount يجلب قيمة count.

استخدم تعريف الصف السابق في جسم البرنامج الرئيسي لإجراء ما يلي:

1. تعريف غرض من النوع شخص يدعى Ahmad ، وإدخال اسم الشخص في هذا الغرض وتحديد تاريخ الميلاد ب 1990/1/1.

2. طباعة جميع المعلومات المتوفرة عن هذا الغرض.

3. طباعة قيمة المتحول الستاتيكي count في الغرض على الشاشة .

4. تعريف غرض ثانٍ من النوع شخص Ali ، وإدخال اسم الشخص في هذا الغرض وتحديد تاريخ الميلاد ب 1980/2/1.

5. طباعة تاريخ ميلاد Ali على الشاشة.

6. طباعة المتحول الستاتيكي count في كل من Ahmad و Ali.

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

```
class date {
```

```
    int day,month,year;
```

```
public:
```

```
    date(int dd , int mm, int yy);
```

```
    date( );
```

```
    void printdate( ) const;
```

```
    void setdate(int dd , int mm, int yy);
```

```
}; //End class
```

```
date::date(int dd , int mm, int yy)
```

```
{  
    day = dd;  
    month = mm;  
    year = yy;  
}
```

```
date::date( ) {day=1;month=1;year=2013;}
```

```
void date::setdate(int dd , int mm, int yy)
```

```
{  
    day = dd;  
    month = mm;  
    year = yy;  
}
```

```
void date:: printdate( ) const
```

```
{  
    cout<<day<< " / " <<month<< " / " <<year<<endl;
```

```
// day=2; error because this function is const;
```

```
}
```

```
class person {
```

```
    static int count;
```


public:

```
    person();  
    char name[10];  
    date birthday;  
    int getcount( );  
    void printp( );  
    ~person( );
```

};//End class

```
int person :: count = 0;  
person::person( )  
{  
    name[0]= '\0';  
    count++;  
}  
int person::getcount( ){return count;}  
person ::~person( ){ -- count; }  
void person :: printp( )  
{  
    cout<<"Name: "<<name<<endl;  
    cout<<"birthday :";  
    birthday.printdate( );  
    cout<<endl;
```

```

}

void main( )
{
    person Ahmad;
    cin>>Ahmad.name;
    Ahmad.birthday.setdate(1,1,1990);
    Ahmad.printp( );
    cout<<"-----"<<endl;
    cout<<Ahmad.getcount( )<<endl;
    cout<<"-----"<<endl;

    person Ali;
    cin>> Ali.name;
    Ali.birthday. setdate (1,2,1980 );
    Ali.printp( );
    cout<<"-----"<<endl;
    cout<<"Ali.count : " << Ali.getcount( ) <<endl;
    cout<<"Ahmad.count:"<<Ahmad.getcount( )<<endl;
    cout<<"-----"<<endl;
} //End main

```

٢-١٩ - التوابع الصديقة والصفوف الصديقة

Friend Function & Friend Classes

التوابع الصديقة هي خاصية جديدة للغة C++ لحماية المعطيات ، حيث تعلن الصفوف صداقة تابع أو أكثر لتسمح له باستخدام المعطيات الأعضاء فيها ولا تسمح لغيره من التوابع بذلك، كما يمكن إعلان صداقة صف بأكمله لصف آخر ، وبالتالي تستطيع توابع الصف الصديق استخدام معطيات الصف الأساسي. يتم تعريف تابع صديق لصف خارج مجال رؤية الصف ، وله الحق في الوصول إلى المعطيات الخاصة للصف . ويمكن التصريح عن صف كامل على أنه صديق لصف آخر ، وللتصريح عن تابع على أنه صديق يجب إضافة التعليمة friend قبل تعريف التابع ضمن الصف ، كما يلي :

friend void function (.) ;

ويستطيع التابع الصديق الوصول إلى أعضاء صف معين دون أن يكون عضواً فيه، أما تعريف الصف على أنه صديق لصف آخر (الصف A) هو كما يلي :

friend class B ;

نقول إن الصف class B هو صديق للصف class A على سبيل المثال .

والسؤال : متى يكون الصف class B صديقاً للصف class A ؟

من أجل ذلك يجب على الصف A أن يتضمن تصريحاً على أن الصف B هو صف صديق . وإن صفة الصداقة ليست صفة متعدية ، مثلاً :
الصف B صديق للصف A والصف C صديق للصف B ، هذا لا يعني أن الصف C صديقاً للصف A .

٢-١٩-١ - التوابع الصديقة Friend Function

تستخدم التتابع الصديقة في مجال التحميل الزائد Overloading ومجال إنشاء أنواع معينة من تتابع الدخل/الخرج I/O Functions . ويعرف التابع الصديق كتابع عادي وليس كتابع عضو ، ولكن يجب وضع تصريح التابع داخل تصريح الصف الذي سيكون صديقاً له مسبقاً بالكلمة المحجوزة **friend** . والشكل العام للتصريح عن تابع صديق لصف ما كما يلي :

friend type function_name(class_name object);

حيث:

friend - الكلمة المحجوزة والتي تستخدم للدلالة على أن التابع المصرح عنه هو تابع صديق .

type - نوع التابع والذي يحدد من خلال نوع النتيجة التي يُعيدها التابع (int , float,....)، وإذا كان لا يعيد قيمة فهو من نوع void .

Function_name - اسم التابع والذي يوضع من قبل المبرمج ويُصح أن يكون هذا الاسم قريباً من محتوى التابع (مهمة التابع).

Class_name - اسم الصف الذي يتضمن التصريح عن التابع الصديق .

Object - اسم غرض من الصف الذي يعتبر التابع صديقاً له .

و يوضح البرنامج التالي في المثال (2 - 24) كيفية استخدام التابع الصديق .

مثال (2 - 24)

// A example of a friend function.

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

```
class myclass {
```

```
int n, d ;
```

```
public:
```

```

myclass (int x , int y ) { n = x ; d =y ; }

// declare a friend of myclass

friend int isfactor( myclass ob);

}; // End class

/* Here is friend function definition. It returns true
   if d is a factor of n. Notice that the keyword
   friend is not used in the definition of isfactor().
*/

int isfactor ( myclass ob )
{
    if ( ! ( ob.n % ob.d ) ) return 1 ;
    else
        return 0 ;
}

void main() {
    myclass ob1(10, 2), ob2(13, 3);
    if (isfactor(ob1) ) cout << " 2 is a factor of 10\n";
    else cout << " 2 is not a factor of 10 \n ";
    if (isfactor(ob2) ) cout << "3 is a factor of 13\n" ;
    else cout << " 3 is not a factor of 13 \n " ;

} // End main

```


Result

2 is a factor of 10

3 is not a factor of 13

Press any key to continue

بما أن التابع () isfactor معرف كتابع صديق للصف myclass فهو يستطيع الوصول إلى الأعضاء الخاصة للصف . إن التابع () isfactor ليس تابعاً عضواً للصف ، لذلك لا يمكن استدعاء التابع الصديق باستخدام المحدد (.) و (->) . إن العبارة التالية خاطئة : obl.isfactor . من الطبيعي أن يمرر إلى هذا التابع الصديق غرض أو أكثر من الصف . لأن التابع الصديق ليس تابعاً عضواً في الصف . عندما يرث صف ما صفاً آخر فإنه لا يرث التوابع الصديقة لأن التابع الصديق يمكن أن يكون صديقاً لأكثر من صف .

مثال (2- 25)

- أنشئ صفاً اسمه root يحتوي على وسيطين x1 , x2 من النوع float كأعضاء خاصة private ويحتوي على التوابع التالية:
- بأن افتراضي يسند القيمة صفر للمعطيات الخاصة.
 - بأن عادي بثلاثة وسطاء ، مهمته حساب جذور معادلة من الدرجة الثانية: $ax^2 + bx + c = 0$;
 - تابع وصول للمعطيات الخاصة اسمه get_x12 ومن النوع void.
 - تابع المدمر لا يفعل شيئاً.

- تابع عضو **print** لطباعة قيم المعطيات الخاصة.
- تابع صديق **fr** لحساب قيمة المقدار $.sd=(x1+x2) \times (x1-x2)$
- في التابع الرئيسي **main** أدخل قيمةً للثوابت **a**, **b**, **c** ومن ثم صرح عن غرض من الصف **root** و استدعي التوابع اللازمة لطباعة قيم الجذور و قيمة المقدار **.sd**

```
#include<iostream.h>
#include<math.h>
class root {
    float x1,x2;
public:
    root( ){x1=0.0;x2=0.0;}
    root( float ,float ,float );
    ~root( ){ }
    void print( ){cout<<"x1="<<x1<<" "<<"x2=" <<x2
                    <<endl ; }
    friend float fr( root ob );
};//End class

root::root( float a, float b, float c ) {
    float delta;
    delta = b*b-4*a*c;
```

```

    if( delta < 0 ){
        x1=0;
        x2=0;
    }
    else
    {
        x1=(-b-sqrt(delta))/(2*a);
        x2=(-b+sqrt(delta))/(2*a);
    }
}

float fr(root ob) {
    float sd;
    if (ob.x1==0 &&ob.x2==0)
        cout<<" no solution"<<endl;
    else
        sd=(ob.x1+ob.x2)*(ob.x1-ob.x2);
    return sd;
}

void main( ) {
    float a,b,c;
    cout<<"Enter the values of a,b,c :" <<endl;

```

```

cin>>a>>b>>c;

root ob(a,b,c);

ob.print( );

cout<<"sd="<<fr(ob)<<endl;

} //End main

```

مثال (2- 26)

اكتب برنامجاً بلغة C++ يحتوي على صفين myclass1 و myclass2. يحتوي الصف myclass1 على متحول خاص Average1 من نوع double وعلى التوابع الأعضاء التالية :

- تابع بان افتراضي يسند القيمة صفر للمتحول Average1 .
- تابع مدمر يطبع رسالة ما عند استدعائه .
- تابع تعديل معطيات اسمه setAverage1 يحتوي على وسيطين الأول مؤشر إلى مصفوفة والثاني يمثل بعد المصفوفة ، ويقوم بحساب المتوسط لعشرة أعداد حقيقية .

□ يحتوي الصف myclass2 على متحول خاص Average2 من نوع double وعلى التوابع الأعضاء التالية :

- تابع بان افتراضي يسند القيمة صفر للمتحول Average2 .
- تابع مدمر يطبع رسالة ما عند استدعائه .
- تابع تعديل معطيات اسمه setAverage2 يحتوي على وسيطين الأول مؤشر إلى مصفوفة والثاني يمثل بعد المصفوفة ، ويقوم بحساب المتوسط

لعشرة أعداد حقيقية .

□ تابع صديق للصف myclass1 و للصف myclass2 من النوع void اسمه fCompare يقوم بمقارنة المتوسطين الحسابيين الناتجين من الصف myclass2 و myclass1 .

□ صرح في التابع main عن مصفوفتين a[10] و b[10] من النوع double ، ثم أدخل عناصر المصفوفتين ، ثم صرح عن أغراض في الصفين لاستدعاء التوابع اللازمة لحساب وطباعة المتوسطين ومقارنتهما من خلال التابع الصديق .

```
#include <iostream.h>
class myclass2 ;
class myclass1 {
    double Average1 ;
public:
    myclass1() { Average1=0 ; cout<< " constructor1 \n " ; }
    void setAverage1(double *x, int n) ; // constructor
    friend void fCompare(myclass1 ob1, myclass2 ob2) ;
    ~myclass1() { cout << " destructor1 \n " ; } // destructor
}; //End Class1

class myclass2 {
    double Average2 ;
public:
    myclass2() { Average2=0 ; cout<< " constructor2 \n " ; }
```



```

void setAverage2(double *y, int n );
friend void fCompare(myclass1 ob1, myclass2 ob2 );
~myclass2( ){cout <<" destructor2 \n" ; } // destructor
};//End Class2

void myclass1:: setAverage1(double *x, int n ) {
    double sum=0;
    for(int i=0; i<n ;i++ )
        sum=sum+x[i];
        Average1 = sum/n ;
    }//End setAverage1

void myclass2:: setAverage2(double *y, int n ) {
    double sum=0;
    for( int i=0; i<n ;i++ )
        sum=sum+y[i];
        Average2 = sum/n ;
    }//End setAverage1

void fCompare( myclass1 ob1, myclass2 ob2 ) {
    if( ob1.Average1 > ob2.Average2 )
        cout<<" The First Average "<< ob1.Average1
        <<" Is Graters thanthe second " <<ob2.Average2
        <<endl;

```

```

else
    cout<<" The First Average "<< ob1.Average1
        <<" Is Less than the second "<<ob2.Average2 <<endl;
}

void main( )
{
    double Array1[5]={1,3,5,7.9 } ;
    double Array2[5]={2,4,6,8.10 } ;
    myclass1 ob1 ;
    myclass2 ob2 ;
    ob1.setAverage1(Array1,5);
    ob2.setAverage2(Array2,5);
    fCompare( ob1 , ob2 ) ;
} //End main

```

Result

The First Average 3.38 Is Less than the second 4.02

Press any key to continue

تمرین (3-2)

أعد المثال السابق ولكن بفرض أن التابع الصديق fCompare هو تابع عضو من الصف myclass1 وبنفس الوقت هو تابع صديق للصف myclass2 .

مثال (2- 27)

- اكتب برنامجاً بلغة ++C يحتوي على صفتين Age و Time حيث:
 - الصف Age يحتوي على : السنة year ، الشهر month ، اليوم day ، كـأعضاء خاصة من النوع int وعلى التوابع التالية:
 - بأن عادي يقوم بإسناد قيم للمعطيات الخاصة.
 - تابع صديق display للصف Age ومن النوع void ، مهمة التابع هي طباعة المعطيات الخاصة لكل من الصفتين.
 - الصف Time يحتوي على : الساعة h ، الدقيقة m ، الثانية s ، كـأعضاء خاصة من النوع int وعلى التوابع التالية:
 - بأن عادي يقوم بإسناد قيم للمعطيات الخاصة.
 - تابع صديق display للصف Time ومن النوع void ، مهمة التابع هي طباعة المعطيات الخاصة لكلا الصفتين .
- في التابع الرئيسي main صرح عن غرضين من النوع Age و Time على التوالي من أجل إدخال المعطيات الخاصة بالصفتين ومن ثم طباعة هذه المعطيات باستخدام التابع الصديق.

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

```
class Time;
```

```

class Age {
    int month, day, year;
public:
    Age( int m,int d,int y){
        month = m;
        day = d;year = y;
    }
    friend void display( Age &, Time & );
};//End class Age

class Time {
    int h,m,s;
public:
    Time( int h1,int m1,int s1)
    {
        h=h1; m=m1; s=s1;
    }
    friend void display(Age &,Time &);
}; // End class Time

void display(Age &a1,Time &t1)
{
    cout<<"\n"<<a1.month<<"/"<<a1.day<<"/"<<a1.year;

```

```

cout<<"\n"<<t1.h<<":"<<t1.m<<":"<<t1.s<<endl;
}

void main() {
    Age ag1(5,10,30);
    Time tm1(12,45,2);
    display(ag1,tm1);
}

```

٢٠-٢ - مصفوفات الأغراض Arrays of Objects

يصرح عن مصفوفات الأغراض كما يصرح عن مصفوفة متحولات من أي نوع آخر ، ويتم الوصول إليها بطريقة الوصول نفسها إلى الأنواع الأخرى . وينشئ البرنامج (2 - 28) التالي مصفوفة ذات أربعة عناصر تحوي أغراضاً من النوع samp ، ثم يجعل المتحول a لكل عنصر يأخذ قيمة من 0 إلى 3 .

مثال (2 - 28)

```

#include <iostream . h >

class samp {
private :
    int a;
public:
    void set_a ( int n) { a = n ; }
    int get_a() { return a ; }
}; // End class

```



```

void main( )
{
    samp ob[4] ;
    int i;
    for(i=0; i<4; i++)
        ob[ i ].set_a( i ) ;
    for(i=0; i<4; i++)
        cout << ob[ i ].get_a( );
    cout << "\n";
} // End main

```

Result

0 1 2 3

Press any key to continue

ويمكن إنشاء مصفوفة من الأغراض إذا كان الصف يحتوي على تابع بانى . و
 يبين المثال (2- 29) كيفية تهيئة المصفوفة ob ، ويظهر هذا البرنامج القيم (-4 , -3 , -2 , -1) على الشاشة والتي مررت إلى التابع البانى للغرض ob (ويمكن التعامل مع مصفوفة أغراض ثنائية البعد) .

مثال (2- 29)

```
#include <iostream.h>
class samp {
private :
    int a;
Public :
    samp ( int n ) { a = n ; }
    int get_a() { return a ; }
}; // End Class

int main() {
    samp ob[4] = { -1, -2, -3, -4 } ;
    int i ;
    for(i=0; i<4; i++)
        cout << ob[ i ].get_a() << ' ' ;
        cout << "\n " ;
    return 0;
} // End main
```

Result

-1 -2 - 3 -4

Press any key to continue

٢-٢١ - استخدام مؤشرات الأغراض

Using Pointer To Objects

كما شرحنا سابقاً يمكن الوصول إلى الأغراض باستخدام المؤشرات (استخدام السهم) (->) بدلاً من النقطة (.) . وبالطريقة نفسها يمكن الوصول إلى الأغراض حيث أن زيادة قيمة المؤشر تجعله يشير إلى الغرض التالي ، وإنقاص قيمة المؤشر تجعله يشير إلى الغرض السابق من النوع نفسه. ويبين المثال (2 - 30) استخدام مؤشرات الأغراض .

مثال (2 - 30)

```
#include <iostream . h >
class point {
private :
    int a , b ;
public :
    point ( int n, int m ) {
        a = n ; b = m ;
    }
    int get_a () { return a ; }
```

```
int get_b () { return b ; }

}; //End class

void main()
{
    point ob[4] = { point (1, 2), point (3, 4), point (5, 6),
    point (7, 8) } ; // End Arrays of Objects
    int i ;
    point *p ;
    p = ob ; // get starting address of array
    for ( i = 0 ; i < 4 ; i++) {
        cout << p->get_a() << ' ' ;
        cout << p->get_b() << "\n" ;
        p++ ; // advance to next object
    } //End for
    cout << endl ;

} // End main
```

Result

```
1 2
3 4
5 6
7 8
```

Press any key to continue

٢-٢٢- المؤشر this

تحتوي لغة C++ على مؤشر خاص يدعى `this [ *this ]` . ويمرر هذا المؤشر إلى أي تابع عضو تلقائياً عند استدعائه ويشير إلى الغرض الذي قام بالاستدعاء . مثلاً العبارة التالية :

`ob.func ( ) ;`

يمرر إلى التابع `func ( )` تلقائياً مؤشر ويشير إلى الغرض `ob` الذي قام بالاستدعاء ، ويدعى هذا المؤشر `this` . وتجدر الإشارة إلى أن المؤشر `this` يمرر فقط إلى التوابع الأعضاء . أما التوابع الأخرى مثل التوابع الصديقة فليس لها مؤشر `this` . ويستخدم المؤشر `this` ضمناً للرجوع إلى المعطيات الأعضاء وإلى التوابع الأعضاء .
يُعد المؤشر `this` ضمناً كافة الأعضاء ضمن الغرض على أنها عناصر تابعة له . الاستخدام الظاهري للمؤشر `this` لتمكين التابع العضو المتعلق بالصف من طباعة قيمة المتحول العضو (`x`) التابع لأحد الأغراض :

`( * this ) . x`

واستخدمنا هنا المؤشر `this` للوصول إلى المتحولات الأعضاء بشكل صريح . في المثال (2 - 31) يقوم الصف `Test` بطباعة `x` بواسطة التابع العضو الثابت `print ( )` ، ثم يستخدم طريقتين للوصول إلى `x` :

١. من خلال المؤشر `this` مع العملية سهم (`->`) .
٢. من خلال العنوان المعطى بواسطة `this` مع استخدام العملية (`.`) .

مثال (2 - 31)

```
#include <iostream.h>

class Test {
public:
    Test( int =0 ); // default constructor
    void print( ) const;
private:
    int x;
}; // End class Test

// constructor
Test :: Test( int value )
{
    x= value;
} // End constructor Test

void Test :: print( ) const
{
```

```
cout << "    x = " << x;
cout << "\n this->x = " << this->x;
cout << "\n(*this).x = " << ( *this ).x << endl;

} // End function print

void main( )
{
    Test op( 200 );
    op .print( );

} // End main
```

Result

x = 200

this -> x = 200

(*this) . x = 200

Press any key to continue

مثال (2 - 32)

الصف inventory يقوم بعملية جرد للمواد ، ويحتوي على متحولات خاصة وهي اسم المادة item و ثمنها cost والموجودات on_hand ، ويحتوي على بان عادي وتابع طباعة show .

```
#include <iostream.h>
#include <cstring.h>
class inventory {
private :
    char item[20] ; double cost ; int on_hand;
public:
    inventory( char *a , double c, int b ) {
        strcpy(item,a);
        cost = c;
        on_hand = b ;
    }
    void show( ) ;
}; // End class

void inventory :: show()
{
    cout << item;
    cout << " : $" << cost;
```

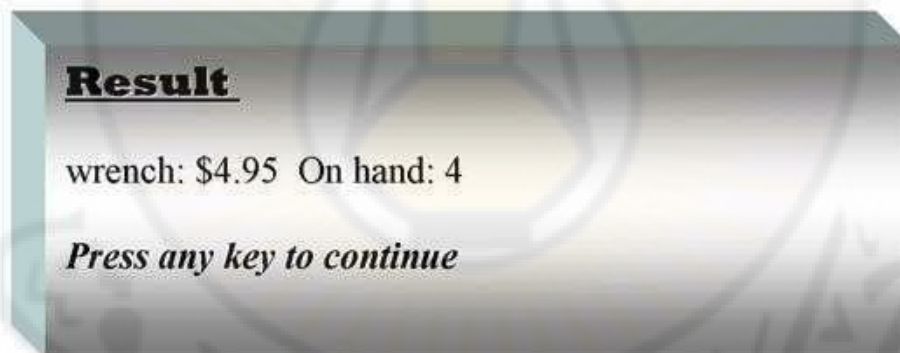
```

    cout << " On hand: " << on_hand << "\n";
}

int main( )
{
    inventory ob("wrench", 4.95, 4);
    ob.show( );
    return 0;

} // End main

```



نلاحظ أن المتحولات الأعضاء `item` ، `cost` ، `on_hand` يشار إليها مباشرة داخل التابع الباني والدالة `show( )` ، وذلك لأن النوابع الأعضاء تُستدعى فقط في حالة الاتصال مع الغرض ، لذلك يستطيع المترجم معرفة الغرض المشار إليه . عند استدعاء تابع عضو يمرر له المؤشر `this` تلقائياً والذي يشير إلى الغرض الذي قام بالاستدعاء . لذلك يمكن إعادة كتابة البرنامج السابق باستخدام المؤشر `this` كما يلي :

```

#include <iostream.h>
#include <cstring.h>
class inventory {
private :
    char item[20] ; double cost ; int on_hand ;
public :
    inventory(char *a , double c, int b)
    {
        strcpy(this -> item, a) ; // access members
        this->cost = c ; // through the this
        this->on_hand = b ; // pointer
    }
    void show( ) ;

}; // End class

void inventory :: show( )
{
    cout << this->item; // use this to access members
    cout << " : $" << this->cost;
    cout << " On hand: " << this->on_hand << "\n";
}

int main( )

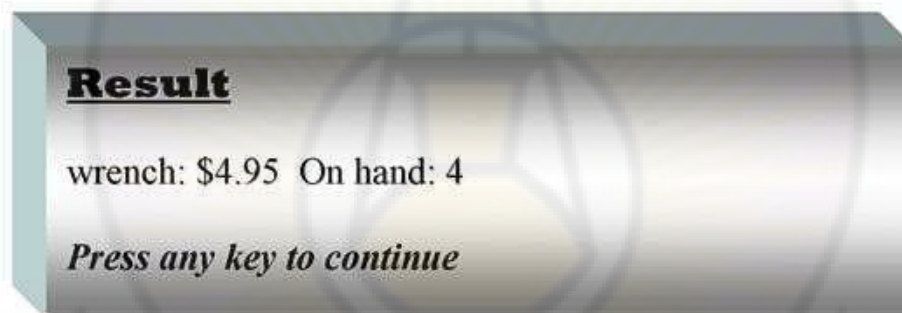
```



```

{
    inventory ob("wrench", 4.95, 4);
    ob.show();
    return 0;
} // End main

```



استخدمنا هنا المؤشر `this` للوصول إلى المتحولات الأعضاء بشكل صريح . إن العبارتين التاليتين داخل التابع `show()` متكافئتين :

`cost = 123.23`

`This -> cost = 123.23`

في الحقيقة تُعد العبارة الأولى الشكل المختصر للعبارة الثانية ، لا يستخدم المبرمجون المؤشر `this` لأن الشكل الأول أسهل .

مثال (2 - 33)

في المثال التالي حول جميع عناوين أعضاء الصف إلى عناوين صريحة بالمؤشر . this

```
#include <iostream.h>

class myclass {
    privat :
        int x, y ;
    public:
        myclass( int a , int b ) ;
        int add ( ) { return x + y ; }
        void show( ) ;
} ;// End class

void myclass :: show ( )
{
    int t ;
    t = add ( ) ; // Call member function
    cout << t << "\n" ;
}

void main()
{
    myclass ob( 10 , 14 ) ;
```

```
Ob.show ( ) ;  
} // End main
```

٢-٢٣ - استخدام العمليتين new و delete

تساعد العملية new على حجز الديناميكي للذاكرة ، ولا حاجة لاستعمال sizeof لحساب عدد البايتات المطلوبة . تعيد new تلقائياً مؤشراً من النوع المحدد ، وتساعد العملية delete على التحرير الديناميكي للذاكرة ، ويمكن تحميل new و delete تحميلاً زائداً ، ولها إمكانية التهيئة الديناميكية للغرض . تقوم العملية new بإنشاء غرض ذي حجم مناسب ، وتستدعي التابع الباني الافتراضي الخاص بالغرض ، وتعيد مؤشراً له نمط الغرض نفسه . مثلاً :

```
int * p ;  
p = new int
```

تُستخدم new للتعامل مع أي نمط char , double , int أو مع أي صف . ولتحرير الفراغ المحجوز للغرض يمكن استخدام العملية delete ، مثلاً :

```
delete p ;
```

ونقوم التعليمة باستدعاء التابع المدمر للغرض المؤشر عليه بواسطة p ثم تحرير الذاكرة المرتبطة به ، فمثلاً :

```
int * p = new int ;  
* p = 5 ;  
* p = * p + 3 ;  
cout << * p ;
```

delete p ;

ويحجز البرنامج في المثال (2 - 34) التالي غرضاً ديناميكياً .

مثال (2 - 34)

```
// Allocating dynamic objects.
#include <iostream.h>
class samp {
private :
    int i, j ;
public:
    void set_ij (int a, int b) { i=a; j=b; }
    int get_product( ) { return i*j ; }
}; //End class

int main( )
{
    samp *p;
    p = new samp ; // allocate object
    if( !p ) {
        cout << "Allocation error\n";
        return 1;
    }
    p->set_ij(4, 5);
```

```
cout << "Product is: " << p->get_product( ) << "\n";
return 0;

} // End main
```

RESULT

Product is: 20

Press any key to continue

وتسمح لغة C++ باستخدام قيم ابتدائية للأغراض المطلوب إنشائها ، فمثلاً :

```
double * p = new double ( 3,14 ) ;
```

أو قيم ابتدائية لصف :

```
time * p = new time ( 12 , 0 , 0 ) ;
```

وهو إعطاء قيمة ابتدائية للوقت وهي 12 ظهراً للغرض المشتق من الصف time وإسنادها بعد ذلك للمؤشر p . ويمكن استخدام new لحجز مصفوفة ديناميكياً .

مثلاً حجز مصفوفة من 10 عناصر من النمط int وإسنادها للمؤشر parray

كما يلي :

```
int * parray = new int [ 10 ] ;
```

وتسند للمؤشر عنوان أول عنصر من عناصر المصفوفة . ولتحرير المصفوفة

نستخدم التعليمة :

```
delete [ ] parray ;
```


إذا كان المؤشر في التعليم السابقة يُوَشر على مصفوفة من الأغراض فإنها تقوم باستدعاء التابع المدمر لهذه الأغراض أولاً ثم تقوم بتحرير الذاكرة . إذا لم تتضمن التعليم السابقة القوسين [] وكان المؤشر ptarray يدل على مصفوفة من الأغراض فإنه سيجري تدمير العنصر الأول فقط بعد استدعاء التابع المدمر . ويوضح المثال (2- 35) تمرير قيمة ابتدائية لغرض محجوز ديناميكياً .

مثال (2- 35)

يمرر البرنامج التالي قيمة ابتدائية إلى غرض محجوز ديناميكياً ، وعندما تُحجز الذاكرة للغرض samp يستدعى التابع الباني تلقائياً ويمرر إليها القيم 5,6 .
// Allocating dynamic objects.

```
#include <iostream.h>
class samp {
Private :
    int i,j;
public:
    samp( int a,int b) { i=a; j=b; }
    int get_product( ) { return i*j; }
};//End main

int main( )
{
    samp *p;
```

```

p = new samp(6, 5); // allocate object with initialization
if( !p ) {
    cout << "Allocation error\n";
    return 1;
}
cout << "Product is: " << p->get_product( ) << "\n";
delete p;
return 0;
} // End main

```

RESULT

Product is: 30

Press any key to continue

٢-٢٤- أعضاء الصفوف الساكنة

يمكن التصريح عن المتحولات الأعضاء على أنها ساكنة static . ويمكن تجنب الكثير من المشاكل الخفية باستخدام الأعضاء الساكنة . عندما نعرف متحول عضو أنه static فإن ذلك يسبب وجود نسخة واحدة من المتحول مهما

كان عدد الأغراض المنشكلة ، أي أن جميع الأغراض تتشارك مع بعضها بمتحول عضو وحيد .

وعند تعريف متحول عضو غير ساكن تتشكل نسخة جديدة من المتحول من أجل كل غرض ، أي كل غرض يمتلك نسخته الخاصة من المتحولات غير الساكنة ، أما المتحول الساكن static فيوجد منه نسخة واحدة تتشارك فيها كل الأغراض من هذا الصف . كما يستخدم نفس المتحول الساكن من قبل جميع الصفوف المشتقة من الصف الذي يحوي العضو الساكن .

وعندما نصرح عن معطيات ساكنة static داخل الصف فإننا لا نعرفها بل نعرفها في مكان آخر خارج الصف ، ويتم ذلك بإعادة التصريح عن المتحول الساكن static باستخدام عامل المجال (::) لتحديد الصف الذي ينتمي إليه . وتأخذ جميع المتحولات الساكنة static القيمة الافتراضية 0 ، لكن نستطيع إعطاء أي قيمة ابتدائية أخرى .

يمكن التصريح عن التوابع الأعضاء أنها ساكنة static ، ولكن هذا الاستخدام ليس شائعاً . تستطيع التوابع الساكنة الوصول إلى المتحولات الساكنة في الصف الذي يحوي التابع فقط ، ولا تملك الأعضاء الساكنة المؤشر this. ليس من المسموح أن تكون التوابع الافتراضية ساكنة ، ولا يمكن تعريف التوابع الساكنة أنها ثابتة const . ويمكن أن تستدعي التوابع الساكنة من خلال غرض من صفها أو بشكل مستقل عن أي غرض .

مثال (2 - 36)

هذا المثال يستخدم متحول عضو من النوع الساكن .

// A static member variable example.

```

#include <iostream.h>

class myclass {
    static int x;
public:
    void setx( int n) { x = n; }
    int getx( ) { return x; }
}; //End Class

// Definition of myclass::x. x is still private to myclass.
int myclass : : x;

int main( ) {
    myclass ob1, ob2;
    ob1.setx(10);
    cout << "ob1.x: " << ob1.getx( ) << '\n'; // displays 10
    cout << "ob2.x: " << ob2.getx( ) << '\n'; // also displays 10
    return 0;
} // End main

```

RESULT

ob1.x: 10

ob2.x: 10

Press any key to continue

نجد أن الغرض ob1 فقط يحدده قيمة المتحول الساكن x ، بما أن x مشترك بين الغرضين ob1 و ob2 وبالتالي مشترك بين جميع أغراض الصف myclass فإن كلا الاستدعاءين للتابع () getx(سيعطي النتيجة نفسها . بما أن أعضاء المعطيات الساكنة تعني ضمناً وجود ذاكرة محجوزة لذلك العضو فإنها تتطلب وجود تعريف منفصل للعضو لحجز مكان التخزين . بما أن المتحول العضو الساكن يكون موجوداً قبل أي غرض من صف ذلك المتحول فمن الممكن الوصول إليه من داخل البرنامج بشكل مستقل عن أي غرض .

لقد أجرينا على المثال (2- 37) السابق بحيث أسندت القيمة 100 إلى المتحول x دون الرجوع إلى أي غرض محدد .

مثال (2- 37)

هذا المثال يستخدم متحول عضو من النوع الساكن .

// Use a static member variable independent of any object..

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

```
class myclass {
```

```
static int x;
```

```
public:
```

```
void setx( int n) { x = n; }
```

```
int getx( ) { return x; }
```

```
};//End Class
```

// Definition of myclass::x. x is still private to myclass.

```
int myclass : : x;
```



```

void main()
{
    myclass ob1, ob2;
    // set x directly
    myclass::x = 100; // no object is referenced.
    cout << "ob1.x: " << ob1.getx() << "\n"; // displays 100
    cout << "ob2.x: " << ob2.getx() << "\n"; // also displays 100
} // End main

```

RESULT

ob1.x: 100

ob2.x: 100

Press any key to continue

٢-٢٥- توابع الأعضاء الساكنة

تملك توابع الأعضاء الساكنة تطبيقات محددة ، لكن من أحد الاستخدامات المفيدة لها التهيئة الأولية للمعطيات الخاصة الساكنة قبل تشكيل أي غرض .
والبرنامج التالي في المثال (2- 38) يوضح توابع الأعضاء الساكنة . وفي هذا البرنامج تتم تهيئة المتحول x باستدعاء التابع (init) قبل إنشاء أي غرض من النوع static_funs_demo .

مثال (2- 38)

```
#include <iostream.h>
class static_func_demo {
privat :
    static int x;
public:
    static void init( int x1) {x = x1;}
    void show( ) {cout << x<< endl;}
}; // End class
int static_func_demo::x; // define x
int main( ) {
    // init static data before object creation
    static_func_demo::init(100);
    static_func_demo x;
    x.show( ); // displays 100
    return 0;
} // End main
```

RESULT

100

Press any key to continue

مثال (2- 39)

اكتب برنامجاً بلغة C++ يقوم بحساب الوقت في أي مدينة ، وإذا أدخلنا الوقت وعدد خطوط العرض بالنسبة لغرينتش . يحتوي الصف على ثلاثة متحولات من نوع int (hour ، minute ، second) ، وعلى متحول من نوع double يدل على عدد خطوط العرض place ، وعلى متحول من نوع char يدل على اسم المدينة المراد معرفة الوقت فيها City . ويحتوي الصف على بانٍ افتراضي وعلى تابع تعديل معطيات اسمه settime، وعلى تابع طباعة اسمه showtime() وعلى تابع مدمر يطبع رسالة ما عند استدعائه .

```
#include <iostream.h>
# include <string.h >
class time {
private :
    char City[200];
    int hour ,minute ,second ;
    double place;
public:
    void settime (char*,int,int,int,double);
    void showtime( );
};// End Class

void time :: settime (char *c ,int h, int m ,int s, double p)
```

```

{
    strcpy( City, c);
    hour=h;
    minute=m;
    second=s;
    place=p;
}

void time::showtime( )
{
    cout <<City<<"(\t"<<hour<<" \t"<<minute
        <<"\t"<<second<<"): "<<endl;
}

void main( )
{
    char city[20];
    cout <<"enter the city you want to know the time in it \n";
    cin.getline (city,sizeof (city),'\n');
    int h,m,s;
    cout <<"enter the hours then the minutes then the seconds in
        GMT \n";
    cin >>h>>m>>s;
    int Line;

```

```

cout <<"enter the time line that the city is in it \n";
cin >>Line;
if ((Line>15) || (Line<-15)) {
    Line =Line /15;
}
h=h+Line;
time st1 ;
st1. settime (city,h,m,s,Line);
st1.showtime( );
} // End main

```

Result

```

enter the city you want to know the time in it
enter the city you want to know the time in it
Moscow
enter the hours then the minutes then the seconds in
GMT
4 30 50
enter the time line that the city is in it
33
Moscow ( 6 30 50):
Press any key to continue

```


٢٦-٢ - المراجع References

تعد المراجع من الميزات الهامة في لغة C++ ، فالمرجع هو مؤشر ضمنى يشير إلى عنوان متحول ، ويتصرف مثل المتحولات الأخرى . إن تمرير الوسائط للتوابع يُعد الأكثر أهمية للمراجع ، ولتعريف متحول أو وسيط مرجعي نضيف الرمز & أمام المتحول .

وعندما نستخدم وسيط مرجعي يمرر المترجم تلقائياً عنوان المتحول المستخدم كمتحول وسيطي، ولا حاجة إلى توليد عنوان المتحول يدوياً باستخدام العامل (&)، كما أن المترجم يستخدم تلقائياً المتحول المشار إليه بالوسيط المرجعي داخل التابع ولا حاجة لتطبيق العامل (*) الذي يشير على المؤشر . تتمتع المراجع بمزايا عديدة نسبة إلى المؤشرات :

❖ يمرر عنوان المتحول تلقائياً إلى التابع الذي يستخدم وسيط مرجعي ولا حاجة إلى تمرير العنوان صراحة .

❖ يرى الكثير من المبرمجين أن الوسائط المرجعية توفر واجهة أكثر أناقة من آلية المؤشرات .

❖ عدم إنشاء نسخة عند تمرير غرض بالمرجع إلى تابع وبالتالي نتجنب المشاكل المرافقة لإنشاء وتدمير هذه النسخة .

ويمكن استخدام المراجع بثلاثة طرق : تمرير المراجع إلى التوابع والأغراض ، إعادة المراجع ، والمراجع المستقلة .

٢-٢٦-١- تمرير المراجع إلى التتابع والأغراض

يوجد نوعين من تمرير المراجع :

أ- تمرير المراجع إلى التتابع: يبين المثال (2- 40) تمرير وسيط للتتابع sumref بالمرجع وتمرير وسيط للتتابع sum بالقيمة .

مثال (2- 40)

```
#include <iostream.h>
int sum( int ) ;
int sumref( int& ) ;
void main ( )
{
    int a=5 , b= 6 ;
    cout << " sum=" << sum(a) << endl;
    cout << "a=" << a << endl;    // a not change
    cout << " sumref=" << sumref(b) << endl;
    cout << "b=" << b << endl;    // b not change
} // End main

int sum ( int x)
{
    return x=x+x ;
}
```

```
int sumref ( int &y )
{
    return y=y+y;
}
```

Result

sum=10

a=5

sumref=12

b=12

Press any key to continue

ب- تمرير المراجع إلى الأغراض

عند تمرير الغرض بالمرجع لا تصنع نسخة من الغرض وإنما يتم التعامل مع النسخة الأصلية للغرض وبالتالي يستدعى التابع المدمر مرة واحدة فقط بينما إذا تم تمرير الغرض بالقيمة يتم استدعاء التابع المدمر مرتين ، مرة عند تدمير نسخة الغرض والثانية عند تدمير الغرض نفسه عند نهاية البرنامج . يبين المثال (2- 41) تمرير غرض من الصف myclass بالمرجع إلى التابع function .

مثال (2- 41)

```
#include <iostream.h>
class myclass {
    int who;
```

public:

```
myclass(int n)
{
    who = n;
    cout << "Constructing " << who << "\n";
}
~myclass( ) { cout << "Destructing " << who << "\n"; }
int id( ) { return who; }
}; //end class
```

// Now, o is passed by reference.

```
void function(myclass &ob)
{
    // note that . operator is still used!!!
    cout << "Received " << ob.id( ) << "\n";
}
```

int main()

```
{
    myclass xob(10);
    function (xob);
    return 0;
} //End main
```

Result

Constructing 1

Received 1

Destructing 1

Press any key to continue

Retiring References ٢-٢٦-٢ - إعادة المراجع

تستطيع التتابع إعادة المراجع ، وهذا مفيد جداً كما سنرى في الفصل الثالث. كما يمكن أن تستخدم للسماح باستخدام التتابع في الوسيط الأيسر من عبارة الإسناد (=) وهذا يعطي أثراً قوياً وفعالاً في الوقت نفسه. يبين المثال (2- 42) تابع يعيد مرجعاً .

مثال (2- 42)

// A simple example of a function returning a reference.

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

```
int &function( ); // return a reference
```

```
int x;
```

```
int main( ) {
```

```
    // assign 100 to reference returned by f( )
```

```
    function( )=100 ;
```



```

cout << "x=" << x << "\n";

return 0;

} // End main

// Return an int reference.
int &function( ) {
    return x ; // returns a reference to x
}

```

Result

x=100

Press any key to continue

إن التابع () function يعيد مرجعاً لقيمة صحيحة . لا يعيد داخل التابع () function قيمة المتحول العام وإنما يعيد عنوان هذا المتحول تلقائياً لذلك فإن العبارة : $function() = 100$ تضع القيمة 100 في x لأن التابع () function يعيد مرجعاً إليها . وبالتالي عندما نستخدم التابع () function على الطرف الأيسر من عبارة الإسناد يتم الإسناد إلى المرجع الذي يعيده التابع () function ، وبما أن التابع () function يعيد مرجعاً إلى x فإن x سيتلقى القيمة 100 . ويتمثل أحد الاستخدامات المفيدة لإعادة المرجع في إنشاء نوع مصفوفة ذات حدود، لكن في لغة C++ إنشاء صف لمصفوفة تقوم بالتحقق الآلي من حدود المصفوفة . يحتوي الصف في المثال (2- 43) على وظيفتين أساسيتين تخزن

الأولى المعلومات داخل المصفوفة وتسترجع الثانية المعلومات منها. وتؤكد هاتان
الوظيفتان من عدم تجاوز المصفوفة في وقت التنفيذ .

مثال (2- 43)

// A simple bounded array example.

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

```
#include <stdlib.h>
```

```
class myarray {
```

```
    int size;
```

```
    char *p;
```

```
public:
```

```
    myarray( int num);
```

```
    ~myarray( ) { delete [ ] p; }
```

```
    char &put( int i);
```

```
    char get( int i);
```

```
}; //end class
```

```
myarray::myarray(int num ) {
```

```
    p = new char [num];
```

```
    if(!p) {
```

```
        cout << "Allocation error\n";
```

```
        exit(1);
```

```
    }
```

```
    size = num;
```

```
}
```

// Put something into the array.

```
char &myarray::put(int i) {  
    if(i<0 || i>=size) {  
        cout << "Bounds error!!!\n";  
        exit(1);  
    }  
    return p[i]; // return reference to p[i]  
}
```

// Get something from the array.

```
char myarray::get(int i) {  
    if(i<0 || i>=size) {  
        cout << "Bounds error!!!\n";  
        exit(1);  
    }  
    return p[i]; // return character  
}
```

```
int main( ) {  
    myarray a(20);  
    a.put(5) = 'X';  
    a.put(10) = 'R';  
    cout << a.get(5)<<"\n" << a.get(10);  
    cout << "\n";  
}
```

// now generate run-time boundary error

```
a.put(15) = '!';  
return 0;  
} //End main
```

Result

X

R

Press any key to continue

ونلاحظ أن التابع put يعيد مرجعاً لعنصر مصفوفة محدد بالوسيط i . ويمكن استخدام هذا المرجع في الطرف الأيسر لعبارة الإسناد ، لتخزين شيء ما في المصفوفة ، وذلك إذا لم تكن قد تجاوزنا حدود المصفوفة . أما التابع get فيعيد القيمة المخزنة في الدليل المحدد ، إذا كان هذا الدليل ضمن حدود المصفوفة ، وتسمى هذه الطريقة بالمصفوفة الآمنة ، وهي إحدى الطرق الممكنة استخدامها عند الحاجة لمصفوفة آمنة .

٢-٢٦-٣-المراجع المستقلة Independent References

تُعد المراجع المستقلة نوعاً آخر من المراجع في لغة C++ ، فالمرجع هو متحول مرجعي يمثل اسماً آخر لمتحول آخر . وبما أنه لا يمكن إسناد قيم جديدة للمراجع ، فيجب تهيئة المراجع المستقلة عند التصريح عنها . وتوجد بعض القيود عند استخدام المراجع منها :

- لا نستطيع التصريح عن مرجع لمرجع .
- لا نستطيع الحصول على عنوان مرجع .

- لا نستطيع تشكيل مصفوفة مراجع ،
 - لا نستطيع إنشاء مرجع لحقل Bit .
 - يجب تهيئة المراجع دائماً إلا إذا كانت أعضاء في صف أو قيمياً معادة أو وسائط تابع .
- وبين المثال (2- 44) مرجعاً مستقلاً .

مثال (2- 44)

```
#include <iostream.h>

void main() {
    int x;
    int &ref = x; // create an independent reference
    x = 10; // these two statements
    ref = 10; // are functionally equivalent
    ref = 100;
    // this prints the number 100 twice
    cout << "x=" << x << "\n"
        << "ref=" << ref << "\n";
} //End main
```

Result

x=100

ref=100

Press any key to continue

مسائل عامة محلولة

١- أنشئ صفتين (You , Me) يحوي كل منهما على تابع عضوي يأخذ مؤشراً إلى غرض من الصف الآخر . انشأ نسخاً من الصفتين داخل التابع main() واستدع التابع العضو الذي سبق ذكره في كل صف.

```
#include <iostream.h>
class You; // Forward declaration
class Me {
private :
public:
    void ProcessYou(You* p) {
        cout << "Processing You at " << p << endl;
    }
}; //End class Me

class You {
public:
    void ProcessMe(Me* p) {
```

```

        cout << "Processing Me at " << p << endl;
    }
}; // End class You

int main( ) {
    Me me;
    You you;
    me.ProcessYou(&you);
    you.ProcessMe(&me);

} // End main

```

Result

Processing You at 0x0012FF78

Processing Me at 0x0012FF7C

Press any key to continue

٢- أنشئ صفاً اسمه Hen ، وضع في داخله صفاً متداخلاً اسمه Nest ، وضع داخل الصف Nest صفاً اسمه Egg ، ويجب أن يحوي كل صف تابعاً عضواً هو () display ، ثم انشأ داخل التابع () main نسخة من كل صف واستدع التابع () display لكل نسخة .

```

#include <iostream.h>
class Hen {
public:
    class Nest {
        int x;
        friend class Hen;
        public:
        class Egg {
            private :
                int y;
                friend class Nest;
                public:
                void display( ) {
                    cout << "Hen::Nest::Egg::display: " << y << endl;
                }
            };//End class Egg
            void initEgg(Egg* e) {
                e->y = 2;
            }
            void display( ) {
                cout << "Hen::Nest::display: " << x << endl;
            }
        };//End class Nest
    };
};

```

```

void initNest(Nest* n) {
    n->x = 1;
}
void display( ) {
    cout << "Hen::display\n";
}
}; // End class Hen
void main( ) {
    Hen h;
    Hen :: Nest n;
    Hen :: Nest :: Egg e;
    h.initNest(&n);
    n.initEgg(&e);
    h.display( );
    n.display( );
    e.display( );
} // End main

```

Result

```

Hen::display
Hen::Nest::display: 1
Hen::Nest::Egg::display: 2
Press any key to continue

```

٣- أنشئ صفأ يحوي أعضاء معطيات موزعة ضمن محددات public و private و protected ، ثم أضف تابعاً عضواً هو (show) ، هذا التابع يقوم بطباعة اسم كل عضو ومعطيات وعنوانه .

```
#include <iostream.h>
class mempfun {
private :
    int x;
protected:
    int y;
public:
    int z;
    void show( ) {
        cout << "x is at " << &x << endl;
        cout << "y is at " << &y << endl;
        cout << "z is at " << &z << endl;
    }
}; // End class mempfun

void main( )
{
    mempfun m;
    m.show( );
} // End main
```


Result

x is at 0x0012FF74

y is at 0x0012FF78

z is at 0x0012FF7C

Press any key to continue

٤- أنشئ صفئ stvalue يحوي على عضو معطيات من النوع int ، ثم اكتب برنامجاً له بحيث يأخذ وسيطاً من النوع int ، ويخزن قيمته في عضو المعطيات ، على الباني والمدمر أن يقوموا بطباعة القيمة الصحيحة مع الرسالة التي يولدونها بحيث يمكننا أن نرى الأغراض عند إنشائها وتدميرها .

```
#include <iostream.h>
class stvalue {
private :
    int x;
public:
    stvalue( int x)
    {
        this->x = x;
        cout << "stvalue(" << x << ")\n";
    }
    ~stvalue( ) {
```

```
        cout << "~stvalue(" << x << ")\\n";  
    }  
}; //End class stvalue
```

```
void main( )
```

```
{  
    { stvalue a1(1);  
      stvalue a2(2);  
      stvalue a3(3);  
      stvalue a4(4);  
      goto jump;  
      stvalue a5(5);  
    }  
}
```

```
jump:
```

```
    stvalue a6(6);  
    cout << "After jumping!\\n";
```

```
} //End main
```

Result

stvalue(1)
stvalue(2)
stvalue(3)
stvalue(4)

~stvalue(4)
~stvalue(3)
stvalue(6)

After jumping!

~stvalue(6)
~stvalue(2)
~stvalue(1)

Press any key to continue

٥- أنشئ صفاً اسمه Manyfunc يحتوي على أربعة توابع أعضاء 0 و 1 و 2 و 3 وسيط من النمط int (أي التابع الأول لا يحتوي على وسيط والثاني له وسيطاً واحداً والثالث له وسيطان والرابع ثلاثة) ، اكتب التابع () main الذي يقوم بإنشاء غرض من الصف Manyfunc ، ثم استدعِ التوابع الأعضاء كلها . وعدل البرنامج السابق بحيث يحتوي الصف تابعاً عضواً واحداً يمتلك ثلاثة وسيطات ، ثم قارن بين الخرجين .

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

```
class Manyfunc {
```

```
public:
```

```

void f( )
{
    cout << "{ } \n"; }
void f( int i) {
    cout << "{ " << i << " } \n";
}
void f( int i, int j)
{
    cout << "{ " << i << ", " << j << " } \n";
}
void f( int i, int j, int k) {
    cout << "{ " << i << ", " << j << ", " << k << " } \n";
}
}; //End class Manyfunc

void main( )
{
    Manyfunc a ;
    a.f( );
    a.f(1);
    a.f(1, 2);
    a.f(1, 2, 3);
}; //End main

```

Result

```
{ }  
{1}  
{1, 2}  
{1, 2, 3}
```

Press any key to continue

البرنامج بعد التعديل هو :

```
#include <iostream.h>  
  
class Manyfunc{  
public:  
    void f(int i=0, int j=0, int k=0) {  
        cout << "{" << i << ", " << j << ", " << k << "}\n";  
    }  
}; //End class Manyfunc  
  
void main( ) {  
    ManyArgs a;  
    a.f( );  
    a.f(1);  
    a.f(1, 2);  
    a.f(1, 2, 3);  
} //End main
```


Result

{0, 0, 0}
{1, 0, 0}
{1, 2, 0}
{1, 2, 3}

Press any key to continue

صحيح أن العمليات داخل التابع () main لم تتغير إلا أن الخرجين المتتابعين يظهران لنا الموضع الذي تكون فيه كل ميزة مناسبة .

=====

٦- أنشئ صفاً اسمه Monitor وظيفته عدّ عدد المرات التي يتم فيها استدعاء التابع العضو () incident ، ثم أضف تابعاً عضواً () print يقوم بطباعة عدد مرات الاستدعاء . وأنشئ تابعاً عاماً (ليس عضواً) يحوي غرضاً من النمط static Monitor . و في كل مرة يتم استدعاء التابع يجب أن يتم استدعاء التابع () incident ، ثم () print لطباعة عدد مرات الاستدعاء . قم بتجريب هذا التابع في ال () main .

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

```
class Monitor {
```

```
private :
```

```
int count;
```

```
public:
```

```
Monitor( ) {
```

```
count = 0;
```

```

    }
    void incident( ) {
        ++count;
    }
    void print( ) {
        cout << count << " incidents\n";
    }
}; //End class Monitor

void f( ) {
    static Monitor m;
    m.incident( );
    m.print( );
}

void main( )
{
    for (int i = 0; i < 5; ++i)
        f( );
} //End main

```

Result

```
Trace::Trace(1)
Trace::Trace(const Trace&)
f(1)
Trace::Trace(2)
Trace::Trace(const Trace&)
Trace::Trace(const Trace&)
Trace::Trace(3)
Trace::Trace(const Trace&)
```

Press any key to continue

٧- أنشئ صفاً يحوي * double يقوم الباني بتهيئة المؤشر *double عبر استدعاء new double ، وإسناد قيمة إلى الذاكرة الناتجة من وسيط الباني . ويطبع المدمر القيمة التي يؤشر إليها ويسند إليها القيمة (1-) ، ويستدعي delete لتحرير ذاكرة التخزين ، ثم يطبع قيمة المؤشر إلى الصفر . أنشئ بعد ذلك تابعاً يأخذ غرضاً من الصف بالقيمة ، ويستدعي هذا التابع في التابع الرئيسي (main) ، ماذا يحدث ؟ أصلح المشكلة عبر كتابة باني نسخ .

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

```
class HasPointer {
```

```
private :
```

```
double * p;
```

```
public:
```

```

HasPointer(double x) {
    p = new double(x);
}
~HasPointer( ) {
    cout << "~HasPointer( )\n";
    delete p;
}
void print( ) {
    cout << *p << endl;
}
}; //End class HasPointer
void f(HasPointer hp)
{
    hp.print( );
}

void main( ) {
    HasPointer hp(5);
    f(hp);
} //End main

```

Result

5

~HasPointer()

~HasPointer()

<access violation error...!!!>

Press any key to continue

عندما نستدعي التابع $f(hp)$ يتم تخزين نسخة من الغرض hp للتابع $f()$ والتي تقوم باستدعاء المدمر عندما لا يبقى هناك حاجة إليها . المشكلة أن المدمر يحذف المؤشر p عندما يتم استدعاء المدمر مرة أخرى لتدمير الغرض hp عندما ينتهي التابع $main()$ فإنه سيحاول حذف المؤشر p مرة أخرى! يكمن الحل في جعل المترجم يقوم بنسخ أكبر الأغراض $HasPointer$ عن طريق كتابة بائي نسخ مناسب مثل :

```
HasPointer( const HasPointer& rhs )
```

```
{
```

```
    P = new double ( *rhs.p );
```

```
}
```

كقاعدة عامة ، إن أي صف يحوي عضو معطيات عبارة عن مؤشر ربما يحتاج إلى بائي نسخ .

=====

٨- اكتب برنامجاً يحتوي على صف اسمه `Myclass2` يحتوي على ثلاثة متحولات n, m, x من النمط `int` ، يحتوي الصف على بانٍ عادي لترميز المعطيات إلى الأغراض عند إنشائها ، وعلى تابع مدمر يطبع رسالة ما عند

استدعائه. وعلى تابع بوسيط واحد اسمه Factorial لحساب العامل، ويعيد قيمة من النوع int ، وعلى تابع آخر اسمه SumFact من النوع void يقوم بحساب وطباعة المقدار التالي : $s = n! + \frac{m!}{x!}$

- أنشئ أغراضاً في التابع main() لاستدعاء توابع الأعضاء للصف MyClass2 وطباعة النتائج .

```
#include <iostream.h>

class myclass {
    int n , m , x;

public:
    myclass( int , int , int ); // constructor
    ~myclass( ); // destructor
    int fact( int );
    void sumfact( ) ;
}; // End class

myclass::myclass( int n1 , int m1 , int x1 )
{
    cout << "In constructor\n";
    n = n1 ; m = m1 ; x = x1 ;
}

myclass::~~myclass( )
{
```

```

    cout << "Destructing...\n";
}
int myclass :: fact(int n){
    if (n == 0 && n == 1)
        return 1;
    else
        return n * fact(n-1);
}
void myclass::sumfact( ){
    double z=fact(x);
    double s =fact(n)+ fact( m)/z ;
    cout<<" s="<<s<<endl ;
}
void main( ) {
    myclass ob( 7 ,6 ,5);
    ob.sumfact( );
} //End main

```

Result

In constructor
s = 118.857
Destructing.

Press any key to continue

٩- اكتب برنامجاً يحتوي على صف اسمه myclass1 يحتوي على ثلاثة متحولات a,b,x من النمط double ، يحتوي الصف على بان عادي لتمير المعطيات إلى الأغراض عند إنشائها ، وعلى تابع مدمر يطبع رسالة ما عند استدعائه . وعلى تابع اسمه functY من النوع double يقوم بحساب المقدار التالي : $y = x^2 + 2x + 10$ ، وعلى تابع آخر اسمه functZ من النوع void يقوم بحساب وطباعة المقدار التالي : $Z = (y - a)^2 + (y + b)^2$.

- أنشئ أغراضاً في التابع () main لاستدعاء توابع الأعضاء للصف MyClass وطباعة النتائج .

```
#include <iostream.h>

class myclass {
    double a , b , x;
public:
    myclass( double , double , double ); // constructor
    ~myclass( ); // destructor
    double functy( );
    void functz( ) ;
}; //End class

myclass::myclass( double a1 , double b1 , double x1 ) {
    a= a1 ; b= b1 ; x = x1 ;
}

myclass::~~myclass( ) {
```

```

    cout << "Destructing...\n";
}
double myclass :: functy( ){
    double y = x*x + 2*x +10 ;
    return y ;
}
void myclass::functz( ){
    double z=( functy( ) - a)*( functy( ) - a) +
        ( functy( ) - b)*( functy( ) - b) ;
    cout<<" z="<<z<<endl ;
}
int main( ) {
    myclass ob( 7 ,6 ,5);
    ob.functz( );
    return 0;
} //End main

```

Result

In constructor

z = 2965

Destructing.

Press any key to continue

- ١٠- اكتب برنامجاً يحتوي صفراً اسمه compute ، يحتوي على وسيطين x2 ، x1 من النوع float كأعضاء خاصة ويحتوي على التوابع العامة التالية:
- تابع عضو calculate من النوع void لحساب جذور معادلة من الدرجة الثانية $ax^2 + bx + c = 0$ ، ثم إسناد النتائج للمعطيات الخاصة (x1 و x2) .
 - تابع عضو لطباعة قيم المعطيات الخاصة وليكن اسمه print ومن النوع void.
 - تابع صديق للصف compute (وسيطه غرض) وليكن اسمه fr ، مهمته حساب المقدار المعرف كما يلي:
$$z = \frac{x1 + 3x2}{15}$$
 - في التابع الرئيسي main صرح عن غرض ob من الصف compute ومن ثم ادخل قيم الأمثال العددية للمعادلة من الدرجة الثانية.
 - يُطلب طباعة القيم: x1 ، x2 و z .

```
#include<iostream.h>
#include<math.h>
class compute {
    float x1,x2 ;
public :
    void calculate ( float , float , float ) ;
    void print ( ) ;
    friend float fr(compute ob );
} ; // End class

void compute::calculate( float a, float b, float c ) {
```



```

float delta;
delta=b*b-4*a*c;
if (delta<0 )
{
    x1=0 ; x2=0 ;
}
else
{
    x1=(-b-sqrt(delta))/(2*a);
    x2=(-b+sqrt(delta))/(2*a);
}
} //End calculate

float fr (compute ob) {
    float z;
    if(ob.x1==0 &&ob.x2==0)
        cout<<"No real solutions"<<endl;
    else
        z=(ob.x1+3*ob.x2)/15;
    return z;
} //End fr

void compute :: print( ) {
    cout << "x1="<<x1<<endl;

```

```

        cout << "x2=" << x2 << endl;
    } //End print

void main( ) {
    float a,b,c;
    compute obl ;
    cout<<"Enter the values of a,b and c";
    cin>>a>>b>>c;
    obl.calculate(a,b,c);
    obl.print( );
    cout<<"\n z=" << fr(obl) << endl;
} // End main

```

- ١١- اكتب برنامجاً يحتوي على صف اسمه Array ، ويحتوي الصف على وسيطين min ، max كأعضاء خاصة ، ويحتوي على الأعضاء العامة التالية:
- بأن افتراضي يسند القيمة صفر للمعطيات الخاصة.
 - بأن عادي وسيطه مصفوفة ويقوم بإيجاد كل من العنصرين الأكبر والأصغر من بين عناصر مصفوفة من الأعداد مكونة من عشرين عنصراً، ومن ثم يُسند النتائج للمعطيات الخاصة max و min على الترتيب.
 - تابع عضو لطباعة قيم المعطيات الخاصة وليكن اسمه print ومن النوع .void
 - تابع مدمر يطبع رسالة ما عند استدعائه.
 - وتابع صديق وسيطه غرض يقوم بحساب القيمة المطلقة لفرق القيمتين العظمى والصغرى

- في التابع الرئيسي main أدخل 20 عدداً صحيحاً و خزنها في مصفوفة أحادية البعد y[20] . بعد ذلك صرح عن غرض ob من الصف Array ومن ثم استدعِ التوابع اللازمة لإيجاد وطباعة العنصرين الأكبر والأصغر .

```
#include<iostream.h>
#include<math.h>
class Array {
    int max, min ;
public:
    Array( ){max=0;min=0;}
    Array( int [ ] );
    void print( ){cout<<"max="<<max<<" "
                <<"min="<<min<<endl ; }
    friend int Fr(Array ) ;
    ~Array(){ cout << "Destructor \n" ;}
};//End class

Array::Array(int x[ ] ) {
    int i, mx=x[0], mn=x[0] ;
    for(i=0;i<20;i++) // حساب العنصر الأكبر
    {
        if(x[i]>mx)
            mx=x[i];
    }
}
```

```

    }
    for(i=0;i<20;i++) // حساب العنصر الأصغر
    {
        if(x[i]<mn)
            mn=x[i];
    }
    max=mx;
    min=mn;
}
int Fr( Array ob) {
    int d;
    d=fabs(ob.max-ob.min);
    return d;
}
void main( ) {
    int y[20];
    cout<<"Enter the numbers \n";
    for(int i=0;i<20;i++)
        cin>>y[i];
    Array ob(y);
    ob.print( );
    cout<<"The difference is "<<Fr(ob)<<endl;
} // End main

```

- ١٢- صفاً اسمه engineer يحتوي على وسيط mark من النوع int كعضو خاص ويحتوي على الأعضاء العامة (public) التالية:
- بان افتراضي يسند القيمة صفر للعضو الخاص mark.
 - بان عادي وسيطه من نوع مصفوفة ويقوم بحساب عدد عناصر المصفوفة التي قيمها أكبر أو تساوي 60.
 - تابع وصول getmark يعيد قيمة mark .
 - تابع (غير عضو) وليكن اسمه counter وسيطه غرض من الصف engineer و يقوم بإسناد القيمة المعادة من التابع getmark إلى متغير t ومن ثم يعيد القيمة t .
 - تابع عضو لتحميل العملية الثنائية (-) بشكل زائد، تابع مدمر لا يفعل شيئاً.
 - يتضمن التابع main إدخال درجات طالب ما لـ 12 مقرر وتخزينها في مصفوفة أحادية البعد m[12] .
 - بعد ذلك التصريح عن غرض ob من الصف engineer ومن ثم استدعاء التابع counter لحساب وطباعة عدد المقررات التي نجح فيها الطالب (على اعتبار أن درجة النجاح في المقرر هي 60) .
 - بعد ذلك يجب التصريح عن الأغراض المناسبة من الصف engineer من أجل استدعاء تابع التحميل الزائد للعملية (-) وذلك باستخدام قيم كيفية لهذه الغاية (هذا الطلب يتم حله بعد فصل التحميل الزائد للعوامل).

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

```
class engineer {
```

```
    int mark;
```



```

public:
    engineer () { mark=0; }
    engineer ( int x[ ] );
    int getmark( ) { return mark; }
    engineer operator -(engineer);
    ~engineer( ) { }
}; //End class

engineer::engineer( int x[ ] )
{
    int i , c = 0 ;
    for(i=0; i<12; i++)
    {
        If ( x[i] >= 60 )
            c+=1;
    }
    mark=c;
}; // End function

int counter(engineer ob) {
    int t;
    t=ob.getmark( );
    return t;
}

```

```

engineer engineer:: operator - (engineer ob1)
{
    engineer t;
    t.mark=mark-ob1.mark;
    return t;
}

```

```

void main() {
    int m[12], m1[12], i;
    cout<<"Enter the marks \n";
    for( i=0;i<12;i++)
        cin >> m[i] ;
    engineer ob( m ) ;
    cout<<"The number of success courses is "
        <<counter(ob)<<endl;
    cout<<"Enter the another marks \n";
    for( i=0;i<12;i++)
        cin >> m1[i] ;
    engineer ob1(m), ob2(m1), ob3;
    ob3=ob1-ob2;
    cout<<"ob3="<<ob3.getmark( )<<endl;
} //End main

```

١٣- اكتب برنامجاً بلغة C++ يعرف صفراً اسمه `queuetype` (رتل) يحتوي على متحولين خاصين هما : بداية الرتل `head` ونهاية الرتل `tail` وهما من نمط `int` وعلى مصفوفة الرتل `queue` وهي من نمط `int` ، ويحتوي وعلى بان افتراضي لإسناد القيمة صفر للمتحويلات الخاصة ، وعلى تابع تعديل معطيات لتخزين القيم في الرتل يدعى `queueq` بوسيط واحد وعلى تابع يعيد قيمة لاستعادة المعطيات من الرتل يدعى `deq` .
أنشئ في التابع الرئيسي `main` غرضين (رتلين) واستدعاء التوابيع اللازمة وطباعة النتائج .

```
#include <iostream.h>
const int SIZE= 10;
class queuetype {
    int queue[SIZE];    // holds the queue
    int head, tail;     // indices of head and tail
public:
    queuetype( );       // constructor
    void queueq(int num); // store
    int deq( ); // retrieve
}; //end class

// Constructor
queuetype::queuetype( ) {
```

```

    head = tail = 0;
}

// Put value on queue.
void queueType::queueq(int num) {
    if(tail+1==head || (tail+1==SIZE && !head)) {
        cout << "Queue is full\n";
        return;
    }
    tail++;
    if(tail==SIZE) tail = 0; // cycle around
    queue[tail] = num;
}

// Remove value from queue.
int queueType::deq()
{
    if(head == tail) {
        cout << "Queue is empty\n";
        return 0; // or some other error indicator
    }
    head++;
    if(head==SIZE) head = 0; // cycle around

```

```

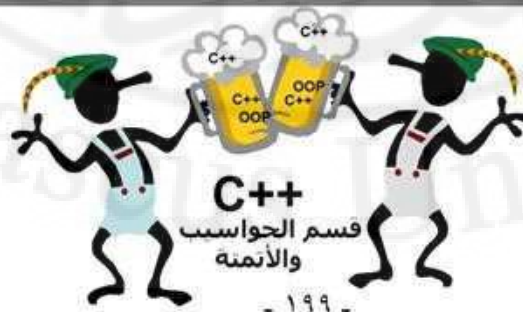
    return queue[head];
}

int main()
{
    queue<int> obq1, obq2;
    int i;
    for(i=1; i<=10; i++)
    {
        obq1.enqueue(i);
        obq2.enqueue(i*i);
    }
    for(i=1; i<=10; i++)
    {
        cout << "Dequeue 1: " << obq1.dequeue() << "\n";
        cout << "Dequeue 2: " << obq2.dequeue() << "\n";
    }
    return 0;
} //End main

```

Result

Queue is full
Queue is full
Dequeue 1: 1
Dequeue 2: 1
Dequeue 1: 2
Dequeue 2: 4
Dequeue 1: 3
Dequeue 2: 9
Dequeue 1: 4
Dequeue 2: 16
Dequeue 1: 5
Dequeue 2: 25
Dequeue 1: 6
Dequeue 2: 36
Dequeue 1: 7
Dequeue 2: 49
Dequeue 1: 8
Dequeue 2: 64
Dequeue 1: 9
Dequeue 2: 81
Queue is empty
Dequeue 1: 0
Queue is empty
Dequeue 2: 0
Press any key to continue



مسائل عامة

١- اكتب برنامجاً بلغة C++ لإنشاء صف أساسي اسمه Department .
يحتوي الصف على ثلاثة متحولات خاصة private نوع char ، وهي : اسم
القسم DepartmentName (قسم هندسة الحواسيب) ، واسم رئيس القسم
CharminName ، وسنة تأسيس القسم year (2003) . ويحتوي الصف
على بانٍ افتراضي و بانٍ عادي لتمثيل المعطيات وتابع طباعة اسمه
. PrintDepartment()
أنشئ في التابع () main أغراضاً للصف Department لتمثيل المعطيات
وطباعة النتائج .

=====

٢- اكتب برنامجاً بلغة C++ للصف الأساسي Book بحيث يحتوي الصف
على المتحولات الخاصة التالية : اسم المؤلف Writer واسم الكتاب name
ودار النشر publishing عدد الصفحات page وهي من النوع char .
ويحتوي الصف على تابع بانٍ عادي وتابع طباعة وتابع مدمر يطبع رسالة ما .
أنشئ في التابع () main أغراضاً للصف Book لتمثيل المعطيات وطباعة
النتائج .

=====

٣- اكتب برنامجاً بلغة C++ للصف الأساسي Company بحيث يحتوي على
المتحولات الخاصة التالية : اسم الشركة namecomp ، واسم صاحب
الشركة الثلاثي (First , Second , last) والمدينة City ، وهي من النوع
char ، ويحتوي الصف على تابع بانٍ وتابع طباعة وتابع مدمر يطبع رسالة ما .

أنشئ في التابع () main أغراضاً للصف Book لتمرير المعطيات وطباعة النتائج .

=====

٤- اكتب برنامجاً بلغة C++ لإنشاء صف أساسي اسمه Faculty يحتوي هذا الصف على ثلاثة متحولات خاصة private من النوع int وهي : عدد الأساتذة Teacher ، وعدد المهندسين Engineer ، وعدد الطلاب Student . ويحتوي الصف على تابع بانٍ افتراضي و بانٍ عادي لتمرير المعطيات وتابع طباعة اسمه PrintFaculty() .

أنشئ في التابع () main أغراضاً للصف Book لتمرير المعطيات وطباعة النتائج .

=====

٥- أنشئ صفاً اسمه numbers يحتوي على وسيطين se , sd كأعضاء خاصة ويحتوي على الأعضاء العامة التالية:

- بانٍ افتراضي يسند القيمة صفر للمعطيات الخاصة.
- بانٍ عادي وسيطه مصفوفة ويقوم بحساب مجموع الأعداد الفردية ومجموع الأعداد الزوجية لمصفوفة من الأعداد مكونة من عشرة عناصر ومن ثم يسند النتائج للمعطيات الخاصة sd و se على الترتيب.
- تابع عضو لطباعة قيم المعطيات الخاصة وليكن اسمه print ومن النوع void.
- تابع مدمر يطبع رسالة ما عند استدعائه.
- في التابع الرئيسي main أدخل 20 عدد صحيح و خزنها في مصفوفة أحادية البعد [20]x . بعد ذلك صرح عن غرض ob من الصف numbers ، و ثم استدعِ التتابع اللازمة لحساب وطباعة المجاميع المطلوبة.

- ٦- أكتب برنامجاً بلغة C++ يتضمن مايلي:
- صفاً اسمه sum يحتوي على وسيطين p , n من النوع float كأعضاء خاصة private ويحتوي على الأعضاء العامة (public) التالية:
- بأن افتراضي يسند القيمة صفر للمعطيات الخاصة.
 - بأن عادي وسيطه مصفوفة ويقوم بحساب مجموع الأعداد الموجبة ومجموع الأعداد السالبة لمصفوفة من الأعداد مكونة من 50 عنصراً ومن النوع float ومن ثم يسند النتائج للمعطيات الخاصة p و n على الترتيب.
 - تابع عضو لطباعة قيم المعطيات الخاصة وليكن اسمه print ومن النوع void.
 - تابع صديق للصف sum (غير عضو) وليكن اسمه fr ، مهمته حساب الفرق بين مجموعي الأعداد الموجبة والأعداد السالبة.
 - تابع مدمر يطبع رسالة ما عند استدعائه.
 - تابعاً رئيسياً main يطلب إدخال 50 عدداً حقيقياً ويخزنها في مصفوفة أحادية البعد n[50] ومن ثم طباعتها. بعد ذلك التصريح عن غرض ob من الصف sum ومن ثم استدعاء التوابيع اللازمة لحساب وطباعة المجاميع المطلوبة والفرق بينهما.

- ٧- أنشئ صفاً اسمه student يحتوي على وسيط smark من النوع int كعضو خاص private ويحتوي على الأعضاء العامة (public) التالية:
- بأن افتراضي يسند القيمة صفر للعضو الخاص smark.
 - بأن عادي وسيطه من نوع مصفوفة ويقوم بحساب عدد عناصر المصفوفة الأكبر أو تساوي 60.

- تابع وصول getsmark يعيد قيمة smark.
- تابع (غير عضو) وليكن اسمه counter وسيطه غرض من الصف student و يقوم بإسناد القيمة المعادة من التابع getsmark إلى متغير t ومن ثم يعيد القيمة t .
- تابع مدمر لا يفعل شيئاً.
- في التابع الرئيسي main أدخل درجات طالب ما في 10 مقررات وخزنها في مصفوفة أحادية البعد [10]x . بعد ذلك صرح عن غرض ob من الصف student ومن ثم استدع التابع counter لحساب وطباعة عدد المقررات التي نجح فيها الطالب (على اعتبار أن درجة النجاح في المقرر هي 60).

=====

- ٨- اكتب برنامجاً يحتوي على الصنفين car و truck حيث أن:
 - ❖ الصف car يحتوي على passengers and speed كأعضاء خاصة من النوع int وعلى التوابع العامة التالية:
 - بأن افتراضي يسند القيمة صفر للأعضاء الخاصة .
 - بأن عادي بوسيطين، مهمته إسناد قيم للأعضاء الخاصة . وتابع مدمر لا يفعل شيئاً .
 - تابع صديق fr للصف car و من النوع int من أجل حساب الفرق بين سرعتي الـ car و الـ truck.
 - ❖ الصف truck يحتوي على weight and speed كأعضاء خاصة من النوع int وعلى التوابع العامة التالية:
 - بأن افتراضي يسند القيمة صفر للأعضاء الخاصة .
 - بأن عادي بوسيطين، مهمته إسناد قيم للأعضاء الخاصة .
 - تابع مدمر لا يفعل شيئاً .

- تابع صديق fr للصف truck و من النوع int من أجل حساب الفرق بين سرعتي الـ car و الـ truck.

❖ في التابع الرئيسي (main) صرح عن غرضين (بقيم مختلفة) لكل من الصفين : car و truck على الترتيب، وبعد ذلك قم بالاستدعاءات اللازمة لحساب الفرق بين سرعتين وطباعة الرسائل المناسبة، حيث أنه إذا كان الفرق بين سرعتين أكبر من الصفر ، فإن الـ car أسرع من الـ truck، وإذا كان الفرق بين سرعتين أصغر من الصفر ، فإن الـ truck أسرع من الـ car، وإلا فإن سرعتين متساويتين.

=====

٩- أنشئ صفاً اسمه HamakEngineer يحتوي على وسيط Good من النوع int كعضو خاص ويحتوي على الأعضاء العامة التالية:

- بان افتراضي يسند القيمة صفر للعضو الخاص.
- باني عادي بوسيطين وسيطه الأول مصفوفة والثاني بعدها ، مهمته حساب عدد عناصر المصفوفة التي تزيد قيمها عن 220 ومن ثم إعطاء النتيجة للعضو الخاص.
- تابع وصول getGood يعيد قيمة Good.
- تابع صديق وليكن اسمه RG ، مهمته حساب الفرق بين قيمة العضو الخاص والعدد الكلي للمجاميع المدخلة ومن ثم إعادة النتيجة .
- تابع مدمر يطبع رسالة ما عند استدعائه.

❖ في التابع الرئيسي main أدخل مجاميع 300 طالباً وطالبة في الثانوية العامة وخرنها في مصفوفة أحادية البعد s[300] . بعد ذلك صرح عن غرض ob من الصف HamakEngineer من أجل حساب عدد الطلاب الذين يحق لهم التسجيل في كلية الهندسة (علماً بأن شرط التسجيل هو أن يزيد مجموع

- الطالب عن 220 درجة) بعد ذلك استدع التتابع اللازمة لطباعة عدد الطلاب المقبولين (Good) وعدد الطلاب غير المقبولين في الكلية.
- استدع تابعي: التحميل الزائد للعملية + و الوصول ،من أجل جمع الغرضين (r1,r1) و (r2,r2) وطباعة ناتج الجمع .

- ١٠- اكتب برنامجاً يحتوي صفاً اسمه Average يحتوي على وسيط av من النوع float كعضو خاص ويحتوي على الأعضاء العامة التالية:
- بأن افتراضي يسند القيمة صفر للعضو الخاص.
 - بأن عادي بوسيطين وسيطه الأول مصفوفة والثاني بعدها، مهمته هي حساب المتوسط الحسابي لعناصر المصفوفة التي تزيد قيمها عن 60 ومن ثم إعطاء القيمة للعضو الخاص.
 - تابع وصول getav يعيد قيمة av.
 - تابع صديق وليكن اسمه RA ، مهمته حساب نصف مربع قيمة العضو الخاص av ومن ثم إعادة النتيجة .
 - تابع مدمر لا يفعل شيئاً.
- ❖ في التابع الرئيسي main أدخل درجات طالب ما في 15 مقرر وخزن هذه الدرجات في مصفوفة أحادية البعد m[15] . بعد ذلك صرح عن غرض ob من الصف Average من أجل حساب قيمة العضو الخاص av ومن ثم استدع التتابع اللازمة لطباعة المعدل العام للطالب المعني (av) وطباعة نصف مربع المعدل العام.

١١- اكتب برنامجاً بلغة c++ لإنشاء صف point يعرف نقطة في مستو، حيث y, x يمثلان إحداثيات النقطة من نمط double، ويحتوي الصف على التوابع التالية:

- بأن عادي مهمته إسناد قيم للأعضاء الخاصة في الصف.
 - بأن افتراضي يقوم بإسناد قيم صف لإحداثيات النقطة .
 - تابع عضو عام اسمه printxy يقوم بطباعة إحداثيات النقطة.
 - تابع عضو عام اسمه printvector يقوم بحساب طولية و زاوية النقطة وطباعتها.
 - تابع عضو عام اسمه vectortopoint بوسيطين هما الطولية والزاوية ويحسب منها إحداثيات النقطة ويسندها إلى x, y .
 - تابع عضو عام اسمه shift بوسيطين ويقوم بإزاحة النقطة بقيمة هذين الوسيطين ويسنده إلى x, y .
 - تابع مدمر يطبع رسالة ما عند استدعائه .
- ❖ نعرف في التابع الرئيسي main نقطة إحداثياتها $A(1,1)$ ثم طباعة إحداثيات وطولية و زاوية هذه النقطة. ثم نعرف نقطة جديدة B بدون إحداثيات ثم طباعة إحداثيات و طولية و زاوية هذه النقطة. يتم إجراء إزاحة النقطة A بمقدار $(2,1)$ ثم طباعة إحداثياتها. إسناد قيمة النقطة A إلى النقطة B ثم طباعة إحداثيات النقطة B .

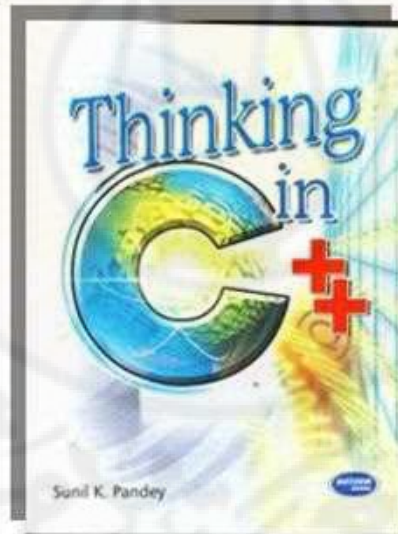




الفصل الثالث

التحميل الزائد للعوامل

OPERATOR OVERLOADING





التحميل الزائد للعوامل

٣-١- مقدمة

تُعد عملية البرمجة بلغة C++ عملية مرتبطة بالأنماط ومقادة بها ، حيث يمكن للمبرمج أن يستخدم الأنماط المعرفة مسبقاً ، ويمكن له أن يعرف أنماط أخرى جديدة . يستطيع المبرمج أيضاً أن يستخدم العمليات المتوفرة مع الأنماط التي يقوم بتعريفها ، ويجب أن نعرف أن لغة C++ لا تسمح بتعريف عمليات جديدة ، وإنما تسمح بتحميل العمليات الموجودة بمعان لم تكن مرتبطة بها من قبل . تُعد عملية التحميل الزائد من الميزات الكبيرة التي تتمتع بها لغة C++ .

٣-٢- مبادئ التحميل الزائد

إن التحميل الزائد للعوامل في الصفوف هو أحد أشكال التحميل الزائد للتوابع مع وجود بعض الفوارق ، وأيضاً التحميل الزائد لأحد العوامل لا يفقد أي شيء من معناه الأصلي ، بل يكتسب معنىً جديداً حسب الصف الذي عُرف من أجله . عند التحميل الزائد لعامل ما يجب إنشاء تابع يكون إما عضواً في صف أو صديقاً له ، ويوجد اختلاف بسيط بين تابع عضو عام في صف وتابع صديق له . ونستطيع باستخدام التحميل الزائد للعوامل إضافة أنواع جديدة إلى البرنامج بطريقة سلسة ومنطقية . والشكل العام لإنشاء تابع عضو لعامل ما هو كما يلي :

```
Return-Type Class _Name :: operator # ( arg -List)
{
    // Operator To be performed ;
}
```


تكون نوع القيمة المعادة من التابع عادة من نوع الصف الذي عرف له التابع ، ويمكن إعادة أي نوع ، والإشارة # تدل على أي نوع من العوامل المراد تحميلها بشكل زائد . مثلاً :

عامل الجمع + operator

عامل الضرب * operator

تتغير لائحة الوسائط arg_List تبعاً لكيفية عمل العامل ونوعه . وعند التحميل الزائد يجب الانتباه إلى أمرين :

١. لا يمكن تغيير أفضلية العملية .

٢. لا يمكن تعديل عدد معاملات العملية .

سنقتصر في هذا الفصل على التحميل الزائد للعوامل الأكثر استخداماً في

C++ . يرث الصف المشتق توابع العوامل ما عدا عامل الإسناد " = " ،

وبشكل عام يستطيع الصف المشتق زيادة تحميل أي عامل يختاره . إن توابع العوامل لا تأخذ وسائط افتراضية .

٣-٣- التحميل الزائد للعوامل الثنائية

Overloading Binary Operators

عند التحميل الزائد للعوامل الثنائية باستخدام تابع عضو في الصف ، فإن

التابع يأخذ وسيطاً واحداً . يستلم هذا الوسيط الغرض الموجود على الجانب الأيمن

للعامل ، أما الغرض على الجانب الأيسر فيستخدم لتوليد الاستدعاء للتابع العامل

ويمرر ضمناً باستخدام المؤشر this ، ويمكن كتابة توابع العامل بعدة أشكال ،

وسنستخدم الشكل الأكثر انتشاراً ، وسنستعرض أهمها .

٣-٣-١ - التحميل الزائد لعملية الجمع " + "

في التحميل الزائد لعملية الجمع تكون الصيغة العامة بالنسبة للصف point هي كما يلي :

point operator + (point ob2) ;

يتم جمع المعامل الأيمن مع المعامل الأيسر وبما أن المعامل الأيسر هو الذي يولد الاستدعاء يجب أن تكون عملية الجمع كما يلي :

x + ob2.x

y + ob2.y

لا يكون الترتيب مهماً في هذه العملية .

مثال (1-3)

يقوم البرنامج التالي بالتحميل الزائد للمعامل " + " في الصف point الذي يستخدم إحداثيات النقطة x , y ، والذي يقوم بجمع غرضين ob1 + ob2.

//EXAMPLE - Overloading Binary Operator (+)

#include <iostream.h>

class point {

private :

int x , y ; // point value .

public :

point() { x = 0 ; y = 0;} // Default constructor

```

    point( int i,int j ) { x=i ; y=j ; }
    void get_xy( int &i , int &j ) { i = x ; j = y ; }
    point operator + ( point ob2 ) ;
}; // End Class

// overload + relative to point class .
point point :: operator + ( point ob2 ) {
    point temp ;
    temp .x = x + ob2.x ;
    temp .y = y + ob2.y ;
    return temp ;
}

void main ( ) {
    point obj1(15,10) ,obj2(8,5), obj3 ;
    int x1 , y1 ;
    obj3 = obj1 + obj2 ; // add tow objects ,
                        // this call operator+( )
    obj3.get_xy( x1 , y1 ) ;
    cout<<" (obj1+obj2) x : "<< x1<<" y : " <<y1<<endl ;
} //End main

```

RESULT

(obj1+obj2) x : 23 y: 15

Press any key to continue

ويعيد التابع () + operator غرضاً من النوع point الذي يحتوي على مجموع الإحداثيات x في x و y في y . وتم استخدام غرض مؤقت temp للاحتفاظ بالنتيجة والذي يعاد من تابع العامل . إن أيّاً من المعاملين لم يتغير وبقيت x و y كما هي ، لذلك نحتاج إلى غرض مؤقت للاحتفاظ بالقيمة . تابع التحميل الزائد يعيد غرضاً من النوع point أي :

$$ob3 = ob1 + ob2 ;$$

لذلك نستطيع استخدام جمع الأغراض في تعبير أكبر . بما أن التابع يعيد غرض من النوع point تعتبر العبارة التالية صحيحة :

$$(ob1 + ob2).get_xy(x1, y1);$$

عندما أنشأنا العامل () + operator لم يكن الترتيب مهماً (ob1 + ob2) أو (ob2 + ob1) أما بالنسبة إلى الطرح يكون الترتيب مهماً في هذه الحالة .

٣-٢- التحميل الزائد لعملية الطرح (-)

لكي يكون التحميل الزائد صحيحاً يجب طرح المعامل الأيمن من المعامل الأيسر ، وبما أن المعامل الأيسر هو الذي يولد الاستدعاء يجب أن تكون عملية الطرح كما يلي :

$$x - ob2.x$$

$$y - ob2.y$$

والشكل العام للتحميل الزائد لعملية الطرح بالنسبة للصف point هو كما يلي :

$$point operator - (point ob2);$$

مثال (2-3)

يقوم البرنامج التالي بالتحميل الزائد للمعامل " - " في الصف point الذي يستخدم إحداثيات النقطة x , y ، والذي يقوم بطرح غرضين ob1 - ob2 .

//EXAMPLE - Overloading Binary Operator (-)

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

```
class point {
```

```
private :
```

```
    int x , y ; // point value .
```

```
public :
```

```
    point( ) { x = 0 ; y = 0 ; } // Default constructor
```

```
    point( int i , int j ) { x = i ; y = j ; }
```

```
    void get_xy( int &i , int &j ) { i = x ; j = y ; }
```

```
    point operator - ( point ob2 ) ;
```

```
}; // End Class
```

```
// overload - relative to point class .
```

```
point point :: operator - ( point ob2 ) {
```

```
    point temp ;
```

```
    temp.x = x - ob2.x ;
```

```
    temp.y = y - ob2.y ;
```

```
    return temp ;
```

```
}
```



```

void main ( ) {
    point obj1(15,5 ) ,obj2(8,10), obj3 ;
    int x1 , y1 ;
    obj3 = obj1 - obj2 ; // sub tow objects ,
        // this call operator -( )
    obj3.get_xy( x1 , y1 ) ;
    cout<<" (obj1 - obj2) x : "<<x1<<" y : "<<y1<<endl ;
} //End main

```

RESULT

(obj1- obj2) x : 7 y: - 5

Press any key to continue

٣-٣-٣- التحميل الزائد لعملية الإسناد (=)

في عملية الإسناد " = " إن الغرض قد عدل بالعملية ، وهذا يتفق مع المعنى الأصلي لعملية الإسناد ، والصيغة العامة لعملية الإسناد بالنسبة للصف point هي كما يلي :

```

point operator = ( point ob2) ;

```

مثال (3-3)

يقوم البرنامج التالي بالتحميل الزائد للعوامل الثنائية في الصف `point` وهي الجمع والطرح والإسناد $(+ , - , =)$.

//EXAMPLE - Overloading Binary Operators (+ , - , =)

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

```
class point {
```

```
private :
```

```
    int x , y ; // point value .
```

```
public :
```

```
    point( ) { x = 0 ; y = 0 ; } // Default constructor
```

```
    point( int i , int j ) { x = i ; y = j ; }
```

```
    void get_xy( int &i , int &j ) { i = x ; j = y ; }
```

```
    point operator + ( point ob2 ) ;
```

```
    point operator - ( point ob2 ) ;
```

```
    point operator =( point ob2 ) ;
```

```
}; // End Class
```

```
point point :: operator + ( point ob2 ) {
```

```
    point temp ;
```

```
    temp .x = x + ob2.x ;
```

```
    temp .y = y + ob2.y ;
```

```

        return temp ;
    }
    point point :: operator - ( point ob2 ) {
        point temp ;
        temp .x = x - ob2.x ;
        temp .y = y - ob2.y ;
        return temp ;
    }
    point point :: operator = ( point ob2 ) {
        x = ob2 . x ;
        y = ob2 . Y ;
        return *this ; // return the object that is assigned
    }

void main ( ) {
    point obj1(15,5),obj2(8,10),obj3, obj4,obj5;
    int x1, y1 ;
    obj3 = obj1 + obj2 ; // add tow objects
    obj4 = obj1 - obj2 ; // sub tow objects
    obj5 = obj1 ; // assign an object
    obj3.get_xy(x1,y1) ;

```

```

cout<<" (obj1+ obj2) x : "<< x1<<" y : "<<
    <<y1<< endl ;
obj4.get_xy(x1,y1) ;
cout<<" (obj1- obj2) x : "<< x1<<" y : "<<
    <<y1<< endl ;
obj5.get_xy(x1,y1) ;
cout<<" (obj5= obj1) x : "<< x1<<" y : "<<
    <<y1<< endl ;
} // End main

```

RESULT

(obj1+ obj2) x : 23 y : 15

(obj1 - o bj2) x : 7 y : -5

(obj5 = obj1) x : 15 y : 5

Press any key to continue

إن التابع يعيد مؤشراً `this *` ، أي التابع () `operator =` يعيد الغرض الذي أسند إليه وذلك للسماح بسلسلة من عمليات الإسناد ، وتسمح عملية إعادة المؤشر `this` من عامل الإسناد للأغراض من النوع `point` أن تستخدم العبارة التالية :

`obj3 = obj2 = obj1 ;`

٣-٣-٤ - المعامل الأيمن في التحميل الزائد من النوع int

يمكن أن يكون المعامل الأيمن في التحميل الزائد من النوع الثابت مثل int ، بدلاً من أن يكون غرضاً من نفس الصف . ويجب أن يكون المعامل الأيسر غرضاً وليس من النوع الثابت . لأن المعامل الأيسر هو الذي سيولد الاستدعاء إلى التابع ، وبالتالي يجب أن يكون غرضاً وليس عدداً ثابتاً . ويمكن أن يكون المعامل الأيمن من النوع int أو double .

مثال (3-4)

يقوم البرنامج التالي بالتحميل الزائد للعوامل الثنائية في الصف point وهي الجمع والطرح (+ و -) وذلك باستخدام المعامل الأيمن من النوع الثابت int .

```
//EXAMPLE - int Type , Overloading Binary
// Operators ( + , - )

#include <iostream.h>

class point {
private :
    int x , y ; // point value .
public :
    point ( ) { x = 0 ; y = 0 ; } // Default constructor
    point ( int i , int j ) { x = i ; y = j ; }
    void get_xy ( int &i , int &j ) { i = x ; j = y ; }
```



```

point operator + ( point ob2 );
point operator + ( int i );

point operator - ( point ob2 );
point operator - ( int i );

}; // End Class

point point :: operator + ( point ob2 ) {
    point temp ;
    temp .x = x + ob2.x ;
    temp .y = y + ob2.y ;
    return temp ;
}

point point :: operator + ( int i ) {
    point temp ;
    temp .x = x + i ;
    temp .y = y + i ;
    return temp ;
}

point point :: operator - ( point ob2 ) {
    point temp ;
    temp .x = x - ob2.x ;
    temp .y = y - ob2.y ;

```

```

        return temp ;
    }

    point point :: operator - ( int i ) {
        point temp ;
        temp .x = x - i ;
        temp .y = y - i ;
        return temp ;
    }

    void main ( ) {
        point obj1(15,5),obj2(8,10), obj3, obj4, obj5, obj6 ;
        int x1, y1 ;

        obj3 = obj1 + obj2 ; // add tow objects
        obj4 = obj1 + 30 ; // add objects + int
        obj5 = obj1 - obj2 ; // sub tow objects
        obj6 = obj1 - 20 ; // sub object - int
        obj3.get_xy(x1,y1) ;
        cout<<" (obj1+ obj2) x : "<<x1<<" y : "<<y1<< endl ;

        obj4.get_xy(x1,y1) ;
        cout<<" (obj1+ int) x : "<<x1<<" y : "<<y1<< endl ;
    }

```

```

obj5.get_xy(x1,y1);
cout<<" (obj1- obj2) x : "<<x1<<" y : “
        <<y1<< endl;
obj5.get_xy(x1,y1);
cout<<" (obj1 - int ) x : "<<x1<<" y : “
        <<y1<< endl;
} // End main

```

RESULT

```

(obj1+ obj2 ) x : 23  y : 15
( obj1 + int ) x : 45  y : 35
( obj1- obj2 ) x : 7   y : - 5
( obj1 - int ) x : -15 y : - 25

```

Press any key to continue

ملاحظة هامة

يمكن استعمال وسيط مرجعي (العنونة) & في توابع التحميل الزائد السابقة ، مثلاً عملية الجمع :

// overload - relative to point class .

```

point point :: operator + ( point &ob2 ) {
    point temp;
    temp.x = x + ob2.x;
}

```

```
temp.y = y + ob2.y ;
return temp ;
}
```

ويكمن السبب في استخدام وسيط مرجعي & (العنونة) في تابع التحميل الزائد للعوامل الثنائية في الكفاءة . لأن تمرير الأغراض كوسائط يستهلك زمناً كبيراً، أما تمرير العنوان فيكون سريعاً . إضافة إلى ذلك تجنب المشاكل الناجمة عن تدمير نسخة المعامل الممررة . لأنه عندما يمرر متحول وسيط بالقيمة تصنع منه نسخة ، وإذا كان للغرض مدمر فإنه يُستدعى عند تدمير النسخة لدى العودة من التابع .

تمرين (1-3)

اكتب برنامجاً بلغة C++ للصف point وأجر عليه التحميل الزائد للعاملين الضرب والقسمة (* و /) .

تمرين (2-3)

لماذا يعتبر استخدام التحميل الزائد التالي غير مناسب ؟

```
point point :: operator % ( point ob )
{ double x ;
  cout << "Enter a number : " ;
```

```

cin >> x ;
cout << " root of " << x << " is " ;
cout << sqrt( x ) ;
}

```

٣-٤ - التحميل الزائد لعوامل المقارنة والمنطقية

Overloading The Relational And Logical Operators

يمكن إجراء التحميل الزائد لعوامل المقارنة والعوامل المنطقية . عند التحميل الزائد للعوامل المنطقية فإنها لن تُعيد غرضاً من نفس الصف التي عرفت له بل ستعيد قيمة صحيحة تشير إما إلى صح True أو إلى خطأ False .

في بعض المترجمات الحديثة ، إن التحميل الزائد للعوامل المنطقية وعوامل المقارنة تعيد قيمة من النوع bool الذي يعرف قيمتين فقط True و False وتحول هذه القيم تلقائياً إلى قيم صحيحة 0 أو 1 .

مثال (3-5)

يبين المثال التالي التحميل الزائد لعوامل الجمع والمساواة والعمليات المنطقية AND (+ و == و &&) في الصف point .

```

// EXAMPLE - Overloading The
//Relational And Logical Operator
#include <iostream.h>
class point {
private :

```



```

    int x, y; //point value

public :

    point( ) { x = 0 ; y = 0 ; }
    point ( int i , int j ) { x = i ; y = j ;}
    void get_xy( int &i , int &j ) { i = x ; j = y ; }

    point operator + ( point ob2) ;
    int operator == ( point ob2) ;
    int operator && ( point ob2) ;
}; // End class

```

//overload + relative to point class .

```

point point:: operator + ( point ob2) {
    point temp ;
    temp .x = x + ob2.x ;
    temp .y = y + ob2.y ;
    return temp ;
}

```

//overload the == operator for point class .

```

int point :: operator == ( point ob2)
{
    return x == ob2.x && y == ob2.y ;
}

```

//overload the && operator for point class .

```

int point :: operator && ( point ob2)
{
    return ( x && ob2.x) && ( y && ob2.y) ;
}

void main ( )
{
    point  obj1( 10,10) ,obj2(5,3),obj3(10,10), obj4(0,-2),obj5;
    int  x , y ;
    obj5=obj1 + obj2 ; // add tow objects , this call
                     // operator+ ( )
    obj5.get_xy(x,y) ;
    cout<<" (obj1+obj2) x: "<< x<<"  y: "<<y<<endl;

    if ( obj1==obj2) cout<<" obj1  same as obj2 \n" ;
    else  cout << "obj1 and obj2 is differ \n " ;

    if ( obj1==obj3) cout<<" obj1 same as obj3 \n" ;
    else cout << "obj2 and obj3 is differ \n " ;

    if ( obj1 && obj2) cout<<" obj1 && obj2 is true \n" ;
    else cout << "obj2 && obj3 is false \n " ;

    if ( obj1 && obj4) cout<<" obj1 && obj4 is true \n" ;

```

```
else cout << "obj1 and obj4 is false \n " ;
```

```
} // End main
```

RESULT

(obj1+obj2) x: 15 y: 13

obj1 and obj2 is differ

obj1 same as obj3

obj1 && obj2 is true

obj1 and obj4 is false

Press any key to continue

مثال (6-3)

اكتب برنامجاً بلغة C++ يعرف صفراً اسمه Aritharray ، يحتوي مصفوفة a كأعضاء خاصة من نمط int ، ويحتوي على بيانٍ افتراضي لإسناد القيمة صفر إلى عناصر المصفوفة a[10]، وتابع تعديل معطيات set وسيطه مصفوفة وعلى تابع طباعة show لطباعة عناصر المصفوفة ، والمطلوب :

- ❖ أجر عملية التحميل الزائد للعملية الثنائية (+) كتابع عضو عام
- ❖ أجر عملية التحميل الزائد للعملية الثنائية (-) كتابع عضو عام
- ❖ أجر عملية التحميل الزائد للعملية الثنائية (==) كتابع عضو عام

أنشئ في التابع الرئيسي main أغراضاً واستدع التابع اللازمة لطباعة جمع
غرضين وطرحهما ومقارنتهما وطباعة النتائج .

```
#include <iostream.h>
class Aritharray {
    int a[10];
public:
    Aritharray( );
    void set(int n[10]);
    void show( );
    Aritharray operator + ( Aritharray ob2 );
    Aritharray operator - ( Aritharray ob2);
    int operator == ( Aritharray ob2);
}; //end class

Aritharray::Aritharray( ) {
    int i;
    for(i=0; i<10; i++) a[i] = 0;
}
void Aritharray::set( int *n) {
    int i;
    for(i=0; i<10; i++) a[i] = n[i];
}
void Aritharray::show( ){
```

```

    int i;
    for(i=0; i<10; i++)
        cout << a[i] << ' ';
    cout << "\n";
}

Aritharray Aritharray::operator+(Aritharray ob2) {
    int i;
    Aritharray temp;
    for(i=0; i<10; i++)
        temp.a[i] = a[i] + ob2.a[i];

    return temp;
}

Aritharray Aritharray::operator - (Aritharray ob2) {
    int i;
    Aritharray temp;
    for(i=0; i<10; i++)
        temp.a[i] = a[i] - ob2.a[i];
    return temp;
}

int Aritharray:: operator == (Aritharray ob2) {
    int i;
    for(i=0; i<10; i++)

```



```

        if(a[i]!=ob2.a[i]) return 0;

    return 1;
}

int main() {
    Aritharray  ob1, ob2, ob3;

    int i[10] = { 10, 12, 53, 14, 5, 16, 70, 28, 19, 100 };
    int j[10] = { 10, 1, 53, 4, 5, 16, 7, 28, 19, 10 };

    ob1.set(i);
    ob2.set(j);
    ob3 = ob1 + ob2;
    ob3.show( );
    ob3 = ob3 - ob1;
    ob3.show( );
    if(ob1==ob2) cout << "ob1 equals ob2\n";
    else cout << "ob1 does not equal ob2\n";

    if(ob2==ob3) cout << "ob2 equals ob3\n";
    else cout << "ob2 does not equal ob3\n";

    return 0;
} //end main

```

RESULT

10 13 106 18 10 32 77 56 38 110

10 1 53 4 5 16 7 28 19 10

ob1 does not equal ob2

ob2 equals ob3

Press any key to continue

٣-٥- التحميل الزائد للعوامل الأحادية

Overloading a Unary Operators

يشبه التحميل الزائد للعوامل الأحادية التحميل الزائد للعوامل الثنائية ، ويتعامل مع وسيط واحد فقط . عندما يتم التحميل الزائد للعامل الأحادي كتابع عضو في صف لا يكون للتابع وسائط لأنه يوجد معامل واحد وهو الذي يولد الاستدعاء لتابع العامل ولا حاجة إلى وسيط آخر .

٣-٥-١- التحميل الزائد للمعامل (++) و (--)

يقوم عامل الزيادة (++) بزيادة المعامل بمقدار واحد ، فهو يعدل الغرض الذي يعمل عليه ، و تابع الزيادة يعيد الغرض بعد زيادة قيمته، و كذلك الأمر بالنسبة للمعامل (--) .

مثال (7-3)

يبين البرنامج التالي التحميل الزائد لمعامل الزيادة (++) ولمعامل النقصان (--) نسبة إلى الصف point .

```
//EXAMPLE - Overloading unary
// Operator ( ++ , -- )
#include <iostream.h>
class point{
private :
    int x, y ; //point value
public :
    point( ) { x=0;y=0;}
    point (int i , int j ) { x = i ; y = j ; }
    void get_xy ( int &i , int &j ) { i = x ; j = y ; }
    point operator ++ ( ) ; // Increment
    point operator -- ( ) ; // Decrement
} ; // End class
//overload ++ for point class .
point point :: operator ++ ( )
{
    x++ ;
    y++ ;
    return *this ; }
```

//overload -- for point class .

Point point :: operator -- ()

```
{ x -- ;  
  y -- ;  
  return *this ;  
}
```

void main ()

```
{   point obj1(10,9) ,obj2(5,3) ;  
    int x,y ;  
    ++obj1 ; // Increment an object  
    obj1.get_xy(x,y) ;  
    cout<<" (obj1) x: "<< x<<" y : "<<y<<endl;  
    --obj2 ; // Decrement an objects  
    obj2.get_xy(x,y) ;  
    cout<<" (obj2) x: "<< x<<" y : "<<y<<endl;  
} // End Class
```

RESULT

(obj1) x: 11 y: 10

(obj2) x: 4 y: 2

Press any key to continue

إن عامل الزيادة أو النقصان يمكن أن يكون من اليمين (الخلف) أو اليسار (الأمام) . وإن المترجمات الحديثة تفرق بين الشكلين . لتحقيق هذا الإنجاز فقد عرفت نسختين من التابع () ++operator وهي كما يلي :

✕ من اليسار :

```
point :: point operator ++ ( ) ;
```

✕ من اليمين :

```
point :: point operator ++ (int notused) ;
```

٣-٥-٢- التحميل الزائد لعملية إشارة السالب (-)

تُعد إشارة السالب عاملاً أحادياً لأنه يغير قيمة الغرض من قيمة موجبة إلى قيمة سالبة أو بالعكس ، وتبقى القيمة المطلقة ثابتة دون تغيير . لنأخذ المثال التالي ونقوم بالتحميل الزائد للعاملين : الطرح " - " وإشارة السالب " - " .

// EXAMPLE

مثال (3-8)

// Overloading a Unary (-)

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

```
class point {
```

```
private :
```

```
int x, y ; //coordinate value
```

```
public :
```

```
point( ) { x=0;y=0;}
```

```
point(int i,int j) { x = i ; y = j ; }
```

```
void get_xy( int &i , int &j ) { i = x ; j = y ; }
```



```

    point operator - ( point ob2 ) ;
    point operator- ( ) ;
}; // End Class

// overload - relative to point class .
point point :: operator - ( point ob2)
{
    point temp ;
    temp .x = x - ob2.x ;
    temp .y = y - ob2.y ;
    return temp ;
}

// overload unary - for point class .
Point point :: operator - ( )
{
    x = -x ;
    y = -y ;
    return *this ;
}

void main ( ) {
    point obj1( 10,8) , obj2(5,3), obj3 ;
    int x , y ;
    obj1 = - obj1 ;    // negation
    obj1.get_xy( x,y ) ;
    cout<<" (obj1) x: "<< x<<" y : "<<y<<endl;

```

```

obj3 = obj1 - obj2 ; // subtraction
obj3.get_xy(x,y) ;
cout<<" (obj3) x: "<< x<<" y : "<<y<<endl;

} // End main

```

RESULT

(obj1) x: -10 y: -8

(obj3) x: -15 y: -11

Press any key to continue

٣-٦ - استخدام التوابع الصديقة

Using Friend Operator Functions

يمكن التحميل الزائد للعوامل في صف باستخدام تابع صديق بدلاً من تابع عضو عام في صف ، كما مر معنا سابقاً . لا تملك التوابع الصديقة كما هو معروف المؤشر **this** ، هذا يعني أنها تستقبل كلا الوسيطين (ob1,ob2) بشكل صريح في حالة العوامل الثنائية ، أما في حالة العوامل الأحادية فتستقبل الوسيط الوحيد بشكل صريح . لا يوجد سبب يدعو لاستخدام التوابع الصديقة بدلاً من التوابع العامة إلا في حالات استثنائية . ويمكن إجراء التحميل الزائد لجميع العوامل الثنائية والأحادية ما عدا عملية الإسناد .

٣-٦-١ - التحميل الزائد للعوامل الثنائية كتابع صديق

يجب التصريح عن المعاملين صراحة ضمن التابع في التحميل الزائد للعوامل الثنائية كتابع صديق ، أي التصريح عن المعامل الأيسر والأيمن كوسطاء للتابع . ويوضح المثال (3-9) التالي التحميل الزائد للعملية الثنائية (+) بالنسبة إلى الصف coord ، باستخدام التابع () + operator كتابع صديق . وبحيث يكون في الحالة الأولى الوسيط الأول غرض ، والوسيط الثاني غرض ، أما الحالة الثانية فيكون الوسيط الأول غرض ، و الوسيط الثاني عدد ثابت int .

مثال (3-9)

//EXAMPLE

// Overloading Binary Operator (+) Using a friend .

#include <iostream.h>

class coord {

private :

int x , y ; //coordinate value

public :

coord() { x=0 ;y=0 ; }

coord(int i, int j) { x = i;y = j ;}

void get_xy() { cout << " x : " << x
<< " y : " << y << endl ;

}

friend coord operator + (coord ob1 , coord ob2) ;

```
        friend coord operator + ( coord ob1 , int i );  
}; // End class
```

```
//overload + using friend .
```

```
coord operator + ( coord ob1 , coord ob2 )  
{   coord temp ;  
    temp .x = ob1.x + ob2.x ;  
    temp .y = ob1.y + ob2.y ;  
    return temp ;  
}
```

```
//overload + using friend .
```

```
coord operator + ( coord ob1 , int i )  
{   coord temp ;  
    temp .x = ob1.x + i ;  
    temp .y = ob1.y + i ;  
    return temp ;  
}
```

```
void main ( ) {  
    coord obj1(10,10) , obj2(5,3) ,obj3 ;  
    obj3=obj1 + obj2 ; // add tow objects  
    cout<<" (obj1+obj2) ";  
    obj3.get_xy( ) ;  
}
```

```

obj3=obj1 + 50 ; // add tow (objects + int)
cout<<" (obj1+50) ";
obj3.get_xy( ) ;
} // End main

```

RESULT

(obj1+obj2) x: 15 y: 13

(obj1+50) x: 60 y: 60

Press any key to continue

نلاحظ أن المعامل الأيسر قد مرر إلى الوسيط الأول ، والمعامل الأيمن قد مرر إلى الوسيط الثاني . كنا وجدنا سابقاً في التحميل الزائد للتتابع الأعضاء ، عندما يكون المعامل الثاني قيمة ثابتة :

$obj1 = obj2 + 20 ;$

$obj1 = 20 + obj2$

نلاحظ أن العبارة الأولى صحيحة أما الثانية فخاطئة ، لأن الغرض في التابع العضو يكون على يسار المعامل ، والثابت على يمينه ما قد يؤدي إلى أخطاء . ويمكن حل المشكلة باستخدام تابع صديق في التحميل الزائد ، وتعريف كلتا الحالتين . كما هو موضح في المثال (4 - 10) التالي .

مثال (10-3)

```
// EXAMPLE - Overloading Binary
// Operator ( + ) Using a friend .
#include <iostream.h>
class coord {
private :
    int x ,y ; //coordinate value
public :
    coord( ) { x = 0 ; y = 0 ; }
    coord(int i,int j ) { x = i ; y = j ; }
    void get_xy( ) {
        cout << " x : " << x << " y : " << y << endl ; }
    friend coord operator + ( int i , coord ob2 ) ;
    friend coord operator + ( coord ob1 , int i ) ;
}; // End class

// overload + using friend .
coord operator + ( int i , coord ob2 )
{ coord temp ;
  temp .x = i + ob2.x ;
  temp .y = i + ob2.y ;
  return temp ;
}
```

//overload + using friend .

```
coord operator + ( coord ob1 , int i )  
{ coord temp ;  
  temp .x = ob1.x + i ;  
  temp .y = ob1.y + i ;  
  return temp ;  
}  
  
void main ( ) {  
  coord obj1(10,10) ,obj2(5,3),obj3;  
  obj3 = obj1 + 100 ; // add object + int  
  cout <<" (obj1+100) " ;  
  obj3.get_xy ( ) ;  
  obj3 = 50 + obj1 ; // add objects + int  
  cout <<" (50 + obj1) " ;  
  obj3.get_xy ( ) ;  
} // End Class
```

RESULT

(obj1+100) x : 110 y : 110

(50+obj1) x : 60 y : 60

Press any key to continue

ملاحظة ٢ :

التحميل الزائد لعملية الضرب كتابع صديق هي كما يلي :

```
friend coord operator * ( coord ob1 , coord ob2 ) ;
```

```
friend coord operator * ( coord ob1 , int i ) ;
```

أما تعريف جسم التابع يكون كما يلي :

```
//overload " * " using friend .
```

```
coord operator * ( coord ob1 , coord ob2 )  
{  
    coord temp ;  
    temp.x = ob1.x * ob2.x ;  
    temp.y = ob1.y * ob2.y ;  
    return temp ;  
}
```

٣-٦-٢ - التحميل الزائد للعوامل الأحادية كتابع صديق

بما أن التابع الصديق لا يملك المؤشر `this` لذا يجب تمرير معامل الزيادة (`++`) أو النقصان (`--`) كوسيط مرجعي (التمرير بالعنوان) لأنه في حالة تمريره بالقيمة لا توجد طريقة للتأثير على الغرض ، أما عند تمريره بالعنوان فإن التعديلات التي يجريها التابع الصديق على الغرض الممرر تنعكس على الغرض المستدعي . و يبين المثال (3-11) التالي كيفية التحميل الزائد للعامل (`++`) باستخدام تابع صديق .

مثال (11-3)

```
// EXAMPLE -- Overloading Unary
//      Operator( ++ ) Using a friend .
#include <iostream.h>
class coord {
private :
    int x , y ; //coordinate value
public :
    coord( ) { x = 0 ; y = 0 ; }
    coord( int i , int j ) {
        x = i ;
        y = j ;
    }

    void get_xy( ) {
        cout << " x : " << x << " y : " << y << endl ;
    }

    friend coord operator ++ ( coord &obl ) ;
}; // End Class

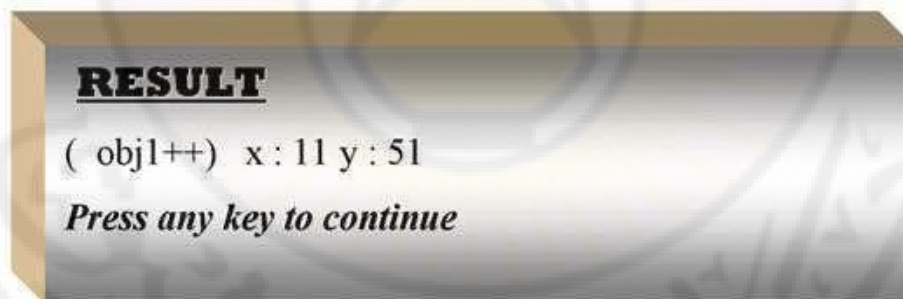
// overload + using friend .
coord operator ++( coord &obl )
{
    obl.x++ ;
```

```

    obj1.y++ ;
    return obj1 ; // Return object, generation the call
}

void main ( )
{
    coord obj1(10,50) ;
    ++ obj1 ; // obj1 passed by Reference
    cout<<" (obj1++) " ;
    obj1.get_xy() ;
} //End main

```



✕ يمكن التمييز بين الزيادة من اليمين والزيادة من اليسار وذلك كما مر معنا سابقاً في التحميل الزائد للعوامل الأحادية كتابع عضو ، وهي كما يلي :

• من اليسار : ++ob

```
coord operator ++ ( coord &ob ) ;
```


• من اليمين : ob++

coord operator ++ (coord &ob , int notused);

□ لا يمكن إجراء عملية التحميل الزائد لعملية الإسناد (=) كتابع صديق ، وإنما فقط كتابع عضو عام فقط .

مثال (3-12)

اكتب برنامجاً بلغة C++ يعرف صفّاً اسمه Aritharray يحتوي مصفوفة a كأعضاء خاصة من نمط int ، ويحتوي على بان افتراض لإسناد القيمة صفر إلى عناصر المصفوفة a[10]، وتابع تعديل معطيات set وسيطه مصفوفة وعلى تابع طباعة show لطباعة عناصر المصفوفة ، والمطلوب :

- ❖ أجر عملية التحميل الزائد للعملية الثنائية (+) كتابع صديق .
- ❖ أجر عملية التحميل الزائد للعملية الثنائية (-) كتابع صديق .
- ❖ أجر عملية التحميل الزائد للعملية الثنائية (==) كتابع صديق .

أنشئ في التابع الرئيسي main أغراضاً واستدع التوابع اللازمة لجمع غرضين وطرحهما ومقارنتهما وطباعة النتائج .

ملاحظة: هذا المثال هو إعادة للمثال (2-7) ولكن أجري التحميل الزائد كتوابع صديقة للصف .

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

```
class Aritharray {
```

```
    int a[10];
```

```
public:
```

```
    Aritharray( );
```

```

void set( int n[10]);
void show( );
friend Aritharray operator+(Aritharray ob1, Aritharray ob2);
friend Aritharray operator-(Aritharray ob1, Aritharray ob2);
friend int operator == (Aritharray ob1, Aritharray ob2);
}; //end class

Aritharray::Aritharray( ) {
    int i;
    for(i=0; i<10; i++) a[i] = 0;
}

void Aritharray::set(int *n) {
    int i;
    for(i=0; i<10; i++) a[i] = n[i];
}

void Aritharray::show( ) {
    int i;
    for(i=0; i<10; i++)
        cout << a[i] << ' ';
    cout << "\n";
}

```

```
Aritharray operator+(Aritharray ob1, Aritharray ob2) {
```

```
    int i;
```

```
    Aritharray temp;
```

```
    for(i=0; i<10; i++)
```

```
        temp.a[i] = ob1.a[i] + ob2.a[i];
```

```
    return temp;
```

```
}
```

```
Aritharray operator-(Aritharray ob1, Aritharray ob2)
```

```
{
```

```
    int i;
```

```
    Aritharray temp;
```

```
    for(i=0; i<10; i++)
```

```
        temp.a[i] = ob1.a[i] - ob2.a[i];
```

```
    return temp;
```

```
}
```

```
int operator == (Aritharray ob1, Aritharray ob2) {
```

```
    int i;
```

```
    for(i=0; i<10; i++)
```

```
        if(ob1.a[i]!=ob2.a[i]) return 0;
```

```
    return 1;
```

```
}
```

```

int main( )
{
    Aritharray ob1, ob2, ob3;
    int i[10] = { 10, 12, 53, 14, 5, 16, 70, 28, 19, 100 };
    int j[10] = { 10, 1, 53, 4, 5, 16, 7, 28, 19, 10 };

    ob1.set( i );
    ob2.set( j );

    ob3 = ob1 + ob2;
    ob3.show( );
    ob3 = ob3 - ob1;
    ob3.show( );

    if(ob1==ob2) cout << "ob1 equals ob2\n";
    else cout << "ob1 does not equal ob2\n";

    if(ob2==ob3) cout << "ob2 equals ob3\n";
    else cout << "ob2 does not equal ob3\n";

    return 0;

} //End main

```

RESULT

10 13 106 18 10 32 77 56 38 110

10 1 53 4 5 16 7 28 19 10

ob1 does not equal ob2

ob2 equals ob3

Press any key to continue

مثال (3-13)

اكتب برنامجاً بلغة C++ يعرف صفّاً اسمه Aritharray يحتوي مصفوفة a كأعضاء خاصة من نمط int ، ويحتوي على بان افتراض لإسناد القيمة صفر إلى عناصر المصفوفة a[10]، وتابع تعديل معطيات set وسيطه مصفوفة وعلى تابع طباعة show لطباعة عناصر المصفوفة ، والمطلوب :

- ❖ أجر عملية التحميل الزائد للعملية الأحادية (++) كتابع عضو عام .
- ❖ أجر عملية التحميل الزائد للعملية الأحادية (--) كتابع صديق .

أنشئ في التابع الرئيسي main أغراضاً واستدع التتابع اللازمة لإجراء عملية الزيادة والنقصان لغرضين وطباعة النتائج .

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

```
class Aritharray {
```

```
int a[10];
```


public:

Aritharray();

void set(int n[10]);

void show();

Aritharray **operator++**();

friend Aritharray **operator--**(Aritharray &ob);

};//end class

Aritharray::Aritharray() {

int i;

for(i=0; i<10; i++) a[i] = 0;

}

void Aritharray::set(int *n) {

int i;

for(i=0; i<10; i++) a[i] = n[i];

}

void Aritharray::show() {

int i;

for(i = 0; i < 10 ; i++)

cout << a[i] << ' ';

cout << "\n";

}

// Overload unary op using member function.

```
Aritharray Aritharray :: operator++( ) {  
    int i;  
    for(i=0; i<10; i++)  
        a[i]++;  
    return *this;  
}
```

// Use a friend.

```
Aritharray operator--(Aritharray &ob) {  
    int i;  
    for(i=0; i<10; i++)  
        ob.a[i]--;  
    return ob;  
}
```

```
int main( ) {  
    Aritharray ob1, ob2, ob3;  
    int i[10] = { 60, 20, 80, 40, 70, 60, 90, 10, 30, 100 };  
    ob1.set( i );  
    ob2.set( i );  
    ob3 = ++ob1;  
    ob3.show( );  
}
```

```
ob3 = --ob1;  
ob3.show( );  
return 0;  
}//End main
```

RESULT

61 21 81 41 71 61 91 11 31 101

60 20 80 40 70 60 90 10 30 100

Press any key to continue



تمارين عامة

١. أجرِ عملية التحميل الزائد للعوامل الثنائية (- و / و *) بالنسبة للصف coord باستخدام التوابع الصديقة .

=====

٢. أجرِ عملية التحميل الزائد في الصف coord ، بحيث تستطيع الأغراض إنجاز عمليات ضرب بقيمة صحيحة بكل إحداثياته مع السماح بالحالتين التاليتين ob * int ، int * ob .

=====

٣. باستخدام التابع الصديق ، بين كيف تتم عملية التحميل الزائد للعملية الأحادية (--) بالنسبة للصف coord بشكلي المعامل كليهما من اليمين واليسار .

=====

٤. أجرِ عملية التحميل الزائد لعوامل الإزاحة << و >> بالنسبة إلى الصف point بحيث نستخدم أنواع العمليات التالية :

ob << integer ;

ob >> integer;

وتأكد من أن هذه العوامل تزيح القيم x و y بالمقدار المحدد .

=====

٥. لدينا النص التالي :

```
class three_d {  
private :
```

```

    int x , y , z ;
public :
    three_d ( int i , int j ,int k ) {
        x = i ; y = j ; z = k ;
    }
    three_d ( ) {
        x = 0 ; y = 0 ; z = 0 ;
    }
    void get ( int &i , int &j , int &k ) {
        i = x ; j = y ; k = z ;
    }
}; // End Class three_d

```

- أ- أجرِ التحميل الزائد للعوامل (+ ، - ، ++ ، --) لهذا الصف ، استخدم شكل واحد للزيادة والنقصان .
- ب- أعد كتابة الصف السابق باستخدام التمرير بالعنوان بدلاً من التمرير بالقيمة .
- ت- ما الاختلاف بين التوابع الصديقة والتوابع الأعضاء ؟

=====

٦. أنشئ صفاً بلغة C++ اسمه strtype والذي يقوم بالتحميل الزائد لما يلي :

- أ- جمع السلاسل الرمزية باستخدام العامل (+) .
- ب- إسناد سلسلة باستخدام العامل (=) .
- ت- مقارنة السلاسل باستخدام العوامل (> ، < ، ==) .
- يحتوي الصف على باني افتراضي وباني عادي وتابع طباعة .

=====

٣-٧- التحميل الزائد لمجاري الدخل والخرج

تتضمن معظم برامج لغة C++ الملف الرأسي <iostream> الذي يتضمن المعلومات الأساسية التي تتطلبها عمليات الدخل/الخرج . نجد ضمن الملف الرأسي <iostream> الأغراض cin و cout و cerr و clog التي ترتبط بمجرى الدخل القياسي ومجرى الخرج القياسي ومجرى الخطأ القياسي الذي يحتفظ فقط بالخطأ الأخير .

وتساعد عملية التحميل الزائد على إنجاز عمليات الدخل/الخرج بشكل مناسب ، لذلك جرى تحميل عملية الإزاحة نحو اليسار (<<) بشكل زائد بحيث أصبحت قادرة على الدلالة على مجرى الخرج ، وهي تُعرف باسم عملية الإدخال ضمن مجرى ، كما تم تحميل عملية الإزاحة نحو اليمين (>>) بشكل زائد بحيث أصبحت قادرة على الدلالة على مجرى الدخل .

٣-٧-١- التحميل الزائد لعملية مجاري الخرج

يمكن إجراء عملية التحميل الزائد لمجاري الدخل والخرج . وتدعى عملية الإخراج في C++ بالإدراج Insertion ويدعى العامل << بعامل الإدراج Insertion Operator أو Insertion . تأتي التسمية من أن عوامل الخرج تقوم بإدراج المعلومات إلى المجرى stream . الشكل العام هو :

```
ostream &operator<<( ostream &stream , class_name ob ) {  
    // Body Of Inserter :  
    return stream ;  
}
```

- ✗ يمثل الوسيط الأول مرجعاً للغرض ostream و هذا يعني أن stream يجب أن تكون مجرى خرج .
- ✗ أما الوسيط الثاني فيستلم الغرض الذي سيصبح خرجاً . يمكن أن يكون الوسيط الثاني مرجعاً (أي تمرير بالعنوان) .

إن تابع الإدراج يعيد مرجعاً (عنوان) للمجرى stream وهو من النوع ostream ، وهذا ضروري عند استعمال التحميل الزائد للعامل << في سلسلة من تعابير الدخل /الخرج مثل :

cout << ob1 << ob2 << ob3 ;

ويفضل أن يقتصر تابع الإدراج على عمليات إخراج المعلومات .

وعندما يكون تابع ما عضو في صف ما ؛ فإن المعامل الأيسر والذي يمرر بشكل ضمني المؤشر this هو الذي يولد الاستدعاء إلى هذا التابع ، هذا يعني أن المعامل الأيسر هو غرض من ذلك الصف ، وبالتالي عند إنشاء تابع إدراج فإن المعامل الأيسر هو مجرى ، والمعامل الأيمن هو الغرض الذي نريد إخراجته ، لذلك لا يمكن أن يكون تابع الإدراج تابع عضو عام . المثال (3- 14) التالي يوضح تابع الإدراج لمجرى الخرج .

مثال (3- 14)

//Use a friend inserter for objects of type coord.

#include <iostream.h>

class coord {

private :

int x, y;

public:

```
coord( ) { x = 0; y = 0; }
```

```
coord(int i, int j) { x = i; y = j; }
```

```
friend ostream &operator <<(ostream &stream, coord ob);
```

```
}; // End Class
```

```
ostream &operator<<(ostream &stream, coord ob)
```

```
{
```

```
stream << ob.x << ", " << ob.y << "\n";
```

```
return stream;
```

```
}
```

```
int main ( )
```

```
{
```

```
coord a(15, 10), b(20, 30);
```

```
cout << a << b;
```

```
return 0;
```

```
} // End main
```

Result

15 , 10

20 , 30

Press any key to continue

ترسل عبارة الدخل/الخرج I/O الموجودة داخل تابع الإدراج في هذه الحالة القيم x و y إلى المجرى stream الممرر إلى التابع . لمزيد من التوضيح نعيد كتابة البرنامج السابق بحيث لا يكون تابع الإدراج تابعاً صديقاً للصف coord و بما أن تابع الإدراج لا يستطيع الوصول إلى الأعضاء الخاصة للصف coord فقد كان المتغيران x و y عامين . والمثال (3- 15) يوضح ذلك .

مثال (3- 15)

```
//Create an inserter for objects of type coord, using a non-
//friend inserter. */
#include <iostream.h>
class coord {
Privat :
public:
    int x , y; // must be public
    coord( ) { x = 0; y = 0; }
    coord ( int i, int j) {
        x = i ;
        y = j;
    }
}; // End Class

//An inserter for the coord class.
ostream &operator<<(ostream &stream, coord ob) {
```

```

stream << ob.x << ", " << ob.y << "\n";
return stream;
}

```

```

int main( )
{
    coord a(10, 15), b(20, 30);
    cout << a << b;
    return 0;
} //End main

```



لا يقتصر عمل تابع الإدراج على إظهار المعلومات النصية ، بل يمكن لتابع الإدراج إنجاز أي عملية ضرورية لإخراج المعلومات بأي شكل مطلوب لأداة أو حالة معينة . يمكن مثلاً إنشاء تابع الإدراج لإرسال المعلومات إلى الرسمة *plotter* . وسيحتاج تابع الإدراج في هذه الحالة إلى إرسال الشيفرة الملائمة لعمل الرسمة إضافة إلى المعلومات المراد إخراجها . ويمكن تصور ذلك من خلال البرنامج (3- 16) التالي الذي ينشئ صفّاً يدعى *triangle* ، ويخزن هذا الصف قاعدة وارتفاع المثلث ، ويقوم تابع الإدراج بإظهار المثلث على الشاشة .

مثال (16-3)

```
//This program draws right triangles
#include <iostream.h>
class triangle {
Private :
    int height, base;
public:
    triangle( int h, int b) {
        height = h;
        base = b;
    }
    friend ostream &operator<<( ostream &stream, triangle ob );
}; // End class

// Draw a triangle.
ostream &operator<<(ostream &stream, triangle ob)
{
    int i, j, h, k;
    i = j = ob.base-1;
    for (h = ob.height-1; h; h--) {
        for( k = i ; k ; k--)
            stream << ' ';
        stream << '*';
    }
}
```

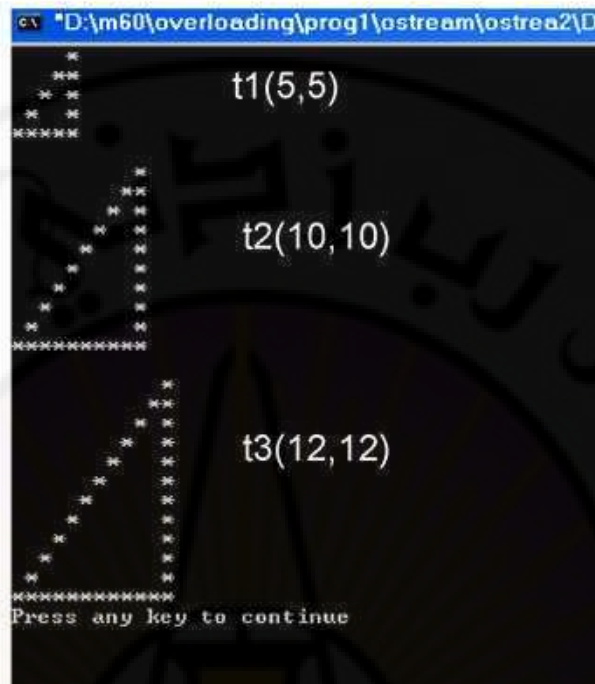


```

    if( j != i) {
        for( k = j-i-1; k; k--)
            stream << ' ';
        stream << '*';
    }
    i--;
    stream << '\n';
}
for( k = 0; k< ob.base ; k++)
    stream << '*';
stream << '\n';
return stream;
}

int main ( )
{
    triangle t1(5, 5), t2(10, 10), t3(12, 12);
    cout << t1;
    cout << endl << t2 << endl << t3;
    return 0;
} // End main

```



٣-٧-٢ - التحميل الزائد لعملية مجاري الدخل

نستطيع إجراء التحميل الزائد لمجری الدخل >> ، كما قمنا بإجراء التحميل الزائد لمجری الخرج . لأن إدخال المعلومات من المجری stream تزيل أو تستخلص المعطيات منه .

الشكل العام هو :

```
istream &operator >> ( istream &stream , class_nam & ob )
{
    // Body Of function :
    return stream ;
}
```

ويعيد التابع مرجعاً (عنوان) إلى المجرى `istream` والذي يمثل مجرى دخل . أي أن الوسيط الأول يجب أن يكون مرجعاً لمجرى الدخل ، أما الوسيط الثاني ويمثل مرجعاً للغرض الذي سيتسلم الدخل . ولا يمكن أن يكون تابع مجرى الدخل عضواً وإنما صديقاً . وعند إنشاء تابع إدراج فإن المعامل الأيسر هو مجرى، والمعامل الأيمن هو الغرض الذي نريد إدخاله ، لذلك لا يمكن أن يكون تابع الإدراج تابع عضو عام . البرنامج في المثال (3- 17) التالي يضيف تابع مجرى الدخل إلى الصف `coord` .

مثال (3- 17)

//Add a friend extractor for objects of type coord.

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

```
class coord {
```

```
Privat :
```

```
int x, y;
```

```
public:
```

```
coord( ) {
```

```
    x = 0; y = 0;
```

```
}
```

```
coord( int i, int j ) {
```

```
    x = i;
```

```
    y = j;
```

```
}
```

```
friend ostream &operator<<(ostream &stream, coord ob);  
friend istream &operator>>(istream &stream, coord &ob);  
}; // End class
```

```
ostream &operator<<(ostream &stream, coord ob)
```

```
{  
    stream << ob.x << ", " << ob.y << "\n";  
    return stream;  
}
```

```
istream &operator>>(istream &stream, coord &ob)
```

```
{  
    cout << "Enter coordinates: ";  
    stream >> ob.x >> ob.y;  
    return stream;  
}
```

```
int main( ) {
```

```
    coord  a(10, 10), b(20, 50);
```

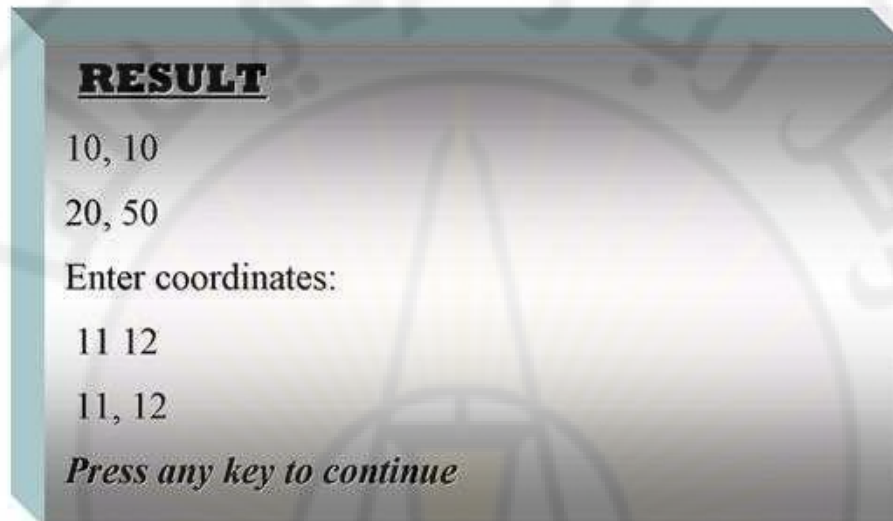
```
    cout << a << b;
```

```
    cin >> a;
```

```
    cout << a<<endl;
```

```
    cin >> b;
```

```
cout << b<<endl;
return 0;
} // End main
```



نلاحظ بأن تابع مجرى الدخل يطلب من المستخدم إدخال المعطيات ، ومع ذلك ليس ضرورياً . البرنامج في المثال (3- 18) التالي يبين أن الصف inventory يخزن اسم المادة item وعدد قطعها onhand وتكلفتها cost ، ويتضمن البرنامج تابع مجرى الإدخال istream وتابع مجرى الإخراج ostream .

مثال (3- 18)

```
#include <iostream.h>
#include <string.h>
class inventory {
private :
```

```

char item[ 40 ]; // name of item
int onhand ; // number on hand
double cost ; // cost of item
public:
    inventory(char *i, int o, double c)
    { strcpy(item, i);
      onhand = o;
      cost = c;
    }
    friend ostream &operator<<(ostream &stream, inventory ob);
    friend istream &operator>>(istream &stream, inventory&ob);
}; // End class

ostream &operator<<(ostream &stream, inventory ob)
{
    stream << ob.item << ": " << ob.onhand;
    stream << " on hand at $" << ob.cost << "\n";
    return stream;
}

istream &operator>>(istream &stream, inventory &ob)
{
    cout << "Enter item name: ";
    stream >> ob.item;
    cout << "Enter number on hand: ";

```



```

    stream >> ob.onhand;
    cout << "Enter cost: ";
    stream >> ob.cost;
    return stream;
}

int main( )
{
    inventory ob(" Hammer ", 4, 12.55 );
    cout << ob;
    cin >> ob;
    cout << ob;
    return 0;
} // End main

```

RESULT

Hammer : 4 on hand at \$12.55

Enter item name: ball

Enter number on hand: 10

Enter cost: 400.5

ball: 10 on hand at \$400.5

Press any key to continue

٣-٧-٣ - قاعدة مجرى dec,oct,hex, showbase

يتوافر في لغة C++ مجرى dec, oct, hex لتحديد نظام العد الواجب استخدامه مع الأعداد الصحيحة (نظام العد العشري dec و نظام العد الثماني oct و نظام العد السداسي عشر hex) . وتتعامل مجاري الإدخال مع الأعداد الصحيحة على أنها تابعة لنظام العد العشري بشكل افتراضي إذا لم يتم تحديد أية خانة من الخانات السابقة .

أما بالنسبة لمجاري الخرج فتتعامل مع المعطيات حسب الصيغة التي تُمرر فيها إلى المجرى، ويتم التعامل مع الأعداد التي تبدأ بصفر 0 على أنها تابعة لنظام العد الثماني ، أما الأعداد التي تبدأ بـ 0x على أنها تابعة لنظام العد السداسي عشر .

يُستخدم مؤشر الصيغة showbase لإجبار البرنامج على إظهار قاعدة الرقم أثناء الخرج ، حيث تتم طباعة الأعداد العشرية بشكل طبيعي ، أما بالنسبة لأعداد النظام الثماني فتطبع مسبقة بصفر 0 ، في حين تطبع الأعداد بنظام السداسي عشر مسبقة بـ 0x . يبين البرنامج في المثال (3- 19) التالي كيفية استخدام مؤشر الصيغة لتحديد عملية الطباعة للأعداد الصحيحة وفقاً لأنظمة العد المختلفة .

مثال (3- 19)

```
//Using stream - manipulator show base
```

```
#include<iostream>
```

```
using namespace std;
```

```
void main ( ) {
```

```
    int x = 150 ;
```

```

// use showbase to show number base
cout<< " printing integer proceeded by their base : "
    << endl << showbase ;
cout<< x << endl; // print decimal value
cout<< oct << x << endl; // print octal value
cout<< hex << x << endl; // print hexadecimal value
} //End main

```

RESULT

printing integer proceeded by their base :

150

0226

0x96

Press any key to continue

٣-٨ - أمثلة على جميع العمليات

يبين البرنامج في المثال (3-20) التالي جميع العمليات كتابع عضو عام في الصف coord .

مثال (3-20)

//EXAMPLE - Overloading Binary Operators and
 // Unary Operators

```

#include <iostream.h>

class coord {
private :
    int x,y; //coordinate value
public :
    coord( ) { x=0;y=0;}
    coord( int i,int j ) { x=i;y=j;}
    void get_xy( int &i , int &j ) { i=x ; j = y ; }

    coord operator + ( coord ob2); // obj1 + obj2
    coord operator - ( coord ob2); // obj1 - obj2
    coord operator + ( int i);      // obj1 + int
    coord operator * ( int i);      // obj1 * int
    coord operator *( coord ob2); // obj1 * obj2
    int operator == ( coord ob2); // obj1 == obj2

    coord operator ++ ( );      // Increment
    coord operator -- ( );      // Decrement
    coord operator - ( );      // negation
}; // End Class

// overload + relative to coord class .
coord coord :: operator + ( coord ob2) {
    coord temp ;

```

```

temp .x = x + ob2.x;
temp .y = y + ob2.y;
return temp;
}
//overload - relative to coord class .
coord coord :: operator - ( coord ob2) {
    coord temp;
    temp .x = x - ob2.x;
    temp .y = y - ob2.y;
    return temp;
}
//overload + for obj1 + int .
coord coord :: operator + ( int i) {
    coord temp;
    temp .x = x + i;
    temp .y = y + i;
    return temp;
}
//overload *for obj1 * int .
coord coord :: operator * ( int i) {
    coord temp;
    temp .x = x * i;
    temp .y = y * i;

```

```

        return temp ;
    }
    //overload + relative to coord class .
    coord coord :: operator * ( coord ob2) {
        coord temp ;
        temp .x = x * ob2.x ;
        temp .y = y * ob2.y ;
        return temp ;
    }
    //overload the == operator for coord class .
    int coord :: operator == ( coord ob2) {
        return  x == ob2.x && y == ob2.y ;
    }
    //overload ++for coord class .
    coord coord :: operator ++ ( ) {
        x++ ;
        y++ ;
        return *this ;
    }
    //overload -- for coord class .
    coord coord :: operator -- ( ) {
        x -- ;
        y -- ;

```



```

        return *this ;
    }
    //overload unary - for coord class .
    coord coord :: operator - ( ) {
        x = -x ;
        y = -y ;
        return *this ;
    }

    void main ( )
    {
        coord obj1(10,10) ,obj2(5,3),obj3;
        int x,y ;
        obj3=obj1 + obj2 ; // add tow objects
        obj3.get_xy(x,y) ;
        cout<<" (obj1+obj2) x: "<< x<<" y: "<<y<<endl;

        obj3=obj1 -obj2 ; // sub tow objects
        obj3.get_xy(x,y) ;
        cout<<" (obj1-obj2) x: "<< x<<" y: "<<y<<endl;

        obj3=obj2 + 100 ; // add objects + int
        obj3.get_xy(x,y) ;
        cout<<" (obj1+ 100) x: "<< x<<" y: "<<y<<endl;
    }

```

```

obj3=obj2 * 100 ; // multiple (objects * int )
obj3.get_xy(x,y) ;
cout<<" (obj2* 100) x: "<< x<<" y: "<<y<<endl;

obj3=obj1 * obj2 ; // multiple tow objectes
obj3.get_xy(x,y) ;
cout<<" (obj1*obj2) x: "<< x<<" y: "<<y<<endl ;
if ( obj1==obj2) cout<<" obj1 same as obj2 \n" ;
else cout << "obj1 and obj2 is differ \n " ;

++obj1 ; // increment an object
obj1.get_xy(x,y) ;
cout<<" (obj1) x: "<< x<<" y: "<<y<<endl;
--obj2 ; // Decrement an objects
obj2.get_xy(x,y) ;
cout<<" (obj2) x: "<< x<<" y: "<<y<<endl;
obj1 = - obj1 ; // negation
obj1.get_xy(x,y) ;
cout<<" (- obj1) x: "<< x<<" y: "<<y<<endl;

} // End main

```

RESULT

```
(obj1+obj2) x: 15 y: 13
(obj1-obj2) x: 5 y: 7
(obj1+ 100) x: 105 y: 103
(obj2* 100) x: 500 y: 300
(obj1*obj2) x: 50 y: 30
obj1 and obj2 is differ
(++obj1) x: 11 y: 11
(-- obj2) x: 4 y: 2
(- obj1) x: -11 y: -11
```

Press any key to continue

يبين البرنامج في المثال (3- 21) التالي جميع العمليات الثنائية والأحادية
كتابع صديق في الصف coord .

مثال (3- 21)

```
//EXAMPLE // Overloading Binary Operator and
// Unary Operators Using a friend .
```

```

include <iostream.h>

class coord {
private :
    int x,y;//coordinate value
public :
    coord( ) { x=0;y=0;}
    coord(int i,int j ) { x=i;y=j;}
    void get_xy( int &i , int &j ) { i = x ; j = y ; }

    friend coord operator + ( coord ob1 , coord ob2 ) ;
    friend coord operator + ( coord ob1 , int i ) ;
    friend coord operator - ( coord ob1 , coord ob2 ) ;
    friend coord operator * ( coord ob1 , coord ob2 ) ;
    friend coord operator * ( coord ob1 , int i ) ;
    friend coord operator ++ ( coord &ob1 ) ;
    friend coord operator -- ( coord &ob1 ) ;
} ; // End class

//overload + using friend .

coord operator + ( coord ob1 , coord ob2 )
{
    coord temp ;
    temp .x = ob1.x + ob2.x ;
    temp .y = ob1.y + ob2.y ;

```

```

        return temp ;
    }
    //overload - using friend.
    coord operator - ( coord ob1 , coord ob2 ) {
        coord temp ;
        temp .x = ob1.x - ob2.x ;
        temp .y = ob1.y - ob2.y ;
        return temp ;
    }
    //overload + using friend .
    coord operator + ( coord ob1 , int i ) {
        coord temp ;
        temp .x = ob1.x + i ;
        temp .y = ob1.y + i ;
        return temp ;
    }
    // overload * using friend .
    coord operator * ( coord ob1 , coord ob2 ) {
        coord temp ;
        temp .x = ob1.x * ob2.x ;
        temp .y = ob1.y * ob2.y ;
        return temp ;
    }
}

```

*//overload * using friend .*

```
coord operator * ( coord ob1 , int i ) {  
    coord temp ;  
    temp .x = ob1.x * i ;  
    temp .y = ob1.y * i ;  
    return temp ;  
}
```

// overload ++ using friend .

```
coord operator ++ ( coord &ob1 ) {  
    ob1.x ++ ;  
    ob1.y ++ ;  
    return ob1 ;  
}
```

//overload -- using friend .

```
coord operator --( coord &ob1 ) {  
    ob1.x -- ;  
    ob1.y -- ;  
    return ob1 ;  
}
```

```
void main ( ) {  
    coord obj1(10,10),obj2(5,3),obj3;  
    int x , y ;
```



```
obj3=obj1 + obj2 ; // add tow objects  
obj3.get_xy(x,y) ;  
cout<<" (obj1+obj2) x: "<< x<<" y: "<<y<<endl;
```

```
obj3=obj1 + 50 ; // add tow (objects + int)  
obj3.get_xy(x,y) ;  
cout<<" (obj1+50) x: "<< x<<" y: "<<y<<endl;
```

```
obj3=obj1 - obj2 ; // subtract tow objects  
obj3.get_xy(x,y) ;  
cout<<" (obj1-obj2) x: "<< x<<" y: "<<y<<endl;
```

```
obj3=obj1 * obj2 ; // multiple tow objects  
obj3.get_xy(x,y) ;  
cout<<" (obj1*obj2) x: "<< x<<" y: "<<y<<endl;
```

```
obj3=obj1 * 50 ; // multiple tow ( object * int i )  
obj3.get_xy(x,y) ;  
cout<<" (obj1*50) x: "<< x<<" y: "<<y<<endl;
```

```
obj1 ++ ; // unary ++  
obj1.get_xy(x,y) ;  
cout<<" (obj1++) x: "<< x<<" y: "<<y<<endl;
```

```
obj2-- ; // unary - -  
obj2.get_xy(x,y) ;  
cout<<" (obj2--) x: "<< x<<" y: "<<y<<endl;  
} // End main
```

RESULT

```
(obj1+obj2) x: 15 y: 13  
(obj1+50) x: 60 y: 60  
  
(obj1-obj2) x: 5 y: 7  
(obj1*obj2) x: 50 y: 30  
(obj1*50) x: 500 y: 500  
  
(obj1++) x: 11 y: 11  
(obj2--) x: 4 y: 2
```

Press any key to continue



مسائل عامة محلولة

١- أكتب برنامجاً بلغة C++ يتضمن صفاً يمثل درجات الحرارة اسمه Temp ويحتوي على وسطين mx , mn من النوع int كأعضاء خاصة ويحتوي على الأعضاء العامة التالية:

- بأن افترضني يسند القيمة صفر للمعطيات الخاصة.
 - بأن عادي بوسطين يسند قيم للمعطيات الخاصة.
 - تابع عضو من النوع void اسمه Test ، مهمته إيجاد العنصر الأكبر لمصفوفة من الأعداد ومن ثم إيجاد العنصر الأصغر لنفس المصفوفة ، ثم يُسند النتائج للمعطيات الخاصة mx و mn على الترتيب .
 - تابع عضو لطباعة قيم المعطيات الخاصة اسمه print .
 - تابع وصول للمعطيات الخاصة من النوع void اسمه get .
 - تابع صديق للصف Temp (غير عضو) اسمه fr مهمته حساب الفرق بين قيم المعطيات الخاصة.
 - أجر عملية التحميل الزائد للعملية الثنائية (*) كتابع صديق .
- ويحتوي البرنامج على تابع رئيسي main من أجل إدخال درجات الحرارة لشهر حزيران (30 يوماً) وتخزينها في مصفوفة أحادية البعد T[30] ومن ثم طباعتها. بعد ذلك صرح عن غرض ob من الصف Temp ومن ثم استدع التتابع اللازمة لحساب و طباعة درجتَي الحرارة العظمى والصغرى لشهر حزيران وكذلك حساب وطباعة الفرق بينهما من خلال التابع الصديق. بعد ذلك صرح عن

الأغراض اللازمة من أجل تنفيذ تابع التحميل الزائد للعملية (*) ، حيث إن الغرض الأول يأخذ الثنائية المكونة من الفرق الناتج نفسه ، بينما الغرض الثاني يأخذ الثنائية المكونة من مربعاته.

البرنامج

```
#include<iostream.h>

class Temp {
    int mx,mn;
public:
    Temp( ){mx =0; mn =0 ;}
    Temp(int m ,int n){ mx = m ;mn = n ;}
    void Test( int [ ],int );
    void print( ){cout<<"mx="<<mx<<" "<<"mn ="
                    <<mn<<endl;
                }
    friend float fr(Temp );
    void get(int &i,int &j){i=mx;j=mn;}
    friend Temp operator *(Temp ,Temp );
};//End Class

void Temp::Test ( int x[ ], int n ) {
    int i,max=x[0],min=x[0];
    for(i = 0 ; i < n ; i++ ) {
        if(x[i]>max)
```

```

        max=x[i];
    }
    for(i=0;i<n;i++) {
        if(x[i]<min)
            min=x[i];
    }
    mx = max;
    mn = min;
}

float fr(Temp ob)
{
    float d;
    d=ob.mx-ob.mn;
    return d;
}

Temp operator *(Temp ob1,Temp ob2)
{
    Temp t;
    t.mx=ob1.mx*ob2.mx;
    t.mn=ob1.mn*ob2.mn;
    return t;
}

```

```

void main( ) {

    int T[30],a,b,d1;
    cout<<"Enter the numbers \n";
    for(int i=0;i<30;i++)
        cin >>T[i];
    for( i=0;i<30;i++)
        cout<<T[i]<< " ";
    cout<<endl;

    Temp ob;
    ob.Test(T,30);
    ob.print( );
    d1=fr(ob);
    cout<<"The value of mx-mn is "<<d1<<endl;

    Temp ob1(d1,d1),ob2(d1*d1,d1*d1),ob3;
    ob3=ob1*ob2;
    ob3.get(a,b);
    cout<<"(ob1+ob2) a:"<<a<<" "<<"b:"<<b<<endl;

} //End main

```


٢- أكتب برنامجاً بلغة C++ يتضمن صفراً اسمه engineer يحتوي على وسيط mark من النوع int كعضو خاص ويحتوي على الأعضاء العامة (public) التالية:

- بأن افتراضي يسند القيمة صفر للعضو الخاص mark.
 - بأن عادي وسيطه من نوع مصفوفة ويقوم بحساب عدد عناصر المصفوفة التي قيمها أكبر أو تساوي 60.
 - تابع وصول getmark يعيد قيمة mark .
 - تابع (غير عضو) اسمه counter وسيطه غرض من الصف engineer و يقوم بإسناد القيمة المعادة من التابع getmark إلى متغير t ومن ثم يعيد القيمة t .
 - أجر عملية التحميل الزائد للعملية الثنائية (-) كنابح عضو عام .
 - تابع مدمر يطبع رسالة ما عند استدعائه.
- ❖ يحتوي البرنامج على تابع رئيسي main يتضمن إدخال درجات طالب ما في 12 مقرر وتخزينها في مصفوفة أحادية البعد m[12] . بعد ذلك التصريح عن غرض ob من الصف engineer ومن ثم استدعاء التابع counter لحساب وطباعة عدد المقررات التي نجح فيها الطالب (على اعتبار أن درجة النجاح في المقرر هي 60)، بعد ذلك يجب التصريح عن الأغراض المناسبة من الصف engineer من أجل استدعاء تابع التحميل الزائد للعملية الثنائية (-) وذلك باستخدام قيم كيفية لهذه الغاية.

❖ البرنامج :

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

```
class engineer {  
    int mark;
```

```

public:
    engineer ( ){mark=0;}
    engineer (int x[ ]);
    int getmark( ){return mark;}
    engineer operator -(engineer);
    ~engineer( ){ cout<<" class Engineer \n "}
};//End Class

engineer::engineer(int x[ ] ) {
    int i,c=0;
    for(i=0;i<12;i++) {
        if(x[i]>=60)
            c+=1;
    }
    mark = c;
}

int counter(engineer ob) {
    int t;
    t= ob.getmark();
    return t;
}

engineer engineer:: operator -(engineer ob1) {
    engineer t;

```

```

        t.mark = mark-ob1.mark;
        return t;
    }

    void main( )
    {
        int m[12],m1[12],i;
        cout<<"Enter the marks \n";
        for( i=0;i<12;i++)
            cin>>m[i];
        engineer ob(m);
        cout<<"The number of success courses is
        "<<counter(ob)<<endl;
        cout<<"Enter the another marks \n";
        for( i=0;i<12;i++)
            cin>>m1[i];
        engineer ob1(m), ob2(m1), ob3;
        ob3=ob1-ob2;
        cout<<"ob3="<<ob3.getmark()<<endl;
    }//End Class

```

٣- أكتب برنامجاً بلغة C++ يتضمن ما يلي:

❖ صفأ اسمه Even_Odd يحتوي على وسيطين odd (فردى) even (زوجى) من نوع int كأعضاء خاصة ويحتوى على الأعضاء العامة التالية:

- بأن افتراضى يسند قيمة الصفر للمعطيات الخاصة.
- بأن عادى وسيطه مصفوفة ويقوم بحساب عدد الأعداد الزوجية وعدد الأعداد الفردية لمصفوفة من الأعداد مكونة من عشرة عناصر ومن النوع الصحيح ومن ثم يُسند النتائج للمعطيات الخاصة even و odd على الترتيب.
- تابع عضو لطباعة قيم المعطيات الخاصة وليكن اسمه print ومن النوع void.
- تابع عضو للوصول إلى لمعطيات الخاصة من النوع void اسمه get.
- أجر عملية التحميل الزائد لعملية الإسناد كتابع عضو (=) .
- تابع صديق للصف Even_Odd (غير عضو) وليكن اسمه fr ، مهمته حساب قيمة المقدار التالى:

$$per = \begin{cases} \frac{even!}{(even - odd)! odd!} \dots \text{if } ..even \geq odd \\ \frac{odd!}{(odd - even)! even!} \dots \text{if } ..even < odd \end{cases}$$

❖ تابعاً لحساب العاملى (!) لأي عدد صحيح اسمه fact وهو غير عضو فى الصف وغير صديق.

❖ تابعاً رئيسياً main لإدخال 10 أعداد صحيحة وتخزينها في مصفوفة أحادية البعد [10]x ومن ثم طباعتها. في هذا التابع يتم التصريح عن أغراض من الصف Even_Odd من أجل حساب وطباعة قيم المعطيات الخاصة التي تم حسابها وأيضاً حساب وطباعة قيمة المقدار per وكذلك تنفيذ عملية التحميل الزائد لعملية الإسناد (=).

❖ البرنامج

```
#include<iostream.h>

float fact(int n) {
    float f=1;
    if(n < 0 )
        return 0;
    if( n == 0)
        return 1;
    for(int i =1; i<= n; i++)
        f = f * i ;
    return f;
}

class Even_Odd {
    int odd,even;
public:
    Even_Odd( ) {odd=0;even=0;}
    Even_Odd(int x[ ]);
    void print( ){cout<<" odd = " << odd << " " }
```

```

        <<"even=" <<even<<endl;}

    void get(int&ev,int&od){ev = even; od = odd ; }

    Even_Odd operator=(Even_Odd);

    friend float fr(Even_Odd ob);

}; //End Class

Even_Odd::Even_Odd(int x[ ]) {
    int i,s1=0,s2=0;
    for(i=0;i<10;i++) {
        if(x[i]%2==0)
            s1=s1+1;
        else
            s2=s2+1;
    }
    even=s1;
    odd=s2;
}

float fr(Even_Odd ob) {
    int e,d;
    float per;
    e = ob.even;
    d = ob.odd;

```



```

if(e >= d)
    per = (float) fact(e)/(fact(e-d)*fact(d));

else
    per = (float) fact(d)/(fact(d-e)*fact(e));
return per ;
}

Even_Odd Even_Odd:: operator=(Even_Odd ob)
{
    even=ob.even;
    odd=ob.odd;
    return * this;
}

void main( )
{
    int x[10],i;
    int e,d;
    cout<<"Enter the numbers \n";
    for( i=0;i<10;i++)
        cin>>x[i];
    cout<<"The entered array is:"<<endl;
    for( i=0;i<10;i++)
        cout<<x[i]<<" ";
}

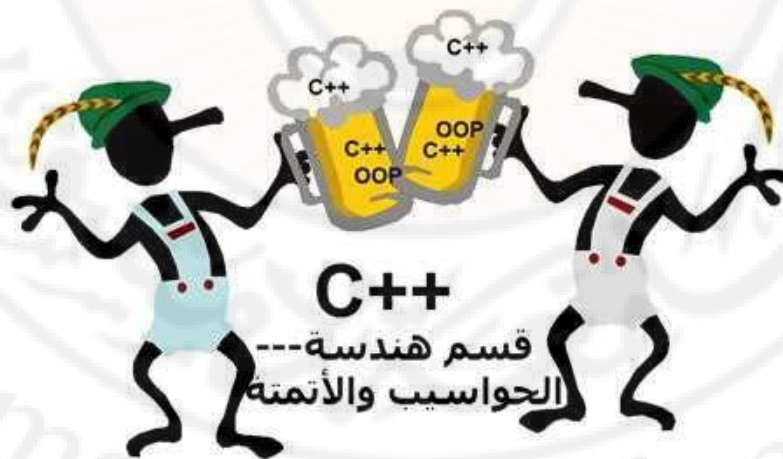
```

```

cout<<endl;
Even_Odd ob(x),ob1;
ob.print( );
cout<<"The value of per is "<<fr(ob)<<endl;
ob1=ob;
ob1.get(e,d);
cout<<"even="<< e <<" "<<"odd="<< d <<endl;

} //End main

```



مسائل عامة

- ١- اكتب برنامج بلغة C++ للصف Point لطباعة إحداثيات النقطة x,y,z من النوع double . بحيث يعطي إحداثيات النقطة في الربع الثالث ، و يحتوي الصف على بان افتراضي وباني عادي لتمرير المعطيات .
- أجر عملية التحميل الزائد لمجرى الخرج كتابع صديق (لعملية الإخراج <<) لطباعة إحداثيات النقطة (x, y, z) .
 - أجر عملية التحميل الزائد للعملية $(=)$ كتابع عضو عام ، والعملية الأحادية $(++)$ من اليمين كتابع صديق في الصف point .
 - أنشئ في التابع $main()$ أغراض للصف point لتمرير المعطيات وطباعة إحداثيات النقطة .

=====

- ٢- اكتب برنامج بلغة C++ للصف Coord لطباعة إحداثيات النقطة x,y,z من النوع int . بحيث يعطي إحداثيات النقطة في الربع الرابع ، و يحتوي الصف على بان افتراضي وبان عادي لتمرير المعطيات .
- أجر عملية التحميل الزائد لمجرى الخرج كتابع صديق (لعملية الإخراج <<) لطباعة إحداثيات النقطة (x, y, z) في الصف Coord .
 - أجر عملية التحميل الزائد للعملية الثنائية $(/)$ كتابع عضو عام ، والعملية الأحادية $(--)$ من اليمين كتابع صديق في الصف Coord .
 - أنشئ في التابع $main()$ أغراض للصف Coord لتمرير المعطيات وطباعة إحداثيات النقطة .

٣- اكتب برنامجاً بلغة C++ لإنشاء صف اسمه `point` يعرف النقطة (x, y) وإحداثياتها من النوع `double` تقع في الربع الثاني ، ويحتوي الصف على بانٍ عادي وبانٍ افتراضي لتحديد القيم الابتدائية للنقطة ويحتوي على تابع مدمر بحيث يطبع رسالة عند استدعائه .ويحتوي على تابع لطباعة إحداثيات النقطة .

- أجر عملية التحميل الزائد في الصف `point` للعمليات التالية:
(* , ++ , ==) .
- أنشئ في التابع `main` أغراضاً للصف `point` لتمثيل المعلومات واستخدام تابع الطباعة لإظهار كافة المعلومات .

=====

٤- اكتب برنامجاً بلغة C++ لإنشاء صفاً اسمه `coord` يعرف نقطة من مستوى إحداثياتها (x, y) من النوع `double` بحيث يحتوي الصف على بانٍ عادي وعلى بانٍ افتراضي لتحديد القيم الابتدائية للنقطة ويحتوي على تابع مدمر بحيث يطبع رسالة ما عند استدعائه .

- أجر عملية التحميل الزائد لمجاري الخرج `ostream` من أجل طباعة إحداثيات النقطة (x, y) .
- أجر عملية التحميل الزائد للعمليات (*) كتابع عضو عام بحيث يتم تمثيل الأغراض بواسطة العنوان والعمليات (++) كتابع صديق في الصف `coord` .
- أنشئ في التابع `main` أغراضاً للصف `coord` لتمثيل المعلومات واستخدام مجاري الخرج لإظهار كافة المعلومات .
- هل يمكن إجراء عملية التحميل الزائد لمجاري الخرج كتابع عضو عام ؟ ولماذا ؟
- متى وكيف يتم استدعاء التوابع البانية والمدمرة ؟

٥- اكتب برنامجاً لإنشاء صفاً بلغة C++ اسمه Time يحتوي على ثلاثة وسطاء (Hour , Minute , Second) من النوع int كأعضاء خاصة ويقوم بإدخال الوقت وطباعته بالشكلين القياسي والعسكري بحيث يتم تعيين قيمة ابتدائية (00 : 00 : 00 AM) للشكل القياسي وقيمة ابتدائية (00 : 00 : 00) للشكل العسكري ، وعند الإدخال يجب أن يطبع القيمة (0) إذا كانت hour و minute و second ذات قيم غير صحيحة .

ويحتوي الصف على بانٍ افتراضي لتحديد القيم الابتدائية للوقت ويحتوي على تابع مدمر بحيث يطبع رسالة ما عند استدعائه ويحتوي على تابع بانٍ عادي لتمرير المعطيات وتابع طباعة للشكل القياسي وتابع طباعة آخر للشكل العسكري .

• أجر عملية التحميل الزائد للعملية (=) كتابع عضو عام والعملية (- -) من اليسار كتابع صديق في الصف .

❖ أنشئ في التابع () main أغراضاً للصف Time لتمرير المعطيات وطباعة الوقت بالشكلين القياسي والعسكري .

❖ هل يتم إجراء عملية التحميل الزائد للتابع المدمر؟ ولماذا ؟

❖ ما هو المؤشر this ؟

=====

٦- اكتب برنامجاً بلغة C++ لإنشاء صفاً اسمه point يعرف نقطة من مستوى إحداثياتها (x , y) من النوع double بحيث يقوم بتعريف نقطة موجبة دوماً مهما كانت قيمة x , y المصرح عنها في الغرض ضمن التابع main ويحتوي الصف على بانٍ افتراضي لتحديد القيم الابتدائية للنقطة ويحتوي على تابع مدمر بحيث يطبع رسالة ما عند استدعائه ، وعلى تابع بانٍ عادي .

❖ أجر عملية التحميل الزائد لمجرى الخرج كتابع صديق (لعملية الإخراج <<) لطباعة إحداثيات النقطة .

- ❖ أجر عملية التحميل الزائد للعملية (=) كتابع عضو عام ، والعملية (++) من اليسار كتابع صديق في الصف point .
- ❖ أجر عملية التحميل الزائد للعملية (/) كتابع عضو عام والعملية (*) كتابع صديق في الصف point .
- ❖ أنشئ في التابع () main أغراض للصف point لتمثيل المعطيات وطباعة إحداثيات النقطة .
- ❖ لماذا لا يأخذ التابع المدمر أي قيمة ولا يعيد أي قيمة ؟
- ❖ هل يتم إجراء عملية التحميل الزائد للتابع الباني والمدمر ؟ ولماذا ؟

=====

٧- اكتب برنامجاً لإنشاء صفّاً بلغة C++ اسمه **strclass** لطباعة سلسلة **str** من نوع **char** (يحتوي الصف على متحول واحد **str**) ، ويحتوي الصف على بانٍ افتراضي لتحديد القيم الابتدائية وعلى تابع مدمر بحيث يطبع رسالة ما عند استدعائه ، وعلى تابع بانٍ عادي لتمثيل المعطيات.

- أجر عملية التحميل الزائد لمجرى الخرج (لعملية الإخراج <<) لطباعة السلسلة كتابع صديق .
- أجر عملية التحميل الزائد لعملية الإسناد (=) كتابع عضو عام .
- أجر عملية التحميل الزائد لعملية المقارنة (==) كتابع عضو عام .
- أجر عملية التحميل الزائد لعملية الجمع (+) كتابع صديق .
- ❖ أنشئ في التابع () main أغراضاً للصف **strclass** لتمثيل المعطيات وطباعة السلسلة .
- ❖ لماذا لا يأخذ التابع المدمر أي قيمة ولا يعيد أي قيمة ؟
- ❖ متى يجب استدعاء التوابع البانية والمدمرة ؟ وكيف ؟

٨- اكتب برنامجاً بلغة C++ لإنشاء صفّاً اسمه point يعرف نقطة ثلاثية البعد إحداثياتها (x , y, z) من النوع double بحيث يقوم بتعريف نقطة سالبة دوماً مهما كانت قيمة x , y, z المصرح عنها في الغرض ضمن التابع main ، ويحتوي الصف على بانٍ افتراضي لتحديد القيم الابتدائية للنقطة وتابع بانٍ عادي و تابع مدمر بحيث يطبع رسالة ما عند استدعائه .

- أجرِ عملية التحميل الزائد لمجرى الخرج كتابع صديق (لعملية الإخراج <<) لطباعة إحداثيات النقطة (x , y, z) .
- أجرِ عملية التحميل الزائد للعملية (=) كتابع عضو عام ، والعملية الأحادية (++) من اليمين كتابع صديق في الصف point .
- أجرِ عملية التحميل الزائد للعملية (/) كتابع عضو عام والعملية الأحادية (-) كتابع صديق في الصف point .
- أنشئ في التابع (main) أغراضاً للصف point لتحرير المعطيات وطباعة إحداثيات النقطة .
- هل يتم إجراء عملية التحميل الزائد للتابع الباني والمدمر؟ ولماذا؟

=====

٩- اكتب برنامجاً بلغة C++ يحتوي على صف أساسي اسمه date ، ويحتوي على ثلاثة وسطاء (day , month , year) من النوع int كأعضاء private ويقوم بإدخال التاريخ وطباعته بالشكل (day : month : year) ويحتوي الصف على بانٍ عادي وبانٍ افتراضي لتحديد القيم الابتدائية للتاريخ وهي (01 : 01 : 2000) ، وقيمة الوسيط day ضمن المجال [1 - 31] ، والوسيط month ضمن المجال [1 - 12] ، والسنة year ضمن المجال [2000 - 2011] .

- أجر عملية التحميل الزائد لمجرى الدخل كتابع صديق (لعملية الإدخال >>) لإدخال المتحولات (day , month , year) من المستخدم (من لوحة المفاتيح) .
- أجر عملية التحميل الزائد لمجرى الخرج كتابع صديق (لعملية الإخراج <<) لطباعة التاريخ (day , month , year) .
- أجر عملية التحميل الزائد للعملية الثنائية (=) كتابع عضو عام ، والعملية الأحادية (++) من اليمين كتابع صديق في الصف date .
- أنشئ أغراضاً في التابع main() لاستدعاء توابع الأعضاء للصف date وطباعة النتائج .

=====

٨- اكتب برنامجاً بلغة C++ يحتوي على صف أساسي اسمه BOX ، ويحتوي الصف على ثلاثة وسطاء خاصة private من النوع char ، وهي : x,y,z التي تمثل أبعاد الصندوق BOX ، ويحتوي الصف على بيانٍ عادي وبيانٍ افتراضي لتحديد القيم الابتدائية لأبعاد الصندوق وهي (1,1,1) .

- أجر عملية التحميل الزائد لمجرى الدخل كتابع صديق (لعملية الإدخال >>) لإدخال المتحولات (x,y,z) من المستخدم (من لوحة المفاتيح) .
- أجر عملية التحميل الزائد لمجرى الخرج كتابع صديق (لعملية الإخراج <<) لطباعة أبعاد الصندوق (x,y,z) .
- أجر عملية التحميل الزائد للعملية الثنائية (=) كتابع عضو عام ، والعملية الثنائية (+) كتابع صديق والعملية الثنائية (=) كتابع عضو عام في الصف BOX .
- أنشئ أغراضاً في التابع main() لاستدعاء توابع الأعضاء للصف BOX ومجرى الدخل والخرج لإدخال وطباعة النتائج .

١٠- اكتب برنامجاً لإنشاء صف اسمه rectangle يعبر عن مستطيل ويمثل المستطيل بإحداثيات نقطتي الزاويتين المتقابلتين في المستطيل النقطة الأولى $A(x1,y1)$ ، والنقطة الثانية $A2(x2,y2)$ ويحتوي الصف على التتابع الأعضاء التالية:

- تابع عضو عام اسمه border لحساب محيط المستطيل.
 - تابع عضو اسمه area لحساب مساحة المستطيل.
 - تابع عضو اسمه $shift(float\ xx, float\ yy)$ لإزاحة المستطيل بمقدار xx و yy ($x=x+xx, y=y+yy$)
 - تابع عضو اسمه $zoom(int\ ratio)$ لتكبير أبعاد المستطيل بنسبة ratio بحيث يبقى مركز المستطيل ثابت بالنسبة لإحداثيات المستوي.
 - تابع عضو اسمه $print$ لطباعة إحداثيات نقاط زوايا المستطيل على الشاشة.
 - أجر عملية التجميل زائد للعملية الثنائية ($=$) للمقارنة بين مساحتي مستطيلين بحيث يعطي 1 عند تساوي المساحتين و 0 فيما عدا ذلك.
- ❖ في التابع main عرف غرضين

`rectangle r1(1,1,2,2), r2(4,10,5,11);`

- ❖ استخدم الصف السابق لتعريف مستطيل ممثل بالنقطتين (1,1) و (2,2) واستخدم جميع الوظائف المعرفة في هذا الصف ثم تحقق من أن هذا المستطيل يساوي المستطيل (10,4) و (5,11) أم لا.

=====

١١- اكتب برنامجاً بلغة C++ يحتوي على صف Complex يمثل عدد عقدي يحتوي على جزء حقيقي Real وجزء تخيلي Img ويحتوي الصف على التتابع التالية :

- تابع بان عام .

- تابع print لطباعة العدد العقدي بالشكل (Real $\pm$ i Img) .
- تابع تحميل زائد للعملية الثنائية (+) لجمع عددين عقديين .
- تابع تحميل زائد للعملية الأحادية (!) لإيجاد المرافق العقدي .
- تابع تحميل زائد للعملية الأحادية (-) من اليسار (أو الأمام) .
- أنشئ أغراضاً في التابع main لاستدعاء التوابع الأعضاء للصف point لإجراء العمليات الأحادية والثنائية وطباعة النتائج .

=====

١٢- اكتب برنامجاً بلغة C++ يحتوي على صف اسمه Vector يمثل شعاع في الفراغ يحتوي على إحداثيات المساقط (X , Y , Z) ويحتوي على التوابع التالية :

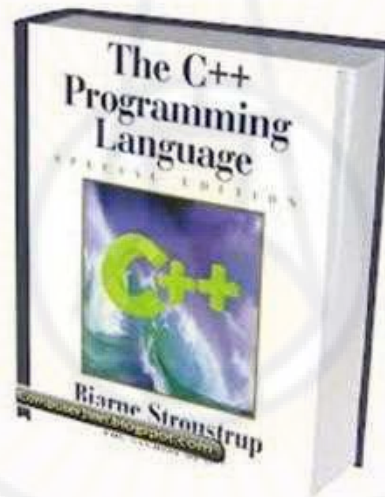
- تابع بانٍ عام .
- تابع تعديل لإحداثيات الشعاع اسمه SetVector .
- تابع التحميل الزائد لمجرى الدخل (>>) كتابع صديق للصف .
- تابع التحميل الزائد لمجرى الخرج (<<) كتابع صديق للصف .
- تابع التحميل الزائد للعملية الثنائية (+) لجمع شعاعين كتابع صديق للصف .
- تابع التحميل الزائد للعملية الثنائية (-) لطرح شعاعين كتابع عضو .
- تابع التحميل الزائد للعملية الثنائية (*) لضرب شعاعين (جداء سلمي) كتابع عضو في الصف .
- تابع التحميل الزائد للعملية الأحادية (-) لعكس جهة الشعاع كتابع عضو .
- تابع التحميل الزائد للعملية الأحادية () لإيجاد طول الشعاع كتابع عضو .

- تابع التحميل الزائد للعملية الثنائية (= =) لمقارنة شعاعين من حيث الطويلة كتابع عضو .
- تابع التحميل الزائد للعملية الأحادية (--) من اليسار كتابع صديق للصف .
- أنشئ أغراضاً في التابع main لاستدعاء توابع الصف وطباعة النتيجة لجميع عمليات التحميل الزائد .





الفصل الرابع
الوراثة
Inheritance





الوراثة

Inheritance

٤-١- مقدمة

تعتبر الوراثة أحد أشكال إعادة استخدام النصوص البرمجية ، حيث يتم إنشاء صفوف جديدة انطلاقاً من صفوف موجودة مسبقاً وتأخذ الصفوف الجديدة صفات وتصرفات الصفوف البرمجية القديمة ، وتضيف إليها إمكانيات جديدة تحتاجها . وتساعد عملية إعادة استخدام النصوص البرمجية على توفير الوقت اللازم لتطوير البرامج ، وتشجع على استخدام برمجيات موثوقة وفعالة ، كما أنها تقلل من الإشكاليات الناتجة عن وضع البرمجيات قيد الاستخدام الفعلي .

وتسمى الصفوف الموجودة مسبقاً بالصفوف الأساسية Base Class وتسمى الصفوف المشتقة عنها بالصفوف المشتقة Derived Class . وتمثل الصفوف المشتقة أغراضاً أكثر تخصصاً من أغراض الصف الأساسي ، كما أنها تتضمن نوابع وإجراءات مخصصة لها ، ونسعى الصف الأساسي الذي يجري اشتقاق صف عنه بالصف الأساسي المباشر Direct Base Class ، أما إذا كان لدينا أكثر من مستوى للوراثة بين الصف الأساسي والصف المشتق فإننا نسمي الصف الأساسي غير المباشر Indirect Base Class .

تدعم لغة C++ الوراثة المتعددة Multiple Inheritance التي يقوم فيها الصف المشتق بالوراثة من أكثر من صف أساسي ، تعد الوراثة المتعددة معقدة ويمكن أن تؤدي إلى أخطاء . وتقدم لغة C++ ثلاثة أنواع من الوراثة ، وهي :

- ❖ الوراثة العامة Public Inheritance
- ❖ الوراثة المحمية Protected Inheritance
- ❖ الوراثة الخاصة Private inheritance

وسنركز في هذا الفصل على الوراثة العامة ، وسنشرح بشكل مختصر بعض الأمور المرتبطة بالنوعين الآخرين .

٤-٢- الصفوف الأساسية والصفوف المشتقة

نعرف الوراثة " هي الآلية التي بواسطتها يمكن لصف ما أن يرث خصائص صف آخر " ، وتسمح الوراثة ببناء بنية هرمية من الصفوف ابتداءً من الصف الأكثر عمومية وانتهاءً بالصف الأكثر خصوصية . يجب تعريف مصطلحين مستخدمين بكثرة في الوراثة:

□ يدعى الصف الذي يورث بالصف الأساسي Base Class .

□ ويدعى الصف الذي يرث بالصف المشتق Derived Class

ويعرف الصف الأساسي Base Class جميع الخواص المشتركة لأي من الصفوف المشتقة أي يقدم وصفاً عاماً لمجموعة من الخصائص ، ويرث الصف المشتق Derived Class الخصائص العامة من الصف الأساسي ويضيف إليها خصائص خاصة به . ولغة C++ توفر وسيلة لتقليل حجم البرنامج حيث يستفاد من بعض التتابع في بعض الصفوف ، وهذا يعني اشتقاق صفوف عديدة من صفوف محددة ثم إضافة المواصفات الإضافية الخاصة بالصف المشتق ، وبهذه الطريقة لا نحتاج إلى إعادة كتابة التتابع مرة أخرى .

ومن الممكن معاملة أغراض الصف الأساسي وأغراض الصف المشتق بشكل متشابه ، حيث يتوفر الجزء المشترك بينهما ضمن تعريف أعضاء الصف

الأساسي ، كما يمكن اعتبار أغراض كافة الصفوف المشتقة كأنها أغراض تابعة للصف الأساسي .

٤-٢-١ - التحكم بالوصول إلى الصف المشتق

Base Class Access Control

عند وراثة صف من صف آخر نستخدم الشكل العام التاليين أجل تعريف الصف المشتق وربطه مع الصف الأساسي :

```
class derived_Class_Name : access Base_Class_Name
{
    . . . . .class Body . . . . .
}; // End derived Class
```

وتمثل طريقة الوصول (access) واحدة من ثلاثة محددات وصول هي :

- ❖ public أي عام .
- ❖ private أي خاص .
- ❖ protected أي محمي .

محدد الوصول يحدد كيف يرث الصف المشتق عناصر الصف الأساسي. وعندما يكون محدد الوصول إلى الصف الأساسي هو public فإن جميع الأعضاء العامة للصف الأساسي تصبح أعضاء عامة للصف المشتق ، وعندما يكون محدد الوصول إلى الصف الأساسي هو private فإن جميع الأعضاء العامة للصف الأساسي تصبح أعضاء خاصة للصف المشتق ، وفي الحالتين كليهما تبقى الأعضاء الخاصة للصف الأساسي خاصة به ، ولا يمكن الوصول إليها من الصف المشتق .

ويجب أن ننتبه إلى أنه عندما يكون محدد الوصول `private` فإن الأعضاء العامة للصف الأساسي تصبح أعضاء خاصة للصف المشتق ولكن تبقى التوابع الأعضاء في الصف المشتق قادرة على الوصول إليها . يوضح المثال (4 - 1) التالي الوراثة بواسطة المحدد الوصول `public` .

مثال (1-4)

```
#include <iostream.h>

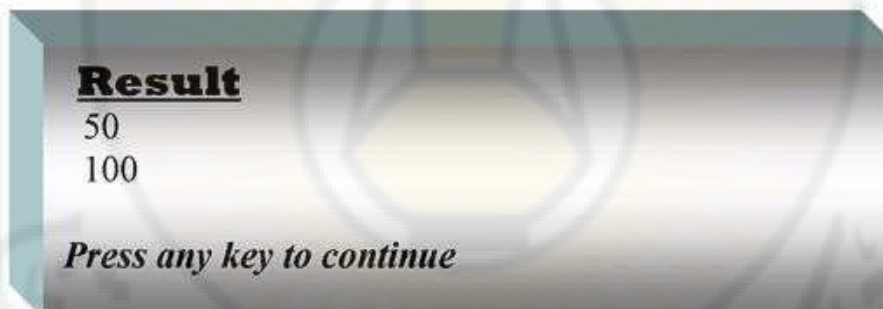
class base {
private :
    int x ;
public :
    void setx( int n ) { x = n ; }
    void printx( ) { cout << x << "\n" ; }
} ; // End class

// Inherit as public.
class derived : public base {
    int y ;
public :
    void sety(int n) { y = n ; }
    void printy( ) { cout << y << "\n" ; }
} ; // End derived Class
```

```

int main( )
{
    derived ob ;
    ob.setx(50) ; // access member of base class
    ob.sety(100) ; // access member of derived class
    ob.printx( ) ; // access member of base class
    ob.printy( ) ; // access member of derived class
    return 0 ;
} // End main

```



وفي الوراثة باستخدام محدد الوصول public تصبح الأعضاء العامة للصف base (مثل : setx() و printx()) أعضاء عامة في الصف المشتق derived . وإن وراثة صف لصف مشتق لا يعني أن الصف المشتق يستطيع الوصول إلى الأعضاء الخاصة للصف الأساسي .

□ مثلاً : لو أضفنا إلى الصف المشتق السابق التابع العام التالي :

```
void showsum ( )
{
    cout << " x+y=" << x+y << "\n" ;
}
```

وفي هذا البرنامج يحاول الصف المشتق derived الوصول إلى العضو الخاص x في الصف الأساسي ، وهذا خطأ لأن العضو الخاص في الصف الأساسي يبقى خاص به مهما كان الشكل الذي تمت به عملية الوراثة . ويوضح المثال (2-4) التالي بأننا لا نستطيع الوصول إلى المعطيات الخاصة وسوف يعطينا خطأ ERROR .

مثال (2-4)

```
#include <iostream.h>
class base {
    int x ;
public:
    void setx(int n) { x = n ; }
    void printx( ) { cout << x << "\n" ; }
}; // End class base

// Inherit as public - this has an error !
class derived : public base {
    int y ;
```

public:

```
void sety( int n) { y = n ; }
```

/ Cannot access private member of base class.*

x is a private member of base and not available

*within derived. */*

```
void show_sum( ) { cout << x+y << '\n' ; } // Error !
```

```
void printy( ) { cout << y << '\n'; }
```

```
}; //End derived class
```

٤-٢-٢ - استخدام الأعضاء المحمية

Using Protected Members

رأينا في المثال (4-2) أن الصف المشتق لا يستطيع الوصول إلى الأعضاء الخاصة للصف الأساسي . هذا يعني أنه إذا احتاج الصف المشتق للوصول إلى بعض الأعضاء الخاصة للصف الأساسي فإن هذه الأعضاء يجب أن تكون عامة . إلا أننا في بعض الأحيان قد نرغب لأحد أعضاء الصف الأساسي أن يبقى خاصاً به ، مع السماح للصف المشتق بالوصول إليه . ومن أجل ذلك نستخدم التعليمة الأساسية `protected` .

ويكافئ محدد الوصول `protected` المحدد `private` ، ما عدا أن الأعضاء المحمية للصف الأساسي مرئية من قبل أعضاء الصف المشتق . في حين أنها غير مرئية لأي صف من خارج الصف الأساسي أو الصفوف المشتقة منه . ويمكن أن تتواجد الكلمة المحجوزة (التعليمة) `protected` في أي مكان في الصف إلا أنها تتواجد عادة بعد التصريح عن الأعضاء الخاصة وقبل الأعضاء العامة .

الشكل العام هو :

```
class class_Name {  
    private :  
        .....  
    protected :  
        .....  
    public :  
        .....  
        .....  
}; // End class Base
```

وعندما يرث صفاً ما صفاً آخر باستخدام محدد الوصول **public** فإن الأعضاء المحمية للصف الأساسي تصبح أعضاء محمية للصف المشتق . أما إذا كانت الوراثة باستخدام محدد الوصول **private** فإنها تصبح خاصة للصف المشتق . وعندما يرث صفاً ما صفاً آخر باستخدام محدد الوصول **protected** عندها تصبح الأعضاء المحمية والعامة للصف الأساسي أعضاء محمية للصف المشتق .

يبين المثال (3-4) كيفية الوصول إلى الأعضاء الخاصة والعامة

والمحمية في الصف **myclass** .

مثال (3-4)

```
#include <iostream.h>  
class myclass {
```



```

private :
    int a ;
protected : // still private relative to myclass
    int b ;
public:
    int c ;
    myclass (int n, int m) { a = n; b = m; }
    int geta( ) { return a; }
    int getb( ) { return b; }
}; // End myclass

void main( ) {
    myclass ob(50, 100);
    // ob.b = 999 ; Error! b is protected and thus private
    ob.c = 300; // OK, c is public
    cout << ob.geta( ) << " ";
    cout << ob.getb( ) << " " << ob.c << "\n";
} // End main

```

Result

50 100 300

Press any key to continue

وضعنا العبارة `ob.b = 999` في سطر تعليق لأنها غير مسموح بها ، لأن `b` عضو محمي ، وبالتالي فهو خاص بالصف `myclass` ، ولا يمكن الوصول إليه من خارجه . لنعود إلى المثال (4-1) بأن نضع المتغير `x1` نوع `protected` كما هو واضح في المثال (4-4) .

وبما أن `x1` عضو محمي فإن الصف `base` قد ورث الصف `derived` باستخدام المحدد `protected` ، فإنها متاحة للاستخدام من قبل التوابيع الأعضاء في الصف `derived` ، لكن لا يمكن الوصول إليها من خارج هذين الصنفين .

مثال (4-4)

```
#include <iostream.h>

class base {
private :
    int x;
protected :
    int x1 ;
public:
    void setx( int n , int n1 ) { x = n ; x1 = n1 ; }
    void printx( ) { cout << x << "\n"; }
}; // End class

// Inherit as public.
class derived : public base {
    int y;
```

public:

```
void sety( int n) { y = n; }  
void printy( ) { cout << y << "\n"; }  
void showsum( )  
{  
    cout << "x1+y = " << x1+y << "\n";  
}
```

}; *// End derived class*

int main()

```
{  
    derived ob;  
    ob.setx(100,150);    // access member of base class  
    ob.sety(200);        // access member of derived class  
    ob.printx( );        // access member of base class  
    ob.printy( );        // access member of derived class  
    ob.showsum( );       // access member of derived class
```

return 0;

} *// End main*

Result

100

200

$x1+y = 350$

Press any key to continue

٤-٣- البواني والمدمرات

Constructors and Destructors

يمكن أن يكون لكل من الصف الأساسي والصف المشتق بانٍ واحدٍ أو أكثر و مدمر واحد لكل منهما . وعندما يكون الصفيين الأساسي والمشتق كليهما توابع بانية ومدمرة فإن التوابع البانية تنفذ بترتيب الاشتقاق نفسه في حين أن التوابع المدمرة تنفذ بالترتيب المعاكس . وهذا يعني أن التابع الباني للصف الأساسي ينفذ قبل التابع الباني للصف المشتق ، أما بالنسبة للتابع المدمر فإن التابع المدمر للصف المشتق ينفذ أولاً ثم التابع المدمر للصف الأساسي . إذا تم تنفيذ التابع المدمر للصف الأساسي أولاً فإنه سيؤدي إلى تدمير الصف المشتق أيضاً؛ لذلك يجب أن ينفذ التابع المدمر للصف المشتق أولاً ثم للصف الأساسي .

يوضح البرنامج (4-5) التالي كيف يتم تنفيذ كل من التوابع البانية والمدمرة للصف الأساسي والصف المشتق ، و كيفية تمرير الوسائط إلى باني الصف المشتق .

مثال (4-5)

`#include <iostream.h>`

```

class base {
public:
    base( ) { cout << "Constructing base class\n"; }
    ~base( ) { cout<<"Destructing base class \n"; }
};// End Base Class

    //Inheritance Class
class derived : public base {
private :
    int j ;
public:
    derived( int n) {
        cout << "Constructing derived class\n";
        j = n;
    }
    ~derived( ) { cout << "Destructing derived class\n"; }
    void printj( ) { cout << j << "\n"; }
}; // End derived class

void main( )
{
    derived ob(100);
    ob.printj( );
} // End main

```


Result

Constructing **base class**

Constructing **derived class**

100

Destructing **derived class**

Destructing **base class**

Press any key to continue

نلاحظ أن التابع الباني ينفذ حسب ترتيب الاشتقاق ، والتابع المدمر ينفذ بعكس ترتيب الاشتقاق ، ونلاحظ أن الوسيط قد مرر إلى باني الصف المشتق بشكل عادي .

في البرنامج (4-6) التالي يأخذ كل من باني الصف الأساسي وباني الصف المشتق وسائط يستخدم كل منهما في هذه الحالة الخاصة الوسيط نفسه ويمرر الصف المشتق الوسيط ببساطة إلى الصف الأساسي .

مثال (4-6)

```
#include <iostream.h>
class base {
private :
    int i;
public:
    base(int n) {
```

```

    cout << "Constructing base class\n";
    i = n;
}
~base( ) { cout << "Destructing base class\n"; }
void printi( ) { cout << i << '\n'; }
} ;// End Base Class

//Inheritance Class
class derived : public base {
    int j;
public:
    derived(int n) : base(n) { // pass arg to base class
        cout << "Constructing derived class\n";
        j = n;
    }
    ~derived( ) { cout << "Destructing derived class\n"; }
    void printj( ) {
        cout << j << '\n';
    }
} ; // End derived Class

int main( ) {
    derived obj(100);
    obj.printi( );
}

```

```
obj.printj( );
return 0;
} // End main
```

Result

```
Constructing base class
Constructing derived class
100
100
Destructing derived class
Destructing base class
Press any key to continue
```

ولا يستخدم التابع الباني للصف الأساسي والصف المشتق في أغلب الأحيان نفس الوسائط ، عندما نحتاج تمرير وسائط إلى كل منهما ، ينبغي تمرير جميع الوسائط التي يحتاجها كل من التابع الباني لكل من الصف الأساسي والمشتق ، ثم يمرر الصف المشتق الوسائط اللازمة إلى الصف الأساسي . يبين البرنامج (4-7) التالي ، كيفية تمرير الوسائط إلى باني الصف المشتق وآخر لباني الصف الأساسي .

مثال (4-7)

```
#include <iostream .h>
class base {
```

private :

int i;

public:

base(int n) {

 cout << "Constructing base class\n";

 i = n;

}

~base() { cout << "Destructing base class\n"; }

void printi() { cout << i << '\n'; }

}; // End Base Class

//Inheritance Class

class derived : public base {

int j;

public:

derived(int n, int m) : base(m) { // pass arg to base class

 cout << "Constructing derived class\n";

 j = n;

}

~derived() { cout << "Destructing derived class\n"; }

void printj() { cout << j << '\n'; }

}; // End derived Class

```

int main( ) {
    derived obj(110,220);
    obj.printi( );
    obj.printj( );
    return 0;
} // End main

```

Result

Constructing **base** class

Constructing **derived** class

220

110

Destructing **derived** class

Destructing **base** class

Press any key to continue

مثال (8-4)

اكتب برنامجاً بلغة ++C للصف Name الذي يعرف طالباً ما ، حيث يقوم الصف Name بإدخال اسم الطالب الثلاثي last , mid , first كثلاث متحولات من نوع char ، أما الصف المشتق address فإنه يرث الصف الأساسي Name (يقوم بوراثة الاسم الثلاثي للطالب) ويضيف اسم الشارع

street والمدينة city والدولة state والرمز البريدي zip ، وهي متحولات من نوع char ، والبرنامج التالي يوضح ذلك .

```
#include <iostream.h>
#include <string.h>
class name
{
public:
    // default constructor
    name( ) { first[ 0]= mid[ 0]=last[ 0]='\0';}
    void set_name(char* ,char* ,char*);
    char* show( );
    ~ name( ){ cout << " constructing name \n" ; }
private:
    char first[20] , mid[20] , last[20] ;
}; // End class name

void name::set_name( char* f, char* m, char* l)
{
    strcpy (first ,f);
    strcpy (mid ,m );
    strcpy (last ,l);
} // End constructor Test
```

```

char* name ::show( )
{
    char* temp = new char[200];
        strcpy (temp , first );
        strcat (temp ," ");
        strcat (temp ,mid );
        strcat (temp ," ");
        strcat (temp ,last );
    return temp ;
} // End function show

// start of class address
class address : public name{
private :
    char street[40] , city[20] , state[20] , zip [20] ;
public :
    address( ) {
        street[ 0]=city[ 0]=state[ 0]=zip[ 0]='\0';}
    void  new_address(char* ,char* , char* ,char* );
    void  new_name( char* , char* , char* );
    char* show_info( );
    ~ address ( )
{

```

```

        cout << " Destructing address \n " ;
    }
}; // End class address

void address :: new_address( char* s , char* c, char* st ,
                             char* z )
{
    strcpy (street,s );
    strcpy (city,c );
    strcpy (state , st );
    strcpy (zip , z );
} // End function

void address :: new_name( char* f, char* m , char* l )
{
    set_name( f , m ,l);
} //End function

char* address :: show_info( )
{
    char* temp = show( ) ;
    strcat (temp , "\n" );
    strcat (temp ,street );
}

```

```

    strcat (temp , "\n" );
    strcat (temp , city );
    strcat (temp , " , " );
    strcat (temp , state );
    strcat (temp , " " );
    strcat (temp , zip);
    return temp ;
} //End function

void main( )
{
    cout<<" Address + Name \n";

    address  xob ;

    xob.new_address( "1234 khlid street " ,
                    " Damascus","Syria  " ,"99887");

    xob.new_name("Saleem","salem","Ali") ;

    cout<< "****"<< xob.show_info( ) <<"\n";

} // End main

```

Result

Address + Name

***Saleem salem Ali

1234 khlid street

Damascus , Syria 99887

Destructing address

constructing name

Press any key to continue

ملاحظات مهمة

□ يمكن كتابة تابع تمرير المعطيات في الصف المشتق كما يلي :

```
void address :: new_address(char* s , char* c, char* st,
                           char* z, char*f, char*m, char*l )
{
    set_name( f , m , l );
    strcpy (street,s );
    strcpy (city,c );
    strcpy (state , st );
    strcpy (zip , z );
} // End function
```


وبذلك نلغي التابع (new_name) من الصف المشتق ، أي يتم إلغاء التابع التالي :

```
void address :: new_name(char* f , char* m , char* l )
{
    set_name( f , m ,l);
} //End function
```

□ أما إذا كان الصف الأساسي يحتوي على تابع تمرير معطيات هو التابع الباني فقط . في هذه الحالة لا يستطيع الصف المشتق أن يرث التابع الباني وبالتالي يجب تمرير المعطيات إلى التابع الباني للصف المشتق ومنه إلى التابع الباني للصف الأساسي بالشكل التالي :

```
class Base {
.....
Base ( int f , int m ) ;.....
} ; // End class Base

class Derived {
.....
Derived( char i ,char j , int f ,int m ) : Base( f , m)
{
.....
}
} ; //End Derived class
```

يوضح المثال (4-9) التالي هذه الفكرة .

مثال (4-9)

```
#include <iostream.h>
#include <string.h>
class name {
public:
    // default constructor
    name( ) { first[20]= mid[20]=last[20]='\0';}
    name(char* ,char* ,char* );
    void printname( );
private:
    char first[20],mid[20],last[20] ;
}; // End class name

//constructor
name :: name( char* f,char* m,char* l){
    strcpy (first ,f);
    strcpy (mid ,m );
    strcpy (last ,l );
} // End constructor

void name ::printname( ) {
    cout<< first<<" "<<mid<<" "<<last<<endl;
} // End function show
```

```

// start of address class
class address : public name {
private :
    char street[40];
    char city[20] ; //extra member
    char state[20];
    char zip [20] ;

public :
    address( ){ street[0]=city[0]=state[0]=zip[0]='\0';}
    address( char* ,char* , char* ,char* ,char* ,char* ,char*);
    void show_info( );
}; // End class address

address :: address(char* s , char* c, char* st,char* z ,
                  char*f , char*m,char*l ) : name ( f,m,l)
{
    strcpy (street,s );
    strcpy (city,c );
    strcpy (state , st );
    strcpy (zip , z );
} // End constructor

```

```

void address :: show_info( )
{
    name :: printname( );
    cout<<street <<" "<<city<<" "<<state<<" "<<zip <<endl;
} //End show_info

void main( )
{
    cout<<"A ddress + Name\n";
    address ob( "1234 khlid street ", " Damascus","Syria ",
                "99887", "Saleem","salem","Ali" );
    ob.show_info( );
} // End main

```

RESULT

Address + Name

Saleem salem Ali

1234 khlid street Damascus Syria 99887

Press any key to continue

مثال (4-10)

اكتب برنامجاً بلغة C++ للصف الأساسي base (المثال (4-4)) ، ويرث الصف المشتق derived الصف الأساسي base، الذي يقوم بطباعة x ، y ومجموع x+y ، حيث المتحول x من نوع private ، والمتحول y من نوع protected ، ويحتوي الصف الأساسي base على بانٍ افتراضي وتابع مدمر وتابع تعديل معطيات base1 وتابع طباعة (printx) ، ويحتوي الصف المشتق derived على بانٍ افتراضي وتابع مدمر وتابع تعديل معطيات derived1 وتابع طباعة (printy) لطباعة المتحول y وتابع (show_sum) لطباعة مجموع x + y

```
#include <iostream.h>
class base {
private :
    int z ;
protected :
    int x
public :
    base( ) { x=0;cout<<"constructing base class \n";}
    ~base( ) {cout<<"constructing base class \n";}
    void base1(int n) { x=n;}
    void printx( ) {cout << x<<"\n";}
}; //End Base Class
```

```

// Inherite
class derived : public base {
    private :
        int y ;
    public :
        derived( ) { y=0; cout<<"constructing derived class \n"; }
        ~derived( ) { cout<<"destructing derived class \n"; }
        void derived1 ( int n) { y = n; }
        void show_sum( ) { cout<< x+y << '\n'; } //error
        void showy( ) { cout << y << '\n'; }
}; // End Derived class

int main ( )
{
    derived ob;
    ob.base1(40);
    ob.derived1 (10);
    ob.show_sum ( );
    ob.showx( );
    ob.showy( );
    return 0
} // End main

```


Result

constructing base class

constructing derived class

50

40

10

destructing derived class

constructing base class

Press any key to continue

مثال (4-11)

أكتب برنامج بلغة C++ للصف الأساسي student يحتوي على المتحولات الخاصة التالية :

❖ First الاسم الأول ، mid الاسم الوسط ، last الاسم الأخير ، id الرقم الجامعي وهي من نمط char .

□ لدينا الصف المشتق course يرث الصف الأساسي student ويحتوي على المتحولات الخاصة التالية :

❖ cid القسم ، description اسم المقرر ، grade النتيجة في المقرر .

□ يحتوي الصف student على تابع تعديل معطيات student1 وعلى تابع

طباعة () printstudent() ، ويحتوي الصف course على تابع باني

عادي وعلى تابع طباعة () showgrade() .

```

#include <iostream.h>
#include <string.h>
class student {
private:
char first[0] ,mid[0], last[0] ,id[0] ;
public:
void student1( char *,char*,char*,char* );
void printstudent( );
}; // End class student

// تابع تعديل معطيات
void student::student1( char *f,char*m,char*l,char*i )
{
strcpy(first ,f);
strcpy(mid ,m );
strcpy(last ,l );
strcpy(id ,i );
} // End student1

void student :: printstudent( )
{
cout<<"name:"<<endl;
cout<< first <<" " << mid<<" " << last

```

```

        <<" "<<id<<endl;
    } // End function print

    class course : public student {
    private:
        char cid[10];
        char description [128] ;
        char grade ;
    public:
        course ( char*,char*,char*,char*,char*,char* );
        void setgrade ( char g )
            { grade = g ; cout << grade << endl ;}
        void showgrade( ) ;
    }; // End class course

    course ::course ( char*f ,char*m ,char*l ,char*si ,
                                char*ci ,char*cd)
    {
        student1 ( f,m,l,si ) ;
        strcpy ( cid,ci);
        strcpy(description ,cd);
    }

```

```

Void course::showgrade( )
{
    student::printstudent( );
    cout << " course identification :" << cid << endl ;
    cout << "===== "<<endl;
    cout<< description<<endl ;
}

void main( ) {
    course xob ( " Ahmad","Salma","Zeinab", "012345678",
                "CIS25c", "Pogramming in C++ " );
    xob.setgrade( 'A');
    xob.showgrade( );
} // End main

```

Result

A

Name:

Ahmad Salma Zeinab 012345678

course identification :CIS 25c

=====

Pogramming in C++

Press any key to continue

لنفرض أن المبرمج أراد أن يجعل العنصر id في القسم الخاص قابلاً للتعامل من قبل الصف course عندها يجب وضع id ضمن محدد الوصول protected وعندها يجب إعادة كتابة تعريف الصف كما يلي :

```
class student {  
    private:  
        char first[20] ;  
        char mid[20] ;  
        char last[20] ;  
    protected :  
        char id[20] ;  
  
    public:  
        void student1( char *,char*,char*,char* );  
        void printstudent( );  
  
}; // End class student
```

إن الكلمة protected تجعل العنصر id ليس في القسم العام ولا الخاص فهو أصبح مرئياً لدى الصفوف المشتقة من الصف الأساسي student. ويمكن جعل id في القسم العام لكنه سوف يكون مرئياً لجميع الصفوف الأخرى الموجودة في البرنامج .

مثال (4-12)

اكتب برنامجاً بلغة C++ للصف الأساسي اسمه Rec يقوم بحساب مساحة مستطيل طوله L وعرضه W ويحتوي على تابع بانٍ عادي لتمرير المعطيات وتابع طباعة . ثم أنشئ صفاً مشتقاً اسمه colorrec يرث الصف الأساسي Rec ويقوم بتلوين المستطيل ويحتوي على وسيط من نوع int هو color ويحتوي الصف على تابع بانٍ عادي لتمرير المعطيات وعلى تابع طباعة.

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

```
class rec {
```

```
public:
```

```
    // constructor
```

```
    rec( float ,float );
```

```
    float getw( ){return w ;}
```

```
    float getl( ){return l ;}
```

```
    void print( ) ;
```

```
private:
```

```
    float w, l ;
```

```
}; // End class rec
```

```
    // constructor
```

```
rec::rec( float w1 ,float l1 )
```

```
{
```

```
    w=w1;l=l1;
```

```
} // End constructor rec
```



```

void rec::print( )
{
    cout<< "area = " << l * w << endl;
} // End function print

    // start of class colorrec
class colorrec : public rec {
private :
    int color ;        //extra member
public :
    colorrec( int col,float w1,float l1 ) : rec(w1,l1)
    {
        color = col ;
    }
    void printcolor( ) { cout << "color="<<color<<"\n";
                        rec :: print( );
                    }
    int getcolor( ){ return color; }
}; // End collerrec

void main( ) {
    colorrec x(10,5,6);
    x.printcolor ( );
    cout<<endl;
} // End main

```

Result

color=10

area = 30

Press any key to continue

مثال (4-13)

اكتب برنامجاً بلغة C++ لإنشاء صف يدعى Building بحيث يخزن عدد الطوابق floors في البناء وعدد الغرف rooms ومساحة الغرف meter بالمتر المربع وهي متغيرات محمية protected . ثم أنشئ صفاً مشتقاً اسمه House يرث الصف Building ويخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة . ثم أنشئ صفاً مشتقاً آخر من الصف الأساسي Building اسمه Office يرث الصف Building ويخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة.

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

```
class building {
```

```
protected:
```

```
int floors;
```

```
int rooms;
```

```
double meter;
```

```
}; //End building
```

```
class house : public building {
```

```
private:
```

```
int bedrooms;
```

```
int bathrooms;
```

```
public:
```

```
house(int f, int r, double m , int br, int bth) {
```

```
    floors = f; rooms = r; meter = m;
```

```
    bedrooms = br; bathrooms = bth;
```

```
}
```

```
void show( ) {
```

```
    cout << "floors: " << floors << "\n";
```

```
    cout << "rooms: " << rooms << "\n";
```

```
    cout << "square meter : " << meter << "\n";
```

```
    cout << "bedrooms: " << bedrooms << "\n";
```

```
    cout << "bathrooms: " << bathrooms << "\n";
```

```
}
```

```
}; // End house class
```

```
class office : public building {
```

```
private:
```

```
int phones ,extinguishers ;
```

public:

```
office( int f, int r, double m, int p, int ext)
{
    floors = f; rooms = r; meter = m;
    phones = p; extinguishers = ext;
}

void show( ) {
    cout << "floors: " << floors << "\n";
    cout << "rooms: " << rooms << "\n";
    cout << "square meter: " << meter << "\n";
    cout << "Telephones: " << phones << "\n";
    cout << "fire extinguishers: ";
    cout << extinguishers << "\n";
}

}; // End office class
```

```
int main( ) {
    house h_ob(2, 12, 5000, 6, 4);
    office o_ob(4, 25, 12000, 30, 8);
    cout << "House: \n";
    h_ob.show( );
    cout << "\nOffice: \n";
    o_ob.show( );
}
```

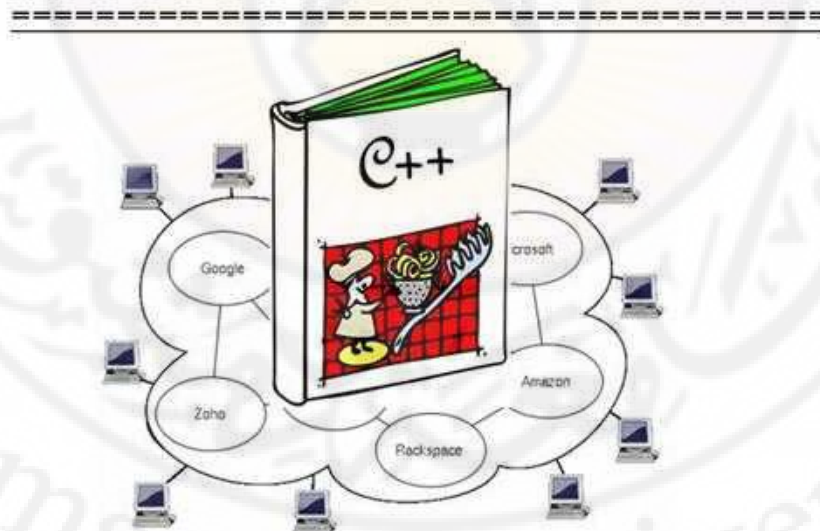
```
return 0;  
} // End main
```

Result

House:
floors: 2
rooms: 12
square meter: 5000
bedrooms: 6
bathrooms: 4

Office:
floors: 4
rooms: 25
square meter: 12000
Telephones: 30
fire extinguishers: 8

Press any key to continue





تمارين



١- اكتب برنامجاً بلغة C++ لإنشاء صف يدعى Building بحيث يخزن عدد الطوابق floors في البناء وعدد الغرف rooms ومساحة الغرف meter بالمتر المربع وهي متغيرات خاصة private ، ويحتوي على بانٍ عادي وتابع طباعة. ثم أنشئ صفاً مشتقاً اسمه House يرث الصف Building ويخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة . ثم أنشئ صفاً مشتقاً آخر من الصف الأساسي Building اسمه Office يرث الصف Building ويخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة.

٢- اشرح ماذا يحدث للأعضاء العامة لصف أساس عندما يرثه صف مشتق باستخدام المحدد public ، وما الذي يحدث للأعضاء الخاصة . وإذا كانت الوراثة باستخدام المحدد private ما الذي يحدث للأعضاء العامة والخاصة .

٣- اشرح ماذا تعني الكلمة المحجوزة protected في الحالتين كلتيهما : عندما تشير إلى أعضاء من صف وعندما تستخدم كمحدد وصول في الوراثة .

٤- عندما يرث صف ما صفاً آخر ، متى يستدعي التابع الباني والتابع المدمر للصف ؟

٤-٤-٤ - الوراثة المتعددة

Multiple Inheritance

توجد طريقتان يستطيع من خلالها الصف المشتق أن يرث عدة صفوف أساسية :

- ١- يمكن أن يكون الصف المشتق صفاً أساسياً لصف مشتق آخر
- ٢- يمكن أن يرث الصف المشتق بشكل مباشر أكثر من صف أساسي

٤-٤-١ - الصف المشتق صف أساسي لصف مشتق آخر

يمكن أن يكون الصف المشتق صفاً أساسياً لصف مشتق آخر مشكلاً بذلك مراتب متعددة المستويات من الصفوف ، ويصبح الصف الأساسي الأصلي صفاً غير مباشر للصف المشتق الثاني ، كما هو مبين بالشكل (٤-١) .



الشكل (٤-١) - يوضح الوراثة المتعددة

وعند استخدام الصف المشتق كصف أساسي لصف مشتق آخر ، فإن البواني للصفوف الثلاثة تستدعي حسب ترتيب الاشتقاق ، أما التابع المدمر فيستدعي بعكس ترتيب الاشتقاق .

نلاحظ أن B1 هو صف أساسي غير مباشر للصف D2 ، وأن D2 يستطيع الوصول إلى الأعضاء العامة لكل من D1 و B1 .
من الشكل (٤-١) نرى أن الباني للصف B1 يستدعي أولاً ثم الباني للصف D1 ثم للصف D2 ، أما المدمر فيستدعي المدمر للصف المشتق D2 ثم للصف D1 ثم للصف الأساسي B1 .
يبين البرنامج في المثال (4-14) التالي كيف يرث الصف المشتق D2 كلاً من D1 و B1 وكيف يرث D1 الصف الأساسي B1 .

مثال (4-14)

// Multiple Inheritance

#include <iostream.h>

class B1 {

int a;

public:

B1(int x) { a = x; }

int geta() { return a; }

}; // End class B1

// Inherit direct base class.

class D1 : public B1 {

int b;

public:

```
D1(int x, int y) : B1(y) { // pass y to B1
    b = x;
}
int getb( ) { return b; }
}; // End Class D1
```

// Inherit a derived class and an indirect base.

```
class D2 : public D1 {
```

```
    int c;
```

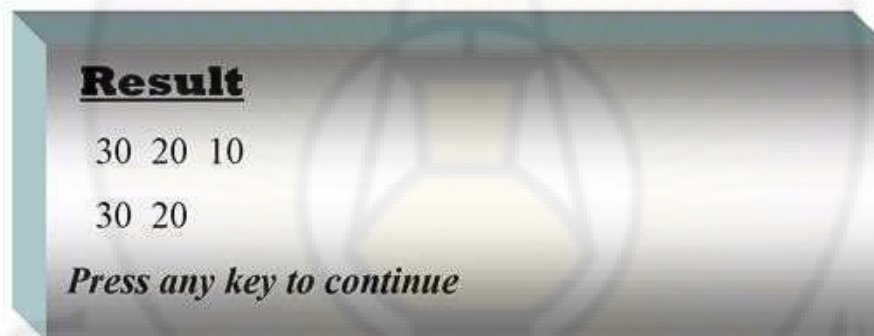
public:

```
D2( int x, int y, int z) : D1(y, z)    // pass args to D1
{
    c = x;
}
/* Because bases inherited as public, D2 has access
   to public elements of both B1 and D1. */
void show( ) {
    cout << geta( ) << ' ' << getb( ) << ' ';
    cout << c << '\n';
}
}; // End Class D2
```

```

int main( )
{
    D2 ob(10, 20, 30);
    ob.show( );
    // geta( ) and getb( ) are still public here
    cout << ob.geta( ) << ' ' << ob.getb( ) << '\n';
    return 0;
} // End main

```



ملاحظات

- عند استخدام الصف المشتق كصف أساسي لصف مشتق آخر فإن البواني للصفوف الثلاثة D2 , D1 , B1 تُستدعى حسب ترتيب الاشتقاق ، أما المدمرات فتُستدعى بشكل معاكس .
- إن البواني لا تورث في الصفوف المشتقة ، لذلك فإن كل صف مشتق يجب أن يصرح عن البواني الخاصة به .

□ توريث الأعضاء الخاصة بواسطة الصف المشتق لكن لا يمكن الوصول إليها من قبل الصف المشتق ، وللوصول إليها علينا التصريح عنها كأعضاء محمية protected .

مثال (4-15)

أعد كتابة المثال (4-13) بحيث يكون الصف Building هو الصف الأساسي ، أما الصف House فهو مشتق من الصف Building ، والصف Office مشتق من الصف House .

Office ← House ← Building

ويكون السؤال كما يلي :

اكتب برنامجاً بلغة C++ لإنشاء صف يدعى Building بحيث يخزن عدد الطوابق floors في البناء وعدد الغرف rooms ومساحة الغرف meter بالمتر المربع وهي متغيرات خاصة private ، ويحتوي على بانٍ عادي وتابع طباعة. ثم أنشئ صفاً مشتقاً اسمه House يرث الصف Building ويخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي متغيرات خاصة، ويحتوي على بانٍ عادي وتابع طباعة . ثم أنشئ صفاً مشتقاً آخر من الصف المشتق الأول House اسمه Office يرث الصف House ويخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة.

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

```
class Building {
```


private:

int Floors;

int Rooms;

double Sqmeter;

public :

Building() { Floors =0;Rooms=0;Sqmeter=0;}

Building (**int** f,**int** r ,**double** m)

{ Floors =f;

Rooms=r;

Sqmeter=m;

}

void Printbuilding()

{

cout << " Rooms =" <<Rooms<< endl;

cout << " Floors =" <<Floors<< endl;

cout << " Sqmeter =" << Sqmeter<< endl;

}

}; // End Building

class House : **public** Building {

private:

int Bedrooms;

int Bathrooms;

public:

```
House ( ) { Bedrooms = 0 ; Bathrooms = 0 ; }
```

```
House(int f, int r, double m , int br, int bth) :
```

```
    Building( f,r,m)
```

```
{  
    Bedrooms = br; Bathrooms = bth;  
}
```

```
void PrintHouse( )
```

```
{  
    Building :: Printbuilding ( ) ;  
    cout << "bedrooms: " << Bedrooms << '\n';  
    cout << "bathrooms: " << Bathrooms << '\n';  
}  
}; // End House
```

```
class Office : public House {
```

```
private:
```

```
    int    Phones;
```

```
    int    Extinguishers;
```

```
public:
```

```
    Office(int f, int r, double m, int br , int bth,
```

```
        int p, int ext) :House ( f,r,m,br,bth)
```

```

    {
        Phones = p; Extinguishers = ext;
    }

void PrintOffice( )
{
    House :: PrintHouse ( ) ;
    cout << "Telephones: " << Phones << "\n";
    cout << "fire extinguishers: ";
    cout << Extinguishers << "\n";
}

}; // End Office

int main( )
{
    Office O_ob(4, 25, 12000, 30, 8,33,44);
    cout << "\nOffice: \n";
    O_ob.PrintOffice( ) ;

    return 0;

}; // End main

```

Result

Office:

Rooms =25

Floors =4

Sqmeter =12000

bedrooms: 33

bathrooms: 44

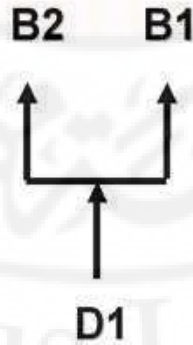
Telephones: 33

fire extinguishers: 44

Press any key to continue

٤-٢-٤ - الصف المشتق يرث أكثر من صف أساسي

يمكن أن يرث الصف المشتق بشكل مباشر أكثر من صف أساسي .
وفي هذه الحالة يتحد صفان أو أكثر لتشكيل الصف المشتق كما هو مبين
بالشكل (٢-٤) .



الشكل (٢-٤) - يوضح الوراثة المتعددة لصفين أساسيين

وعندما يرث صف أساسي وبشكل مباشر عدة صفوف أساسية نستخدم التصريح التالي :

```
class derived_name : Access B1 , Access B2,... , Access Bn
{
    // Body Of Class
}
```

وتمثل B1 , B2 , ... Bn أسماء الصفوف الأساسية ، ويمثل Access محدد الوصول الذي يختلف من صف إلى آخر . وعند وراثة صفوف متعددة أساسية و يتم تنفيذ البواني حسب ترتيبها في عبارة التصريح من اليسار إلى اليمين ، أما المدمرات بترتيب معاكس .

الصف المشتق يمرر الوسائط اللازمة لكل صف أساس باستخدام الشكل التالي :

```
derived _ Constructor ( arg_List ) : B1(arg_List) ,
                                B2(arg_List) , .... , Bn(arg_List)
{
    // Body Of Derived Class Constructor
}
```

ملاحظة

عندما يرث صف مشتق ترتيب ما من الصفوف فإن كل صف مشتق في السلسلة يجب أن يمرر للصف الأساسي السابق الوسائط التي يحتاجها .

مثال (4-16)

أعد كتابة المثال (4-14) السابق حيث يرث الصف المشتق D2 كلاً من الصفتين الأساسيين B1 و D1 .

```
// Multiple Inheritance
```

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

```
class B1 {
```

```
private :
```

```
    int a;
```

```
public:
```

```
    B1(int x) { a = x; }
```

```
    int geta() { return a; }
```

```
}; // End class B1
```

```
// Inherit direct base class.
```

```
class D1 : public B1 {
```

```
private :
```

```
    int b;
```

```
public:
```

```
    D1( int x, int y) : B1(y)    // pass y to B1
```

```
{
```

```
    b = x;
```

```
}
```

```

    int getb( ) { return b; }
}; // End Class D1

// Inherit a derived class and an indirect base.
class D2 : public D1 {
private :
    int c;
public:
    D2( int x, int y, int z) : D1(y, z) // pass args to D1
    {
        c = x ;
    }
    /* Because bases inherited as public, D2 has access
       to public elements of both B1 and D1. */
    void show( ) {
        cout << geta( ) << ' ' << getb( ) << ' ';
        cout << c << '\n';
    }
}; // End Class D2

int main( )
{
    D2 ob(1, 2, 3);

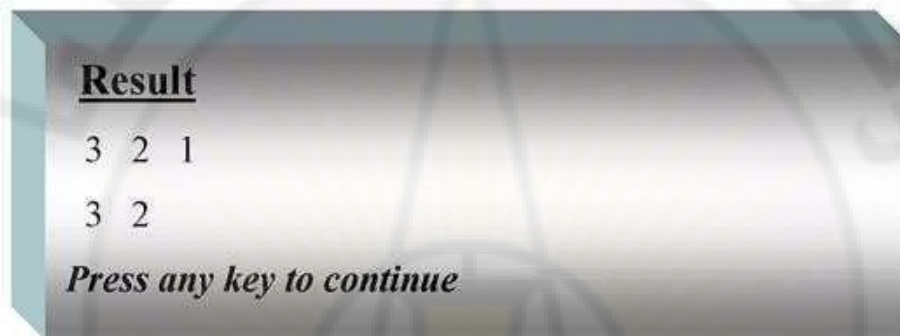
```



```

ob.show( );
// geta( ) and getb( ) are still public here
cout << ob.geta( ) << ' ' << ob.getb( ) << '\n';
return 0;
} // End main

```



مثال (4-17)

أعد كتابة المثال (4-15) بحيث يكون الصف Building هو الصف الأساسي الأول، والصف House الصف الأساسي الثاني، والصف Office هو مشتق من الصف House والصف Building ، كما هو مبين في الشكل (4-3) .



الشكل (4-3) - يوضح الوراثة لصفين

ويكون السؤال كما يلي :

اكتب برنامجاً بلغة ++C لإنشاء صفين أساسيين : يدعى الصف الأساسي الأول **Building** بحيث يخزن عدد الطوابق floors في البناء وعدد الغرف rooms ومساحة الغرف meter بالمتر المربع وهي متغيرات خاصة private ، ويحتوي على بانٍ عادي وتابع طباعة، والصف الأساسي الثاني اسمه **House** يخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة . ثم أنشئ صفاً مشتقاً يرث الصفين الأساسيين **Building** و **House** اسمه **Office** يخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة.

البرنامج

```
#include <iostream.h>

// first Base Class
class Building {
private:
    int Floors ;
    int Rooms ;
    double Sqmeter ;
public :
    Building( ) { Floors =0;Rooms=0;Sqmeter=0;}
    Building ( int f ,int r ,double m )
    {
        Floors =f;
        Rooms=r;
```

```

        Sqmeter=m;
    }
    void Printbuilding( )
    {
        cout << " Rooms =" <<Rooms<< endl;
        cout << " Floors =" <<Floors<< endl;
        cout << " Sqmeter =" << Sqmeter<< endl;
    }
}; // End Building

// Second Base Class
class House {
private :
    int Bedrooms;
    int Bathrooms;
public:
    House ( ) { Bedrooms = 0 ; Bathrooms = 0 ; }
    House( int br, int bth)
    {
        Bedrooms = br; Bathrooms = bth;
    }
    void PrintHouse( ) {
        cout << "bedrooms: " << Bedrooms << "\n";
    }
};

```

```

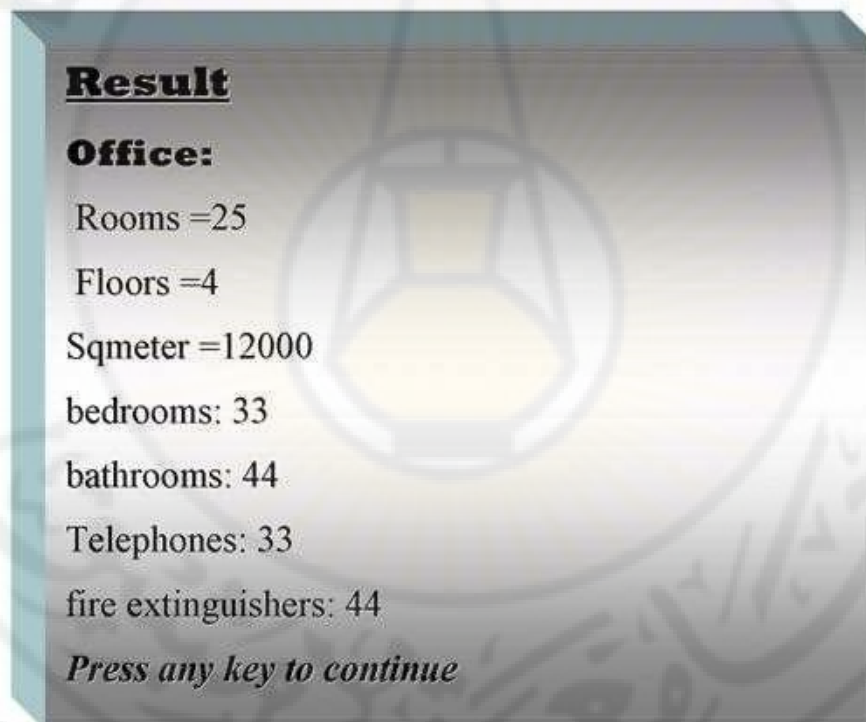
        cout << "bathrooms: " << Bathrooms << "\n";
    }
}; //End House
// Derived Class
class Office : public Building , public House {
private :
    int Phones;
    int Extinguishers;
public:
    Office( int f, int r, double m, int br, int bth, int p, int ext):
        Building( f,r,m) , House( br, bth)
    {
        Phones = p; Extinguishers = ext;
    }
    void PrintOffice( )
    {
        Building :: Printbuilding ( ) ;
        House ::PrintHouse( ) ;
        cout << "Telephones: " << Phones << "\n";
        cout << "fire extinguishers: ";
        cout << Extinguishers << "\n";
    }
}; // End Office

```

```

int main( )
{
    Office Office-ob(4, 25, 12000, 30, 8,44 ,33 );
    cout << "\n Office: \n";
    Office-ob.PrintOffice( );
    return 0;
} //End main

```



يوضح المثال (4 - 18) التالي كيف يتم استدعاء البواني والمدمرات في المثال (4 - 17) السابق .

مثال (18-4)

```
#include <iostream.h>

// first Base Class
class B1 {
public:
    B1( ) { cout << "Constructing B1\n"; }
    ~ B1( ) { cout << "Destructing B1\n"; }
}; // End Class B1

// second Base Class
class B2 {
private:
    int b;
public:
    B2( ) { cout << "Constructing B2\n"; }
    ~ B2( ) { cout << "Destructing B2\n"; }
}; // End class B2

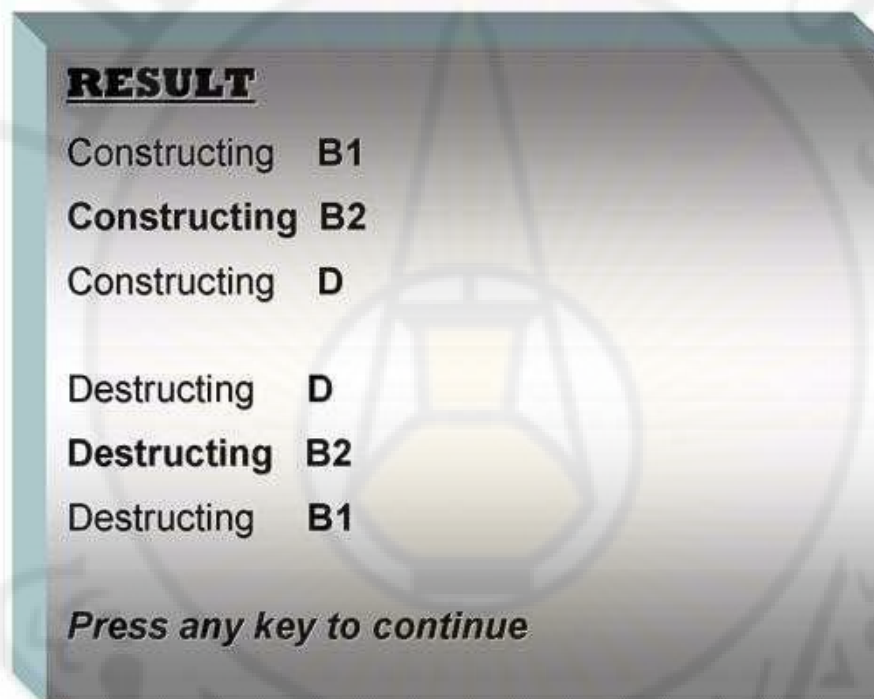
//Inherit two base classes.
class D : public B1, public B2 {
public:
    D( ) { cout << "Constructing D\n"; }
    ~ D( ) { cout << "Destructing D\n"; }
}; // End Class D
```



```

int main( ) {
    D ob;
    return 0;
} // End main

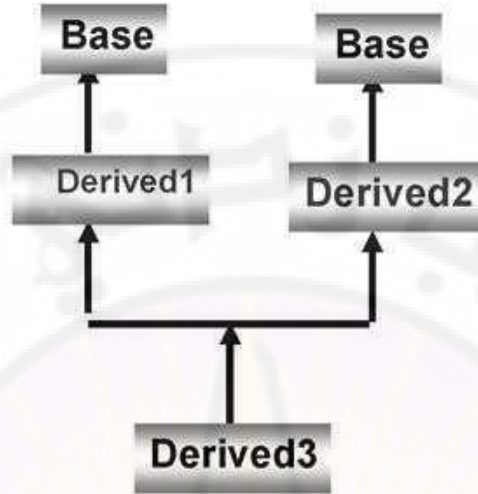
```



٤-٤-٣- الصفوف الأساسية الافتراضية

Virtual Base Class

ليكن لدينا المخطط الوراثي المبين في الشكل (٤-٤) التالي :



الشكل (٤-٤) - المخطط الوراثي

يرث الصفان المشتقان Derived1 و Derived2 الصف الأساسي Base وفي الوقت نفسه يرث الصف Derived3 الصفين Derived1 و Derived2 مباشرة ، هذا يعني أن الصف Derived3 يرث الصف الأساس Base مرتين بشكل غير مباشر .

الأولى : عن طريق الصف Derived1

والثانية : عن طريق Derived2

وبالتالي سيحدث نوع من الالتباس عندما يستخدم الصف Derived3 أحد أعضاء الصف الأساسي Base ، حيث أن Derived3 يتضمن نسختين منه . ولحل هذا الالتباس تضمنت C++ آلية يتم بواسطتها تضمين نسخة واحدة من الصف الأساسي الافتراضي Virtual Base Class ، وفي هذه الحالة يمكن أن يرث صف مشتق بشكل غير مباشر صفاً أساسياً أكثر من مرة وبذلك نستخدم الوراثة الافتراضية لمنع تشكيل نسختين أو أكثر من الصف الأساسي في أي صف مشتق يرث هذا الصف الأساسي بشكل غير مباشر .

مثال (4-19)

يستخدم هذا المثال صف أساسي افتراضي لمنع تواجد نسختين من الصف Base في الصف المشتق Derived3 .

// This program uses a virtual base class.

#include <iostream.h>

class base {

public:

int i;

};*// End class base*

// Inherit base as virtual.

class derived1 : virtual public base {

public:

int j;

};*// End class derived1*

// Inherit base as virtual, here too.

class derived2 : virtual public base {

public:

int k;

};*// End class derived2*

/ Here, derived3 inherits both derived1 and derived2.*

However, only one copy of base is present.

**/*

```
class derived3 : public derived1, public derived2 {
```

```
public:
```

```
    int product( ) { return i * j * k; }
```

```
}; End class derived3
```

```
int main( ) {
```

```
    derived3 ob;
```

```
    ob.i = 100; // unambiguous because only
```

```
                // one copy present
```

```
    ob.j = 7;
```

```
    ob.k = 8;
```

```
    cout << "Product is " << ob.product( ) << "\n";
```

```
    return 0;
```

```
} // End main
```

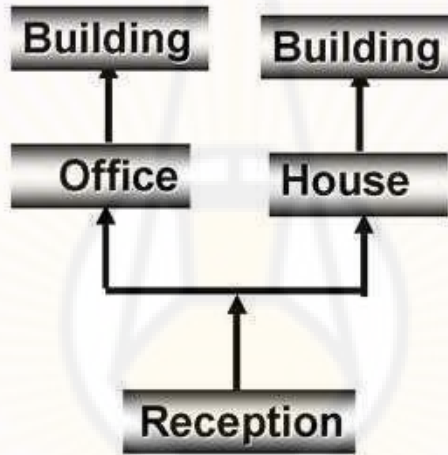
Result

Product is 5600

Press any key to continue

مثال (4-20)

أعد كتابة المثال (4-13) بحيث يكون الصف **Building** هو الصف الأساسي (صف أساسي افتراضي) ، أما الصف **House** مشتق من الصف **Building** والصف **Office** مشتق من الصف **Building** أيضاً ، أما الصف **Reception** يرث الصف **House** و الصف **Office** ، كما هو موضح في الشكل (4-5)



الشكل (4-5) - يبين الوراثة المتعددة (الصف الافتراضي)

اكتب برنامجاً بلغة C++ لإنشاء صف يدعى **Building** بحيث يخزن عدد الطوابق **floors** في البناء وعدد الغرف **rooms** ومساحة الغرف **meter** بالمتر المربع وهي متغيرات خاصة **private** ، ويحتوي على باني عادي وتابع طباعة. ثم أنشئ صفاً مشتقاً اسمه **House** يرث الصف **Building** ويخزن عدد غرف النوم **Bedrooms** وعدد الحمامات **Bathrooms** وهي متغيرات خاصة ، ويحتوي على باني عادي وتابع طباعة . ثم أنشئ صفاً مشتقاً آخر من **Building**

اسمه Office يرث الصف Building ، ويخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة ، ويحتوي على بانٍ عادي وتابع طباعة. ثم أنشئ صفاً يرث الصفين House و Office اسمه Reception يخزن عدد الموظفين Employed وعدد الغرف FreeRoom الفارغة وهي متغيرات خاصة ويحتوي على بانٍ عادي وتابع طباعة .

البرنامج :

والصف Reception يرث الصفين House و Office وبالتالي سوف يكون لديه نسختين من الصف الأساسي Building ، ولحل هذا الالتباس يجب تضمين التعليمة Virtual في الصف House والصف Office . كما هو مبين في البرنامج التالي .

```
#include <iostream.h>
class Building {
private:
    int Floors , Rooms ;
    double Sqmeter;
public :
    Building( ) { Floors =0;Rooms=0;Sqmeter=0;}
    Building ( int f ,int r ,double m )
    { Floors =f;
      Rooms=r;
      Sqmeter=m;
    }
```



```

void Printbuilding( )
{
    cout << " Rooms =" << Rooms << endl;
    cout << " Floors =" << Floors << endl;
    cout << " Sqmeter =" << Sqmeter << endl;
}
}; //End Class Building

```

```

class House : virtual public Building {
private :
    int Bedrooms;
    int Bathrooms;
public:
    House ( ) { Bedrooms = 0 ; Bathrooms = 0 ; }
    House( int f, int r, double m , int br, int bth) :
        Building( f,r,m)
    {
        Bedrooms = br; Bathrooms = bth;
    }
void PrintHouse( ) {
    Building ::Printbuilding ( ) ;
    cout << "bedrooms: " << Bedrooms << "\n";
    cout << "bathrooms: " << Bathrooms << "\n";
}

```

```

    }
}; // End Class House

class Office : virtual public Building {
private :
    int Phones;
    int Extinguishers;
public:
    Office( int f, int r, double m, int p, int ext) :
        Building ( f, r, m)
    {
        Phones = p; Extinguishers = ext;
    }
    void PrintOffice() {
        Building ::Printbuilding ( ) ;
        cout << "Telephones: " << Phones << "\n";
        cout << "fire extinguishers: ";
        cout << Extinguishers << "\n";
    }
}; // End Class Office

```

```

class Reception : public House, public Office {
private :

```

```

    int Employed,FreeRoom;
public :
    Reception( int f, int r, double m , int br, int bth,int p,int
               ext,int w ,int free ) : House ( f,r,m,br,bth) ,
                                       Office ( f,r,m,p,ex
    {
        Employed = w; FreeRoom = free;
    }
    void PrintReception( ) {
        Office :: PrintOffice( ) ;
        House :: PrintHouse( );
        cout << "Employed: " << Employed << "\n";
        cout << "FreeRoom: " << FreeRoom << "\n";
    }
}; // End Class Reception

int main( )
{
    Reception ob(4, 100, 12000.5, 30, 8,33,44,33,5);
    cout << "\n Reception: \n";
    ob.PrintReception( );
    return 0;
} // End main

```

Result

Reception:

Rooms =100

Floors =4

Sqmeter =12000.5

bedrooms: 30

bathrooms: 8

Telephones: 53

fire extinguishers: 44

Employed: 33

FreeRoom: 5

Press any key to continue



مسائل عامة محلولة

١ - أنشئ هرمية صفوف من ثلاثة مستويات مع بوان ومدمرات افتراضية ، تطبع رسائل على الخرج تحقق من أنه من أجل غرض من الصف الأكثر اشتقاقاً ، يتم استدعاء البواني الافتراضية والمدمرات الثلاثة تلقائياً ، فسر ترتيب الاستدعاء الذي حصل .

```
#include <iostream.h>

class A {
public:
    A( ) {cout << "A::A( )\n";}
    ~A( ) {cout << "A::~~A( )\n";}
}; //End class A

class B : public A {
public:
    B( ) {cout << "B::B( )\n";}
    ~B( ) {cout << "B::~~B( )\n";}
}; // End class B

class C : public B {
```

public:

```
C( ) {cout << "C::C( )\n";}  
~C( ) {cout << "C::~~C( )\n";}  
}; // End class C
```

void main()

```
{  
    C ob ;  
} // End main
```

Result

```
A::A( )  
B::B( )  
C::C( )  
C::~~C( )  
B::~~B( )  
A::~~A( )
```

Press any key to continue

يتم دائماً بناء مكونات الغرض قبل بناء الغرض نفسه . بتطبيق هذا المبدأ بشكل عودي نستنتج أنه يجب أولاً إنشاء الأغراض الجزئية في قمة هرمية الوراثة ، وتليها الأغراض الأدنى منها ، وهكذا ، أما المدمرات فيتم دائماً بعكس ترتيب البناء .

٢- أنشئ صفاً فيه تابعان عضوان ساكنان ، قم بالوراثة من هذا الصف ، وأعد تعريف أحد هذين التابعين . أظهر أنه سيتم إخفاء التابع الآخر في الصف الموروث .

```
#include <iostream.h>

class Base {
public:
    static void f( ) {
        cout << " Base::f( ) \n";
    }
    static void g( ) {
        cout << " Base::g( ) \n";
    }
}; //End class Base

class Derived : public Base {
public:
    static void g( int ) {
        cout << "Derived::g( int ) \n";
    }
}; // End class Derived

void main( ) {
    Derived a ;
```

```

a.f( );
a.g( 1 );
// Derived::g( ); // Error: too few parameters
} // End main

```

Result

Base::f()

Derived::g(int)

Press any key to continue

لا فائدة من الوسيط الموجود في التابع () g في الصف الموروث Derived ، حيث سيؤدي تعريف أي تابع يحمل اسم تابع موجود في الصف الأساسي إلى إخفاء جميع التوابع المحملة بشكل زائد من الاسم نفسه في الصفوف الأساسية ، لأن الصفوف المشتقة تشكل مجال رؤية داخلي جديد .

٣- أنشئ صفًا يحوي عضو معطيات وصفًا مشتقًا يضيف عضو معطيات آخر . اكتب تابع غير عضو يأخذ غرضاً من الصف الأب بالقيمة ويطبع حجم ذلك الغرض باستخدام sizeof ، وقم داخل التابع () main بإنشاء غرض من الصف المشتق ، وأطبع حجمه ، واستدع تابعك ، وفسر ما يحدث .

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

```
class Base {
```

```

private :
    int x;
}; // End class Base

class Derived : public Base {
    private :
        float y;
}; // End class Derived

void mySize(Base b) {
    cout << "mySize == " << sizeof b << endl;
}

void main( )
{
    Base b ;
    Derived d ;

    cout << "sizeof b == " << sizeof b << endl;
    cout << "sizeof d == " << sizeof d << endl;
    mySize(b);
    mySize(d);

} // End main

```

Result

sizeof b == 4

sizeof d == 8

mySize == 4

mySize == 4

Press any key to continue

٣- مهمة هذا التمرين إنشاء نموذج تصميم (design pattern) يدعى proxy ، ابدأ من الصف الأب والذي هو subject وضع فيه ثلاثة توابع : f() و g () و h () . ثم أنشئ الصفوف proxy و Implementation1 و Implementation2 ، واجعلها ترث من الصف subject . يجب أن يحتوي الصف proxy على مؤشر إلى الصف subject ، وعلى جميع التوابع الخاصة بالصف proxy أن تقوم بجميع الاستدعاءات من خلال المؤشر إلى subject . يأخذ باني الصف proxy مؤشراً إلى subject بحيث يتم تحميله داخل الصف proxy (عن طريق الباني عادة) . أنشئ في التابع الرئيسي (main) غرضين من الصف proxy مستخدماً التحققين المختلفين . ثم عدل الصف proxy بحيث يصبح بإمكانك تغيير التحقق ديناميكياً .

البرنامج :

```
#include <iostream.h>
class Subject {
public:
    virtual void f( ) = 0;
```

```

virtual void g( ) = 0;
virtual void h( ) = 0;
}; // End class Subject

class Proxy : public Subject {
    Subject* pSubject;
public:
    Proxy(Subject* pSubject = 0) {
        This ->pSubject = pSubject;
    }
    void setSubject(Subject* pSubject) {
        this ->pSubject = pSubject;
    }
    void f( ) {
        pSubject ->f( );
    }
    void g( ) {
        pSubject ->g( );
    }
    void h( ) {
        pSubject ->h( );
    }
}; // End class Proxy

```

```
class Implementation1 : public Subject {
```

```
public:
```

```
void f( ) {  
    cout << "Implementation1::f\n";
```

```
}
```

```
void g( ) {  
    cout << "Implementation1::g\n";
```

```
}
```

```
void h( ) {  
    cout << "Implementation1::h\n";
```

```
}
```

```
}; // End class Implementation1
```

```
class Implementation2 : public Subject {
```

```
public:
```

```
void f( ) { cout << "Implementation2::f\n"; }
```

```
void g( ) { cout << "Implementation2::g\n"; }
```

```
void h( ) { cout << "Implementation2::h\n"; }
```

```
}; // End class Implementation2
```

```
void main( ) {
```

```
    Implementation1 impl1;
```

```
    Proxy proxy(&impl1);
```



```

proxy.f( );
proxy.g( );
proxy.h( );
Implementation2 impl2;
proxy.setSubject(&impl2);
proxy.f( );
proxy.g( );
proxy.h( );
} // End main

```

Result

```

Implementation1::f
Implementation1::g
Implementation1::h
Implementation2::f
Implementation2::g
Implementation2::h

```

Press any key to continue

من الطبيعي جعل الصف `subject` مجرداً طالما أنه يمثل الواجهة الوظيفية التي نريدها . تزود الصفوف `Implementation1` و `Implementation2` بتحقيقات مختلفة للواجهة نفسها بالطبع . يجب أيضاً على الصف `proxy` أن يحقق الواجهة

بشكل كامل طالما أنه النمط الأساسي الذي يستخدمه الزبون . وثأتي مرونة هذا النموذج من إمكانية تغيير التحقيق دون تغيير الغرض الذي يستخدمه الزبون.

=====

٤- أنشئ هرمية وراثية صفها الأساسي Rodent (قارض) مكونة من الصفوف : (.. , Hamster , Gerbil , Mouse) . وزود الصف الأساسي بطرائق شائعة لجميع الصفوف من النمط Rodent ، وأعد تعريف هذه الطرائق في الصفوف المشتقة لانجاز سلوكيات مختلفة اعتماداً على النمط المعين من الصف Rodent . أنشئ مصفوفة من المؤشرات إلى النمط Rodent ، واملأها بأنماط معينة مختلفة منه ، واستدع طرائق الصف الأساسي لترى ماذا يحدث عدل بعد ذلك الصف Rodent بحيث يصبح صفاً أساسياً مجرداً نقياً .

البرنامج :

```
#include <iostream.h>
#include <vector>
#include <stddef.h> // For size_t
using std :: vector ;
class Rodent {
public:
    virtual ~Rodent( ) {cout << "~Rodent\n";}
    virtual void eat( ) = 0;
    virtual void infest( ) = 0;
    virtual void squeak( ) = 0;
}; // End class Rodent
```

```

class Mouse : public Rodent {
public:
    ~Mouse( ) {cout << "~Mouse\n";}
    void eat( ) {cout << "Mouse::eat\n";}
    void infest( ) {cout << "Mouse::infest\n";}
    void squeak( ) {cout << "Mouse::squeak\n";}
}; // End class Mouse

```

```

class Gerbil : public Rodent {
public:
    ~Gerbil( ) {cout << "~Gerbil\n";}
    void eat( ) {cout << "Gerbil::eat\n";}
    void infest( ) {cout << "Gerbil::infest\n";}
    void squeak( ) {cout << "Gerbil::squeak\n";}
}; // End class Gerbil

```

```

class Hamster : public Rodent {
public:
    ~Hamster( ) {cout << "~Hamster\n";}
    void eat( ) {cout << "Hamster::eat\n";}
    void infest( ) {cout << "Hamster::infest\n";}
    void squeak( ) {cout << "Hamster::squeak\n";}
}; // End class Hamster

```

```

void doRodentThings(vector < Rodent* >& v) {
    for (size_t i = 0; i < v.size( ); ++i)
    {
        v[i]->eat( );
        v[i]->infest( );
        v[i]->squeak( );
    }
}

```

```

void main( )
{
    vector< Rodent* > v;
    v.push_back(new Mouse);
    v.push_back(new Gerbil);
    v.push_back(new Hamster);
    doRodentThings(v);

    // Clean up:
    for (int i = 0; i < v.size( ); ++i)
        delete v[i];

} // End main

```

Result

Mouse::eat

Mouse::infest

Mouse::squeak

Gerbil::eat

Gerbil::infest

Gerbil::squeak

Hamster::eat

Hamster::infest

Hamster::squeak

~Mouse

~Rodent

~Gerbil

~Rodent

~Hamster

~Rodent

Press any key to continue

يوضح البرنامج السلوك المتعدد الأشكال للتوابع الافتراضية . على الرغم من وجود شعاع من المؤشرات إلى الأغراض Rodent إلا أن الشيفرة الفعلية التي يتم تنفيذها مختلفة بشكل ديناميكي حسب نمط المؤشر * Rodent طالما وُجد لدى الصف Rodent تابع افتراضي ننقي فهو مجرد، هذا يعني أنه لا يمكن إنشاء أي تابع نسخ من الصف Rodent والهدف من وجوده فقط تعريف واجهة تستطيع الصفوف المشتقة تحقيقها .

٥- أكتب برنامجاً بلغة C++ يتضمن الصفوف التالية:

- صفئاً أساسياً area ، ويحتوي على الأعضاء المحمية w (القاعدة) و h (الارتفاع) وهي من النوع float . ويحوي هذا الصف على التوابع العامة التالية:

- بانٍ افتراضي يُسند الصفر للمعطيات المحمية.
- بانٍ عادي بوسيطين من النوع float يسند قيم للمعطيات المحمية.
- تابع وصول getwh من النوع void للوصول للمعطيات المحمية.
- تابعاً صديقاً (غير عضو) لإجراء التحميل للعملية (+) بشكل زائد.
- صفئاً مشتقاً اسمه R يرث الصف area وراثته العامة ويحتوي على أعضاء عامة فقط وهي:

- بانٍ عادي بوسيطين من النوع float يسند قيم للمعطيات المحمية للصف الأساسي area .
- تابع عضو areaF لحساب مساحة المستطيل.
- صفئاً مشتقاً اسمه T يرث الصف الأساسي area وراثته العامة ويحتوي على أعضاء عامة فقط وهي:
- بانٍ عادي بوسيطين من النوع float يسند قيم للمعطيات المحمية للصف الأساسي area .
- تابع عضو areaF لحساب مساحة المثلث.

- في التابع الرئيسي main صرح عن غرضين من الصنفين: R و T على الترتيب مع إعطاءهم قيم محددة وفقاً للتوابع البانية، ثم استدع ما يلزم من التوابع لحساب المساحات المطلوبة وطباعتها. صرح عن غرضين (مع وسطاء) من الصف area وعن غرض ثالث (بدون وسطاء) من الصف نفسه بعد ذلك استدع تابع التحميل الزائد للعملية (+) وتابع الوصول من أجل

جمع الغرضين الذين تم التصريح عنهما ، مع العلم أن الغرض الأول يأخذ الثنائية (r1,r1) ، بينما الغرض الثاني يأخذ الثنائية (t1,t1). وحيث أن r1 و t1 يمثلان مساحتي المستطيل والمثلث على الترتيب.

البرنامج:

```
#include<iostream.h>
class area {
    protected:
        float w,h;
    public:
        area( ){w = 0; h = 0; }
        area( float i, float j ){ w = i; h = j; }
        void getwh( float &i, float &j ){ i = w; j = h; }
        friend area operator +(area ,area );
};//The end of area class

class R : public area {
    public:
        R(float ,float );
        float areaF();
};//The end of R class

class T : public area {
    public:
```

```

        T(float ,float );
        float areaF( );
};//The end of T class

R::R(float w1,float h1) {
    w=w1;
    h=h1;
}

T::T(float w1,float h1) {
    w=w1;
    h=h1;
}

float R::areaF( ) {
    return w*h;
}

float T::areaF( ) {
    return 0.5*w*h;
}

area operator +(area ob1,area ob2)
{
    area temp;
    temp.h=ob1.h+ob2.h;
    temp.w=ob1.w+ob2.w;
}

```

```

        return temp;
    }

    void main( )
    {
        R  r(10.0,5.0);
        T  t(6.0,4.0);
        float x,y,r1,t1;
        r1=r.areaF( );
        t1=t.areaF( );
        cout<<" Rectangle: "<<r1<<endl;
        cout<<" Triangle: "<<t1<<endl;

        area  ob1(r1,r1),ob2(t1,t1),ob3;
        ob3=ob1+ob2;
        ob3.getwh(x,y);
        cout<<"(ob1+ob2) Rectangle+ Triangle:"<<y<<" "
            <<"Rectangle+ Triangle:"<<x<<endl;

    }//The end of main function

```

٦- أكتب برنامجاً بلغة C++ يتضمن ما يلي:

- صفاً أساسياً اسمه Product يحتوي على وسيطين year , model من النوع int كأعضاء خاصة من النوع protected ويحتوي على الأعضاء العامة (public) التالية:

- بان افتراضي يسند القيمة صفر للمعطيات الخاصة.
- بان عادي بوسيطين من النوع int يسند قيم للمعطيات الخاصة.
- تابع عضو print من النوع void لطباعة قيم المعطيات الخاصة.
- تابع مدمر لا يفعل شيئاً.

- صفاً مشتقاً من الصف الأساسي Product (وراثته عامة) اسمه computer ، يحتوي على الأعضاء الخاصة التالية : الاسم name (لا يتجاوز 25 حرفاً) ، السرعة speed ، حجم القرص الصلب hard (يقاس GB) ، ويحتوي على الأعضاء العامة (public) التالية :
- بان عادي بثلاثة وسطاء ، اثنين منهما خاصة بالتابع الباني للصف الأساسي، مهمة الباني استقبال سلسلة محارف والتحقق من كون طولها لا يتجاوز 24 حرفاً ، ومن ثم نسخها إلى العضو الخاص name ، بالإضافة إلى الطلب من المستخدم إدخال قيم لبقية المعطيات الخاصة بالصف المشتق .

- تابع عضو print من النوع void لطباعة قيم المعطيات الخاصة.
- تابع صديق اسمه fr ، مهمته حساب عدد الأقراص الليزرية CD (بفرض أن سعة القرص هي 800 MB (ميغا بايت)) التي تكافئ حجم الـ hard.

- تابع عضو اسمه extra بوسيطين من النوع float ، مهمته معرفة الفرق بعدد الأقراص بين نوعين من الهاردات (Two) . hards

- تابعاً مدمراً لا يفعل شيئاً.

- صرح في التابع الرئيسي main عن غرضين من الصف computer وتمرر لهما القيم الموضحة بالجدول التالي:

Name	Model	Year
pentum3	1	1998
Pentum4	2	2000

ثم استدعِ التوابع اللازمة لطباعة قيم المعطيات الخاصة بالصفين كليهما والتي تم إدخالها من الجدول ومن لوحة المفاتيح، وأيضاً يجب استدعاء التابع الصديق fr والتابع العضو extra لتنفيذ المطلوب .

البرنامج :

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

```
#include<string.h>
```

```
class Product {
```

```
protected:
```

```
int model;
```

```
int year;
```

```
public:
```

```

Product( ){model=0;year=0;}
Product(int m, int y ){model = m; year = y; }
void print( ) {
    cout<<"model="<<model<<"
    year="<<year<<endl;
}
~Product( ){ }
};//The end of Product class

```

```

class computer : public Product {
private:
    char name[25];
    int speed,hard;
public:
    computer(char *l , int m, int y): Product(m,y) {
        int length=strlen(l);
        if(length>24)
            length=24;
        strncpy(name,l,length);
        name[length]='\0';
        cout<<"Enter speed,hard:\n";
        cin>>speed>>hard;
    }
}

```



```

void print( ) {
    Product :: print( );
    cout<<"name:"<<name<<" speed="
        <<speed<<" hard="<<hard<<endl;
}
float extra(float,float);
friend float fr( computer ob );
    ~ computer( ) {    }
};//end computer class

float fr( computer ob){
    float kb;
    kb = (float)((ob.hard*1024)/800);
    return kb;
}

float computer :: extra(float cd1,float cd2) {
    float cd;
    cd=cd2-cd1;
    return cd;
}

void main( )
{
    computer ob1("pentum3",1, 1995),

```

```

computer ob2 (" pentum4 " , 2,2000);
    ob1.print( );
    ob2.print( );
float n1,n2;
    n1=fr(ob1);
    n2=fr(ob2);
    cout<<"size1="<<n1<<" CDs"<<endl;
    cout<<"size2="<<n2<<" CDs"<<endl;
    cout<<"size2-size1="<<ob1.extra(n1,n2)
        <<" CDs"<<endl;

} // end main

```

٧- أكتب برنامجاً بلغة C++ يتضمن الصفوف التالية:

- صفئاً أساسياً vehicle (مركبة) يحتوي على الأعضاء المحمية (عدد العجلات) و r (الوزن) وهي من النوع int . ويحتوي هذا الصف على التوابع العامة التالية:

- بأن افتراضي يُسند الصفر للمعطيات المحمية.
- بأن عادي بوسيطين من النوع int يسند قيم للمعطيات المحمية.
- تابع وصول (get_wr) من النوع void للوصول للمعطيات المحمية.

- تابع صديق (غير عضو) لإجراء عملية التحميل للعملية (*) بشكل زائد.

- تابع () printf للطباعة.

- صفًا مشتقًا اسمه bus (باص) يرث الصف الأساسي vehicle وراثته عامة ويرث من الصف الأساسي عدد العجلات و الوزن و يضيف معلومة (عضو خاص) من النوع الصحيح int هي: عدد الركاب pass و يحتوي على أعضاء عامة هي: بانٍ عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات المحمية للصف الأساسي vehicle وتابع عضو print() للطباعة.

- صفًا مشتقًا اسمه truck (جرار) يرث الصف الأساسي vehicle وراثته عامة ويرث من الصف الأساسي عدد العجلات و الوزن و يضيف معلومة (عضو خاص) من النوع الصحيح int هي: الحمولة load و يحتوي على أعضاء عامة هي: بانٍ عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات المحمية للصف الأساسي vehicle وتابع عضو print() للطباعة.

- صرح في التابع الرئيسي main عن غرضين من الصنفين: bus و truck على الترتيب لتمرير القيم التالية:

vehicle	W	r	Pass	Load
Bus	4	600	10	---

Truck	6	1400	---	4000
-------	---	------	-----	------

ثم صرح عن الأغراض المناسبة من الصف `vehicle` وذلك من أجل استدعاء تابع التحميل الزائد للعملية (*) ، مع العلم أنه يمكن إعطاء الثنائية (4,600) للغرض الأول ، بينما الغرض الثاني يأخذ الثنائية (6,1400) . ويطلب طباعة قيم المعطيات الخاصة لكل الصفوف وأيضاً نتيجة ضرب الغرضين .

البرنامج :

```
#include<iostream.h>
class vehicle {
    protected:
        int w,r;
    public:
        vehicle( ) { w = 0, r = 0; };
        vehicle(int w1,int r1) { w = w1, r = r1; };
        void get_wr(int &i,int&j){I = w ; j = r ; }
        friend vehicle operator *(vehicle ob1,vehicle ob2);
        void printv( ) {cout<<"Wheels:"<< w << " "
                        << "range:" << r<<endl;}
};//The end of vehicle class

class bus : public vehicle{
    int pass;
```

```

public:
bus(int p1,int w1,int r1) : vehicle(w1,r1) {   pass=p1;
}
void print();
};//The end of bus class

void bus::print() {
printv( );
cout<<"passengers:"<<pass<<endl;
}
class truck : public vehicle {
int load;
public:
truck(int l1,int w1,int r1) : vehicle(w1,r1) {
    load=l1;
}
void print();
};//The end of truck class

void truck::print() {
printv( );
cout<<"loadlimit:"<<load<<endl;
}
vehicle operator *(vehicle ob1,vehicle ob2) {

```

```

    vehicle temp;
    temp.w=ob1.w*ob2.w;
    temp.r=ob1.r*ob2.r;
    return temp;
}
void main( ) {
    int ww, rr ;
    bus c(10,4,600);
    truck t(4000,6,1400);
    cout<<" \n bus: "<<endl<<endl;
    c.print( );
    cout<<" \n truck: "<<endl<<endl;
    t.print( );
    vehicle ob1(4,600),ob2(6,1400) , ob3;
    ob3=ob1*ob2;
    ob3.get_wr(ww,rr);
    cout<<"(ob1+ob2)\n wheels:"<<ww<<" "
    <<" range: "<<rr<<endl;
} //The end of main function

```


٨- أكتب برنامجاً بلغة ++C يتضمن الصفوف التالية:

- صفئاً أساسياً اسمه product (اسم سلعة مصنعة) يحوي المعطيات المحمية التالية : الرقم المميز ref : عدد صحيح ، وزن السلعة weight : عدد حقيقي ، ويحتوي على:

- تابع print لطباعة معلومات الصف .
- بانٍ عادي يطلب إدخال قيم المعطيات الخاصة بالصف عن طريق لوحة المفاتيح.
- تابع مدمر لا يفعل شيئاً.
- توابع وصول للمعطيات الخاصة .
- صفئاً مشتقاً اسمه pen (قلم) يرث الصف الأساسي product وراثته عامة، ويحتوي على المعطيات الخاصة التالية:
- العلامة التجارية mark - سلسلة محارف ، واللون الخارجي ex : عدد صحيح ، ولون الحبر ink: عدد صحيح ، ويرث جميع معطيات الصف product ويحتوي على :

- تابع print لطباعة معلومات الصف .
- تابع بانٍ عادي بحيث يأخذ قيمة للمتحول mark من التابع main، بينما يطلب إدخال بقية القيم عن طريق لوحة المفاتيح.
- تابع مدمر لا يفعل شيئاً.
- توابع وصول للمعطيات الخاصة .
- صرح في التابع الرئيسي main عن الأغراض اللازمة من أجل تمرير القيم للمعطيات الخاصة للصفين كليهما (بفرض لدينا سلعتان هما : bic و pencil وكل سلعة لها: رقم مميز + وزن + علامة تجارية + لون خارجي + لون حبر) ومن ثم استدع ما يلزم من التوابع من أجل طباعة هذه القيم .

البرنامج :

```
#include<iostream.h>
#include<string.h>
class Product {
protected:
    int ref;
    float weight;
public:
    Product( ){
        cout<<"Enter the values of ref,weight:\n";
        cin>>ref>>weight;
    }
    void print( ){ cout<<"Ref= "<<ref<<
        " Weight= "<< weight<<endl;
    }
    int getRef( ){return ref; }
    float getWeight( ){return weight;}
    ~Product( ){ }
};//The end of Product class
```

```
class Pen : public Product {
private:
```

```

    char mark[25];
    int ex,ink;
public:
    Pen(char *s) : Product( ) {
        int stlength=strlen(s);
        if(stlength>24)
            stlength=24;
        strncpy(mark,s,stlength);
        mark[stlength]='\0';

        cout<<" Enter the values of external
        color,ink color:\n";
        cin>>ex>>ink;
    }
    void print( ) {
        Product::print( );
        cout<<"mark:"<<mark<<"external color="
        <<ex<<" Inc color="<<ink<<endl;
    }
    int getEx( ){return ex;}
    int getIn( ){return ink;}

    ~Pen( ){ }
};//The end of Pen class

```

```

void main( ) {
    Pen b("bic"),p("pencil");
    b.print( );
    cout<<b.getEx( )<<" "<<b.getIn( )<<endl;
    p.print( );
    cout<<p.getEx( )<<" "<<p.getIn( )<<endl;

} //The end of main function

```

٩- أكتب برنامجاً بلغة C++ يتضمن الصفوف التالية:

- صفّاً أساسياً اسمه car يحتوي على الأعضاء المحمية w (عدد العجلات (الوزن) و r (وهي من النوع int . ويحتوي هذا الصف على التوابع العامة التالية :

- بأن افترضني يُسند الصفر للمعطيات المحمية، بأن عادي بوسيطين من النوع int يسند قيم للمعطيات المحمية.
- تابع وصول getwr من النوع void للوصول للمعطيات المحمية.
- تابع صديق (غير عضو) لتحميل العملية (+) بشكل زائد.
- تابع (printcar) للطباعة.

- صفّاً مشتقاً اسمه Taxi يرث الصف الأساسي car وراثته عامة ويرث من الصف الأساسي عدد العجلات و الوزن ويضيف معلومة (عنصر خاص) من النوع الصحيح int هي: عدد الركاب P و يحتوي على الأعضاء العامة التالية :

- بانٍ عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات المحمية للصف الأساسي car .
- تابع عضو print () للطباعة.
- صفًا مشتقًا اسمه Pickup يرث الصف الأساسي car وراثته عامة ويرث من الصف الأساسي عدد العجلات و الوزن ويضيف معلومة (عنصر خاص) من النوع الصحيح int هي: الحمولة load و يحتوي على الأعضاء العامة التالية :
- بانٍ عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات المحمية للصف الأساسي car .
- تابع عضو print () للطباعة.
- صرح في التابع الرئيسي main عن غرضين من الصنفين Taxi و Pickup على الترتيب لتمرير القيم التالية:

car	W	R	P	load
Taxi	4	1200	4	---
Pickup	6	2500	---	4000

ثم صرح عن الأغراض اللازمة من الصف car من أجل تنفيذ تابع التحميل الزائد للعملية (+) ، مع العلم أن الغرض الأول يأخذ الثنائية (4,1200) ، بينما الغرض الثاني يأخذ الثنائية (6,2500)، ويطلب طباعة نتيجة الجمع.

البرنامج :

```
#include<iostream.h>
class car {
protected:
    int w,r;
```


public:

```
car( ){ w=0, r=0;}  
car(int w1 ,int r1){w=w1; r=r1;}  
void getwr(int &i,int&j){i=w;j=r;}  
friend car operator +(car ,car );  
void printcar ( ) {  
    cout<<" Wheels:"<<w<< endl;  
    cout <<" range:"<<r<<endl;  
}  
};//The end of car class
```

```
class Taxi : public car {  
    private :  
        int pp;  
    public:  
        Taxi(int p1,int w1,int r1):car(w1,r1){  
            pp = p1;  
        }  
  
    void print() {  
        printcar( ) ;  
        cout<<" passengers: "<< pp<<endl ;  
    }  
}
```



```
};//The end of Taxi class
```

```
class Pickup : public car {  
    int load;  
public:  
    Pickup(int l1,int w1,int r1):car(w1,r1){  
        load = l1;  
    }  
    void print( ) {  
        printcar( );  
        cout<<"loadlimit: "<<load<<endl;  
    }  
};//The end of Pickup class
```

```
car operator +(car ob1,car ob2)  
{  
    car temp;  
    temp.w=ob1.w+ob2.w;  
    temp.r=ob1.r+ob2.r;  
    return temp;  
}
```

```
void main()  
{
```

```

int w,r;
Taxi taxi(4,4,1200);
Pickup bic(4000,6,2500);
cout<<" \n Taxi: "<<endl<<endl;
taxi.print( );
cout<<" \n printcar( ): "<<endl<<endl;
bic.print( );
car ob1(4,1200),ob2(6,2500),ob3;
ob3=ob1+ob2;
ob3.getwr(w,r);
cout<<"(ob1+ob2)\n wheels:"<<w<< " "
<<"range:"<<r<<endl;

} //The end of main function

```

١٠- أكتب برنامجاً بلغة C++ يتضمن الصفوف التالية:

- صف أساسي area يحتوي على الأعضاء المحمية width (القاعدة) و height (الارتفاع) وهي من النوع float . ويحتوي الصف على التوابع العامة التالية:

- بانّي افتراضي يُسند الصفر للمعطيات المحمية.
- بانّي عادي بوسيطين من النوع float يسند قيم للمعطيات المحمية.
- تابع وصول getwh من النوع void للوصول للمعطيات المحمية.

- تابع صديق (غير عضو) لتحميل العملية (*) (بشكل زائد).
- صف مشتق اسمه tetragon يرث الصف الأساسي area وراثته عامة ويحتوي فقط على الأعضاء العامة التالية :
 - باني عادي بوسيطين من النوع float يسند قيم للمعطيات المحمية للصف الأساسي area .
 - تابع عضو T_area لحساب مساحة مربع.
- صف مشتق اسمه right_Triangle يرث الصف الأساسي area ويحتوي فقط على الأعضاء العامة التالية :
 - باني عادي بوسيطين من النوع float يسند قيم للمعطيات المحمية للصف الأساسي area .
 - تابع عضو T_area لحساب مساحة مثلث قائم الزاوية.
- صرح في التابع الرئيسي main عن غرضين من الصفين: tetragon و right_Triangle على الترتيب مع إعطاءهم قيم محددة وفقاً للتابع البانية ، ثم استدعي ما يلزم من توابع لحساب المساحات المطلوبة وطباعتها. ثم صرح عن غرضين (مع وسطاء) من الصف area وعن غرض ثالث (بدون وسطاء) من نفس الصف. بعد ذلك استدعي تابع التحميل الزائد للعملية (*) وتابع الوصول من أجل ضرب الغرضين الذين تم التصريح عنهما ، مع العلم أن الغرض الأول يأخذ الثنائية (t , t) ، بينما الغرض الثاني يأخذ الثنائية (r t , r t). وحيث أن t و rt يمثلان مساحتي المربع والمثلث القائم على الترتيب.

البرنامج :

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

```
class area {
```

```
protected:
```

```

    float width,height;
public:
    area( ){width=0;height=0;}
    area(float i,float j){
        width=i; height=j;
    }
    void getwh(float &i, float&j) {
        i = width;
        j=height;
    }
    friend area operator *(area ob1,area ob2);
};//The end of area class

class tetragon : public area {
public:
    tetragon(float w,float h);
    float T_area( );
};//The end of tetragon class

class right_Triangle : public area {
public:
    right_Triangle(float w,float h );
    float T_area( );

```

```
};//The end of right_Triangle class
```

```
tetragon::tetragon(float w,float h) {  
    width=w;  
    height=h;  
}  
right_Triangle::right_Triangle(float w,float h) {  
    width=w;  
    height=h;  
}  
float tetragon::T_area() {  
    return width*height;  
}  
float right_Triangle::T_area() {  
    return 0.5*width*height;  
}  
area operator *(area ob1,area ob2) {  
    area temp;  
    temp.height=ob1.height*ob2.height;  
    temp.width=ob1.width*ob2.width;  
    return temp;  
}
```

```

void main( ) {
    tetragon ob_t(15.0,15.0);
    right_Triangle ob_r(20.0,10.0);
    float w,h,t,rt;
    t=ob_t.T_area( );
    rt=ob_r.T_area( );
    cout<<" The area of tetragon is "<<t<<endl;
    cout<<"The area of right_Triangle is "
        << rt<<endl;
    area ob1(t,t),ob2(rt,rt),ob3;
    ob3=ob1*ob2;
    ob3.getwh(w,h);
    cout<<"(ob1*ob2)\n tetragon*right_Triangle : "
        << h<<" "<<"\n tetragon* right_Triangle:"
        <<w<<endl;

    }//The end of main function

```

١١- اكتب برنامجاً بلغة C++ يحتوي على صف أساسي اسمه FClass يحتوي على وسيطين x , y من النوع float كأعضاء خاصة private ويحتوي على الأعضاء العامة (public) التالية:

- بأن افترضني يسند القيمة صفر للمعطيات الخاصة.
- بأن عادي بوسيطين من النوع float يسند قيم للمعطيات الخاصة.

- تابع عضو Fx لحساب المقدار $x = a + b$.
- تابع عضو Fy لحساب المقدار $y = 2x + 5$.
- تابع وصول getxy من النوع void للوصول إلى المعطيات الخاصة.

- تابع صديق (غير عضو) لتحميل العملية + بشكل زائد.
- أنشئ صفاً مشتقاً اسمه SClass يرث الصف الأساسي FClass وراثته عامة ، لا يحتوي على أي عضو خاص، أما الأعضاء العامة (public) هي عبارة عن تابع عضو واحد Fz والذي يقوم بحساب قيمة المقدار $z = x^2 + y^2$.

- صرح في التابع الرئيسي main عن متغيرين من النوع الحقيقي rx , ry وغرضين (بدون وسطاء) من الصنفين FClass و SClass ، ثم استدعي ما يلزم من التوابيع من أجل حساب وطباعة قيمة المقدار z ، ضع نتيجة استدعاء التابع Fx للمتغير rx وضع نتيجة استدعاء التابع Fy للمتغير ry ، ثم أطبع قيم x و y و z . أيضاً صرح عن غرضين (مع وسطاء- وفقاً للتابع الباني العادي) من الصف FClass وعن غرض ثالث (بدون وسطاء) من الصف FClass و استدعي تابع التحميل الزائد للعملية (+) كتابع صديق وتابع الوصول من أجل جمع الغرضين الذين تم التصريح عنهما ، حيث أن الغرض الأول يأخذ الثنائية (rx , rx) ، بينما الوسيط الثاني يأخذ الثنائية (ry , ry)

البرنامج :

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

```
class Fclass {
```

```
    float x,y;
```

```

public:
Fclass( ){x=0;y=0;}
Fclass(float i,float j){
    x=i;y=j;
}
float Fx(float ,float );
float Fy(float );
void getxy(float&i,float&j){
    i=x;j=y;
}
friend Fclass operator+( Fclass obFx ,Fclass obFy);
};//The end of Fclass class
class SClass : public Fclass {
public:
    float Fz( );
};//The end of Class
float Fclass::Fx(float a,float b)
{
    x=a+b;
    return x;
}
float Fclass::Fy(float x)
{
    y=2*x+5;
}

```

```

    return y;
}
float SClass::Fz( )
{
    return Fx(2,3)*Fx(2,3)+Fy(Fx(2,3))*Fy(Fx(2,3));
}
Fclass operator + (Fclass obFx,Fclass obFy) {
    Fclass temp;
    temp.x=obFx.x+obFy.x;
    temp.y=obFx.y+obFy.y;
    return temp;
}
void main( ) {
    float rx,ry;
    Fclass obf;
    SClass obs;
    float x,y;
    rx=obf.Fx(2,3);
    ry=obf.Fy(rx);
    cout<<"x= "<<rx<<endl;
    cout<<"y= "<<ry<<endl;
    cout<<"Z= "<<obs.Fz( )<<endl;
    Fclass obFx(rx,rx),obFy(ry,ry),obFz;

```

```

obFz=obFx+obFy;
obFz.getxy(x,y);
cout<<"(obFx+obFy)x:"<<x<<" "<<"y:"<<y<< endl;

```

}//The end of main function

١٢- اكتب برنامجاً بلغة C++ يحتوي على صف أساسي kia ويحتوي على الأعضاء المحمية wheels (عدد العجلات) و range (الوزن) وهي من النوع int . ويحتوي هذا الصف على التوابع العامة التالية:

- بأن افترضني يُسند الصفر للمعطيات المحمية.
- بأن عادي بوسيطين من النوع int يسند قيم للمعطيات المحمية.
- تابع وصول getwr من النوع void للوصول للمعطيات المحمية.
- تابع صديق (غير عضو) لتحميل العملية (+) بشكل زائد.
- تابع () show للطباعة.
- أنشئ صفاً مشتقاً اسمه kiaTaxi يرث الصف الأساسي kia وراثته عامة ويرث من الصف الأساسي عدد العجلات و الوزن ويضيف معلومة (عنصر خاص) من النوع الصحيح int هي: عدد الركاب numP ويحتوي على الأعضاء العامة التالية :
- بأن عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات المحمية للصف الأساسي kia .
- تابع عضو () show للطباعة.
- أنشئ صفاً مشتقاً اسمه kiaBic يرث الصف الأساسي kia وراثته عامة ويرث من الصف الأساسي عدد العجلات و الوزن ويضيف معلومة (عنصر

- خاص (من النوع الصحيح int هي: الحمولة loadL ويحتوي الصف على الأعضاء العامة التالية:
- بان عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات المحمية للصف الأساسي kia .
 - تابع عضو () show للطباعة.
 - صرح في التابع الرئيسي main عن غرضين من الصنفين: kiaTaxi و kiaBic على الترتيب لتمرير القيم التالية:

kia	wheels	Range	numP	loadL
kiaTaxi	4	1000	4	لا شيء
kiaBic	6	2000	لا شيء	5000

ثم صرح عن غرضين (مع وسطاء) من الصف kia وعن غرض ثالث (بدون وسطاء) من نفس الصف. بعد ذلك استدعي تابع التحميل الزائد للعملية (+) ، وتابع الوصول من أجل جمع الغرضين ، مع العلم أن الغرض الأول يأخذ الثنائية (4,1000) ، بينما الغرض الثاني يأخذ الثنائية (6,2000)، ويطلب طباعة نتيجة الجمع.

البرنامج :

```
#include<iostream.h>
class kia{
protected:
    int wheels,range;
public:
    kia(){ wheels=0, range=0;};
```



```

kia(int w ,int r);
void getwr(int &i,int&j){i=wheels;j=range;}
friend kia operator +(kia ob1,kia ob2);
void showk( );
};//The end of kia class

kia::kia(int w ,int r){
    wheels=w;
    range=r;
}
void kia::showk( ){
    cout<<"Wheels:"<<wheels<<endl;
    cout<<"range:"<<range<<endl;
}
class kiaTaxi : public  kia {
    int nummP;
public:
    kiaTaxi(int p,int w,int r):kia(w,r) {
        nummP=p;
    }
    void show( );
};//The end of kiaTaxi class

```



```

void kiaTaxi::show( ) {
    showk( );
    cout<<"passengers:"<<nummP<<endl;
}

```

```

class kiaBic : public kia {
    int loadL;
public:
    kiaBic(int l,int w,int r):kia(w,r){
        loadL=l;
    }
    void show( );

```

```

};//The end of kiaBic class

```

```

void kiaBic::show( ){
    showk( );
    cout<<"loadlimit:"<<loadL<<endl;
}

```

```

kia operator +(kia ob1,kia ob2) {
    kia temp;
    temp.wheels=ob1.wheels+ob2.wheels;
    temp.range=ob1.range+ob2.range;
    return temp;
}

```

```

}

void main( ) {
    int w,r;
    kiaTaxi taxi(4,4,1000);
    kiaBic bic(5000,6,2000);
    cout<<" \n kiaTaxi: "<<<endl<<<endl;
    taxi.show( );
    cout<<" \n kiaBic: "<<<endl<<<endl;
    bic.show( );
    kia ob1(4,1000),ob2(6,2000),ob3;
    ob3=ob1+ob2;
    ob3.getwr(w,r);
    cout<<"(ob1+ob2)\n wheels:"<<w<< " "<<"range:"
        <<r<<endl;
} //The end of main function
=====

```

١٣- اكتب برنامجاً بلغة C++ يحتوي على صف أساسي اسمه first ويحتوي على وسيطين x , y من النوع int كأعضاء خاصة private ويحتوي على الأعضاء العامة (public) التالية:

- بأن افتراضي يسند القيمة صفر للمعطيات الخاصة.
- بأن عادي بوسيطين من النوع int يسند قيم للمعطيات الخاصة.

- تابع عضو fx لحساب المقدار $x = a + b$.
- تابع عضو fy لحساب المقدار $y = x + 10$.
- تابع وصول `getxy` من النوع `void` للوصول إلى المعطيات الخاصة.

- تابع صديق (غير عضو) لتحميل العملية (+) بشكل زائد.
- أنشئ صفاً مشتقاً اسمه `second` يرث الصف الأساسي `first` وراثته عامة ، لا يحتوي على أي عضو خاص ، أما أعضائه العامة (`public`) هي عبارة عن تابع عضو واحد وليكن `fz` الذي يقوم بحساب قيمة المقدار ($z = x + 2y$) .

في التابع الرئيسي `main`:

- صرح عن متغيرين من النوع الصحيح `r1, r2`.
- صرح عن غرضين (بدون وسطاء) من الصفين `first` و `second`
- استدعي ما يلزم من التوابيع من أجل حساب وطباعة قيمة المقدار `z` ، ضع نتيجة استدعاء التابع `fx` للمتغير `r1` وضع نتيجة استدعاء التابع `fy` للمتغير `r2`.
- صرح عن غرضين (مع وسطاء - وفقاً للتابع الباني العادي) من الصف `first` وعن غرض ثالث (بدون وسطاء) من الصف `first` .
- استدعي تابع التحميل الزائد للعملية (+) (التابع الصديق) وتابع الوصول من أجل جمع الغرضين ، حيث أن الغرض الأول يأخذ الثنائية (`r1, r1`) ، بينما الوسيط الثاني يأخذ الثنائية (`r2, r2`).

البرنامج :

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

```
class first {
```

```

    int x,y;
public:
    first( ){x=0;y=0;}
    first(int i,int j){x=i;y=j;}
    int f1(int a,int b);
    int f2(int x);
    void getxy(int&i,int&j){i=x;j=y;}
    friend first operator+(first obf1,first obf2);
};//The end of first class

class second : public first {
public:
    int f3( );
};//The end of second class

int first::f1(int a,int b) {
    x=a+b;
    return x;
}

int first::f2(int x) {
    y=x+10;
    return y;
}

```

```

int second::f3( ) {
    return f1(2,3)+2*f2(f1(2,3));
}
first operator+(first obf1,first obf2) {
    first temp;
    temp.x=obf1.x+obf2.x;
    temp.y=obf1.y+obf2.y;
    return temp;
}

```

```

void main( ) {
    int r1,r2;
    first obf;
    second obs;
    int x,y;
    r1=obf.f1(2,3);
    r2=obf.f2(r1);
    cout<<"x= "<<r1<<endl;
    cout<<"y= "<<r2<<endl;
    cout<<"Z= "<<obs.f3( )<<endl;
    first obf1(r1,r1),obf2(r2,r2),obf3;
    obf3=obf1+obf2;
    obf3.getxy(x,y);
}

```

```
cout<<"(obf1+obf2) x:"<<x<<" "<<"y:"<<y<<endl;
```

```
}//The end of main function
```



مسائل عامة

١- اكتب برنامجاً بلغة C++ لإنشاء صف اسمه Book ، يحتوي على ثلاثة وسطاء خاصة من النوع char ، وهي : اسم الكتاب NameBook واسم دار النشر Publishing وسنة النشر Year ، يحتوي هذا الصف على باني افتراضي وتابع تعديل معطيات وتابع طباعة اسمه PrintBook .

- استخدم الوراثة العامة لإنشاء صف مشتق من الصف الأساسي اسمه AddressBook ويرث من الصف الأساسي اسم الكتاب ودارا لنشر وسنة النشر ويضيف معلومات على هذا الكتاب وهي : اسم المؤلف Writer والدولة Country والمدينة City وهي متحولات خاصة من نمط char ، ويقوم الصف المشتق بإدخال المعلومات السابقة عن الكتاب وطباعتها ، ويحتوي الصف المشتق على باني عادي وتابع طباعة اسمه ShowBook .

صرح ضمن التابع main عن أغراض لتمرير المعلومات إلى الصفوف واستخدام تابع الطباعة لإظهار كافة المعلومات .

=====

٢- اكتب برنامجاً بلغة C++ لإنشاء صفين أساسيين هما :

- الصف الأساسي الأول اسمه CourseName ، يحتوي هذا الصف على ثلاثة متحولات خاصة private من النوع char ، هي أسماء ثلاثة مقررات للطالب : courseName1 و courseName2 و courseName3 .

ويحتوي الصف على باني افتراضي وباني عادي وتابع طباعة اسمه Print() . يطبع أسماء المقررات .

- **الصف الأساسي الثاني اسمه Course** ، يحتوي على ثلاثة متحولات خاصة private نوع int ، وهي درجات الطالب للمقررات الثلاثة :
course1 و course2 و course3. ويحتوي الصف على باني افتراضي و تابع تعديل معطيات setCourse و تابع طباعة اسمه () Print لطباعة درجات الطالب في المقررات الثلاثة .
- **يرث الصف المشتق StudName الصف الأساسي الأول CourseName** والصف الأساسي الثاني Course وراثته عامة public. ويحتوي الصف StudName على ثلاثة متحولات خاصة private من نوع char ، وهي الاسم الثلاثي للطالب : firstName و secondName و lastName . ويحتوي الصف على باني افتراضي و باني عادي و تابع طباعة اسمه () Print لطباعة الاسم الثلاثي للطالب وأسماء المقررات ودرجاتها .
- **صرح ضمن التابع main عن أغراض لتمرير المعطيات إلى الصف المشتق** واستخدم تابع الطباعة لإظهار كافة المعلومات اللازمة .

=====

٣- اكتب برنامجاً بلغة C++ يحتوي على :

- ⓧ **صف أساسي أول اسمه date** ، يحتوي على ثلاثة وسطاء (day , month , year) من النوع int كأعضاء private ويقوم بإدخال التاريخ وطباعته بالشكل (year : month : day) ويحتوي الصف على باني افتراضي لتحديد القيم الابتدائية للتاريخ وهي (2000 : 01 : 01) وقيمة الوسيط day ضمن المجال [1 - 31] والوسيط month ضمن المجال [1 - 12] والسنة year ضمن المجال [2000 - 2012] ويحتوي الصف على باني عادي وعلى تابع طباعة () printf .

✗ **صف أساسي ثاني اسمه time** يحتوي على ثلاثة وسطاء (Hour, Minute, Second) من النوع int كأعضاء private ويقوم بإدخال الوقت وطباعته بالشكلين القياسي والعسكري ، ويحتوي على بان افتراضي لطباعة القيمة الابتدائية ، وعلى تابعي طباعة هما () printu لطباعة الوقت بالشكل العسكري و () prints لطباعة الوقت بالشكل القياسي ، حيث الوسيط Hour ضمن المجال [0 - 24] أما Minute و Second ضمن المجال [0 - 59] .

✗ **صف مشتق اسمه Airline** (مكتب الخطوط الجوية) ، ويرث الصنفين الأساسيين السابقين (وراثته عامة) ويحتوي على وسيطين كأعضاء private هما : اسم الشخص المسافر name واسم الدولة الذي سيسافر إليها NameCountry وهي من النوع char ، ويقوم الصنف بطباعة اسم المسافر واسم الدولة والتاريخ والوقت الذي سيغادر فيه المسافر . ويحتوي الصنف على تابع باني عادي وتابع طباعة () printAir

❖ صرح ضمن التابع main عن أغراض لتمرير المعطيات إلى الصنف المشتق واستخدام تابع الطباعة لإظهار كافة المعطيات في الصنف المشتق .
❖ ماذا يحدث للأعضاء العامة والخاصة لصف أساسي يرثه صف مشتق باستخدام المحدد public ؟

❖ كيف تستدعي التوابع البانية والمدمرة للصفوف المشتقة في البرنامج السابق ؟

=====

٤- اكتب برنامجاً بلغة ++C لإنشاء صف أساسي اسمه Faculty يحتوي هذا الصنف على ثلاثة متحولات خاصة private من النوع int وهم : عدد الأساتذة Teacher ، وعدد المهندسين Engineer ، وعدد الطلاب Student .

ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه
() PrintFaculty.

- يرث الصف المشتق Department الصف الأساسي Faculty وراثته عامة ويحتوي على ثلاثة متحولات خاصة private نوع char وهم : اسم القسم DepartmentName، واسم رئيس القسم CharminName ، وسنة تأسيس القسم year. ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه () PrintDepartment .

- يرث الصف المشتق Building الصف المشتق Department ويحتوي على ثلاثة متحولات خاصة private نوع int وهم : عدد المكاتب Office، وعدد المخابر Laboratory ، وعدد القاعات الدراسية Room. ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه () PrintBuilding .

- صرح ضمن التابع main عن أغراض لتمرير المعطيات إلى الصف المشتق واستخدام تابع الطباعة لإظهار كافة المعلومات اللازمة .
- ماذا يحدث للأعضاء العامة والخاصة لصف أساسي يرثه صف مشتق باستخدام المحدد public ؟

=====

٥- اكتب برنامجاً لإنشاء صفين أساسيين بلغة C++ الصف الأساسي الأول اسمه Faculty ، ويحتوي على ثلاثة وسطاء كأعضاء خاصة من النوع int وهي : عدد الأساتذة Teacher ، وعدد المهندسين Engineer ، وعدد الطلاب Student، ويحتوي هذا الصف على بان افتراضي و بان لتمرير المعطيات وتابع طباعة اسمه () PrintFaculty . أما الصف الأساسي الثاني اسمه Building ، ويحتوي على ثلاثة وسطاء كأعضاء خاصة من النوع int

وهي : عدد المكاتب Office ، وعدد المخابر Laboratory ، وعدد القاعات الدراسية Room ، ويحتوي الصف المشتق على بان افتراضي وعلى و بان عادي وعلى تابع طباعة اسم () PrintBuilding

- استخدم الوراثة العامة لإنشاء صف مشتق اسمه Department ويرث الصفين الأساسيين Faculty و Building ويحتوي على ثلاثة متحولات خاصة private نوع char وهي : اسم القسم DepartmentName ، واسم رئيس القسم CharminName ، وسنة تأسيس القسم year ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه () PrintDepartment .

- صرح ضمن التابع main عن أغراض لتمرير المعطيات إلى الصف المشتق واستخدم تابع الطباعة لإظهار كافة المعلومات اللازمة .
- متى وكيف تستدعي النوايع البانية والمدمرة للصفوف المشتقة في البرنامج السابق ؟

=====

٦- اكتب برنامجاً لإنشاء صفاً بلغة C++ اسمه **Building** ، يحتوي على ثلاثة وسطاء كمتحولات private بحيث يخزن عدد الطوابق floors في البناء وعدد الغرف rooms وهي من النوع int ومساحة الغرف sqmeter بالمتر المربع وهي من النوع double . ويحتوي على بان عادي وعلى تابع طباعة printBuilding

- ثم أنشئ صفاً مشتقاً اسمه **House** يرث الصف **Building** ويخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي متغيرات خاصة (private) من النوع int . ويحتوي على بان عادي وعلى تابع طباعة printHouse .

- ثم أنشئ صفاً مشتقاً اسمه **Office** يرث الصف **Building** ويخزن عدد أجهزة الإطفاء **Extinguishers** وعدد الهواتف **Phones** وهي متغيرات خاصة (**private**) من النوع **int** . ويحتوي على بانٍ عادي وعلى تابع طباعة **printOffice** .
- صرح ضمن التابع **main** عن غرض في الصف المشتق **Office** لتمثيل المعلومات إلى الصفوف واستخدام تابع الطباعة لإظهار كافة المعلومات .
- ماذا يحدث للأعضاء العامة والخاصة لصف أساسي يرثه صف مشتق باستخدام المحدد **public** ؟

=====

- ٧- اكتب برنامجاً ليحتوي على صف أساسي اسمه **student** يحتوي على المعطيات الخاصة التالية: الاسم الأول **first** و الكنية **last** من نمط **char**، أما الرقم الجامعي **id** من نمط **int**، ويحتوي على الأعضاء العامة التالية:
- بانٍ افتراضي يسند المحرف الصفري أو الصفر للمعطيات الخاصة وذلك وفقاً لنوع المعطيات الخاصة.
 - بانٍ عادي بثلاثة وسطاء يسند قيم للمعطيات الخاصة.
 - تابع عضو لطباعة قيم المعطيات الخاصة اسمه **print** .
 - صفاً مشتقاً اسمه **course** يرث الصف الأساسي **student** وراثته العامة، ويحتوي على المعطيات الخاصة التالية: رمز المقرر **cid** واسم المقرر **name** من نمط **char** ، ودرجة الطالب **mark** من نمط **float** ويحتوي على الأعضاء العامة التالية:
 - بانٍ عادي يسند قيم للمعطيات الخاصة لكلا الصنفين.
 - تابع تعديل لدرجة الطالب في المقرر .

- تابع عضو اسمه count ، مهمته تحديد عدد المحارف للسلسلة .
- تابع عضو لطباعة قيم المعطيات الخاصة بالصف المشتق اسمه print مستخدماً ضمنه تابع الطباعة المعروف في الصف الأساسي student .

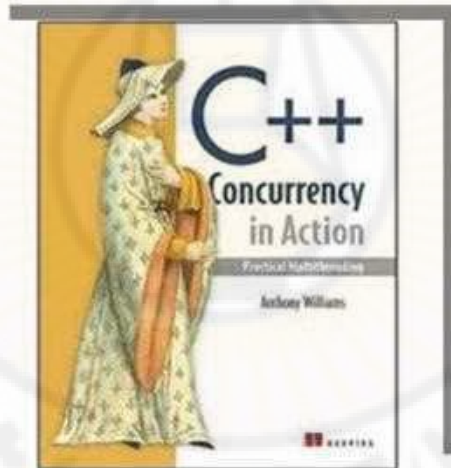
• يتم في التابع الرئيسي main التصريح عن الغرض المناسب من أجل تمرير قيم للمعطيات الخاصة بالصفين الأساسي والمشتق وهذه القيم مبيّنة في الجدول التالي:

الاسم الأول	الكنية	الرقم الجامعي	رمز المقرر	اسم المقرر	درجة الطالب
Ahmad	Salem	12345	Prg2	Programming in C++	75.0

وبعد ذلك يتم استدعاء ما يلزم من التوابيع من أجل: تعديل درجة الطالب لتصبح 85.0 بدلاً من 75.0 ، وأيضاً طباعة قيم كافة المعطيات الخاصة لكلا الصفين قبل التعديل وبعده ثم طباعة عدد المحارف التي يتكون منها اسم المقرر .



الفصل الخامس
تعدد الأشكال
Polymorphism





تعدد الأشكال

Polymorphism

٥-١ - مقدمة

إن تعدد الأشكال في لغة C++ هي الميزة الأساسية الثالثة للبرمجة غرضية التوجه OOP ، في حين أن الميزتين الأولى هما الصفوف والوراثة. تقوم التغليف (الصفوف) بإنشاء أنماط معطيات جديدة عبر تجمع الخصائص والسلوكيات ، وتسمح ميزة التحكم بالوصول بفصل الواجهة عن التنفيذ عبر جعل المتحولات خاصة private . وبالتالي يعني هذا النوع من التنظيم الميكانيكي منطقاً جاهزاً لمن له خبرة في البرمجة البنيوية ، أما التوابع الافتراضية فتتعامل مع الفصل باستخدام مصطلحات الأنماط types . رأينا في الفصل الرابع كيف تسمح الوراثة بمعاملة الغرض كأنه ينتمي لنمطه الأصلي أو نمط الصف الأساسي . وتعد هذه المقدرة أساسية لأنها تسمح بأن يتعامل معاملة العديد من الأنماط كما لو كانت نمطاً وحيداً . ويسمح التابع الافتراضي لنمط ما أن يعبر عن تميزه عن نمط آخر مشابه ما دام مشتق من الصف الأساسي نفسه . ويتم التعبير عن هذا التميز من خلال الاختلافات في سلوك التوابع التي يمكن استدعاؤها بواسطة الصف الأساسي .

وتساعد تقنية تعدد الأشكال في تطوير برامج عامة بدلاً من كتابة برامج خاص لهدف معين ومحدد ، و تتعامل تقنية تعدد الأشكال مع مؤشرات على الصف الأساسي ومع العنوان (Reference) للدلالة على هذا الصف . إن فهم تعدد الأشكال يبدأ من فهم التوابع الافتراضية (Virtual) ومتى نحتاج إلى استخدامها .

وإن تعدد الأشكال هو معرفة كيفية انجاز خدمة مبنية ضمن الغرض نفسه وليس في الشيفرة التي تستخدم الغرض ، و استدعاء تابع محدد أو عملية محددة يمكن أن يأخذ عدداً غير محدد من الأشكال المختلفة . و تعدد الأشكال ممكن باستخدام التوابع الافتراضية .

٥-٢ - المؤشرات إلى الصفوف المشتقة

Pointers To Derived Classes

تعد المؤشرات إلى الصفوف المشتقة في أن المؤشر المعروف الذي يشير إلى صف أساسي يمكن أن يشير إلى صف مشتق من هذا الصف الأساسي ، فعلى سبيل المثال لدينا الصف الأساسي base والصف المشتق derived من الصف الأساسي base ، يمكن أن نكتب ما يلي :

```
base *p ; // base class to pointer
base ob1 ; // object of type base
derived ob2 ; // object of type derived
P = & ob1 ; // p points to base object
P = & ob2 ; // p points to derived object
```

ومن الواضح أن المؤشر من الصف الأساسي يمكن أن يشير إلى غرض من أي صف مشتق من الصف الأساسي دون توليد أي خطأ . ونستطيع استخدام مؤشر من الصف الأساسي للإشارة إلى صف مشتق إلا أننا نستطيع الوصول إلى أعضاء الغرض المشتق الموروثة من الصف الأساسي (الأعضاء العامة) فقط . لأن المؤشر في الصف الأساسي لا يملك إلا معلومات عن أعضاء الصف

الأساسي ولا يعرف أي شيء عن الأعضاء التي أضيفت بعد عملية الاشتقاق .
مع أنه من الممكن أن يشير مؤشر الأساس (مؤشر موجود في الصف الأساسي)
إلى غرض مشتق إلا أن العكس غير صحيح ، فالمؤشر في الصف المشتق لا
يستخدم للوصول إلى غرض من صف أساسي .

يوضح المثال (5- 1) التالي كيف يمكن استخدام مؤشر صف أساسي

ل للوصول إلى صف مشتق .

مثال (5 - 1)

// EXAMPLE 1 THIS IS SHORT Program

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

```
class Base {
```

```
private :
```

```
int x ;
```

```
public :
```

```
Base( ){
```

```
x=0 ;
```

```
cout<<"constructing base class \n";
```

```
}
```

```
void setx(int i) { x=i;}
```

```
void int getx( ){return x;}
```

```
}; //End Class
```



```

// Inherit
class derived : public Base {
private :
    int y ;
public :
    derived( ){
        y=0;
        cout<<"constructing derived class \n";
    }
    void sety ( int i) {
        y=i;
    }
    int gety( ) {
        return y;
    }
}; // End class

void main ( ) {
    Base *p;           // pointer to base type
    Base b_ob ;        // object of base .
    derived d_ob ;     // object of derived
    // use p to access base object

```

```

p = &b_ob ;
p-> setx(100);    // access base object
cout << " Base Object x : " << p-> getx( ) << "\n";
// use p to access derived object
p = &d_ob ;      // pointer to derived object
p-> setx(199);    // access derived object
cout << " Base Object x : " << p-> getx( ) << "\n";
// can't use p to set y , so do it directly
d_ob.sety(188);
cout << " Derived Object x : " << p-> getx( ) << "\n";
cout << " Derived Object y : " << d_ob. gety( ) << "\n";
} // End main

```

Result

constructing base class
 constructing base class
 constructing derived class

Base Object x : 100

Base Object x : 199

Derived Object x : 199

Derived Object y : 188

Press any key to continue

ولا يوجد أية فائدة من استخدام مؤشر الصف الأساسي في البرنامج سوى توضيح كيفية استخدامه للإشارة إلى صف مشتق .

٥-٣- التتابع الافتراضية

Virtual Functions

يتم بناء تعددية الأشكال من خلال التتابع الافتراضية ، وعندما يتم استدعاء أحد التتابع الافتراضية باستخدام مؤشر على صف أساسي فإن لغة C++ تقوم باختيار التابع المُعاد تعريفه والملائم للصف المشتق المرتبط بالغرض . ويتم أحياناً تعريف تابع عضو عام غير افتراضي ضمن الصف الأساسي ويعاد تعريفه في الصف المشتق ، فإذا تم استدعاء مثل هذا التابع من خلال مؤشر على صف أساسي فإنه يتم استدعاء نسخة الصف الأساسي من ذلك التابع، أما إذا تم استدعاء التابع من خلال مؤشر على صف مشتق فإنه يتم استخدام نسخة الصف المشتق ، عندئذ نسمي هذا التصرف غير متعدد الأشكال .

التابع الافتراضي virtual function عبارة عن تابع عضو يصرح عنه داخل الصف الأساسي ثم يعاد تعريفه داخل الصف المشتق . لإنشاء تابع افتراضي يجب أن يسبق التصريح عنه بالكلمة المحجوزة **virtual** أي :

`virtual void getxy( ) ;`

وعندما يرث صف مشتق صفاً أساسياً يحتوي على تابع افتراضي يقوم الصف المشتق الجديد بإعادة تعريف التابع الافتراضي داخل الصف المشتق ليقوم بإنجاز وظيفة أخرى يحددها له الصف المشتق ، وكل إعادة تعريف لهذا التابع في كل صف مشتق هي عبارة عن تحديد لعمله بما يتفق خصيصاً مع هذا الصف المشتق . و إعادة تعريف التابع الافتراضي في الصف المشتق لا يحتاج إلى

استخدام الكلمة المحجوزة `virtual` ، ويتم استدعاء التابع الافتراضي كما يستدعي أي تابع آخر ، إلا أن ما يميزه عن غيره يكمن في عملية استدعائه عن طريق المؤشر .

إن نوع الغرض الذي يشير إليه المؤشر عند حدوث الاستدعاء هو الذي يحدد النسخة التي ستنفذ ، وبالتالي عندما نشق صفين أو أكثر من صف أساسي يحوي تابعاً افتراضياً ، فإنه عندما يشير مؤشر الأساس على غرضين مختلفين ستنفذ نسختين مختلفتين من التابع الافتراضي هذه هي الطريقة التي تتجز من خلالها تعدد الأشكال في زمن التنفيذ ، ويسمى الصف الذي يحوي التابع الافتراضي بالصف متعدد الأشكال `Polymorphism Class` . ويبين المثال (2-5) استخدام التابع الافتراضي في الصف الأساسي والصفوف المشتقة.

مثال (2 - 5)

```
/* THIS IS SHORT PROGRAM THAT USES A VIRTUAL  
FANCTION */
```

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

```
class base {
```

```
public :
```

```
int x;
```

```
base(int i) { x = i ;}
```

```
virtual void func( )
```

```
{
```

```
cout << "Using base version of func( ) : " ;
```

```
cout << x << endl ;
```

```

    }
}; // End Class

// Inherit
class derived1 : public base {
public :
    derived1 ( int i ) : base(i) { }
    void func( )
    {
        cout << "Using derived1's version of func( ) : ";
        cout << x*x << endl ;
    } // End Virtual function
}; // End class .

// Inherit
class derived2 : public base {
public :
    derived2 ( int i ) : base( i ) { }
    void func( )
    {
        cout << "Using derived2's version of func( ) : ";
        cout << x+x << endl ;
    } // End Virtual function
};

```



```
}; // End class .
```

```
void main ( ) {
```

```
    base *p;           // pointer to base type
```

```
    base ob( 10 ) ;
```

```
    derived1 d_ob1( 10) ; // object of derived1
```

```
    derived2 d_ob2( 10) ; // object of derived2
```

```
    // use p to access base object
```

```
        p = &ob ;
```

```
        p-> func( ) ; // use base's func( )
```

```
    // use p to access derived1 object
```

```
        p = &d_ob1 ;
```

```
        p-> func( ) ; // use derived1's func( )
```

```
    // use p to access derived2 object
```

```
        p = &d_ob2 ;
```

```
        p-> func( ) ; // use derived2's func( )
```

```
} // End main
```


Result

Using base version of func() : 10

Using derived1's version of func() : 100

Using derived2's version of func() : 20

Press any key to continue

كما نلاحظ أن البرنامج السابق يتكون من ثلاثة صفوف ، الصف الأول هو الصف الأساسي base ويحتوي على التابع الافتراضي func () ، ويرث هذا الصف صفتين مشتقتين هما derived1 و derived2 . ويستدعي كل من هذين الصفتين المشتقتين التابع الافتراضي func () ويحدد كل واحد عمله بطريقته الخاصة .

صرّح في التابع الرئيسي main () عن المؤشر p الذي يُوّشر إلى أغراض من الصفوف base و derived1 و derived2 . في البداية يسند إلى p عنوان الغرض ob من النوع base ، وعند استدعاء التابع func () باستخدام المؤشر p تستخدم النسخة المعرفة من base ، بعد ذلك يسند إلى p عنوان الغرض d_ob1 ويتم استدعاء التابع الافتراضي func () مرة ثانية. وبما أن نوع الغرض الذي يشير إليه المؤشر p سوف يحدد النسخة التي ستستدعى فإنه في هذه المرة سيتم استدعاء التابع func () المعاد تعريفه في الصف المشتق derived1 . وأخيراً سيسند عنوان الغرض d_ob2 من النوع derived2 إلى المؤشر p ويتم استدعاء التابع func () وفي هذه المرة يتم تنفيذ نسخة التابع الموجودة في الصف المشتق derived2 .

□ ملاحظة مهمة :

إن ترتيب التوابع الافتراضية حسب ترتيب الوراثة عندما لا يتجاوز (override) صف مشتق التابع الافتراضي (أي لا يحتوي الصف المشتق على التابع الافتراضي) ، يتم استخدام التابع الافتراضي المعروف داخل الصف الأساسي ، والمثال (3 - 5) التالي يوضح ذلك .

مثال (3 - 5)

```
#include <iostream.h>
class base {
public:
    int x ;
    base(int i) { x = i ; }
    virtual void func( ) {
        cout << "Using base version of func( ): " ;
        cout << x << "\n";
    }
}; // End class base
class derived1 : public base {
public:
    derived1(int i) : base( i ) { }
    void func( )
{
```

```

    cout << "Using derived1's version of func( ): ";
    cout << x*x << "\n";
}
}; // End derived1 Class

class derived2 : public base {
public:
    derived2(int i) : base(i) { }
    // derived2 does not override func( )
}; // End derived2 Class

int main( ) {
    base *p ;
    base ob( 10 );
    derived1 d_ob1( 20 );
    derived2 d_ob2( 5 );

    p = &ob ;
    p->func( );    // use base's func( )
    p = &d_ob1 ;
    p->func( ) ;    // use derived1's func( )
    p = &d_ob2 ;
    p->func( ) ;    // use base's func( )
    return 0;
} // End main

```

Result

Using base version of func() : 10
Using derived1's version of func() : 400
Using base version of func() : 5

Press any key to continue

وفي هذا المثال لا يتجاوز (override) الصف derived2 التابع الافتراضي func () ، لذلك عند إسناد عنوان الغرض d_ob2 إلى المؤشر p تنفذ نسخة التابع () func الموجودة في الصف الأساسي .

٥-٤- الصفوف المجردة

Abstract Classes

عندما نفكر بالصف على أنه نمط فإننا نفترض أننا سوف نحصل منه على أغراض من ذلك النمط ، وسوف نبين فيما يلي بعضاً من الحالات التي يفيد فيها أن نعرف الصفوف التي لا يقوم المبرمج باستخراج أغراض منها ، عندئذ تسمى هذه الصفوف بالصفوف المجردة Abstract Class ويتم استخدامها كصفوف أساسية في عمليات التوريث للصفوف المشتقة منه . وتوافر الصفوف المجردة أرضية مناسبة لاشتقاق صفوف أخرى وتوريثها واجهات وتوابع ملائمة لعملها .

وعندما يحتوي الصف على تابع افتراضي واحد على الأقل فإنه يسمى صفاً مجرداً Abstract Class . وبما أن الصف المجرد يحتوي على تابع افتراضي واحد على الأقل ، فإن الصفوف المشتقة من هذا الصف عندما تتجاوز (override) هذه التوابع تعيد تعريفها بما يتفق عمل كل صف مشتق ، وعندما

لا يتجاوز (override) صف مشتق التابع الافتراضي يتم استخدام التابع الافتراضي المعرف داخل الصف الأساسي ، ويمكن لأي صف مشتق أن لا يتجاوز التابع الافتراضي في الصف الأساسي .

والبرنامج في المثال (5 - 4) ينشئ صفاً أساسياً اسمه area يحتفظ ببيدي شكل ما ، ويصرح أيضاً عن تابع افتراضي اسمه (geatarea) والذي يعيد عند تجاوزه من قبل الصف المشتق مساحة نوع الشكل المعرف في الصف المشتق في هذه الحالة يحدد تصريح التابع داخل الصف الأساسي شكل الواجهة في حين أنه أوكل التمثيل الفعلي لهذه الواجهة إلى الصفوف التي ورثت الصف الأساسي .

ويحتوي الصف الأساسي area على متحولين خاصين dim1 و dim2 من نمط double ويحتوي على تابع تعديل معطيات بوسيطين ، وتابع وصول للمتحويلات الخاصة ويقوم التابع الافتراضي بحساب محيط المستطيل ، وعندما يتجاوز الصف المشتق الأول التابع الافتراضي في الصف الأساسي يقوم بحساب مساحة المستطيل ، أما عندما يتجاوز الصف المشتق الثاني التابع الافتراضي يقوم بحساب مساحة المثلث .

// Create an abstract class.

مثال (5 - 4)

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

```
class area {
```

```
private :
```

```
double dim1, dim2; // dimensions of figure
```

```
public:
```

```
void setarea(double d1, double d2)
```

```

{
    dim1 = d1;
    dim2 = d2;
}

void getdim ( double &d1, double &d2)
{
    d1 = dim1;
    d2 = dim2;
}

virtual double getarea( )    // virtual function
{
    حساب محيط مستطيل
    return 2*(dim1 + dim2);
}

};// End Class area

class rectangle : public area {
public:
    double getarea( )    // virtual function
    {
        حساب مساحة مستطيل
        double d1 , d2 ;
        getdim(d1, d2);
        return d1 * d2;
    }
};// End Class rectangle

```



```

class triangle : public area {
public:
    double getarea( )    // virtual function
    {
        double d1, d2;
        getdim(d1, d2);
        return 0.5 * d1 * d2;
    }
}; // End Class triangle

```

حساب مساحة مثلث

```

int main() {
    area *p;
    area b ;
    rectangle r ;
    triangle t ;
    p = &b;
    p->setarea(3.3, 4.5);

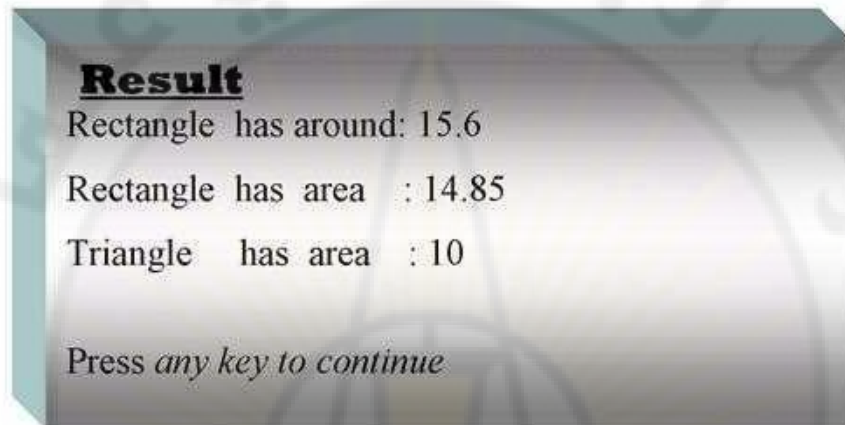
    cout << "Rectangle has around: " << p->getarea( ) << endl ;
    p = &r;
    p->setarea(3.3, 4.5);
    cout << "Rectangle has area: " << p->getarea( ) << endl ;
    p = &t;

```

```

p->setarea(4.0, 5.0);
cout << "Triangle has area: " << p->getarea( ) << endl ;
return 0;
} // End main

```



مثال (5 - 5)

اكتب برنامجاً ينشئ صففاً أساسياً يدعى number يحتفظ بقيمة عدد صحيح عشري ويحتوي على تابع افتراضي اسمه (shownum) ، ثم أنشئ صفين outhex و outoct يرثان الصف number ويتجاوزان التابع الافتراضي (shownum) بحيث يتم تحويل العدد الصحيح العشري ما يكافئه في النظام السداسي عشر والنظام الثماني على التوالي .

البرنامج

```

#include <iostream.h>
class number {

```

protected :

int i;

public:

number() { i = 0 ;}

number(**int** x) { i = x; }

virtual void shownum() { cout << i << '\n'; }

} ; *// End Class number*

class outhex : **public** number {

public:

outhex(**int** n) : number(n) { }

void shownum() {

cout << hex << i << '\n';

}

} ; *// End Class Outhex*

class outoct : **public** number {

public:

outoct(**int** n) : number(n) { }

void shownum () { cout << oct << i << '\n'; }

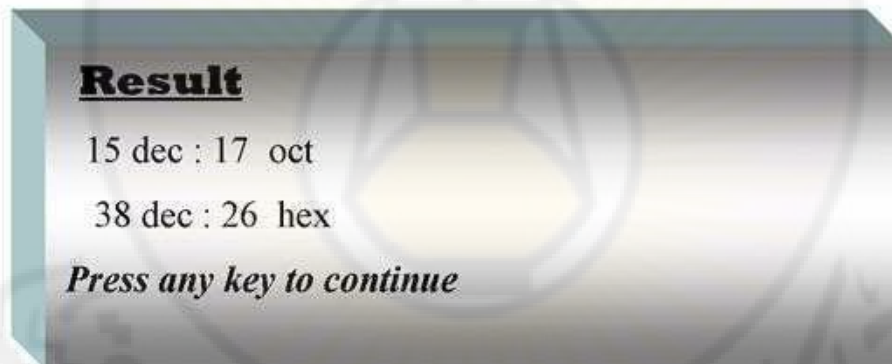
} ; *// End outhex Class*

int main() {

```

number *p ;
outoct O( 15 ) ;
outhex h( 38 ) ;
p = & O ;
p->shownum( ) ;
p = & h ;
p->shownum( ) ;
return 0;
} // End main

```



مثال (5 - 6)

اكتب برنامجاً بلغة C++ لإنشاء صف أساسي مجرد اسمه Decimal يحتوي على وسيط واحد كعضو خاص x من النوع int ويحتوي الصف على بان افتراضي وعلى تابع لتعديل المعطيات اسمه **number** وتابع افتراضي (virtual) اسمه shownum() والذي سيقوم بطباعة العدد العشري .

- اشتق من الصف الأساسي Decimal صفًا اسمه **HexDecima** وعند تجاوز الصف المشتق التابع الافتراضي (`shownum()`) من الصف الأساسي يقوم بحساب وطباعة الرقم العشري بالنظام السداسي عشر .
- اشتق من الصف **HexDecimal** صفًا اسمه **Octal** وعند تجاوز الصف المشتق التابع الافتراضي (`shownum()`) يقوم بحساب وطباعة الرقم العشري بالنظام الثماني .
- أنشئ غرضاً في التابع `main` للصفوف المشتقة واستدع نوابغ الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي لطباعة النتائج .

البرنامج

```
#include <iostream.h>

class Decimal {
private :
    int x ;
public:
    Decimal ( ) { x=0 ;}
    void number(int i) { x = i; }
    virtual void shownum( )
    {
        cout << " Decimal "; cout << x << "\n";
    }
}
```

```
void getdim ( int & d1)
```

```
{  
    d1 = x;  
}
```

```
};//End class Decimal
```

```
class HexDecimal : public Decimal {
```

```
public:
```

```
void shownum( )
```

```
{  
    int d1 ;  
    getdim(d1);  
    cout << " HexDecimal ";  
    cout << hex << d1 << "\n"; }  
}
```

```
};//End class Hexdecimal
```

```
class Octal : public HexDecimal {
```

```
public:
```

```
void shownum( )
```

```
{  
    int d2 ;  
    getdim(d2);
```



```

        cout << " Octal ";
        cout << oct << d2 << '\n';
    }
}; //End class Octal

int main( )
{
    Decimal *p;
    Decimal n;
    p = &n ;
    p-> number( 200 );
    p-> shownum( );

    Octal o ;
    p = &o ;
    p-> number( 200 );
    p-> shownum( );

    HexDecimal h ;
    p = &h ;
    p-> number(200 );
    p-> shownum( );
    return 0;
} // End main

```

Result

Decimal 200

Octal 310

HexDecimal c8

Press any key to continue

تمرين (5 - 1)

اكتب برنامجاً ينشئ صفئاً أساسياً اسمه `dist` يخزن المسافة بين نقطتين في متغير من النوع `double` و أنشئ في الصف `dist` تابع اسمه `( trav_time )` يظهر الوقت اللازم لقطع هذه المسافة بافتراض أن المسافة بالميل والسرعة 60 ميل/الساعة . أنشئ صفئاً مشتقاً اسمه `metric` يتجاوز التابع `( trav_time )` بحيث تعطي الزمن بافتراض أن المسافة معطاة بالكيلومتر والسرعة تساوي 100 Km / h .

٥-٥- الصفوف المجردة النقية

Pure Abstract Classes

في بعض الأحيان لا يكون للتابع الافتراضي أي عمل ذي معنى لينجزه في الصف الأساسي ، عندئذ يقوم الصف الأساسي بتأمين نواة مؤلفة من مجموعة توابع ومتغيرات أعضاء يحتاجها الصف المشتق أو الصفوف المشتقة ، وعندما لا يكون للتابع الافتراضي في الصف الأساسي أي عمل ذو معنى ، يتوجب على كل صف مشتق أن يتجاوز هذا التابع . للتأكد من حدوث ذلك فعلاً أوجدت لغة

C++ التوابع الافتراضية النقية (الصرفه) Pure virtual functions . لا تملك التوابع الافتراضية النقية أي تعريف في الصف الأساسي ، ويتضمن الصف الأساسي فقط التصريح الأولي للتابع .
لتعريف تابع افتراضي نقي هو كما يلي :

virtual type func_name (parameter – list) = 0 ;

الجزء المهم من هذا التصريح هو جعل التابع الافتراضي يساوي الصفر ، وهذا يخبر المترجم أنه لا يوجد أي جسم للتابع في الصف الأساسي ، و يجبر كل صف مشتق من الصف الأساسي على تجاوز هذا التابع وإلجبار الصف المشتق على إعادة تعريف التابع بالشكل الذي يناسبه وفي حال عدم حصول ذلك سيولد خطأ في الترجمة .

وعندما يحتوي الصف الأساسي على تابع افتراضي نقي واحد على الأقل فإنه يسمى صفًا مجرداً نقياً **Pure Abstract Class** . وبما أن الصف المجرد يحتوي على الأقل على تابع افتراضي نقي بدون جسم فإنه يُعد من الناحية التقنية نوعاً غير كامل ولا يمكن إنشاء أي أغراض منه . أي أن الصفوف المجردة النقية وجدت فقط لتكون صفاً أساسياً لصفوف مشتقة أخرى .
نعيد كتابة البرنامج في المثال (5-4) للصف **area** وجعله صفًا مجرداً نقياً ، كما هو موضح في المثال (5-7) .

مثال (5 - 7)

```
// Create an abstract class.  
#include <iostream.h>  
class area {
```

private :

double dim1, dim2; *// dimensions of figure*

public:

void setarea(**double** d1, **double** d2)

```
{  
    dim1 = d1;  
    dim2 = d2;  
}
```

void getdim(**double** &d1, **double** &d2)

```
{  
    d1 = dim1;  
    d2 = dim2;  
}
```

virtual double getarea() = 0 ; *// pure virtual function*

};//End class area

class rectangle : **public** area {

public:

double getarea() {

double d1, d2;

getdim(d1, d2);

return d1 * d2;

}

};// End class rectangle

```
Class triangle : public area {
```

```
public:
```

```
    double getarea( )
```

```
    {
```

```
        double d1, d2;
```

```
        getdim(d1, d2);
```

```
        return 0.5 * d1 * d2;
```

```
    }
```

```
}; // End class triangle
```

```
void main( ) {
```

```
    area *p ;
```

```
    rectangle r ;
```

```
    triangle t ;
```

```
    p = &r;
```

```
    p ->setarea(3.3, 4.5);
```

```
    cout << "Rectangle has area: " << p->getarea( ) << '\n';
```

```
    p = &t;
```

```
    p ->setarea(4.0, 5.0);
```

```
    cout << "Triangle has area: " << p->getarea( ) << '\n';
```

```
    } // End main
```

Result

Rectangle has area: 14.85

Triangle has area : 10

Press any key to continue

نلاحظ أن التابع () `getarea` أصبح تابعاً افتراضياً نقياً وعلى كل صف مشتق أن يتجاوز هذا التابع .

عندما يرث صف مشتق تابعاً افتراضياً من الصف الأساسي ثم يستخدم الصف المشتق كصف أساسي لصف مشتق آخر فإن التابع الافتراضي الموروث يمكن تجاوزه في الصف المشتق الثاني . إذا احتوى الصف الأساسي B على تابع افتراضي اسمه () `f` ، وورث الصف D1 الصف الأساسي B وورث الصف D2 الصف المشتق D1 ، أي :

$B \leftarrow D1 \leftarrow D2$

فإن كلا من D1 و D2 يستطيعان تجاوز التابع الافتراضي () `f` كل واحد ضمن صفه ، كما هو موضح في المثال (5 - 8) .

مثال (5 - 8)

// Virtual functions retain their virtual nature when inherited.

#include <iostream.h>

class base {

public:

virtual void func()

{

cout << "Using base version of func()\n";

}

}; // End class

class derived1 : public base {

public:

void func()

{

cout << "Using derived1's version of func()\n";

}

}; // End class derived1

// Derived2 inherits derived1.

class derived2 : public derived1 {

public:

void func()

{

cout << "Using derived2's version of func()\n";

}

}; // End class derived2

```
int main() {  
    base *p ;  
    base ob ;  
    derived1 d_ob1 ;  
    derived2 d_ob2 ;  
    p = &ob;  
        p->func( );    // use base's func( )  
    p = &d_ob1;  
        p->func( );    // use derived1's func( )  
    p = &d_ob2;  
        p->func( );    // use derived2's func( )  
    return 0;  
} // End main
```

Result

Using **base** version of func()

Using **derived1's** version of func()

Using **derived2's** version of func()

Press any key to continue

ويرث الصف derived1 في هذا البرنامج الصف الأساسي base ويتجاوز التابع الافتراضي fun () ويعيد تعريفه بما يناسبه، في حين يرث الصف derived2 الصف derived1 ويتجاوز كذلك التابع الافتراضي func () ويعيد تعريفه بما يناسبه. لأن التتابع الافتراضية مرتبة حسب ترتيب الاشتقاق ، وإذا لم يتجاوز derived2 التابع func () سيستخدم التابع func () للصف derived1 بدلاً من ذلك ، وفي حال عدم تجاوز التابع func () من قبل أي من الصفين derived1 و derived2 فإن جميع الإشارات إلى هذا التابع ستعود إلى تلك المعرفة في الصف الأساسي base .

مثال (5 - 9)

المثال التالي يوضح كيف يمكن استخدام تابعين افتراضيين في الصف المجرد النقي . يتم كتابة الصف area في المثال (5-9) بطريقة ثانية تختلف قليلاً عن البرنامج في المثال (5-7). حيث يحتوي الصف الأساسي على متحولين d1,d2 من نمط **protected** ، وعلى بان افتراضي وتابع تعديل معطيات . ويقوم الصف المشتق الأول بحساب مساحة ومحيط مستطيل (أبعاده d1,d2) وعند تجاوز التابعين الافتراضيين ، أما الصف المشتق الثاني فيقوم بحساب مساحة ومحيط دائرة قطرها d1 عند تجاوز التابعين الافتراضيين. حيث يستخدم التابع الافتراضي getarea1() لحساب المساحة. أما التابع getarea2() يستخدم لحساب المحيط .

البرنامج :

```
// EXAMPLE Create an Abstract Class
```

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

```

class Area {
protected :
    double d1 , d2 ;
public :
    Area ( ) { d1=1;d2=1 ;}
    void setarea(double dim1 , double dim2)
    {
        d1= dim1 ;
        d2 = dim2 ;
    }
    // pure virtual function1
    virtual double getarea1( ) = 0 ;
    // pure virtual function2
    virtual double getarea2( ) = 0 ;
}; // End class Area

```

```

//Inherite
class derived1 : public Area {
public :
    double getarea1 ( )
    {
        return d1*d2 ;
    }
}

```

```

    }
    double getarea2 ( )
    {
        return 2*(d1+d1);
    }
}; // End class

class derived2 : public Area {
public :
    double getareal ( )
    {
        return (3.14*d1*d1);
    }
    double getarea2 ( )
    {
        return 2*3.14*d1 ;
    }
}; // End class .

void main ( ) {
    Area *p; // pointer to area
    derived1 r;
    derived2 c;

    //use p to access derived1 object

```

```
p = &r ;
```

```
P->setarea(3.4, 4.5 ) ;
```

```
cout << " Rectangle has area : "<<p-> getareal( )  
<< endl;
```

```
cout << " Rectangle has around : "<<p-> getarea2( )  
<< endl;
```

```
// use p to access derived2 object
```

```
p = &c ;
```

```
p->setarea( 4.0 , 5.0 ) ;
```

```
cout << " Circal has area : " << p-> getareal( )  
<< endl;
```

```
cout << " Circal has aroud : " <<p-> getarea2( )  
<< endl;
```

```
} // End main
```

Result

Rectangle has area : 15.3

Rectangle has around : 13.6

Circal has area : 62.8

Circal has aroud : 25.12

Press any key to continue

٥-٦- تطبيقات تعدد الأشكال

Applying Polymorphism

إن تعدد الأشكال هي العملية التي يتم بواسطتها تطبيق واجهة مشتركة لعدة حالات متشابهة في الشكل ولكن مختلفة من الناحية التقنية ، وتكمن أهمية تعدد الأشكال في أنها تبسط الأنظمة المعقدة .تسمع طريقة تعدد الأشكال بتوضيح العلاقات المنطقية بين الأفعال المتشابهة ما يجعل البرنامج سهل الفهم والصيانة. وهناك مصطلحان يرتبطان مع البرمجة غرضية التوجه بشكل عام ومع ++C بشكل خاص وهما الربط المبكر Early Binding ، والربط المتأخر Late Binding . حيث يشير الربط المبكر إلى الأحداث التي يمكن أن تعرف زمن الترجمة ، ويشير بشكل خاص إلى استدعاءات التوابع التي يمكن معرفة وجهتها في زمن الترجمة ، بما فيها التوابع العادية وتوابع التحميل الزائد والتوابع الأعضاء غير الافتراضية والتوابع الصديقة ، وعند ترجمة الأنواع السابقة من التوابع فإن المعلومات اللازمة عن عناوينها تكون متوافرة في زمن الترجمة . إن الفائدة الأساسية هو فعاليتها ، إذ يعد استدعاء التوابع المربوطة في زمن الترجمة الأسرع بين أنواع استدعاءات التوابع أما سينتها فهي عدم المرونة .

بينما يشير الربط المتأخر إلى الأحداث التي يجب حصولها في زمن التنفيذ، أي أن عناوين و استدعاءات التوابع لا تعرف حتى يتم تنفيذ البرنامج . وتعد التوابع الافتراضية في لغة ++C أغراضاً ذات ربط متأخر ، فعند الوصول إلى تابع افتراضي عبر مؤشر إلى صف أساسي فإن البرنامج سيحدد في زمن التنفيذ إلى أي نوع من الأغراض يشير المؤشر ، واعتماداً على ذلك ينتقي نسخة التابع التي ستنفذ .الفائدة الأساسية من الربط المتأخر هي المرونة في زمن التنفيذ وبذلك يصبح حراً بالاستجابة إلى الأحداث العشوائية دون الحاجة إلى كتابة جزء كبير

من نص البرنامج ، أما سيئتها فهي وجود الزمن الكبير الزائد المترافق مع استدعاء التابع وهذا ما يجعل الاستدعاءات أبطأ من تلك التي تحدث مع الربط المبكر .

يعرف المثال (6-10) صف مجرد اسمه list (قائمة) من أجل قيم صحيحة int ، تتألف واجهة القائمة list من تابعين افتراضيين نقيين هما : store و retrieve يُستدعى التابع الأول store لتخزين قيمة ، ولاستعادة هذه القيمة أو قيمة أخرى من القائمة يُستدعى التابع retrieve ، لا يعرف الصف الأساسي list التابع الافتراضية الموجدة فيه ، بينما يعرف كل صف مشتق نوع القائمة التي تمثله . يتم تمثيل نوعين من القوائم هما : الرتل queue و المكسد stack ، رغم أن القائمتين تعملان بشكل مختلف إلا أن الوصول إلى أي منهما بحاجة إلى استخدام الواجهة نفسها .

// Demonstrate virtual functions.

مثال (5 - 10)

```
#include <iostream>
#include <cstdlib>
#include <cctype>
using namespace std;
class list {
public:
    list *head;    // pointer to start of list
    list *tail;    // pointer to end of list
    list *next;    // pointer to next item
    int num;       // value to be stored
```

```
list( ) { head = tail = next = NULL; }
virtual void store(int i) = 0;
virtual int retrieve( ) = 0;
}; //End class list
```

```
// Create a queue type list.
class queue : public list {
public:
    void store(int i);
    int retrieve( );
}; //end class queue
```

```
void queue::store(int i) {
    list *item;
    item = new queue;
    if( ! item ) {
        cout << "Allocation error.\n";
        exit(1);
    }
    item->num = i;
```

```
// put on end of list
if(tail) tail->next = item;
```

```

tail = item;
item->next = NULL;
if(!head) head = tail;
}
int queue::retrieve() {
    int i;
    list *p;
    if(!head) {
        cout << "List empty.\n";
        return 0;
    }

    // remove from start of list
    i = head->num;
    p = head;
    head = head->next;
    delete p;
    return i;
} //end retrieve

// Create a stack type list.
class stack : public list {
public:

```

```
void store(int i);  
int retrieve( );  
};//end class stack
```

```
void stack::store(int i) {  
    list *item;  
    item = new stack;  
    if(!item) {  
        cout << "Allocation error.\n";  
        exit(1);  
    }  
    item->num = i;  
  
    // put on front of list for stack-like operation  
    if(head) item->next = head;  
    head = item;  
    if(!tail) tail = head;  
}//end store
```

```
int stack::retrieve() {  
    int i;  
    list *p;  
    if(!head) {
```

```

    cout << "List empty.\n";
    return 0;
}
// remove from start of list
i = head->num;
p = head;
head = head->next;
delete p;
return i;
} //end retrieve

void main() {
    list *p;
    stack sob;
    queue qob;
    char ch;
    int i;
    for(i=0; i<10; i++) {
        cout << "Stack or Queue? (S/Q): ";
        cin >> ch;
        ch = tolower(ch);
        if(ch=='q') p = &qob;
        else p = &sob;
    }
}

```



```

    p->store(i);
}
cout << "Enter T to terminate\n";
for(;;) {
    cout << "Remove from Stack or Queue? (S/Q): ";
    cin >> ch;
    ch = tolower(ch);
    if(ch=='t') break;
    if(ch=='q') p = &qob;
    else p = &sob;
    cout << p->retrieve( ) << "\n";
}
cout << "\n";
} //End main

```

يبين التابع الرئيسي main كيف يمكن معالجة الأحداث العشوائية في زمن التنفيذ بواسطة التتابع الافتراضية وتعدد الأشكال في زمن التنفيذ . ينفذ البرنامج حلقة التكرار من 0 حتى 9 وفي كل مرة تكرر يسألنا البرنامج عن نوع القائمة التي سينتقيها لوضع القيمة فيه هل هي المكس؟ أم الرتل ؟. وفقا لأجابتنا سوف يشير المؤشر p إلى الغرض الصحيح ثم تخزن قيمة i وعند انتهاء الحلقة تبدأ حلقة جديدة تطلب منا تحديد أي قائمة ستسحب منها قيمة .



الأسئلة



١. ما التابع الافتراضي Virtual Function .
٢. ما التوابع التي لا يمكن جعلها افتراضية .
٣. كيف تساهم التوابع الافتراضية في إنجاز تعدد الأشكال في زمن التنفيذ ؟
٤. ما التابع الافتراضي الصرف أو النقي .
٥. ما الصف المجرد النقي ؟ وما الصف المتعدد الأشكال ؟
٦. هل يمكن وراثة التابع الافتراضي ؟
- ٧.





مسائل عامة محلولة



- ١- اكتب برنامجاً بلغة C++ يعرف صفّاً أساسياً مجرداً اسمه **Credit** يحتوي على ستة وسطاء $x1, x2, x3, x4, x5, x6$ من النوع **int** كأعضاء محمية **protected** وهي عبارة عن درجات المواد الدراسية للفصل الدراسي الأول ، حيث $x1, x2, x3, x4$ هي مقررات اختصاصية، ويحتوي الصف على بان افتراضي وعلى تابع لتعديل المعطيات اسمه **setCredit()** وعلى تابع افتراضي (**virtual**) اسمه **vertCredit()** سيقوم بطباعة درجات الطالب.
- اشتق من الصف الأساسي صفّاً مشتقاً اسمه **derived1** وعند تجاوز الصف المشتق التابع الافتراضي **vertCredit()** من الصف الأساسي سيقوم بحساب معدل درجات الطالب في المقررات الاختصاصية. ويطبع كلمة ناجح **Passed** إذا كان معدلة أكبر أو يساوي 60، ويطبع كلمة راسب **Failed** إذا كان أصغر من 60.
- اشتق من الصف **derived1** صفّاً مشتقاً اسمه **derived2** وعند تجاوز الصف المشتق التابع الافتراضي **vertCredit()** يقوم بحساب معدل الطالب في المقررات الستة ويطبع كلمة ناجح **Passed** إذا كان معدلة أكبر أو يساوي 60، ويطبع كلمة راسب **Failed** إذا كان أصغر من 60.
- أنشئ أغراضاً في التابع **main()** لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج .

```

#include <iostream.h>

class Credit {
protected :
    double x1 ,x2,x3,x4,x5,x6 ;
public :
    Credit ( ) {
        x1=x2=x3=x4=x5=x6=0.0 ;
    }
    void setCredit (double x11 ,double x21, double x31,
                    double x41,double x51,double x61)
    {
        x1=x11;
        x2=x21;
        x3=x31;
        x4=x41;
        x5=x51;
        x6=x61;
    }
    virtual void vertCredit( ) // virtual function
    {
        cout << " x1 = " << x1 << endl ;
        cout << " x2 = " << x2 << endl ;
        cout << " x3 = " << x3 << endl ;
    }

```

```

        cout << " x4 = " << x4 << endl ;
        cout << " x5 = " << x5 << endl ;
        cout << " x6 = " << x6 << endl ;
    }
}; // End Class Credit

// Inherite
class derived1 : public Credit {
public :
    void vertCredit ( )
    {
        double Rate1 = (x1+x2+x3+x4)/4 ;
        cout << " Rate1 = " << Rate1 ;
        if ( Rate1 >= 60.0 ) {
            cout << "    Passed " ;
            cout << endl ; }
        else { cout << " Failed " ;
            cout << endl ;
        }
    }
}; // End Class derived1 .

```

```

class derived2 : public derived1 {
public :
    void vertCredit ( )
    {
        double Rate = (x1+x2+x3+x4+x5+x6)/6.0 ;
        cout << " Rate = " << Rate ;
        if ( Rate >= 60.0 ) {
            cout << "   Passed " ;
            cout << endl ;}
        else { cout << " Failed " ;
            cout << endl ;}
    }
}; // End class derived2 .

```

```

void main ( ) {
    Credit *p; // pointer to Credit
    Credit c;
    derived1 s ;
    derived2 r ;

```

// use p to access Credit object

```

p = &c ;

```



```

p-> setCredit(50,60,40,80,90,77 ) ;
cout << " print course  : x1+2+x3... = ";
cout<< endl;
p-> vertCredit ( );
cout<< endl;

// use p to access derived1 object
p = &s ;
p-> setCredit( 50,60,40,80,90,77) ;
cout << "Summing courses : ";
p-> vertCredit ( );
cout<< endl;

// use p to access derived2 object
p = &r ;
p-> setCredit( 50,60,40,80,90,77) ;
cout << "Rate of Summing : ";
p-> vertCredit ( );
cout<< endl;

}// End main

```

Result

print course : x1+2+x3... =

x1 = 50

x2 = 60

x3 = 40

x4 = 80

x5 = 90

x6 = 77

Summing courses : Rate1 = 57.5 Failed

Rate of Summing : Rate = 66.1667 Passed

Press any key to continue

٢- اكتب برنامجاً بلغة C++ لإنشاء صف أساسي مجرد اسمه **Formula**

يحتوي على ثلاثة وسطاء خاصة **private** ، حيث x من نمط **double** و n و m من نمط **int** ، ويحتوي الصف على باني افتراضي وعلى تابع تعديل معطيات اسمه () **SetFormula** وتابع افتراضي (virtual) اسمه () **Print** يقوم بحساب ($m! + n!$) في الصف الأساسي .

• اشتق من الصف الأساسي صفاً مشتقاً اسمه **FormulaSum** وعند تجاوز الصف المشتق التابع الافتراضي () **Print** من الصف الأساسي سيقوم بحساب

المقدار $\sum_{i=1}^n x^i$ ويطبع الناتج .

• اشتق من الصف **FormulaSum** صفاً مشتقاً اسمه **FormulaFact** وعند تجاوز الصف المشتق التابع الافتراضي **Print()** يقوم بحساب المقدار: $\frac{m!}{n!}$ ويطبع النتيجة .

▪ أنشئ أغراضاً في التابع **main()** لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج .

١. إذا لم يتجاوز الصف المشتق **FormulaSum** التابع الافتراضي **Print()** ، ماذا يحصل؟ وضح ذلك؟

٢. إذا لم يتجاوز الصف المشتق **FormulaFact** التابع الافتراضي **Print()** ، وكان الصف صفاً مجرداً نقياً ، ماذا يحصل؟ وضح ذلك؟

٣. كيف يمكن جعل الصف المجرد السابق صفاً مجرداً نقياً؟ وضح ذلك؟

البرنامج :

```
#include <iostream.h>
#include <math.h>
class Formula {
private :
    double x ;
    int n , m ;
public :
    Formula ( ){ x=0.0 ; n=0; m=0 ;}
    void SetFormula( double x1 , int n1 ,int m1 )
    {
        x= x1 ; n = n1 ; m=m1 ;
    }
}
```

```

virtual void Print( ) {
    double x1 ; int n1 ,m1;
    getdim( x1 , n1 , m1);
    cout << " Factory : " ;
    int factn=1 ; double factm = 1;
    for( int i=1;i<= n1 ; i++) factn = factn * i ;
    for( int j=1;j<= m1 ; j++) factm = factm * j ;
    cout << " factn+factm= "<<factn+factm<<endl ;
}

void getdim( double &x1 , int &n1,int &m1 )
{
    x1=x ; n1=n ; m1=m;
}

}; // End class

```

// Inherits

```

class FormulaSum : public Formula {
public :
void Print() {
    double x1 ;
    int n1,m1 ;
    getdim( x1 , n1 , m1);
    cout << " Summing : " ;

```

```

    double sum=0.0 ;
    for( int i=1;i<=n1 ; i++ )  sum = sum + pow(x1,i) ;
    cout << "x1="<<x1<<" n1="<<n1 << " sum= "
        <<sum << endl ;
    }
}; // End class .

// Inherits
class FormulaFact : public FormulaSum {
public :
void Print( ) {
    double x1 ;
    int n1 ,m1;
    getdim( x1 , n1 , m1) ;
    cout << " Factory : " ;
    int factn=1 ; double factm = 1; double factDiv ;
    for( int i=1;i<=n1 ; i++ ) factn = factn * i ;
    for( int j=1;j<=m1 ; j++ ) factm = factm * j ;
    factDiv = factm / factn ;
    cout << " factn= "<<factn<<" factm= "<<factm<<endl
        << " factDiv= "<<factDiv << endl ;
    }
}; // End class .

```

```

void main ( ) {
    Formula *p ;           // pointer to Formula type
    Formula obase ;        // object of Formula
    obase. SetFormula ( 40.0 ,10 ,20) ;
    FormulaSum obsum ;     // object of FormulaSum
    obsum. SetFormula ( 40.0 ,10 ,20) ;
    FormulaFact obfact ;   // object of FormulaFact
    obfact . SetFormula ( 40.0 , 10 , 20) ;

    // use p to access Formula object
    p = &obase ;
    p-> Print( );

    // use p to access FormulaSum object
    p = &obsum ;
    p-> Print( );

    // use p to access FormulaFact object
    p = &obfact ;
    p-> Print( );

} // End main

```


Result

Factory : $\text{factn} + \text{factm} = 2.4329\text{e}+018$

Summing : $x1 = 40$, $n1 = 10$, $\text{sum} = 1.07546\text{e}+016$

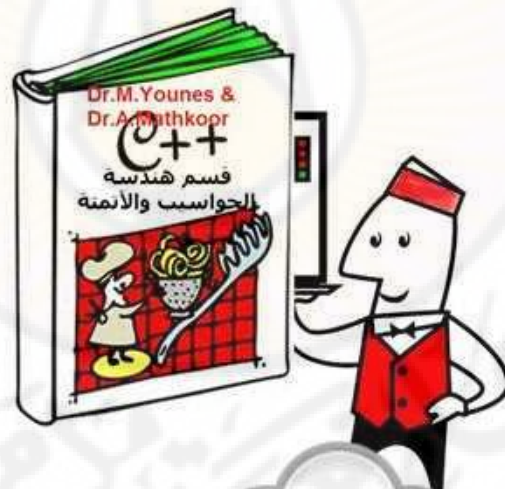
Factory : $\text{factn} = 3628800$, $\text{factm} = 2.4329\text{e}+018$

factDiv = $6.70443\text{e}+011$

Press any key to continue

ملاحظة :

يترك للطالب الإجابة على الأسئلة الثلاث الموجودة في المسألة .





مسائل عامة



- ١- اكتب برنامجاً لإنشاء صف أساسي مجرداً بلغة C++ اسمه equation يحتوي على ثلاثة وسطاء x, y, z من النوع double كأعضاء محمية ويحتوي الصف على بان افتراضي وعلى بان عادي وعلى تابع افتراضي اسمه $equat()$ يقوم بحساب وطباعة مجموع مربعات x, y, z .
- اشتق من الصف الأساسي equation صفاً مشتقاً اسمه $derived1$ ، وعند تجاوز الصف المشتق التابع الافتراضي $func()$ يقوم بحساب وطباعة حجم هرم ارتفاعه z وقاعدته مستطيل أبعاده x, y .
 - اشتق من الصف $derived1$ صفاً مشتقاً اسمه $derived2$ ، وعند تجاوز الصف المشتق التابع الافتراضي $fun()$ يقوم بحساب وطباعة حجم متوازي المستطيلات أبعاده x, y, z .
 - أنشئ أغراضاً في التابع $main()$ لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج.

- ٢- اكتب برنامجاً بلغة C++ لإنشاء صف أساسي مجرد اسمه formula يحتوي على ثلاثة وسطاء A, B, C من النوع float كأعضاء محمية ويحتوي الصف على بان افتراضي وعلى تابع تعديل معطيات set وعلى تابع افتراضي اسمه $form()$ والذي يقوم بحساب وطباعة المتوسط الحسابي للأعداد A, B, C .
- اشتق من الصف الأساسي formula صفاً مشتقاً اسمه $derived1$ ، وعند تجاوز الصف المشتق التابع الافتراضي $form()$ يقوم بحساب وطباعة مجموع مربعات الأعداد A, B, C .

- اشتق من الصف الأساسي formula صفاً مشتقاً اسمه derived2، وعند تجاوز الصف المشتق التابع الافتراضي () form يقوم بحساب وطباعة مجموع الأعداد A , B , C .
- أنشئ أغراضاً في التابع () main لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج .

=====

٣- اكتب برنامجاً بلغة C++ لإنشاء صف أساسي مجرد نقي اسمه Formula يحتوي على ثلاثة وسطاء من نمط private ، حيث x من نمط double و n و y من نمط int ، ويحتوي الصف على بانٍ افتراضي وعلى تابع تعديل معطيات اسمه () SetFormula وتابع افتراضي (virtual) اسمه () Print .

- اشتق من الصف الأساسي صفاً مشتقاً اسمه FormulaSum وعند تجاوز الصف المشتق التابع الافتراضي () Print من الصف الأساسي سيقوم بحساب: $\sum_{i=1}^n (2x \times i)$ ويطبع الناتج .

- اشتق من الصف FormulaSum صفاً مشتقاً اسمه FormulaFact وعند تجاوز الصف المشتق التابع الافتراضي () Print يقوم بحساب المقدار: $\frac{y!}{n!}$ ويطبع النتيجة .

أنشئ أغراضاً في التابع () main لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج .

- إذا لم يتجاوز الصف المشتق FormulaSum التابع الافتراضي () Print ، ماذا يحصل؟ وضح ذلك؟
- إذا لم يتجاوز الصف المشتق FormulaFact التابع الافتراضي () Print ، ماذا يحصل؟ وضح ذلك؟

- كيف يمكن جعل الصف المجرد النقي السابق صفًا مجرداً ؟ وضح ذلك ؟

=====

٤- اكتب برنامجاً بلغة C++ لإنشاء صف أساسي مجرد نقي اسمه Credit يحتوي على ثلاثة وسطاء خاصة private من نمط char وهي اسم الطالب الثلاثي first و second و last ، ويحتوي أيضاً على خمسة وسطاء خاصة private من النمط int ، وهي عبارة عن درجات المقررات الدراسية للفصل الدراسي الأول x1,x2,x3,x4,x5 ، حيث x1,x2,x3 هي مقررات اختصاص ويحتوي الصف على تابع تعديل معطيات اسمه () setCredit وتابع افتراضي (virtual) اسمه () PrintCredit .

- اشتق من الصف الأساسي صفًا مشتقاً اسمه studcomp وعند تجاوز الصف المشتق التابع الافتراضي () PrintCredit من الصف الأساسي سيقوم بحساب معدل درجات الطالب في المقررات الاختصاصية ، ويطبع اسم الطالب ومعدله في المقررات الاختصاصية وكلمة ناجح Passed إذا كان معدله أكبر أو يساوي 60، ويطبع كلمة راسب Failed إذا كان أصغر من 60.

- اشتق من الصف studcomp صفًا مشتقاً اسمه studcourse وعند تجاوز الصف المشتق التابع الافتراضي () PrintCredit يقوم بحساب معدل الطالب في المقررات الخمسة ويطبع اسم الطالب ومعدله وكلمة ناجح Passed إذا كان معدله أكبر أو يساوي 60، ويطبع كلمة راسب Failed إذا كان أصغر من 60.

- أنشئ أغراضاً في التابع () main لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج .

=====

- ٥- أنشئ صفاً أساسياً مجرداً بلغة C++ اسمه Student يحتوي على ثلاث درجات D1,D2,D3 لثلاثة مقررات وهي وسطاء من النوع Double كأعضاء محمية protected ، ويحتوي على بان افتراضي وعلى تابع لتعديل المعطيات Setstud وتابع افتراضي virtual واحد اسمه Print() يقوم بحساب وطباعة المتوسط الحسابي للدرجات (معدل الطالب) .
- اشتق من الصف الأساسي Student صفاً مشتقاً اسمه Maximum ، و عند تجاوز الصف المشتق التابع الافتراضي Print() يقوم بحساب وطباعة أكبر درجة .
 - اشتق من الصف المشتق الأول Max صفاً مشتقاً اسمه Minimum ، و عند تجاوز الصف المشتق الثاني التابع الافتراضي Print() ، يقوم بحساب وطباعة أصغر درجة .
 - أنشئ غرضاً في التابع main() لاستدعاء توابع الأعضاء للصفوف المشتقة باستخدام مؤشرات الصف الأساسي وطباعة النتائج .
١. إذا لم يتجاوز الصف المشتق الأول Maximum التابع الافتراضي وكان الصف Student صفاً مجرداً، ماذا يحصل؟ وضح ذلك؟
 ٢. كيف يمكن جعل الصف المجرد السابق صفاً مجرداً نقياً (صرف) ؟ وضح ذلك ؟
 ٣. إذا لم يتجاوز الصف المشتق الثاني Minimum التابع الافتراضي ، وكان الصف Student صفاً مجرداً نقياً ماذا يحصل؟ وضح ذلك؟
 ٤. ما هي التوابع التي لا يمكن جعلها افتراضية .



الفصل السادس

القوالب ومعالجة الاستثناءات

Template and Exception Handling





القوالب ومعالجة الاستثناءات

Template and Exception Handling

٦-١ - مقدمة

من الإمكانيات المهمة في لغة C++ القوالب Templates، وتساعد القوالب على تحديد مجال كامل من التوابع المرتبطة (المحملة بشكل زائد، التي تسمى بتوابع القوالب) أو مجال من الصفوف المرتبطة (التي تسمى بصفوف القوالب).

تمثل معالجة الاستثناءات نظاماً جزئياً في C++ يسمح بمعالجة الأخطاء التي تحدث في زمن التنفيذ بطريقة بنيوية ومحكومة و إنها تؤتمت معظم عمليات معالجة الأخطاء التي كانت تكتب يدوياً مهما كان حجم البرنامج .

٦-٢ - قوالب التوابع

Function Templates

تستخدم التوابع المحملة بشكل زائد للقيام بعمليات متشابهة على معطيات من أنماط مختلفة . لكن إذا كانت هذه العمليات متطابقة من أجل عدة أنماط معطيات ، ويمكن تحقيق ذلك بشكل أفضل وأكثر ملائمة باستخدام إمكانية بناء قوالب التوابع Function Templates حيث يكتب المبرمج تعريفاً وحيداً لقالب تابع ما . ثم تقوم لغة C++ بتوليد توابع منفصلة لمعالجة كل نمط من أنماط المعطيات المستخدمة عند استدعاء التابع . أي أن تعريف قالب وحيد لتابع يعطي إمكانية للحصول على مجموعة من التوابع .

يتم إنشاء التابع العام باستخدام الكلمة المحجوزة `template` ، وتستخدم لإنشاء قالب أو هيكل يصف ما يجب أن يفعله التابع تاركاً مهمة إضافة التفاصيل اللازمة للمترجم ، والشكل العام لقالب التابع كما يلي :

```
template < class Type > ret-type func-name ( parameter List )
{
    // Body of function
}
```

- ❖ يمثل `Type` اسم نوع المعطيات المستخدم من قبل التابع .
 - ❖ تبدأ تعريفات قوالب التوابع جميعها بالتعليمة `template` التي يرد بعدها قائمة بأنماط وسطاء التابع الوهمية بين الإشارتين `< ... >` .
 - ❖ يجب أن تسبق الكلمة المحجوزة `class` اسم كل وسيط وهمي ، وإذا لم تكتب الكلمة المفتاحية `class` قبل نمط أي وسيط وهمي ضمن تعريف قالب التابع سوف يحدث خطأ قواعدي .
 - ❖ يمكن أن تكون أنماط الوسطاء أي نمط من الأنماط المعروفة في لغة C++ ، ثم يتم بعدها تعريف جسم التابع .
- على الرغم من شيوع استخدام الكلمة المحجوزة `class` لتحديد نوع عام في تصريح القالب إلا أننا نستطيع استخدام الكلمة المحجوزة `typename` .

مثال (1-6)

اكتب برنامجاً بلغة C++ لإنشاء قالب عام لتابع التبديل `swap` لتبديل قيم المتحولين الممررين `a` و `b` .

وبما أن عملية التبديل مستقلة عن نوع المتحولات فإنه من المناسب إنجاز هذه العملية بواسطة القوالب .

```
#include <iostream.h>

// function template swapargs definition
template < class X >
void swap( X &a, X &b ) {
    X temp ;
    temp = a ;
    a = b ;
    b = temp ;
} // end function swapargs

void main( ) {
    int i = 25 , j = 50 ;
    double x = 25.55 , y = 50.22 ;
    cout << "Original i,j: " <<i<< " " <<j << endl;
    // call integer function-template specialization
    swap( i , j );
    cout << "Swap intègres : " <<i<< " " <<j<< endl;
    cout << "Original x , y : " <<x<< " , " <<y << endl;
    // call double function-template specialization
    swap( x , y );
    cout << "Swap doubles : " <<x<< " , " <<y<< endl;
} // end main
```

Result

Original i , j : 25 50

Swap integers : 50 25

Original x , y : 25.55 , 50.22

Swap doubles : 50.22 , 25.55

Press any key to continue

وتستخدم الكلمة المحجوزة (التعليمة) `template` لتعريف قالب التابع العام ،

حيث يقوم المترجم عند ترجمة السطر التالي بشيئين :

- تشكيل قالب
- بداية تعريف تابع عام

```
template< class X > void swap ( X &a, X &b )
```

• يمثل `X` اسم النوع العام للتابع ، بعد ذلك يتم التصريح عن التابع

(`swap`) باستخدام الصف `X` كنوع معطيات للقيم المراد تبديلها

.يستدعى التابع (`swap`) في التابع الرئيسي (`main`) باستخدام نوعين

مختلفين من المعطيات `int` و `float`. بما أن التابع (`swap`) هو قالب

عام فإن المترجم سينشئ تلقائياً نسختين من التابع (`swap`) :

- الأولى تستبدل القيم الصحيحة .
- الثانية تستبدل القيم الحقيقية .

ويمكن استخدام التعليمة `typename` بدلاً من التعليمة `class` لتحديد النوع العام في تعريف القالب ، كما هو مبين في التصريح الآخر للتابع `swap ( )` كما يلي :

```
// Use typename
template < typename X >
void swap ( X &a , X &b {
    X temp ;
    temp = a ;
    a = b ;
    b = temp ;
} // end function swap
```

ويجب أن تأتي الكلمة المحجوزة `template` قبل بداية تعريف التابع مباشرة ، ولا يجوز أن تكون هناك أي تعليمة بين الكلمة المحجوزة `template` وبداية تعريف التابع، فالتعبير التالي غير صحيح :

```
// This will not compile
template < typename X >
int z ; // This is an Error
void swap ( X &a , X &b {
    X temp ;
    temp = a ;
    a = b ;
    b = temp ;
} // end function swap
```


ويمكن تعريف أكثر من نوع عام في عبارة `template` وذلك على شكل لائحة يفصل بين عناصرها فواصل و ينشئ البرنامج التالي قالب لتابع يملك نوعين عامين للمعطيات المراد طباعتها .

```
#include <iostream.h>
template< class type1 , class type2 >
void myfunc ( type1 x , type2 y )
{
    cout << x << " " << y << endl ;
} // End function
void main ( )
{
    myfunc ( 20 , " Hi " ) ;
    myfunc ( 20.55 , 20L ) ;
} // End main
```

و تستبدل الأسماء `type1` و `type2` بواسطة المترجم بأنواع المعطيات `int` و `double` بالنسبة لـ `type1` أما `type2` فتستبدل بـ `char*` و `Long` ، وذلك عندما ينشئ المترجم النسخة المخصصة من التابع `myfunc ( )` .
يشبه التابع العام تابع التحميل الزائد إلا أنه أكثر تحديداً ، بينما ينفذ تابع التحميل الزائد أفعالاً مختلفة في جسم التابع . أما التوابع العامة فيجب أن تنجز العمل نفسه في جميع النسخ ، فمثلاً ، لا يمكن استبدال توابع التحميل الزائدة التالية بتوابع عامة لأنها لا تقوم بالعمل نفسه .

```
void outdata( int i )
```

```
{  
    cout << i ;  
}
```

```
void outdata( double d )
```

```
{  
    cout << setprecision(10) << setfill( ' #' ) ;  
    cout << d;  
    cout << setprecision(6) << setfill( ' ' );  
}
```

ومع أن التابع قالب يستطيع التحميل الزائد لنفسه كلما احتاج لذلك إلا أننا نستطيع أيضاً إجراء التحميل الزائد لأحد هذه التوابع بشكل صريح. إذا قمنا بالتحميل الزائد لتابع قالب فإن تابع التحميل الزائد سوف يتجاوز تابع القالب نسبة إلى تلك النسخة المحدودة . لدينا مثلاً هذه النسخة من المثال (6 - 1) .

//Overriding a template function.

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

```
template <class X>
```

```
void swap ( X &a , X &b ) {
```

```
    X temp;
```

```

temp = a;
a = b;
b = temp;
}

```

//This overrides the generic version of swapargs().

```

void swap ( int a, int b)

```

```

{
    cout << "this is inside swap (int , int ) \n";
}

```

```

int main( ) {

```

```

    int i =25 , j =50;

```

```

    float x = 25.55 , y =50.22 ;

```

```

    cout << "Original i, j: " << i << ' ' << j << endl;

```

```

    cout << "Original x, y: " << x << ' ' << y << endl;

```

```

    swap (i, j); // this calls the explicitly

```

```

        // overloaded swapargs()

```

```

    swap (x, y); // swap floats

```

```

    cout << "Swapped i, j: " << i << ' ' << j << endl;

```

```

    cout << "Swapped x, y: " << x << ' ' << y << endl;

```

```

    return 0;

```

```

} //End main

```

Result

```
Original i , j : 25 50
Original x , y : 25.55 , 50.22
this is inside swap (int,int)
Swapped i, j: 25 50
Swapped x, y: 25.55 50.22
Press any key to continue
```

توضح التعليقات أن استدعاء التابع (i , j) swap يؤدي إلى استدعاء نسخة التحميل الزائد من التابع () swap المعروف في البرنامج .أي أن المترجم لن يقوم بتوليد نسخة من تابع القالب () swap لأن تابع التحميل الزائد قد تجاوز تابع القالب بشكل صريح ، ويسمح التحميل الزائد اليدوي لتابع القالب المبنية في هذا المثال بصنع تابع قالب يناسب حالة معينة ، وبشكل عام إذا تطلب البرنامج وجود نسخة مختلفة من تابع ما لأنواع معطيات مختلفة فإنه يتوجب استخدام تابع تحميل زائد عوضاً عن القالب .

مثال (2-6)

يقوم البرنامج التالي بتعريف قالب تابع من النمط T لحساب القيمة العظمى لثلاثة أعداد من النمط المحدد . ويصرح قالب التابع عن وسيط وهمي من النوع T وبالتالي يتم استدعاء التابع maximum ضمن التابع الرئيسي main ويقوم

باستبدال نمط المعطيات T بنمط المعطيات المستخدم أثناء عملية الاستدعاء وذلك من أجل خلق نسخة كاملة عن التابع لحساب القيمة العظمى من بين ثلاثة أعداد من نمط محدد. نستطيع أن نعتبر القوالب مولدات للتوابع . في هذا المثال ثلاث طرق لاستدعاء التابع maximum في التابع الرئيسي main مع الأنماط char , double , int ، ما يؤدي إلى خلق ثلاث نسخ عن هذا التابع من أجل كل نمط .

```
#include <iostream.h>

// definition of function template maximum
template < class T > // or template< typename T >
T maximum( T x1, T x2, T x3 ) {
    T max = x1;
    if ( x2 > max )
        max = x2;
    if ( x3 > max )
        max = x3;
    return max;
} // end function template maximum

void main( ) {
    // demonstrate maximum with int values
    int i1, i2, i3;
    cout << "Input three integer values: ";
    cin >> i1 >> i2 >> i3;
```



```
// invoke int version of maximum
cout << "The maximum integer value is: "
    << maximum( i1, i2, i3 );

// demonstrate maximum with double values
double d1, d2, d3;
cout << "\n\nInput three double values: ";
cin >> d1 >> d2 >> d3;

// invoke double version of maximum
cout << "The maximum double value is: "
    << maximum( d1, d2, d3 );

// demonstrate maximum with char values
char ch1, ch2, ch3;
cout << "\n\nInput three characters: ";
cin >> ch1 >> ch2 >> ch3;

// invoke char version of maximum
cout << "The maximum character value is: "
    << maximum( ch1, ch2, ch3 ) << endl;
} // end main
```


Result

Input three integer values: 22 44 11
The maximum integer value is: 44

Input three double values: 14.5 12.5 66.5
The maximum double value is: 66.5

Input three characters: m y f
The maximum character value is: y

Press any key to continue

مثال (3-6)

يمكن أن نكتب قالب تابع من أجل عملية فرز عناصر المصفوفة ثم نقوم C++ بتوليد توابع منفصلة لفرز عناصر مصفوفة من النمط int أو من النمط float أو من النمط char . وهكذا .

ويتضمن القالب printArray تصريحاً عن الوسيط الوهمي T ، حيث T أي نمط للمصفوفة التي سيتم طباعة عناصرها مرتبة بشكل تصاعدي بموجب التابع printArray() . يتم الرجوع إلى T على أنها نمط الوسيط .

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

```
// function template printArray definition
```

```
template< class T >
```

```

void printArray( T *array, int count ) {
T Temp;
    for(int i=0; i<count-1; i++) // ArraySize-1
    {
        for(int j=0 ; j<count-1-i ; j++)
        {
            if(array[j] > array[j+1])
            {
                Temp = array[j];
                array[j] = array[j+1];
                array[j+1] = Temp;
            }
        }
    }
    for ( int i1 = 0; i1 < count ; i1++ )
        cout << array[ i1 ] << " ";
        cout << endl;
} // end function printArray

```

```

void main( ) {
    const int aCount = 5, bCount = 7, cCount = 7;
    int a[ aCount ] = { 5, 2, 7, 4, 5 };

```

```

double b[ bCount ] = { 4.1, 2.2, 8.3, 4.4, 1.5, 6.6, 7.7 };
char c[ cCount ] = "Younse"; // 7th position for null
cout << "Array a contains:" << endl;
    // call integer function-template specialization
printArray( a, aCount );
cout << "Array b contains:" << endl;
    // call double function-template specialization
printArray( b, bCount );
cout << "Array c contains:" << endl;
    // call character function-template specialization
printArray( c, cCount );
} // end main

```

Result

Array a contains:

2 4 5 5 7

Array b contains:

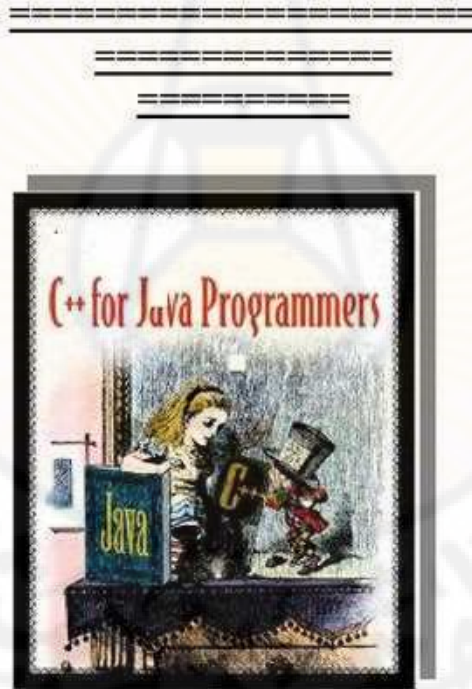
1.5 2.2 4.1 4.4 6.6 7.7 8.3

Array c contains:

Y e n o s e

Press any key to continue

وعندما يلتقط المترجم استدعاء للتابع (`printArray`) ضمن نص البرنامج فإنه يقوم باستبدال `T` ضمن قالب التابع بنمط الوسيط الممر فعلياً . ثم تقوم `C++` بإنشاء نسخة كاملة عن التابع لطباعة عناصر المصفوفة ذات النمط المحدد بوساطة نمط الوسيط . يجري بعدها ترجمة التابع الجديد ويقوم البرنامج بإنشاء ثلاث نسخ من التابع (`printArray`) ، واحدة لطباعة عناصر المصفوفة المرتبة من النمط `int` والثانية لطباعة عناصر المصفوفة المرتبة من النمط `double` والثالثة لطباعة عناصر المصفوفة المرتبة من النمط `char` .



مسائل عامة محلولة

١ - اكتب برنامجاً بلغة C++ لتعريف قالب تابع يدعى min() بحيث يعيد القيمة الصغرى من وسائطه . في التابع الرئيسي (main) أنشئ ثلاثة أغراض الأول يعيد القيمة الصغرى للمتحولين من النمط double والغرض الثاني يعيد القيمة الصغرى للمتحولين من النمط int والثالث يعيد القيمة الصغرى للمتحولين من النمط char.

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

```
template <class X >
```

```
X min( X a, X b)
```

```
{
```

```
    if( a <= b)
```

```
        return a;
```

```
    else
```

```
        return b;
```

```
}
```

```
int main() {
```

```
    cout << min(25.2, 29.55);
```

```
    cout << endl;
```

```
    cout << min(30, 40);
```

```

cout << endl;
cout << min('Y', 'M');
cout << endl;
return 0;
} //End main

```



٢- يُعد التابع (find) مرشحاً جيداً لكي يكون قالباً. يبحث هذا التابع عن غرض في مصفوفة ويعيد إما دليل الغرض المطابق أو يعيد (-1) إذا لم يجد غرضاً مطابقاً و فيما يلي تعريف أولي لنسخة محددة من التابع (find) :

```
int find( int object , int * list , int size )
```

حول التابع (find) إلى تابع عام ثم وضع الحل ببرنامج (يحدد الوسيط size عدد عناصر المصفوفة) .

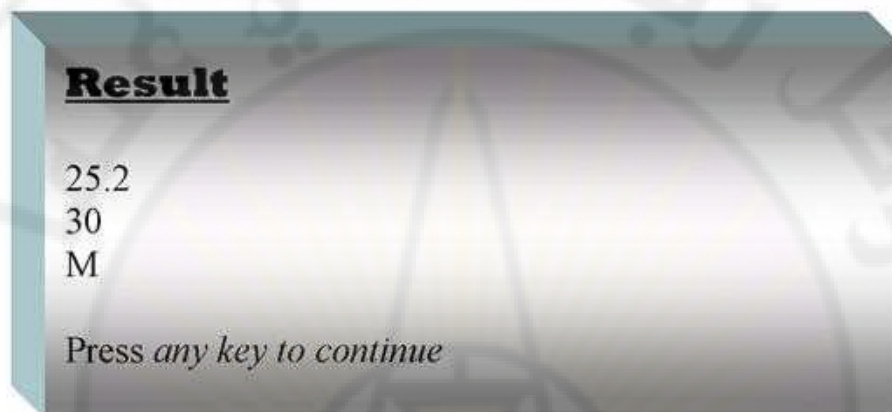

```

#include <iostream.h>
#include <string.h>
template < class X >
int find( X object, X *list, int size)
{
    for( int i=0; i<size; i++)
        if( object == list[i])
            return i;
    else
        return -1;
}

int main( )
{
    int a[ ] = {10, 22, 22, 40};
    char *c = "this is a test";
    double d[ ] = {10.10, 20.20, 30.30};
    cout << find(22, a, 4);
    cout << endl;
    cout << find( 'a', c , (int) strlen(c));
    cout << endl;
    cout << find(3.0, d, 3);
}

```

```
cout << endl;  
return 0;  
} //End main
```



٣- اشرح السبب الذي يجعل للتابع العام General Function قيمة وتساعد في تبسيط النص المصدري للبرنامج الذي نكتبه .

الجواب :

التوابع العامة مهمة لأنها تسمح للمبرمج بتعريف خوارزميات عامة يمكن تطبيقها على أنواع مختلفة من المعطيات . كما تساعد التوابع العامة على تنفيذ مبدأ واجهة واحدة ، وطرائق متعددة ، الذي يمثل المبدأ الأساسي في لغة C++ .



٦-٣ - قوالب الصفوف

Template Classes

إضافة إلى إمكانية تعريف تابع عام فإننا نستطيع أيضاً تعريف صف عام، عند إنشاء صف عام فإننا ننشئ صفاً يعرف جميع الخوارزميات المستخدمة من قبل ذلك الصف ولكن النوع الفعلي للمعطيات التي سيتعامل معها سيحدد كوسيط عند إنشاء أغراض من هذا الصف .

وتتجلى قوالب الصفوف عند إنشاء صف يحوي منطقاً قابلاً للتعميم ، فمثلاً خوارزمية الرتل Queue من أجل القيم الصحيحة يمكن استخدامها أيضاً من أجل إنشاء رتل من الرموز أو القيم الحقيقية و يمكننا بواسطة قالب الصف إنشاء صف يمثل رتل أو غير ذلك من أجل أي نوع من المعطيات . ويولد المترجم تلقائياً نوع الغرض الصحيح اعتماداً على النوع الذي تحدده عند إنشاء الغرض .
الشكل العام لقالب الصف :

```
template <class Type >
```

```
class class_name{
```

```
..... // Body of class
```

```
.....
```

```
};
```

حيث Type اسم النوع الذي سيحدد عند تمثيل الصف ، ويمكننا تعريف أكثر من نوع معطيات عام واحد باستخدام قائمة تفصل بين عناصرها فواصل ، حالما ننشئ قالب الصف نستطيع أن ننشئ مثال محدد من ذلك الصف باستخدام الشكل العام التالي :

```
class_name < type > ob ;
```

حيث إن `type` هو اسم نوع المعطيات الذي سيعمل معه الصف ، وكذلك تصبح التوابع العامة في قالب الصف تلقائياً توابع عامة ولا حاجة إلى تحديدها بوضوح باستخدام `template`. وبالتالي فإن لغة `C++` تقدم مكتبة من صفوف القوالب تسمى مكتبة القوالب القياسية `Standard Template Library` واختصاراً `STL` . تقدم هذه المكتبة نسخاً عامة لأكثر الخوارزميات وبنى المعطيات استخداماً. ينشئ المثال (4-6) صفّاً بسيطاً يمثل قائمة وحيدة الارتباط ، ثم يوضح البرنامج كيفية استخدام هذا الصف العام لإنشاء قائمة مترابطة تخزن رموزاً .

مثال (4-6)

```
// A simple generic linked list.
#include <iostream.h>
template < class T > class list {
    T data ;
    list *next;
public:
    list( T d);
    void add( list *node ) {node -> next = this; next = 0; }
    list *getnext( ) { return next; }
    T getdata( ) { return data ; }
}; //End class
```

```
template <class T> list< T>::list( T d)
```

```
{
```

```
    data = d;
```

```
    next = 0;
```

```
}
```

```
int main( ) {
```

```
    list< char > start('a');
```

```
    list< char > *p, *last;
```

```
    // build a list
```

```
    last = &start;
```

```
    for(int i=1; i < 26; i++) {
```

```
        p = new list<char> ('a' + i);
```

```
        p->add(last);
```

```
        last = p;
```

```
    }
```

```
    // follow the list
```

```
    p = &start;
```

```
    while(p) {
```

```
        cout << p->getdata( );
```

```
        p = p->getnext( );
```

```
    }
```

```
    cout<< endl ;
```

```

// int list
list<int> start1(1);
list<int> *p1, *last1;
// build a list
last1 = &start1;
for(int i1=1; i1 < 26; i1++) {
    p1 = new list<int> (1 + i1);
    p1->add(last1);
    last1 = p1; }
// follow the list
p1 = &start1;
while(p1) {
    cout << p1->getdata( );
    p1 = p1->getnext( );
}
cout<< endl ;
return 0;
} //End main

```

Result

abcdefghijklmnopqrstuvwxy

1234567891011121314151617181920212223242526

Press any key to continue

إن التصريح عن قالب صف مشابه للتصريح عن قالب تابع . و النوع الفعلي للمعطيات المخزنة في القائمة list هو نوع عام في تصريح الصف ، ولا يحدد نوع المعطيات الفعلي إلا عند التصريح عن غرض في القائمة list . في المثال (4-6) تم إنشاء الأغراض والمؤشرات داخل التابع الرئيسي (main) ، وتم تحديد نوع المعطيات من نمط char ضمن علامتي أكبر و أصغر < char > . هذا البرنامج يبني قائمة list مترابطة تحتوي على الأحرف الأبجدية ثم يظهرها ، ونستطيع تغيير نوع المعطيات المخزنة في القائمة list بتغيير نوع المعطيات المحدد عند إنشاء أغراض من النمط int و نستطيع مثلاً إنشاء غرض آخر يخزن قيمة صحيحة باستخدام التصريح التالي :

```
List < int > int_start(1) ;
```

وكذلك نستطيع استخدام القائمة list لتخزين أنواع المعطيات التي ننشأها بأنفسنا ، ويمكن استخدام البنية التالية لتخزين معلومات عن عناوين على سبيل المثال :

```
struct add {  
char name[ 40 ] ;  
char street[ 50 ] ;  
char city[ 30 ] ;  
char state[ 50 ] ;  
char zip[ 20 ] ;  
}
```

ثم نستخدم التصريح لإنشاء غرض list يخزن أغراض من النوع add كما يلي :

```
List < add > obj ( structvar ) ;
```

حيث إن structvar تحتوي على البنية add .

٦-٣-١ - قالب الصف Stack

فيما يلي مثال آخر عن قالب الصفوف، وهو الصف stack (المكدس) وهو عبارة عن بنية معطيات يمكن إدخال عناصر ضمنها وإخراجها منه وفقاً لطريقة العمل : الداخل أخيراً خارج أولاً أي Last In First Out بشكل مستقل عن نمط المعطيات التي يتعامل معها ، ولكن عندما نبدأ بالتفكير بإنشاء مكدس فعلي للمعطيات يجب أن نهتم بتحديد نمط المعطيات ، ويساعد ذلك على خلق فرصة جديدة لإعادة استخدام البرمجيات .

ونحتاج فقط إلى طريقة لوصف المكدس كصف ثم نشق منه مكدسات للتعامل مع أنماط معينة للمعطيات ، وتسمى هذه الإمكانية المتوفرة في لغة C++ بقوالب الصفوف . كما تسمى قوالب الصفوف أحياناً بالأنماط ذات الوسطاء لأنها تتطلب نمطاً أو أكثر كوسطاء لها لتحديد كيفية تخصيص قالب صف ليصبح صفاً ، ويقوم المبرمج الذي يرغب باستخدام قوالب الصفوف بكتابة تعريف قالب صف فقط ، وفي كل مرة يحتاج فيها إلى نسخة صف مرتبطة بنمط معين فإنه يحدد ذلك إلى المترجم الذي يقوم بدوره بتوليد النص البرمجي اللازم لذلك الصف .

مثال (5-6)

هذا المثال هو إعادة للصف stack الذي تم شرحه في الفصل الثاني ، والفرق هو أننا جعلنا الصف stack هنا قالب صف ، وبالتالي يمكن استخدامه لتخزين أي نوع من المعطيات ، وفي هذا المثال سوف ننشئ مكدساً لتخزين المعطيات من نمط char ومكدساً ثانياً لتخزين المعطيات من نمط double ومكدساً آخراً لتخزين المعطيات من نمط int.

// This function demonstrates a generic stack.

#include <iostream.h>

const int size= 10 ;

// Create a generic stack class

template < **class** **Type** > **class** stack {

Type stk[size]; *// holds the stack*

int tos; *// index of top-of-stack*

public:

void init() { tos = 0; } *// initialize stack*

void push(**Type** ch); *// push object on stack*

Type pop(); *// pop object from stack*

};*//end class*

// Push an object.

template < **class** **Type** >

void stack< **Type** >::push(**Type** ob) {

if(tos==size) {

cout << "Stack is full.\n";

return ;

}

stk[tos] = ob;

tos++;

}

// Pop an object.

```
template < class Type >
Type stack< Type >::pop( )
{
    if(tos == 0) {
        cout << "Stack is empty.\n";
        return 0;    // return null on empty stack
    }
    tos--;
    return stck[tos];
}
```

```
int main( ) {
    // Demonstrate character stacks.
    stack< char > s1, s2;    // create two stacks
    int i;
        // initialize the stacks
    s1.init( );
    s2.init( );
    char cs1,cs2 ;
    cout<<" Enter char value \n";
    for(i=0; i<3; i++) {
        cin >> cs1 >> cs2;
```

```

s1.push(cs1);
s2.push(cs2);
}

for(i=0; i<3; i++) cout << "Pop s1: " << s1.pop() << "\n";
cout << "-----\n";
for(i=0; i<3; i++) cout << "Pop s2: " << s2.pop() << "\n";
cout << "-----\n";

    // demonstrate double stacks
stack< double > ds1, ds2;    // create two stacks
    // initialize the stacks
ds1.init( );
ds2.init( );
double d1,d2 ;
cout<<" Enter double value \n";
for(i=0; i<3; i++) {
    cin >> d1 >> d2;
    ds1.push(d1);
    ds2.push(d2);
}

for(i=0; i<3; i++) cout << "Pop ds1: " << ds1.pop( ) << "\n";
cout << "-----\n";

```



```

for(i=0; i<3; i++) cout << "Pop ds2: " << ds2.pop( ) << "\n";
cout << "-----\n";

    // demonstrate int stacks
stack< double > ints1, ints2;    // create two stacks

    // initialize the stacks
ints1.init( );
ints2.init( );
int int1,int2 ;
cout<<" Enter int value \n";
for(i=0; i<3; i++) {
    cin >> int1 >> int2;
    ints1.push(int1);
    ints2.push(int2);
}

for(i=0; i<3; i++) cout << "Pop ds1: " << ints1.pop( ) << "\n";
cout << "-----\n";
for(i=0; i<3; i++) cout << "Pop ds2: " << ints2.pop( ) << "\n";
cout << "-----\n";
return 0;
} //End main

```


Result

Enter char value

M A M O U N

Pop s1: U

Pop s2: N

Pop s1: M

Pop s2: O

Pop s1: M

Pop s2: A

Enter double value

11.5 12.5 13.5 14.5 15.5 16.5

Pop ds1: 15.5

Pop ds2: 16.5

Pop ds1: 13.5

Pop ds2: 14.5

Pop ds1: 11.5

Pop ds2: 12.5

Enter int value

10 20 30 40 50 60

Pop ds1: 50

Pop ds2: 60

Pop ds1: 30

Pop ds2: 40

Pop ds1: 10

Pop ds2: 20

Press any key to continue

إن الصف stack يحتوي على متحولين خاصين هما [stck] و top ، تقوم المصفوفة [stck] بالاحتفاظ بالرموز المدفوعة إلى المكس و top دليل قمة المكس . يحتوي الصف stack على التوابع العامة التالية () init() لتهيئة المكس و () push لدفع قيمة إلى المكس و () pop لسحب قيمة من المكس .

ويقوم التابع () main بإنشاء مكدين S1, S2 ، ثم تدفع ثلاثة معطيات من نمط char إلى كل مكديس و تجدر الإشارة إلى أن المكدين S1, S2 منفصلين عن بعضهما بحيث لا تؤثر القيم المدفوعة إلى S1 على القيم المدفوعة إلى S2 أي أن كلاً من الغرضين يملك نسخة خاصة من المتحولين [stck و top . يُعد هذا المبدأ أساساً لفهم الأغراض على الرغم من أن جميع أغراض الصف تشترك بالتتابع العامة نفسها ، إلا أن كل غرض ينشئ ويحتفظ بأعضاء معطياته الخاصة .

وإذا كان لدينا مثلاً قالب الصف stack فإنه يمكن أن يصبح الأساس لبناء عدة مكديسات لأنماط مختلفة للمعطيات مثل : مكديس لمعطيات من النمط float أي stack of float أو من النمط int أي stack of int أو من النمط char أي stack of char أو من النمط Employee أي stack of Employee وغيرها .

ونلاحظ في تعريف قالب الصف stack في المثال (6-5) يأخذ هذا القالب الشكل العام لتعريف الصف ولكنه يبدأ بالجزء الرأسي التالي :

template < class Type >

للدلالة على أن التعريف التالي هو تعريف لقالب صف ذي وسيط من النمط Type ، ويدل النمط Type على نمط المكديس المطلوب إنشاءه . ويمكن للمبرمج أن يستخدم أي متحول آخر يرغب به بدل المتحول Type ، ويتم استخدام نمط الوسيط Type ضمن تعريف قالب الصف وضمن تعريف التتابع الأعضاء المرتبطة بقالب الصف .

يوجد شرطان أساسيان لاستخدام المكديس لتخزين عناصر من نمط مختلف عن الأنماط الأولية ، وهما :

- يجب أن يتوافر لدينا تابع بان ناسخ

- ويجب أن يكون قادراً على التعامل مع عملية الإسناد .

ويتضح من الصف stack والصف list أن قوالب التوابع وقوالب الصفوف تؤمن أداة قوية يمكن استخدامها لتقليص زمن البرمجة ، حيث يمكن تعريف الشكل العام للخوارزمية لاستخدامه مع أي نوع من المعطيات ، وبالتالي يوفر علينا مشقة كتابة إجراءات منفصلة لكل نوع معطيات نحتاج أن تعمل معه الخوارزمية .

٦-٣-٢ - قالب الصف لعدد من أنواع المعطيات

يمكن أن يملك قالب الصفوف على أكثر من نوع من المعطيات ، كما هو معروف لدينا في الصفوف ، ويتم ذلك بالتصريح عن كافة أنواع المعطيات المطلوبة في الصف في قائمة يفصل بين عناصرها فواصل ضمن التصريح template . المثال (6-6) ينشئ صفاً يستخدم نوعين من المعطيات .

مثال (6-6)

اكتب برنامجاً بلغة C++ لإنشاء قالب صف اسمه myclass يحتوي على نوعين من المعطيات الخاصة x,y ويحتوي على بان بوسيطين وتابع طباعة show . في التابع الرئيسي main صرح عن غرضين من المعطيات ، حيث يستخدم الغرض ob1 معطيات صحيحة int ومعطيات حقيقية double ، أما الغرض ob2 يستخدم معطيات محرفية من نمط char ومعطيات كسلسلة محرفية كمؤشر من نمط char* . ثم يستدعي كل من الغرضين تابع الطباعة لطباعة النتيجة . وفي كلتا الحالتين يولد المترجم تلقائياً المعطيات والتوابع المناسبة لتلائم الطريقة التي أنشئت فيها الأغراض .

/ This example uses two generic data types in a class definition.*

**/*

#include <iostream.h>

template < class T1, class T2 >

class myclass {

T1 x ;

T2 y ;

public:

myclass(T1 a , T2 b) { x = a ; y = b; }

void show() { cout << x << ' ' << y << '\n'; }

};//End class

void main() {

myclass< int , double > ob1(75, 70.5);

myclass< char * , char > ob2("The Result of Test is " , 'B');

ob1.show(); *// show int, double*

ob2.show(); *// show char, char **

}//End main

Result

75 70.5

The Result of Test is B

Press any key to continue

مثال (7-6)

اكتب برنامجاً بلغة C++ يعرف قالب الصف Point ويعرف النقطة (x , y) واقعة في الربع الأول (تعريف نقطة موجبة دوماً مهما كانت قيمة x و y) . ويحتوي على بان افتراضي يسند القيمة صفر للمتحويلات الخاصة ، وبان عادي بوسيطين لتمرير المعطيات إلى الأغراض عند إنشاءها ولتحويل النقطة دوماً إلى الربع الأول ، ويحتوي على تابعي وصول للمعطيات الخاصة . في التابع main نعرف غرضين من الصف Point ، الغرض الأول يعرف نقطة إحداثياتها من نمط int ويطبعها ، و الغرض الثاني يعرف نقطة إحداثياتها من نمط double ويطبعها .

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

```
template < class T >
```

```
class point {
```

```
private :
```

```
    T x,y ;
```

```
public :
```

```
    point ( ) { x=0; y=0 ;}
```

```

point(T new_x, T new_y) {
    if ( new_x < 0 )
        new_x *= -1;
    if ( new_y < 0 )
        new_y *= -1;
    x=new_x;
    y=new_y; }
T get_x() { return x; }
T get_y() { return y; }
}; // End class

void main( ) {
    point < int > obji(-1,-20); // int point
    cout << "obj1=" << obj1.get_x( ) << ",";
    cout << obj1.get_y( ) << " // int point" << endl;

    point < double > objd(-1.5,-20.5); // double point
    cout << " obj2 = " << obj2.get_x( ) << ",";
    cout << obj2.get_y( ) << " // double point" << endl;

} // End main

```


Result

```
obj1= 1, 20 // int point  
obj2 = 1.5 ,20.5 // double point  
Press any key to continue
```

تمرين (6-1)

اكتب برنامجاً بلغة C++ لإنشاء قالب صف يدعى input، يحتوي على متحول خاص data وباني بثلاث وسائط كما يلي :

input(char *s, X min, X max)

بحيث ينفذ الإجراءات التالية عند استدعاء الباني :

- إدخال قيمة data من المستخدم
- يعيد طلب المعطيات من المستخدم إذا لم تكن القيمة ضمن مجال محدد مسبقاً عند إنشاء الغرض .

يصرح عن الأغراض من نوع الصف input كما يلي :

input ob(" prompt message ", min-value , max-value)

تمثل message prompt رسالة تطلب من المستخدم إدخال قيمة ، وتحدد min-value القيم الصغرى و max-value القيم العظمى المقبولة .



٦-٤ - معالجة الاستثناءات

Exception Handling

تؤمن لغة C++ آلية جاهزة لمعالجة الأخطاء وتسمى معالجة الاستثناءات Exception Handling و نستطيع باستخدام هذه الآلية إدارة أخطاء زمن التنفيذ والاستجابة لها بسهولة أكبر ، وتبنى معالجة الأخطاء في لغة C++ على ثلاث كلمات محجوزة وهي :

- التعليمة try
- التعليمة catch
- التعليمة throw

وفي الحالة العامة تكون عبارات البرنامج المراد مراقبتها من أجل الأخطاء محتواة في كتلة try ، وعند حدوث استثناء ما داخل الكتلة try يرمى هذا الاستثناء باستخدام التعليمة throw ، ويلتقط الاستثناء باستخدام العبارة catch ويتابع .
إن أي عبارة ترمي استثناء يجب أن تنفذ داخل الكتلة try ، وكل استثناء يجب أن يلتقط من قبل عبارة catch والتي تلي مباشرة عبارة try التي رمت الاستثناء ، والشكل العام لعبارة catch و try موضح كما يلي :

```
try {  
    // try Block  
}  
catch ( type1 arg. )  
{  
    // catch Block  
}
```

•
•
•
catch (typeN arg.)

```
{  
    // catch Block  
}
```

ويجب أن تحتوي كتلة **try** على جزء البرنامج المراد مراقبته من الأخطاء ، ويمكن أن يكون هذا الجزء صغيراً يراقب عدة عبارات داخل تابع واحد ، أو يمكن أن يكون عاماً بحيث يحاط التابع **main** بكتلة **try** وهذا يؤدي إلى مراقبة البرنامج بأكمله .

وعندما يرمى استثناء ما فإنه يلتقط بواسطة عبارة **catch** المناسبة والتي تقوم بمعالجة الاستثناء ، ويمكن أن يرافق عبارة **try** عدة عبارات **catch** . ويحدد نوع الاستثناء عبارة **catch** المستخدمة ، أي إذا كان نوع المعطيات المحدد بواسطة **catch** يوافق نوع الاستثناء فإن عبارة **catch** تلك تنفذ وتهمل باقي العبارات . وعند النقاط استثناء يستلم الوسيط **arg** قيمته ، وإذا لم نكن نريد الوصول إلى الاستثناء نفسه نحدد فقط النوع **type** في عبارة **catch** ويبقى **arg** اختيارياً ، ويمكن التقاط أي نوع من المعطيات بما في ذلك الصفوف التي ننشئها والشكل العام لعبارة **throw** كما يلي :

throw exception ;

ويجب أن تنفذ عبارة **throw** إما داخل الكتلة **try** المناسبة أو من أي تابع يستدعيها البرنامج داخل الكتلة بشكل مباشر أو غير مباشر ، أما **exception**

فتمثل قيمة الاستثناء المرمي . إن رمي الاستثناء لا يوافق أي عبارة catch قد يسبب توقف غير عادي للبرنامج . يبين المثال (8-6) الطريقة التي تعمل بها معالجة الاستثناءات في C++ .

مثال (8-6)

```
// A simple exception handling example.
#include <iostream.h>
int main( ){
    cout << "start\n";
    try {                // start a try block
        cout << "Inside try block\n";
        throw 10;        // throw an error
        cout << "This will not execute";
    }

    catch ( int i ) {    // catch an error
        cout << "Caught One! Number is: ";
        cout << i << "\n";
    }
    cout << "end\n";
    return 0;
} //End main
```

Result

start

Inside try block

Caught One! Number is: 10

end

Press any key to continue

نلاحظ وجود كتلة try تحوي ثلاث عبارات بالإضافة إلى عبارة catch (int i) التي تعالج استثناء صحيحاً ، وستنفذ عبارتان فقط من العبارات الثلاث داخل الكتلة try ، الأولى عبارة cout والثانية عبارة throw . وعندما يرمى استثناء ينتقل التحكم إلى التعبير catch وتنتهي الكتلة try ، أي أن لا تستدعي عبارة catch بل ينتقل تنفيذ البرنامج إليها وبالتالي لا تنفذ عبارة cout الثانية . بعد تنفيذ عبارة catch يتابع تنفيذ البرنامج العبارات التي تلي catch . إلا أن كتلة catch تنتهي غالباً باستدعاء التابع (exit) أو (abort) أو أي تابع آخر يسبب إنهاء البرنامج .

ويجب أن يوافق نوع الاستثناء النوع المحدد في عبارة catch ، إذا غيرنا النوع المحدد في عبارة catch في المثال (6-8) إلى double فإن الاستثناء لن يلتقط ويحدث إنهاء غير عادي للبرنامج ، ويوضح المثال (6-9) هذا التغيير .

// This example will not work

#include <iostream.h>

int main(){

cout << "start\n";

مثال (6-9)


```

try {           // start a try block
    cout << "Inside try block\n";
    throw 10;    // throw an error
    cout << "This will not execute";
}

catch ( double i ) {    // won't work
    cout << "Caught One! Number is: ";
    cout << i << "\n";
}
cout << "end\n";
return 0;
} //End main

```

Result

Abnormal program termination

Press any key to continue

يمكن رمي استثناء من عبارة خارج الكتلة try طالما كانت تلك العبارة داخل تابع يستدعي من الكتلة try ويوضح ذلك المثال (10-6) .

مثال (10-6)

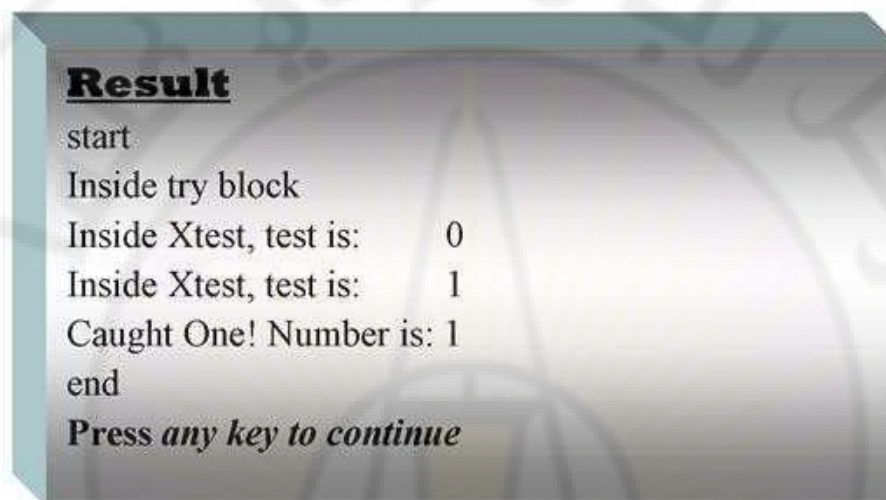
// Throwing an exception from a function outside the try block

```
#include <iostream.h>

void Xtest( int test)
{
    cout << "Inside Xtest, test is: " << test << "\n";
    if( test ) throw test;
}

int main() {
    cout << "start\n";
    try {                // start a try block
        cout << "Inside try block\n";
        Xtest(0);
        Xtest(1);
        Xtest(2);
    }
    catch ( int i ) {    // catch an error
        cout << "Caught One! Number is: ";
        cout << i << "\n";
    }
}
```

```
cout << "end\n";
return 0;
} //End main
```



يمكننا أن نجعل كتلة try محلية في تابع معين وفي هذه الحالة يعاد تهيئة معالجة الاستثناء الخاصة بهذا التابع في كل مرة يستدعى فيها ذلك التابع ، وبوضح ذلك المثال (11-6) .

مثال (11-6)

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

```
// A try/catch can be inside a function other than main( ).
```

```
void Xhandler(int test)
```

```
{
```

```
try{
```

```
        if( test ) throw test;
    }
    catch( int i ) {
        cout << "Caught One! Ex. #: " << i << "\n";
    }
}

int main() {
    cout << "start\n";
    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);
    cout << "end";
    return 0;
} //End main
```

Result

```
start
Caught One! Ex. #: 1
Caught One! Ex. #: 2
Caught One! Ex. #: 3
end
Press any key to continue
```

ونلاحظ من نتائج البرنامج عندما تم رمي ثلاث استثناءات فإن التابع يعود بعد كل استثناء ، وعند استدعاء التابع ثانية تعاد تهيئة معالجة الاستثناءات . ويمكن إرفاق أكثر من catch بكلمة try ، لكن يجب في هذه الحالة أن تلتقط كل catch نوعاً مختلفاً من الاستثناءات ، ويوضح المثال (6-12) استثناءات صحيحة وأخرى محرفية .

مثال (6-12)

```
#include <iostream.h>

// Different types of exceptions can be caught.

void Xhandler(int test)
{
    try{
        if ( test ) throw test;
        else throw "Value is zero";
    }
    catch( int i ) {
        cout << "Caught One! Ex. #: " << i << "\n";
    }
    catch( char *str) {
        cout << "Caught a string: ";
        cout << str << "\n";
    }
}
```

```

int main() {
    cout << "start\n";
    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);
    cout << "end\n";
    return 0;
} //End main

```

Result

```

start
Caught One! Ex. #: 1
Caught One! Ex. #: 2
Caught a string: Value is zero
Caught One! Ex. #: 3
end
Press any key to continue

```

ونلاحظ أن كل عبارة catch تستجيب إلى النوع الخاص بها ، وبشكل عام يتم التحقق من تعابير catch حسب ترتيب توضعها في البرنامج ، ولا تنفذ إلا عبارة catch الموافقة وتهمل جميع العبارات المتبقية .

٦-٤-١ - معالج الاستثناءات

Exception Handler

يوجد العديد من الميزات الإضافية لمعالجة الاستثناءات في لغة C++ ما يجعلها أكثر سهولة وملائمة للاستخدام ، قد نحتاج في بعض الحالات إلى وجود معالج استثناءات واحد يلتقط جميع الاستثناءات بدلاً من النقاط نوع معين ، ويتم ذلك بسهولة وذلك باستخدام الشكل العام التالي لمعالج الاستثناءات :

```
catch( ..... ) {  
    // process all exceptions  
}
```

ويمكن تحديد نوع الاستثناءات الذي يمكن أن يرميها التابع إلى البرنامج الجزئي subroutine المستدعى ، أي يمكن التحكم بنوع الاستثناءات الذي يستطيع تابع ما رميها إلى الخارج ، كما يمكن منع التابع من رمي أي استثناء. لتطبيق هذه القيود يجب إضافة عبارة throw إلى تعريف التابع ويصبح الشكل العام لمعالج الاستثناءات كما يلي :

```
Ret-type func-name ( arg-list ) throw ( type - list )  
{  
    // Process  
}
```

وفي هذه الحالة يمكن رمي الأنواع المصرح عنها في القائمة type - list فقط خارج التابع ، وبالتالي يؤدي رمي أي نوع آخر إلى توقف غير طبيعي للبرنامج ، أما إذا كان غير مطلوب من التابع رمي أي استثناء عندها يجب استخدام قائمة فارغة . إذا كان المطلوب رمي استثناء من داخل معالج استثناء ما يمكن إنجاز ذلك باستدعاء عبارة throw وحدها دون ذكر أي استثناء ، ما يؤدي إلى تمرير

الاستثناء الحالي إلى سلسلة try/catch خارجية . ويوضح المثال (6-13)
عمل العبارة catch(....) .

مثال (6-13)

```
// This example catches all exceptions.
#include <iostream.h>
void Xhandler( int test)
{
    try{
        if(test == 0) throw test;           // throw int
        if(test == 1) throw 'b';           // throw char
        if(test == 2) throw 125.25;        // throw double
    }
    catch(...) {                          // catch all exceptions
        cout << "Caught One!\n";
    }
} //end Xhandler

int main() {
    cout << "start\n";
    Xhandler(0);
    Xhandler(1);
```

```
Xhandler(2);  
cout << "end\n";  
return 0;  
} //End main
```

Result

```
start  
Caught One!  
Caught One!  
Caught One!  
end  
Press any key to continue
```

ونلاحظ أن عبارات throw الثلاثة قد التقطت بواسطة عبارة catch واحدة .
وأحد الاستخدامات المهمة والمفيدة لعبارة catch(...) استخدامها كآخر عبارة
في مجموعة عبارات catch . في هذه الحالة تؤمن هذه العبارة معالجة افتراضية
مفيدة تتمثل في التقاط جميع الاستثناءات غير المعالجة و نعدل البرنامج في
المثال (6-13) بحيث تلتقط الاستثناءات الصحيحة صراحة ولكنها تعتمد على
عبارة catch(...) لالتقاط كافة الاستثناءات الأخرى كما هو مبين في
المثال (14) .

مثال (14-6)

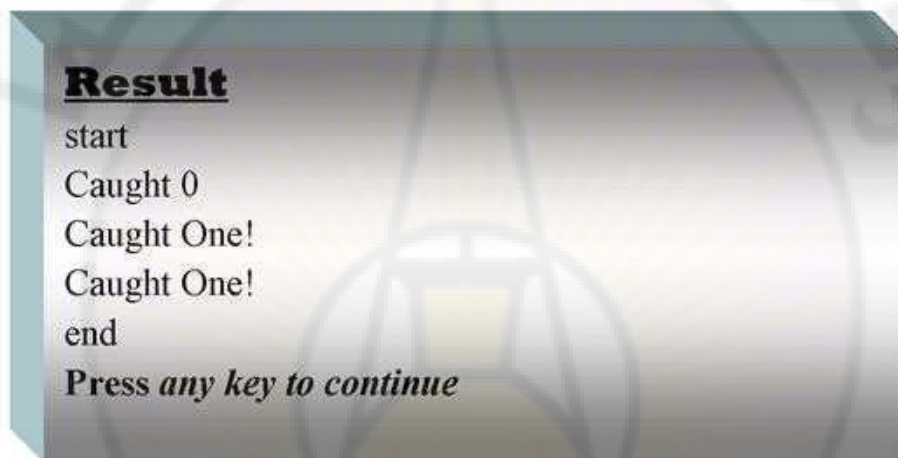
```
// This example uses catch(...) as a default.  
#include <iostream.h>  
void Xhandler(int test)  
{  
    try{  
        if(test == 0) throw test;           // throw int  
        if(test == 1) throw 'b';           // throw char  
        if(test == 2) throw 125.25; // throw double  
    }  
    catch( int i ) {                       // catch an int exception  
        cout << "Caught " << i << "\n";  
    }  
    catch(...) {                            // catch all other exceptions  
        cout << "Caught One!\n";  
    }  
}  
}  
  
int main( ) {  
    cout << "start\n";  
    Xhandler(0);  
    Xhandler(1);
```

```

Xhandler(2);
cout << "end\n";
return 0;

} //End main

```



يبين البرنامج في المثال (6-14) أن استخدام catch(...) كمعالج افتراضي يعد طريقة جيدة للنقاط جميع الاستثناءات التي لا نرغب بمعالجتها صراحة ، كما أن النقاط جميع الاستثناءات يمنع توقف البرنامج عند حدوث استثناء غير معالج، ويوضح البرنامج في المثال (6-15) كيف يتم حصر الاستثناءات التي يمكن أن يرميها تابع ما .

مثال (6-15)

```

// Restricting function throw types.
#include <iostream.h>

```

```

    // This function can only throw ints, chars, and doubles.
void Xhandler(int test) throw (int , char, double) {
    if(test == 0) throw test;    // throw int
    if(test == 1) throw 'c';    // throw char
    if(test == 2) throw 125.25; // throw double
} //end Xhandler
int main() {
    cout << "start\n";
    try{
        Xhandler(0);    // also, try passing 1 and 2 to Xhandler()
    }
    catch( int i ) {
        cout << "Caught int\n";
    }
    catch( char c ) {
        cout << "Caught char\n";
    }
    catch( double d ) {
        cout << "Caught double\n";
    }
    cout << "end\n";
    return 0;
} //End main

```


Result

start

Caught int

end

Press any key to continue

ويستطيع التابع (Xhandler) أن يرمي استثناءات من الأنواع int و char و double ، إذا حاولنا رمي أي نوع آخر فإن البرنامج سيتوقف بشكل غير عادي. وبالتالي يمكن حصر أنواع الاستثناءات التي يمكن أن نرميها من التابع إلى الكتلة try التي يستدعيها ، بينما تستطيع كتلة try داخل تابع ما رمي أي نوع من الاستثناءات طالما أنها تلتقط داخل ذلك التابع ، وتطبق القيود عند رمي استثناء إلى خارج التابع .

ونستطيع إعادة رمي أي استثناء (كما وجدنا سابقاً) والسماح لعدة معالجات بالوصول إلى الاستثناء ، ويمكن لأحد معالجات الاستثناء أن يعالج أحد جوانب استثناء ما في حين يهتم معالج آخر بجانب آخر منه . ويمكن إعادة رمي الاستثناء فقط من كتلة catch، ولن يلتقط الاستثناء من قبل العبارة catch نفسها وإنما سينتشر إلى عبارة catch خارجية ، ويوضح البرنامج في المثال (6-16) إعادة رمي استثناء من النوع char* .

مثال (6-16)

// Example of "rethrowing" an exception.

#include <iostream.h>

```

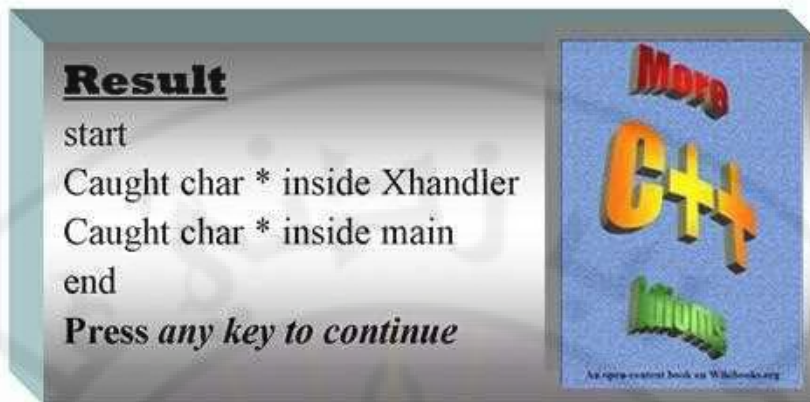
void Xhandler()
{
    try {
        throw "welcome";    // throw a char *
    }
    catch( char *) {        // catch a char *
        cout << "Caught char * inside Xhandler\n";
        throw ;            // rethrow char * out of function
    }
} //end Xhandler

```

```

int main() {
    cout << "start\n";
    try{
        Xhandler( );
    }
    catch( char *) {
        cout << "Caught char * inside main\n";
    }
    cout << "end\n";
    return 0;
} //End main

```



تمرين (6-2)

□ ما الخطأ في البرنامج الجزئي التالي ؟

```
try{
    //.....
    throw 20 ;
}
catch ( int *p ) {
    //.....
}
```

□ بين طريقة لتصحيح البرنامج الجزئي السابق .

□ ما تعبير **catch** الذي يلتقط كل أنواع الاستثناءات ؟

=====

=====

٦-٤-٢- معالجة الاستثناءات التي يرميها new

عند بداية ظهور C++ كانت new تعيد null عند الفشل ، إلا أن هذا الفعل قد تغير لاحقاً بحيث يسبب فشل new رمي استثناء ، كما أن اسم الاستثناء قد تغير عدة مرات ، وأخيراً فقد تقرر أن فشل new سوف يولد استثناء بشكل افتراضي ، ويمكن إعادة مؤشر null بشكل اختياري و وبالتالي تقدم المترجمات المختلفة العامل new بطرق مختلفة . وعندما لا تستطيع new حجز موقع في الذاكرة تقوم لغة C++ القياسية برمي الاستثناء bad-alloc وإذا لم تلتقط هذا الاستثناء سيتوقف البرنامج ، وللوصول إلى هذا الاستثناء يجب أن يتضمن الملف الرأسي `#include <new>` في البرنامج .

ويمكن في لغة C++ القياسية جعل new تعيد المؤشر null بدلاً من رمي استثناء عند الفشل في حجز موقع في الذاكرة . هذا الشكل مفيد عند ترجمة برنامج قديم بلغة C++ باستخدام مترجم حديث ، ويمكن الاستفادة منه عند استبدال التابع `malloc()` بالعامل new كما هو مبين فيما يلي :

`p-var = new (nothrow) type ;`

حيث `p-var` متحول مؤشر من النمط `type` ، ويعمل الشكل `nothrow` للعامل new مثل نسخة new الأصلية . وبما أن new تعيد null عند الفشل يمكن إسقاطها من البرامج القديمة دون الحاجة إلى إضافة معالجة استثناءات ، إلا أن الاستثناءات تقدم بديلاً أفضل من أجل البرامج الحديثة . ويبين المثال (6-17) كيف تستخدم العبارة new كتلة try/catch لمراقبة حالة الفشل في حجز ذاكرة .

مثال (6 - 17)

```
#include <iostream>
```

```
#include <new>
```

```

using namespace std;

int main( ) {
    int *p;
    try {
        p = new int;           //allocate memory for int
    }
    catch (bad_alloc xa) {
        cout << "Allocation failure.\n";

        return 1;
    }
    for( *p=0;*p<10;*p++)
        cout << *p<<" ";
    delete p;                 //free the memory
    return 0;
} //End main

```

وعند حصول أي فشل في حجز الذاكرة سيلتقط بواسطة العبارة `catch` .

بما أن البرنامج في المثال (6-17) لن يفشل في الظروف العادية ، فإن البرنامج في المثال (6-18) يوضح قدرة `new` على رمي الاستثناءات عن طريق إحداث فشل قسري في حجز الذاكرة ، ويتم ذلك عن طريق حجز الذاكرة حتى استنفادها .

مثال (18-6)

```
// Force an allocation failure.
#include <iostream >
#include <new>
using namespace std;
int main( ) {
    double *p;
    // this will eventually run out of memory
    do {
        try {
            p = new double[100000];
        }
        catch (bad_alloc xa) {
            cout << "Allocation failure.\n";
            return 1;
        }
        cout << "Allocation OK.\n";
    }
    while(p);
    return 0;
}//End main
```


ويبين البرنامج في المثال (6-19) كيفية استخدام `new( nothrow )` بشكل اختياري، ويعمل هذا البرنامج مثل البرنامج في المثال (18-6) ويؤدي إلى فشل عملية حجز الذاكرة .

مثال (6-19)

```
// Demonstrate the new(nothrow) alternative.
#include <iostream>
#include <new>
using namespace std;
int main( ) {
    double *p;
    // this will eventually run out of memory
    do {
        p = new(nothrow) double[100000];
        if(p) cout << "Allocation OK.\n";
        else cout << "Allocation Error.\n";
    }
    while(p);
    return 0;
}
```



مسائل عامة محلولة

١- اكتب برنامجاً بلغة C++ لتعريف قالب صف يدعى رتل `queue` ، أنشئ في التابع (`main`) غرضين ، الغرض الأول من نمط `int` (رتل من نمط `int`) ، أما الغرض الثاني من نمط `char` (رتل من نمط `char`) .

البرنامج

```
// Create a generic queue.
#include <iostream.h>
#define SIZE 100
template < class Qtype >
class q_type {
    Qtype queue[SIZE];    // holds the queue
    int head, tail;        // indices of head and tail
public:
    q_type( ) { head = tail = 0; }
    void q( Qtype num);    // store
    Qtype deq( );          // retrieve
}; //End class
```

// Put value on queue.

```
template < class Qtype >
void q_type< Qtype > :: q( Qtype num)
{
    if(tail+1==head || (tail+1==SIZE && ! head)) {
        cout << "Queue is full.\n";
        return;
    }
    tail++;
    if(tail==SIZE) tail = 0;    // cycle around
    queue[tail] = num;
}
```

// Remove value from queue.

```
template < class Qtype >
Qtype q_type < Qtype >::deq()
{
    if(head == tail) {
        cout << "Queue is empty.\n";
        return 0;    // or some other error indicator
    }
}
```

```

    head++;
    if(head == SIZE) head = 0; // cycle around
    return queue[head];
}

```

```

int main() {
    q_type<int> q1;
    q_type<char> q2;
    int i;
    for( i=1; i<=10; i++) {
        q1.q(i);
        q2.q(i-1+'A');
    }

    for(i=1; i<=10; i++) {
        cout << "Dequeue 1: " << q1.deq() << "\n";
        cout << "Dequeue 2: " << q2.deq() << "\n";
    }
    return 0;
} //End main

```

Result

Dequeue 1: 1
Dequeue 2: A
Dequeue 1: 2
Dequeue 2: B
Dequeue 1: 3
Dequeue 2: C
Dequeue 1: 4
Dequeue 2: D
Dequeue 1: 5
Dequeue 2: E
Dequeue 1: 6
Dequeue 2: F
Dequeue 1: 7
Dequeue 2: G
Dequeue 1: 8
Dequeue 2: H
Dequeue 1: 9
Dequeue 2: I
Dequeue 1: 10
Dequeue 2: J

Press any key to continue

٢- اكتب برنامجاً بلغة C++ لتعريف قالب تابع يدعى mode بحيث يعيد القيمة الأكثر تكراراً في المصفوفة a. أنشئ في التابع الرئيسي (main) غرضين ، الغرض الأول مصفوفة من نمط int (int a[]) ، أما الغرض الثاني فمصفوفة من نمط char (char *p) .

```

#include <iostream.h>
#include <string.h>
// A generic mode finding function.
template < class X >
X mode(X *data, int size)
{
    register int t, w;
    X md, oldmd;
    int count, oldcount;
    oldmd = 0;
    oldcount = 0;
    for(t=0; t<size; t++) {
        md = data[t];
        count = 1;
        for(w = t+1; w < size; w++)
            if(md==data[w]) count++;
        if(count > oldcount) {
            oldmd = md;
            oldcount = count;
        }
    }
    return oldmd;
}
//end mode

```



```

int main() {
    int a[ ] = { 10, 20, 30, 40, 20, 30, 20, 20, 10, 50};
    char *p = "Wellcome mmmm To Programming";

    cout << "mode of a: " << mode(a, 10) << endl;
    cout << "mode of p: " << mode(p, (int) strlen(p)) << endl;

    return 0;
} //End main

```

Result

mode of a: 20

mode of p: m

Press any key to continue

٣- اكتب برنامجاً بلغة C++ لتعريف قالب تابع يدعى sum بحيث يعيد مجموع قيم عناصر المصفوفة data. أنشئ في التابع الرئيسي (main) غرضين ، الغرض الأول مصفوفة a من نمط int ، أما الغرض الثاني فمصفوفة b من نمط double .

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

```
template < class X >
```

```
X sum(X *data, int size)
```

```
{  
    int i;  
    X result = 0;  
    for(i=0; i<size; i++)  
        result += data[i];  
    return result;  
}
```

```
int main() {  
    int a[ ] = {15, 20, 13, 54,25};  
    double b[ ] = {2.5, 30.5, 20.5, 14.5};  
    cout << sum(a, 5) << endl;  
    cout << sum(b, 4) << endl;  
    return 0;  
}  
//End main
```

Result

127

68

Press any key to continue

٤- اكتب برنامجاً بلغة C++ لتعريف قالب تابع يدعى bubble تمثل خوارزمية الترتيب الفقاعي bubble sort بحيث يتم ترتيب المصفوفة data بشكل تصاعدي. أنشئ في التابع الرئيسي (main) ثلاثة أغراض ، الغرض الأول مصفوفة a من نمط int ، أما الغرض الثاني مصفوفة b من نمط double ، والغرض الثالث قمصوفة c من نمط char.

```
#include <iostream.h>

// A generic bubble sort.
template < class X >
void bubble(X *data, int size)
{
    register int a, b;
    X t;
    for(a=1; a < size; a++)
        for(b=size-1; b >= a; b--)
            if(data[b-1] > data[b]) {
                t = data[b-1];
                data[b-1] = data[b];
                data[b] = t;
            }
}
//end bubble
```

```

int main() {
    int a[] = {30, 25, 50, 66, 11, 18, 59, 33, 76, 19};
    double b[] = {11.25, 25.5, 12.2, 53.3, 50.0};
    char c[] = {'k', 'g', 'a', 'q', 'f'};

    int j;
    bubble(a, 10); // sort ints
    bubble(b, 5);  // sort doubles
    bubble(c, 5);  // sort chars

    for(j=0; j<10; j++) cout << a[j] << ' ';
    cout << endl;
    for(j=0; j<5; j++) cout << b[j] << ' ';
    cout << endl;
    for(j=0; j<5; j++) cout << c[j] << ' ';
    cout << endl;
    return 0;
} //End main

```

Result

11 18 19 25 30 33 50 59 66 76

11.25 12.2 25.5 50 53.3

a f g k q

Press any key to continue

٥ - اكتب برنامجاً بلغة C++ لتعريف قالب صف يدعى stack يستطيع تخزين أزواج من الأغراض بأنماط مختلفة ، [وهو إعادة للمثال (5)] ولكن يتعامل مع ثنائيات من أنواع مختلفة من المعطيات [، ويحتوي على تابع init لإعطاء قيمة ابتدائية لمؤشر المكس ، وتابع تعديل معطيات push لدفع المعطيات إلى المكس ، وتابع pop من أجل طباعة معطيات المكس . أنشئ في التابع الرئيسي main() غرضين ، الغرض الأول يمثل مكس نمط char ، أما الغرض الثاني يمثل مكس من نمط double ، واستدعاء ما يلزم من توابع الأعضاء لترميز المطيات إلى المكسين وطباعة النتائج .

```

/* This function demonstrates a generic stack that
   holds two values. */
#include <iostream.h>
#define SIZE 10
// Create a generic stack class
template < class StackType > class stack {
    StackType stck[SIZE][2];           // holds the stack
    int tos;                           // index of top-of-stack
public:
    void init( ) { tos = 0; }
    void push(StackType ob, StackType ob2);
    StackType pop( StackType &ob2);
}; //end class

```

// Push objects.

```
template < class StackType >
void stack<StackType> :: push(StackType ob, StackType ob2)
{
    if(tos == SIZE) {

        cout << "Stack is full.\n";
        return ;
    }
    stck[tos][0] = ob;
    stck[tos][1] = ob2;
    tos++;
} //end push
```

// Pop objects.

```
template < class StackType >
StackType stack<StackType>::pop(StackType &ob2)
{
    if(tos == 0) {
        cout << "Stack is empty.\n";
        return 0; // return null on empty stack
    }
}
```



```
tos--;  
ob2 = stck[tos][1];  
return stck[tos][0];  
}//end pop
```

```
int main( ) {  
    // Demonstrate character stacks.  
    stack< char > s1, s2;           // create two stacks  
    int i;  
    char ch;  
        // initialize the stacks  
    s1.init( );  
    s2.init( );  
  
    s1.push('M', 'u');  
    s1.push('A', 'o');  
    s1.push('M', 'Y');  
  
    s2.push('N', 's');  
    s2.push('U', 'e');  
    s2.push('O', 'n');
```

```

for(i=0; i<3; i++) {
    cout << "Pop s1: " << s1.pop(ch);
    cout << " " << ch << "\n";
}
for(i=0; i<3; i++) {
    cout << "Pop s2: " << s2.pop(ch);
    cout << " " << ch << "\n";
}
cout << "-----\n";

    // demonstrate double stacks
stack<double> ds1, ds2;           // create two stacks
double d;
    // initialize the stacks
ds1.init( );
ds2.init( );
ds1.push(60.6, 50.5);
ds2.push(16.0, 15.0);
ds1.push(40.4, 30.3);
ds2.push(14.0, 13.0);
ds1.push(20.2, 10.1);
ds2.push(12.0, 11.0);
for(i=0; i<3; i++) {
    cout << "Pop ds1: " << ds1.pop(d);

```

```

    cout << " " << d << "\n";
}
cout << "-----\n";
for(i=0; i<3; i++) {
    cout << "Pop ds2: " << ds2.pop(d);
    cout << " " << d << "\n";
}
cout << "-----\n";
return 0;
} //End main

```

Result

Pop s1: M Y
 Pop s1: A o
 Pop s1: M u
 Pop s2: O n
 Pop s2: U e
 Pop s2: N s

 Pop ds1: 20.2 10.1
 Pop ds1: 40.4 30.3
 Pop ds1: 60.6 50.5

 Pop ds2: 12 11
 Pop ds2: 14 13
 Pop ds2: 16 15

Press any key to continue

٦- اكتب برنامجاً بلغة C++ لتعريف قالب صف يدعى stack يستطيع تخزين أغراض بأنماط مختلفة ، [وهو إعادة للمثال (6-5)] ، بحيث يعالج حالة امتلاء وفراغ المكسد كاستثناءات ، ويحتوي على تابع init لإعطاء قيمة ابتدائية لمؤشر المكسد ، وتابع تعديل معطيات push لدفع المعطيات إلى المكسد ، وتابع pop من أجل طباعة معطيات المكسد . أنشئ في التابع الرئيسي (main) غرضين ، الغرض الأول يمثل مكسد نمط char ، أما الغرض الثاني يمثل مكسد من نمط double ، واستدعاء ما يلزم من توابع الأعضاء لترميز المطيات إلى المكسدين وطباعة النتائج .

```
/* This function demonstrates a generic stack that includes  
exception handling. */  
#include <iostream.h>  
#define SIZE 10  
// Create a generic stack class  
template < class StackType > class stack {  
    StackType stek[SIZE]; // holds the stack  
    int tos; // index of top-of-stack  
public:  
    void init( ) { tos = 0; } // initialize stack  
    void push(StackType ch); // push object on stack  
    StackType pop( ); // pop object from stack  
};//end class
```

// Push an object.

```
template < class StackType >
void stack< StackType >::push(StackType ob)
{
    try {
        if(tos == SIZE) throw SIZE;
    }
    catch(int) {
        cout << "Stack is full.\n";
        return;
    }
    stck[tos] = ob;
    tos++;
}
```

// Pop an object.

```
template <class StackType>
StackType stack< StackType > :: pop()
{
    try {
        if(tos == 0) throw 0;
    }
    catch(int) {
```

```

    cout << "Stack is empty.\n";
    return 0;           // return null on empty stack
}
tos--;
return stck[tos];
}

int main( ) {
    // Demonstrate character stacks.
    stack< char > s1, s2;           // create two stacks
    int i ;
    // initialize the stacks
    s1.init( );
    s2.init( );
    s1.push('m');
    s2.push('n');
    s1.push('a');
    s2.push('u');
    s1.push('M');
    s2.push('o');
    for(i=0; i<3; i++) cout << "Pop s1: " << s1.pop( ) << "\n";
    for(i=0; i<4; i++) cout << "Pop s2: " << s2.pop( ) << "\n";
    cout<< "-----\n";
}

```



```

        // demonstrate double stacks
stack< double > ds1, ds2;      // create two stacks
        // initialize the stacks
ds1.init( );
ds2.init( );

ds1.push(60.6);
ds2.push(50.5);
ds1.push(40.4);
ds2.push(30.3);
ds1.push(20.2);
ds2.push(10.1);

for(i=0; i<3; i++) cout << "Pop ds1: "
                        << ds1.pop( )<<endl;
cout<< "-----\n";
for(i=0; i<4; i++) cout << "Pop ds2: "
                        << ds2.pop( )<<endl;
cout<< "-----\n";

return 0;
} //End main

```

Result

Pop s1: M

Pop s1: a

Pop s1: m

Pop s2: o

Pop s2: u

Pop s2: n

Stack is empty.

Pop s2:0

Pop ds1: 20.2

Pop ds1: 40.4

Pop ds1: 60.6

Pop ds2: 10.1

Pop ds2: 30.3

Pop ds2: 50.5

Stack is empty.

Pop ds2: 0

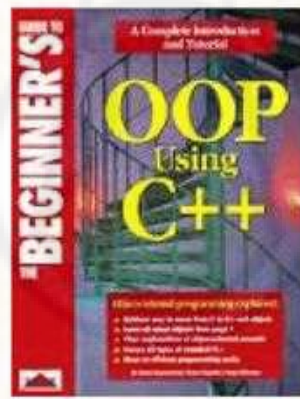
Press any key to continue



الفصل السابع

معالجة الملفات

Files Processing





معالجة الملفات

Files Processing

٧-١ - مقدمة

يتناول هذا الفصل العديد من التوابع التي تتعامل مع الملفات وكيفية تخزين المعطيات (كتابة المعطيات) في الملفات واسترجاعها (قراءة المعطيات) من خلال توابع وصفوف لغة C++ . و للتعامل مع صفوف الملفات لا بد أن يشتمل البرنامج على الملف `fstream.h` وذلك باستعمال الأمر `#include` ، حيث يحتوي هذا الملف على تعريف صفوف الملفات مثل `ifstream` و `ofstream` والتي تحتوي على توابع التعامل مع الملفات.

٧-٢ - تعريف الملف

File Definition

الملف هو مجموعة من المعلومات ، وعادة يمكن أن تخزن في أقراص تخزين الحاسوب المتنوعة، وحيث يتم تخزين المعلومات في الملفات وبعد ذلك يتم استرجاعها.

٧-٣ - أسماء الملفات

File Names

جميع الملفات يتم إعطاؤها أسماء معينة من أجل التعامل معها من قبل المستخدم ونظام التشغيل. ويوضح لنا الجدول (٩-١) التالي ، أسماء بعض الملفات وامتداداتها (أسماءها الموسعة) بالإضافة إلى محتوى كل ملف.

File Name and Extension اسم الملف وامتداده	File Contents محتويات الملف
MYPROG.BAS	BASIC program
MENU.BAT	DOS Batch File
INSTALL.DOC	Documentation File
CRUNCH.EXE	Executable File
BOB.HTML	HTML (Hypertext Markup Language) File
3DMODEL.JAVA	Java program or applet
INVENT.OBJ	Object File
PROG1.PRJ	Borland C++ Project File
ANSI.SYS	System Device Driver
README.TXT	Text File

الجدول (٧-١) - يوضح الملفات وامتداداتها

٧-٤ - عملية استخدام الملفات

يتم استخدام الملف ببساطة في البرنامج من خلال العمليات الثلاثة التالية:

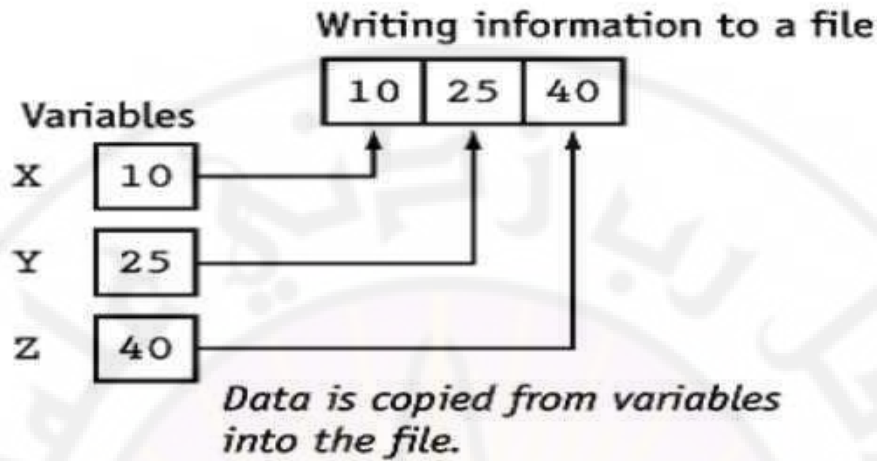
١- الملف يجب أن يكون مفتوحاً. إذا كان الملف غير موجوداً، فإن

عملية فتحه تعني خلق الملف .

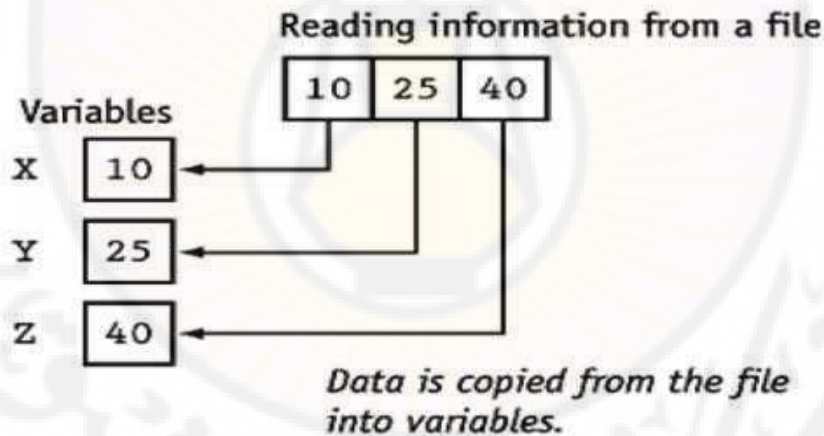
٢- يتم تخزين المعلومات في الملف ، و تتم القراءة منه.

٣- عندما ينتهي البرنامج من استخدام الملف، فإن الملف يجب أن يُغلق.

ومن خلال الشكلين (٧-١) و (٧-٢) يتم توضيح عملية الكتابة في الملف وأيضاً عملية القراءة منه.



الشكل (٧-١) - يوضح كتابة المعلومات في الملف



الشكل (٧-٢) - يوضح قراءة المعلومات من الملف

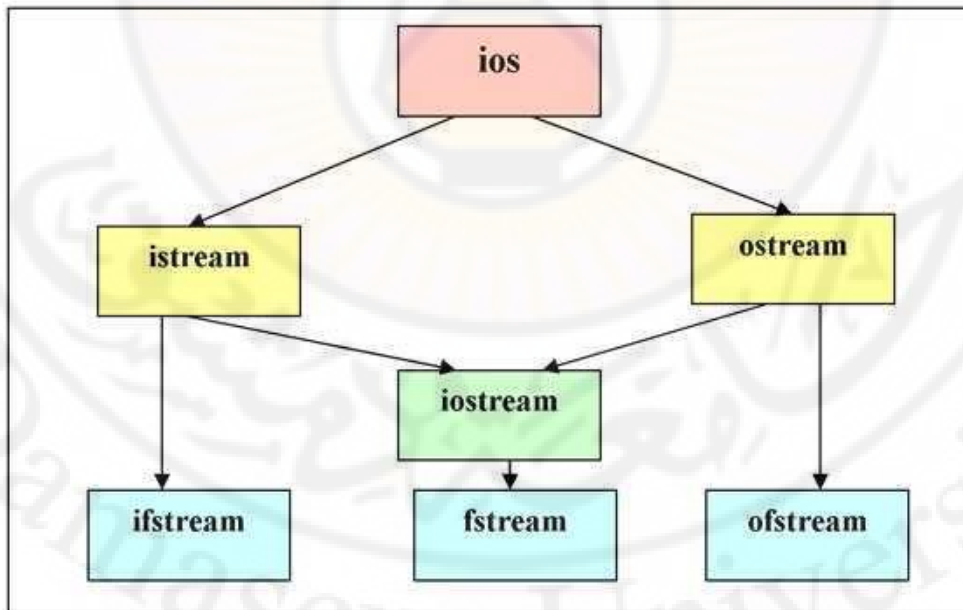
ملاحظة (١)

لا بد من التنكير، أنه قبل تنفيذ إدخال الملفات وإخراجها ، يجب وضع البرنامج اللازم بشكل دقيق ، والوصول إلى الملف يتم من خلال تضمين الملف fstream.h في البرنامج المعني.

٧-٥- مكتبة الإدخال والإخراج في لغة C++

تعتبر عمليات الإدخال والإخراج من أكثر عمليات لغة C++ مرونة، ولا يمكن اعتبارها من العمليات المعرفة في لغة C++ بل إنها محققة باستخدام تقنيات هذه اللغة، فهي في الواقع ليست سوى توابع أعضاء في صفوف تحاكي مجاري الدخل/الخرج التي يمكن أن تكون وحدات الإدخال والإخراج القياسية أو ملفات فيزيائية على وسائط التخزين الثانوية.

تتألف مكتبة الإدخال والإخراج في لغة C++ من عدة ملفات رأسية تتضمن تعاريف صفوف الدخل/الخرج وعملياتها، ويكتفى هنا بإعطاء مخطط جزئي من مخطط الوراثة الذي يربط بعض تلك الصفوف ببعضها الآخر.



الشكل (٧-٣) - يوضح مخطط جزئي من مخطط وراثة صفوف الدخل/الخرج

وتدعم هذه الصفوف عمليات الإدخال والإخراج الأساسية وعمليات الإدخال والإخراج على الملفات، وبعض عمليات الإدخال والإخراج الخاصة بالسلاسل المحرفية، كما تتضمن ملفات المكتبة أربعة أغراض ترتبط بأجهزة الإدخال والإخراج القياسية وهي:

cin - هو غرض من الصف istream ويرتبط بالدخل القياسي ، وهو يقابل stdin في لغة C .

cout - هو غرض من الصف ostream ويرتبط بالخروج القياسي ، وهو يقابل stdout في لغة C .

شكل الفتح (فتح الملف) الافتراضي Default Open Mode	نوع الملف File Type
الملف يكون مفتوحاً من أجل الإخراج فقط (هذا يعني أن المعلومات يمكن كتابتها في الملف ولكن لا يمكن القراءة منه) إذا لم يكن الملف موجوداً ، فيتم إنشائه. وإذا كان الملف موجوداً بالأساس ، فإن محتوياته يتم مسحها .	ofstream
الملف يكون مفتوحاً من أجل الإدخال فقط (هذا يعني أن المعلومات يمكن قراءتها فقط من الملف ولكن لا يمكن الكتابة فيه). في هذه الحالة محتوى الملف تتم قراءتها من البداية. إذا كان الملف غير موجوداً ، فإن التابع open يخفق في عمله.	ifstream

الجدول (٧-٢) - يوضح الصنفين ifstream و ofstream

cerr - هو غرض من الصف ostream ويكافئ stderr في لغة C .
 clog - هو غرض من الصف ostream ، ولا يوجد مكافئ له في لغة C .
 والجدول (٧-٢) يوضح لنا بعض المعلومات التي تخص الصفين ifstream و ofstream :
 كنا قد تطرقنا في الشكل (٧-٣) إلى الصف ios ، هذا الصف يعتبر الصف الأساس الذي تُشتق منه الصفوف الأخرى، ويتضمن غرض هذا الصف مجموعة من العلامات التي تميز حالة مجرى الدخل أو الخرج، وكل هذه العلامات مخزنة في حقل واحد من النمط int ، ويمكن قراءة قيم هذه العلامات بواسطة التوابع الأعضاء ، والجدول (٧-٣) يبين لنا القيم المُعادة من خلال استخدام مثل هذه التوابع الأعضاء.

القيمة المُعادة	اسم التابع العضو
يعيد هذا التابع القيمة (1) عند بلوغ نهاية الملف وإلا فهو يعيد القيمة (0)	int eof()
يعيد هذا التابع القيمة (1) عند محاولة تنفيذ عملية غير صحيحة ، كمحاولة تجاوز نهاية الملف.	int bad()
يعيد هذا التابع القيمة (1) إذا لم ينجح في تنفيذ إحدى عمليات الإدخال أو الإخراج أو إذا كانت قيمة التابع bad() تساوي (1).	int fail()
يعيد هذا التابع القيمة (1) إذا كانت قيم التوابع السابقة كلها مساوية للصفر.	int good()

الجدول (٧-٣) - يوضح القيم المُعادة لبعض التوابع الأعضاء

٧-٦- فتح ملف

Opening a File

عند فتح ملف يتم تحديد سبب الفتح، هل هو لإدخال معطيات في الملف أم لقراءة معطيات منه أم للقراءة والكتابة (input, output, input & output). ويتم ذلك بالتعامل مع الصف المختص بالعملية المطلوبة، فعلى سبيل المثال، يُستخدم الصف ifstream للإدخال، والصف ofstream يُستخدم للإخراج والصف fstream يُستخدم للإدخال والإخراج ويتضح ذلك من خلال التعليمات البرمجية التالية:

- 1: ifstream in;//input
- 2: ofstream out;//output
- 3: fstream io;//input/output

من خلال هذه التعليمات ، نلاحظ أنه في السطر الأول تم التصريح عن الغرض in من الصف ifstream وبالتالي يمكن استخدام هذا الغرض لفتح ملف للإدخال ، وبصورة مشابهة يمكن الحديث عن السطرين الثاني والثالث. وقبل أن تتم عملية إدخال المعطيات إلى الملف أو القراءة منه، يجب فتح الملف أولاً.

٧-٦-١- تابع فتح الملف open()

يُستخدم التابع open() لفتح الملف حسب الحالة المطلوبة ويتم تحديد ذلك في قائمة وسطاء التابع، ويأخذ التابع الشكل العام التالي:

void open(char *file name,int mode, int Access);

حيث:

file name: هو اسم الملف المطلوب فتحه، ويتم كتابة المسار الكامل للملف.
mode: يحدد هذا الوسيط الحالة المطلوب فتح الملف لها وهي حالة من عدة حالات موضحة في الجدول (٧-٤).

File Mode Flag الحالة	Meaning المعنى (الاستخدام)
ios::app	إضافة معلومات إلى محتويات الملف
ios::ate	المعطيات يمكن أن تكتب في أي مكان ضمن الملف
ios::binary	الكتابة في الملف/القراءة من الملف بالنظام الثنائي
ios::in	فتح الملف من أجل الإدخال (الكتابة)
ios::nocreate	عدم فتح الملف إذا لم يكن موجوداً من قبل
ios::noreplace	عدم فتح الملف إذا كان موجوداً
ios::out	فتح ملف للإخراج (القراءة)
ios::trunk	إذا كان الملف موجوداً ، فإن محتوياته سوف تُحى.

الجدول (٧-٤) – يبين حالات فتح الملف

Access: هذا الوسيط يحدد طبيعة الملف وهي حالة من مجموعة الحالات الموضحة في الجدول (٧-٥).

الحالة	المعنى
0	ملف عادي (Normal)
1	ملف للقراءة فقط (Read)
2	ملف مخفي (Hidden)
3	ملف نظام (System)

الجدول (٧-٥) - يوضح حالات استخدام المحدد Access

٧-٦-٢ - حالات استخدام التابع open()

فيما يلي سنقدم مجموعة من الحالات لاستخدام التابع open .
الحالة الأولى: فتح ملف للإخراج.

لهذه الغاية لنأخذ المثال التالي:

```
1:ofstream out1;  
2:out1.open("EX1",ios::out,0);
```

في هذا المثال تم استخدام التابع `open` لفتح ملف باسم `EX1` لحالة الإخراج (`out`) وطبيعة الملف (`0`). من خلال السطر الأول نلاحظ أنه تم التصريح عن الغرض `out1` من الصف `ofstream`، لذلك إذا تم استخدام التابع `open` مع هذا الغرض بوسيط واحد هو اسم الملف ، يتم فتح الملف للإخراج كما في الحالة التالية:

```
out1.open("EX1");
```

في هذه الحالة يتم فتح ملف باسم `EX1` للإخراج لأنها الحالة الافتراضية لغرض من الصف `ofstream` .

الحالة الثانية: فتح ملف للإدخال.

لهذه الغاية نأخذ المثال التالي:

```
1:ifstream in1;
```

```
2:in1.open("EX2",ios::in,0);
```

هذه الحالة هي نفس الحالة الأولى مع فارق بسيط هو استبدال الصف `ofstream` المستخدم للإخراج بالصف `ifstream` في السطر الأول، وفي السطر الثاني تم استبدال الحالة `ios::out` بالحالة `ios::in`.

الحالة الثالثة: فتح ملف للإدخال والإخراج.

لهذه الغاية نأخذ المثال التالي:

```
1:fstream io1;
```

```
2:io1.open("EX3",ios::in|ios::out);
```

في هذه الحالة تم التصريح عن الغرض `io1` من الصف `fstream` وذلك لاستخدامه في فتح ملف للإدخال والإخراج، وفي السطر الثاني تم فتح ملف باسم `EX3` للإدخال والإخراج وذلك باستخدام الحالة `ios::in|ios::out` .
ومن أجل اختبار فتح الملف ، فإن المثال التالي يختبر هل الملف مفتوح أم لا؟

```
1:if(!io1)
```

```
2:cout<<"Cannot Open File \n";
```

في السطر الأول يتم اختبار الغرض io1 بالتعليمة if والعملية ! ، فإذا كانت جملة الشرط صحيحة ، أي أن الغرض غير صحيح، فهذا يعني أنه لم يتم فتح الملف وبالتالي يتم طباعة رسالة خطأ في السطر الثاني.

مثال (1-7)

البرنامج التالي يوضح لنا ، كيفية التصريح عن غرض من الصف `fstream` ومن ثم فتح ملف.

```
// This program demonstrates the declaration of an ifstream
```

```
// object and the opening of a file.
```

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

```
#include <fstream.h>
```

```
int main( ) {
```

```
    ifstream dataFile; // Declare file stream object
```

```
    char fileName[81];
```

```
    cout << "Enter the name of a file you wish to open\n";
```

```
    cout << "or create: ";
```

```
    cin.getline(fileName, 81);
```

```
    dataFile.open(fileName, ios::out);
```

```
    cout << "The file " << fileName << " was opened.\n";
```

```
    return 0;
```

```
}//The end of main function
```

وعند تنفيذ البرنامج على الحاسوب ، سوف نحصل على الخرج التالي:

Result

Enter the name of a file you wish to open

or create: mystuff.dat

The file mystuff.dat was opened.

Press any key to continue

فتح الملف عند التصريح

يمكن فتح الملف عند التصريح عنه ، والشكل العام لهذا الاستخدام البرمجي هو كما يلي :

```
fstream dataFile("names.dat", ios::in | ios::out);
```

مثال (2-7)

البرنامج التالي يوضح لنا فتح الملف في لحظة التصريح عن الغرض من نوع مجرى الملف.

```
// This program demonstrates the opening of a file at the
```

```
// time the file stream object is declared.
```

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

```
#include <fstream.h>
```

```

int main( )
{
    fstream dataFile("c:files_c++\\names.dat", ios::in |
    ios::out);
    cout << "The file names.dat was opened.\n";
    return 0;
} //The end of main function

```

حيث أن العبارة البرمجية:

```

fstream dataFile("c:files_c++\\names.dat", ios::in | ios::out);

```

تعني إعطاء علم للمترجم بأننا نرغب بفتح ملف للكتابة (كتابة المعلومات في الملف) والقراءة (قراءة المعلومات من الملف)، اسمه names.dat وسيتم تخزينه في المجلد files_c++ (تسمية اختيارية للمجلد) والمحتوى في السواعة C أو أي مجلد آخر. وعند تنفيذ هذا البرنامج سنحصل على الخرج التالي:

Result

The file names.dat was opened.

Press any key to continue

ومن أجل اختبار أخطاء فتح الملفات نستخدم الكتلة البرمجية التالية:

//Testing for Open Errors

```
dataFile.open("c:files_c++\\cust.dat", ios::in);  
if (!dataFile)  
{  
    cout << "Error opening file.\n";  
}
```

ولابد من التنويه، إلى أنه يوجد طريقة أخرى لاختبار أخطاء فتح الملفات تتمثل في استخدام الكتلة البرمجية التالية:

//Another way to Test for Open Errors

```
dataFile.open("c:files_c++\\cust.dat", ios::in);  
if (dataFile.fail())  
{  
    cout << "Error opening file.\n";  
}
```

٧-٧- إغلاق الملف

Closing a File

الملف يجب أن يُغلق عندما ينتهي البرنامج من استخدامه لهذا الملف. والمثال (3-7) يوضح لنا كيفية إغلاق الملف المستخدم من قبل البرنامج .

مثال (7-3)

البرنامج التالي يوضح لنا كيفية استخدام مفهوم إغلاق الملف عندما ينتهي البرنامج من استخدام الملف المعني.

// This program demonstrates the close function.

```
#include <iostream.h>
#include <fstream.h>
int main( )
{
    fstream dataFile;
    dataFile.open("c:files_c++\\testfile.txt", ios::out);
    if (!dataFile) {
        cout << "File open error! " << endl;
        return 1;
    }
    cout << "File was created successfully.\n";
    cout << "Now closing the file.\n";
    dataFile.close();
    return 0;
} //The end of main function
```

وعند تنفيذ هذا البرنامج سنحصل على الخرج التالي:

Result

File was created successfully.

Now closing the file.

Press any key to continue

٧-٨ - استخدام المعامل (<<) من أجل الكتابة في الملف

Using (<<) to Write Information to a File

إن عملية الإدخال باستخدام المجرى << يمكن استخدامها من أجل كتابة المعلومات في الملف.

مثال (7-4)

البرنامج التالي يوضح لنا كيفية استخدام المجرى << من أجل كتابة المعلومات في الملف.

// This program uses the << operator

//to write information to a file.

#include <iostream.h>

#include <fstream.h>

int main() {

fstream dataFile;

char line[81];

dataFile.open("c:files_c++\\demofile.txt", ios::out);

if (!dataFile) {

```

        cout << "File open error!" << endl;
        return 1;
    }
    cout << "File opened successfully.\n";
    cout << "Now writing information to the file.\n";
    dataFile << "Jones\n";
    dataFile << "Smith\n";
    dataFile << "Willis\n";
    dataFile << "Davis\n";
    dataFile.close( );
    cout << "Done.\n";
    return 0;
} //The end of main function

```

وعند تنفيذ هذا البرنامج سنحصل على الخرج التالي:

Result
 File opened successfully.
 Now writing information to the file.
 Done.
Press any key to continue

الخرج الذي سيتم إدخاله إلى الملف **demofile.txt** هو التالي:

Jones

Smith

Willis

Davis

و يوضح الشكل (7-4) كيفية توضع المحارف المكونة للمعلومات التي تم إدخالها إلى الملف ، وبالتالي في ذاكرة الحاسوب .

J	o	n	e	s	\n	S	m	i	t	h	\n	W	i	l
---	---	---	---	---	----	---	---	---	---	---	----	---	---	---

l	i	s	\n	D	a	v	i	s	\n	<EOF>
---	---	---	----	---	---	---	---	---	----	-------

الشكل (7-4) - يوضح كيفية توضع المحارف المكونة للمعلومات

مثال (7-5)

البرنامج التالي يقوم بفتح ملف بالاسم demofile.txt ومن ثم كتابة بعض المعلومات فيه ، بعد ذلك يتم إغلاقه. ومن جديد يتم فتحه وإضافة معلومات جديدة إليه ، ومن ثم إغلاقه بشكل نهائي.

/ This program writes information to a file, closes the file,
then reopens it and appends more information. */*

#include <iostream.h>

#include <fstream.h>

int main()

{

 fstream dataFile;

 dataFile.open("c:files_c++\\demofile.txt",
 ios::out);

 dataFile << "Jones\n";

 dataFile << "Smith\n";

 dataFile.close();

 dataFile.open("c:files_c++\\demofile.txt",
 ios::app);

 dataFile << "Willis\n";

 dataFile << "Davis\n";

 dataFile.close();

return 0;

}//The end of main function

وعند تنفيذ هذا البرنامج سنحصل على الخرج التالي:



٧-٩- شكل الخرج في الملف

File Output Formatting

إن شكل إخراج المعلومات في الملف يمكن تصميمه بنفس الأسلوب المستخدم لتصميم شكل إخراج المعلومات على الشاشة.

مثال (٦-٧)

البرنامج التالي يستخدم التابع العضو precision لغرض من ملف مجرى الدخل لأجل صياغة خرج الملف.

/ This program uses the precision member function of a file stream object to format file output. */*

#include <iostream.h>

#include <fstream.h>


```

int main( )
{
    fstream dataFile;
    float num = 123.456;
    dataFile.open("c:files_c++\\numfile.txt", ios::out);
    if (!dataFile)
    {
        cout << "File open error!" << endl;
        return 1;
    }
    dataFile << num << endl;
    dataFile.precision(5);
    dataFile << num << endl;
    dataFile.precision(4);
    dataFile << num << endl;
    dataFile.precision(3);
    dataFile << num << endl;
    return 0;
} //The end of main function

```

وعند تنفيذ هذا البرنامج سنحصل على النتيجة التالية:

Result

//Contents of File numfile.txt

123.456

123.46

123.5

124

Press any key to continue

مثال (7-7)

من خلال البرنامج التالي سنقوم بالتصريح عن غرض من الصفfstream ومن ثم التصريح عن مصفوفة ثنائية البعد وإدخال قيم عناصرها ، بعد ذلك سنقوم بعملية إخراج عناصر المصفوفة في ملف من خلال الغرض outFile .

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

```
#include <fstream.h>
```

```
#include <iomanip.h>
```

```
int main()
```

```
{
```

```
    fstream outFile("c:files_c++\\table.txt", ios::out);
```

```
    int nums[3][3] = { 2897, 5, 837, 34, 7, 1623,  
                      390, 3456, 12 };
```

```

// Write the three rows of numbers
for (int row = 0; row < 3; row++)
{
    for (int col = 0; col < 3; col++)
    {
        outFile << setw(4) << nums[row][col]
        << " ";
    }
    outFile << endl;
}
outFile.close( );
return 0;
} //The end of main function

```

وعند تنفيذ هذا البرنامج سنحصل على الخرج التالي (محتوى الملف) :

Result

//Contents of File TABLE.TXT

2897

5

837

34

7

1623

390

3456

12

Press any key to continue

ويوضح الشكل (٥-٧) توضع المعطيات في الذاكرة .

2	8	9	7					5			8	3	7		\n
---	---	---	---	--	--	--	--	---	--	--	---	---	---	--	----

		3	4					7		1	6	2	3		\n
--	--	---	---	--	--	--	--	---	--	---	---	---	---	--	----

	3	9	0		3	4	5	6				1	2		\n
--	---	---	---	--	---	---	---	---	--	--	--	---	---	--	----

<EOF>

الشكل (٥-٧) - يوضح توضع المعطيات في الذاكرة

٧-١٠ - استخدام المعامل >> من أجل القراءة من ملف

Using >> to Read Information from a File

إن عملية الإخراج باستخدام المجري >> يمكن استخدامها من أجل قراءة المعلومات من الملف.

مثال (7-8)

البرنامج التالي يوضح لنا كيفية استخدام المجري >> من أجل قراءة المعلومات من الملف.

```
/* This program uses the >> operator to read information from
a file. */
```

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

```
#include <fstream.h>
```

```
int main( )
```

```
{
```

```

fstream dataFile;
char name[81];
dataFile.open("c:files_c++\\demofile.txt", ios::in);
if (!dataFile)
{
    cout << "File open error! " << endl;
    return 1;
}
cout << "File opened successfully.\n";
cout << "Now reading information from the
    file.\n\n";
for (int count = 0; count < 4; count++)
{
    dataFile >> name;
    cout << name << endl;
}
dataFile.close( );
cout << "\nDone.\n";
return 0;
} //The end of main function

```

وعند تنفيذ هذا البرنامج سنحصل على النتيجة التالية:

Result

//Program Screen Output

File opened successfully.

Now reading information from the file.

Jones

Smith

Willis

Davis

Done.

Press any key to continue

٧-١١ - تحديد نهاية الملف

Detecting the End of a File

عند قراءة محتويات ملف لا بد من التأكد من عدم الوصول إلى نهاية الملف وعند الوصول إلى نهاية الملف يتم التوقف عن القراءة، ويتم تحديد نهاية الملف باستخدام التابع `eof()` . ولهذه الغاية نستخدم التعليمة:

```
if (inFile.eof())  
    inFile.close();
```


مثال (7-9)

البرنامج التالي يستخدم غرض مجرى الملف مع التابع العضو eof () من أجل تحديد نهاية الملف.

/ This program uses the file stream object's eof()
member function to detect the end of the file.*/*

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

```
#include <fstream.h>
```

```
int main( )
```

```
{
```

```
    fstream dataFile;
```

```
    char name[81];
```

```
    dataFile.open("c:files_c++\\demofile.txt", ios::in);
```

```
    if (!dataFile)
```

```
    {
```

```
        cout << "File open error!" << endl;
```

```
        return 1;
```

```
    }
```

```
    cout << "File opened successfully.\n";
```

```
    cout << "Now reading information from the  
file.\n\n";
```

```
    dataFile >> name; // Read first name from the file
```

```
while (!dataFile.eof( ))
{
    cout << name << endl;
    dataFile >> name;
}
dataFile.close( );
cout << "\nDone.\n";
return 0;
} //The end of main function
```

وعند تنفيذ هذا البرنامج سنحصل على النتيجة التالية:

Result

//Program Screen Output

File opened successfully.

Now reading information from the file.

Jones

Smith

Willis

Davis

Done.

Press any key to continue

ملاحظة (٢)

لا بد من التنويه إلى أنه في لغة C++ ، نهاية الملف لا تعني أن البرنامج موجود في الجزء الأخير من معلومات (بيانات) الملف. التابع eof () يُعيد القيمة true عندما لا يوجد أي معلومات أخرى للقراءة .

١٢-٧ - التتابع الأعضاء لكتابة وقراءة الملفات

Member Functions for Reading and Writing Files

يتم فتح ملف نصي للكتابة فيه أو القراءة منه باستخدام تتابع لغة C++.

مثال (7-10)

من خلال البرنامج التالي سنقوم بفتح ملف نصي للكتابة فيه ومن ثم اختبار فتح الملف ، وفي النهاية إغلاق الملف.

// This program writes information to a file , closes the file

```
1:#include <iostream.h>
2:#include <fstream.h>
3:int main( )
4:{
5:    ofstream out1;
6:    out1.open("c:files_c++\\Store.txt",ios::out);
7:    if (!out1)
8:    {
```

```

9:         cout << "Cannot Open Store File..\n" << endl;
10:         return 1;
11:     }
12:     out1<<99.95<<endl;
13:     out1<<15.9<<endl;
14:     out1.close( );
15:     return 0;
16: }//The end of main function

```

نوضح في هذا البرنامج ما يلي:

- في السطرين 5 و 6 يتم التصريح عن غرض باسم out1 من الصف ofstream مع إعطائه قيمة ابتدائية هي الاسم Store ويتم إرسالها إلى تابع باني الصف والذي يقوم بفتح ملف للكتابة باسم Store .
- في السطر 7 يتم اختبار الغرض out1 إذا كان غير صحيح ، فهذا يعني عدم إمكانية فتح الملف ويتم طباعة رسالة بذلك في السطر رقم 9 .
- في السطرين 12 و 13 يتم الكتابة في الملف Store وذلك من خلال الغرض out1 .
- في السطر 14 يتم إغلاق الملف وذلك باستخدام التابع close من خلال الغرض out1 .

وعند تنفيذ هذا البرنامج سنحصل على النتيجة التالية:

Result

99.95

15.9

Press any key to continue

بالتأكيد هذه المعلومات ستكون مخزنة في الملف Store وليس على شاشة تنفيذ البرنامج. في هذا المثال رأينا كيف تم فتح ملف باسم Store والكتابة فيه ، و سنوضح في المثال (7 - 11) كيف يمكن فتح هذا الملف وقراءة معطياته وطباعتها على الشاشة.

مثال (7-11)

البرنامج التالي يوضح لنا كيفية فتح الملف الذي تم إنشاؤه في المثال (7 - 10) وكتابة بعض المعطيات فيه (إدخال المعطيات) ، ثم إغلاقه.

// This program writes information to a file , closes the file

```
1:#include <iostream.h>
2:#include <fstream.h>
3:int main( )
4:{
5:    ifstream in;
6:    in.open("c:files_c++\\Store.txt",ios::in);
7:    if (!in)
8:    {
9:        cout << "Cannot Open Store File..\n" << endl;
10:        return 1;
11:    }
12:    float Length, width;
```



```

13:   in>>Length>>Width;
14:   cout<<" Length=" << Length<<endl;
15:   cout<<" Width= " << Width <<endl;
16:   in.close( );
17:   return 0;
18: }//The end of main function

```

نوضح في هذا البرنامج ما يلي:

- في السطرين 5 و 6 يتم التصريح عن غرض بالاسم `in` من الصف `ifstream` مع إعطائه قيمة ابتدائية هي الاسم `Store` ويتم إرسالها إلى تابع باني الصف لفتح ملف بهذا الاسم .
- في السطر 13 يتم قراءة المعطيات من الملف المشار إليه بالغرض `in` ووضع المعطيات في المتغيرات `Length` و `Width` .
- في السطرين 14 و 15 يتم طباعة قيم المتغيرات `Length` و `Width` على الشاشة.
- وعند تنفيذ هذا البرنامج سنحصل على الخرج التالي (على الشاشة) .

Result

//Program Screen Output

Length=99.95

Width=15.9

Press any key to continue

مثال (7-12)

يوضح البرنامج التالي لنا كيفية استخدام المجرى << من أجل كتابة المعلومات في ملف ما.

/ This program uses the << operator to write information to a file. */*

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

```
#include <fstream.h>
```

```
int main( )
```

```
{
```

```
    fstream dataFile;
```

```
    char line[80];
```

```
    dataFile.open("c:files_c++\\murphy.txt", ios::out);
```

```
    if (!dataFile)
```

```
    {
```

```
        cout << "File open error!" << endl;
```

```
        return 1;
```

```
    }
```

```
    cout << "File opened successfully.\n";
```

```
    cout << "Now writing information to the file.\n";
```

```
    dataFile << "JayneMurphy47JonesCircleAlmond,NC28702\n";
```

```
    dataFile.close();
```

```
return 0;
```

```
}//The end of main function
```

وعند تنفيذ البرنامج سنحصل على النتيجة التالية:

Result

//Program Screen Output

File opened successfully.

Now writing information to the file.

Press any key to continue

مثال (7-13)

البرنامج التالي يوضح لنا كيفية استخدام غرض من الصف `fstream` مع التابع العضو `eof()` للوصول إلى نهاية الملف، الذي تم إنشاؤه في المثال (7-12).

*// This program uses the file stream object's eof() member function to detect the end of the file. */*

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

```
#include <fstream.h>
```

```
int main( )
```

```
{
```

```
    fstream nameFile;
```

```
    char input[80];
```

```

nameFile.open("c:files_c++\\murphy.txt", ios::in);
if (!nameFile)
{
    cout << "File open error!" << endl;
    return 1;
}
nameFile >> input;
while (!nameFile.eof( ))
{
    cout << input<<endl;
    nameFile >> input;
}
nameFile.close( );
return 0;
} //The end of main function

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output

JayneMurphy47JonesCircleAlmond,NC28702

Press any key to continue

٧-١٣ - التابع العضو getline

يستخدم هذا التابع في كتابة (إدخال) سطر كامل في الملف متضمناً الفراغات والشكل العام لاستدعاء هذا التابع العضو هو التالي:

```
dataFile getline(str, m, '\n');
```

حيث:

str - هذا الوسيط يدل على اسم مصفوفة المحارف أو المؤشر الذي يُشير إلى جزء من الذاكرة. والمعلومات التي سيتم قراءتها من الملف سيتم تخزينها في هذه المصفوفة.

m - هذا الوسيط يدل على حجم المصفوفة **str** . في هذه الحالة يتم قراءة **m-1** حرفاً كحد أقصى.

'\n' - هذا الوسيط يمثل المحرف الذي يحدد نهاية السطر. وعند الوصول إليه ، فإنه يقوم بوقف عملية القراءة من الملف قبل الوصول إلى العدد الأعظمي من المحارف. هذا الوسيط يعتبر افتراضياً، أي يمكن الاستغناء عنه.

dataFile - غرض من الصف **fstream** .

مثال (7-14)

البرنامج التالي يوضح لنا كيفية استخدام غرض من الصف **fstream** مع التابع العضو **getline()** من أجل قراءة سطر من المعلومات من ملف تم فتحه.

```
/* This program uses the file stream object's getline member  
function to read a line of information //from the file. */
```

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

```
#include <fstream.h>
```

```

int main( )
{
    fstream nameFile;
    char input[80];
    nameFile.open("c:files_c++\\murphy1.txt",
        ios::in);
    if (!nameFile)
    {
        cout << "File open error!" << endl;
        return 1;
    }
    // use \n as a delimiter
    nameFile.getline(input, 80, '\n');
    while (!nameFile.eof( ))
    {
        cout << input << endl;
        // use \n as a delimiter
        nameFile.getline(input, 80, '\n');
    }
    nameFile.close();
} //The end of main function

```

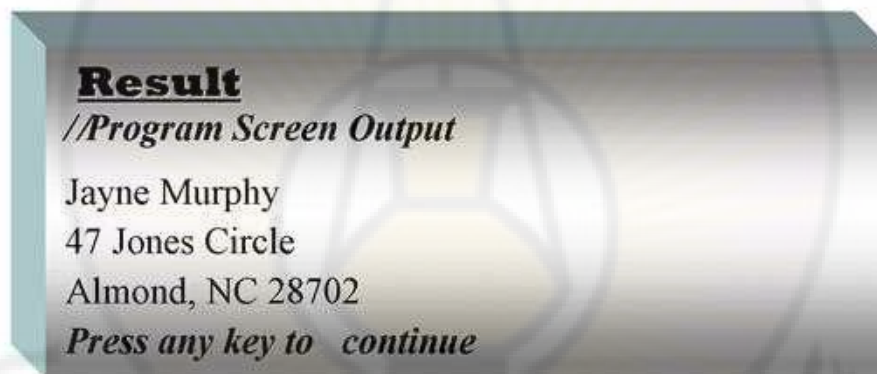
بالرجوع إلى البرنامج في المثال (7-12) ، سنقوم باستبدال السطر:


```
dataFile << "JayneMurphy47JonesCircleAlmond,NC28702\n";
```

بالكتلة البرمجية التالية:

```
dataFile <<("JayneMurphy47 \n");  
dataFile<<("JonesCircle \n");  
dataFile<<("Almond,NC28702 \n");
```

بعد القيام بهذه التعديلات وتنفيذ البرنامج في المثال (7-14) ، سنحصل على
الخرج التالي (على الشاشة):



٧-١٤ - التابعان get() و put()

المعطيات النصية هي المعطيات المخزنة على شكل حروف وأرقام ولكن هناك شكل آخر وهو تخزين المعطيات بالنظام الثنائي Binary System ، وفي هذه الفقرة سنوضح كيفية كتابة وقراءة المعطيات باستخدام النظام الثنائي، ولتحقيق هذه الغاية يتوفر في لغة C++ التابعين get() و put() .

٧-١٤-١ - التابع العضو get ()

يتم استدعاء التابع get(ch) للقراءة من الملف وذلك من خلال استخدام غرض من الصف ifstream ، ووضع المعطيات المقروءة في المتغير ch (حرف بحرف). والشكل العام لاستدعاء هذا التابع هو التالي:
inFile.get(ch);

مثال (7-15)

البرنامج التالي يطلب من المستخدم إدخال اسم الملف الذي يرغب بفتحه، بعد ذلك سيتم فتح الملف المطلوب وإخراج محتوياته على الشاشة.

/ This program asks the user for a file name. The file is opened and its contents are displayed on the screen. */*

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

```
#include <fstream.h>
```

```
int main( )
```

```
{
```

```
    fstream file;
```

```
    char ch, fileName[51];
```

```
    cout << "Enter a file name: ";
```

```
    cin >> fileName;
```

```
    file.open(fileName, ios::in);
```

```
    if (!file) {
```

```

        cout << fileName << " could not be opened.\n";
        return 1;
    }
    file.get(ch); // Get a character
    while (!file.eof() )
    {
        cout << ch;
        file.get(ch); // Get another character
    }
    file.close( );
    return 0;
} //The end of main function

```

٧-١٤-٢- التابع العضو put()

يتم استخدام التابع put() لكتابة حرف بحرف في ملف ما تم فتحه. والشكل العام لاستخدام هذا التابع هو كما يلي :

```
outFile.put(ch);
```

مثال (7-16)

البرنامج التالي يوضح لنا كيفية استخدام التابع put () من أجل الكتابة في الملف الذي تم فتحه وذلك حرفاً حرفاً .

// This program demonstrates the put member function.

```

#include <iostream.h>
#include <fstream.h>
void main( ) {
    fstream dataFile("c:files_c++\\sentence.txt",
    ios::out);
    char ch;
    cout << "Type a sentence and be sure to end it
    with a ";
    cout << "period.\n";
    while (1)
    {
        cin.get(ch);
        dataFile.put(ch);
        if (ch == '.')
            break;
    }
    dataFile.close( );

} //The end of main function

```

وعند تنفيذ البرنامج ستحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output with Example Input

Type a sentence and be sure to end it with a period.

I am on my way to becoming a great programmer.

//Resulting Contents of the File SENTENCE.TXT:

I am on my way to becoming a great programmer.

Press any key to continue

مثال (7-17)

يتم في البرنامج التالي استخدام الملفات الثنائية Binary Files التي تحتوي على معطيات غير نصية والتي ليست بالضرورة تكون مخزنة على شكل نصي كُتب بالنظام ASCII، والشكل العام لفتح مثل هذا النوع من الملفات هو التالي:

```
file.open("c:files_c++\\NUMS.DAT ", ios::out | ios::binary);
```

// This program uses the write and read functions.

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

```
#include <fstream.h>
```

```
void main( ) {
```

```
    fstream file("c:files_c++\\NUMS.DAT", ios::out |  
                ios::binary );
```

```

int buffer[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
cout << "Now writing the data to the file.\n";
file.write((char*)buffer, sizeof(buffer));
file.close();

// Reopen the file.
file.open("c:files_c++\\NUMS.DAT", ios::in);
cout << "Now reading the data back into
memory.\n";
file.read((char*)buffer, sizeof(buffer));
for (int count = 0; count < 10; count++)
    cout << buffer[count] << " ";
cout<<endl;
file.close( );
} //The end of main function

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output

Now writing the data to the file.

Now reading the data back into memory.

1 2 3 4 5 6 7 8 9 10

Press any key to continue

٧-١٥ - ملفات الوصول العشوائي

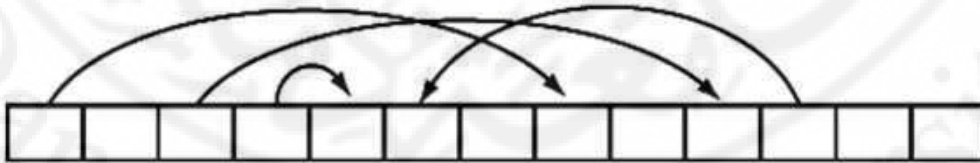
Random Access Files

من خلال جميع الأمثلة السابقة لاحظنا كيف يمكن إنشاء الملفات التسلسلية وكيف يمكن البحث عن المعلومات داخل هذه الملفات. وفي هذه الفقرة سنتناول نوع جديد من الملفات ، ألا وهو ملفات الوصول العشوائي. تستخدم ملفات الوصول العشوائي في التطبيقات التي تحتاج إلى وصول لحظي إلى سجل معين من المعطيات، حيث يمكن الوصول إلى أي سجل ضمن هذه الملفات بشكل مباشر (وسريع) بدون أية عملية بحث ضمن السجلات الأخرى. ويبين الشكل (٦-٧) مقارنة بين الملفات التسلسلية والملفات العشوائية ، ويبين الجدول (٦-٧) حالات استخدام التتابع beg و end و cur أما الجدول (٧-٧) يبين حالات استخدام التابع () seekg .

Sequential Access



Random Access



الشكل (٦-٧) - يبين مقارنة بين الملفات التسلسلية والملفات العشوائية

Mode Flag شكل الحالة	Description الوصف
ios::beg	التغيير يُحسب من بداية الملف The offset is calculated from the beginning of the file.
ios::end	التغيير يُحسب من نهاية الملف The offset is calculated from the end of the file.
ios::cur	التغيير يُحسب من نهاية الملف The offset is calculated from the end of the file.

الجدول (٩-٦) - يبين حالات استخدام التتابع beg و end و cur

Statement التعليمة	How it Affects the Read/Write Position معنى استخدام التعليمة للقراءة/ للكتابة
File.seekp(32L, ios::beg);	هذه التعليمة تضع موضع الكتابة في البايت رقم 33 (32 بايت) من بداية الملف .

<code>file.seekp(-10L, ios::end);</code>	هذه التعليمة تضع موضع الكتابة في البايت رقم 11 (10 بايت) من نهاية الملف.
<code>file.seekp(120L, ios::cur);</code>	هذه التعليمة تضع موضع الكتابة في البايت رقم 121 (120 بايت) من الموضع الحالي.
<code>file.seekg(2L, ios::beg);</code>	هذه التعليمة تضع موضع القراءة إلى البايت رقم 3 (2 بايت) من بداية الملف.
<code>file.seekg(-100L, ios::end);</code>	هذه التعليمة تضع موضع القراءة إلى البايت رقم 101 (100 بايت) من نهاية الملف.
<code>file.seekg(40L, ios::cur);</code>	هذه التعليمة تضع موضع القراءة إلى البايت رقم 41 (40 بايت) من الموضع الحالي.
<code>file.seekg(0L, ios::end);</code>	هذه التعليمة تضع موضع القراءة إلى نهاية الملف.

الجدول (٧-٧) - يبين حالات استخدام التابع () seekg

مثال (7-18)

البرنامج التالي يوضح لنا كيفية الوصول إلى البيانات عشوائياً في الملف باستخدام التابع ().seekg

// This program demonstrates the seekg function.

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

```
#include <fstream.h>
```

```
void main( )
```

```
{
```

```
    fstream file("sentence.txt", ios::in);
```

```
    char ch;
```

```
    file.seekg(5L, ios::beg);
```

```
    file.get(ch);
```

```
    cout << "Byte 5 from beginning: " << ch << endl;
```

```
    file.seekg(-10L, ios::end);
```

```
    file.get(ch);
```

```
    cout << "Byte 10 from end: " << ch << endl;
```

```
    file.seekg(3L, ios::cur);
```

```
    file.get(ch);
```

```
    cout << "Byte 3 from current: " << ch << endl;
```

```
    file.close( );
```

```
}//The end of main function
```

وبفرض أنه لدينا الملف sentence.txt والذي يحتوي على العبارة التالية:

I am on my way to becoming a great programmer

فعند تنفيذ البرنامج سنحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output

Byte 5 from beginning: o

Byte 10 from end: a

Byte 3 from current: r

Press any key to continue

٧-١٦ - إنشاء السجلات باستخدام البنى

Creating Records with Structures

في هذه الفقرة سنتناول كيف يمكن استخدام مفهوم البنى Structures من أجل تخزين السجلات في الملف ، وحيث أن هذه السجلات تكون مكونة من حقول تملك أنواع مختلفة من المعطيات . ولهذه الغاية لنأخذ التعريف البرمجي التالي لبنية اسمها Info :

```
struct Info
{
    char name[50];
    int age;
    char address1[50];
    char address2[50];
    char phone[14];
}
```

};

طالما أن السجلات يمكن أن تتضمن خليط من المعطيات المختلفة الأنواع، فإنه من أجل فتح ملف ما لتخزين مثل هكذا نوع من السجلات لا بد من استخدام المحدد `ios::binary` ضمن العبارة اللازمة والمستخدم لفتح الملفات ، فعلى سبيل المثال:

```
fstream people("people.dat", ios::out | ios::binary);
```

مثال (7-19)

البرنامج التالي يوضح لنا كيفية مفهوم البنى Structures من أجل تخزين سجل من المعلومات في ملف يتم فتحه لهذه الغاية.

/ This program demonstrates the use of a structure variable to store a record of information to a file. */*

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

```
#include <fstream.h>
```

```
#include <ctype.h> // for toupper
```

```
// Declare a structure for the record.
```

```
struct Info
```

```
{
```

```
    char name[50];
```

```
    int age;
```

```
    char address1[50];
```

```
    char address2[50];
```

```
    char phone[14];
```



```

};
int main( )
{
    fstream people("c:files_c++\\people.dat", ios::out
    | ios::binary);
    Info person;
    char again;
    if (!people)
    {
        cout << "Error opening file. Program
        aborting.\n";
        return 1 ;
    }
    do
    {
        cout << "Enter the following information
        about a "
            << "person:\n";
        cout << "Name: ";
        cin.getline(person.name, 50);
        cout << "Age: ";
        cin >> person.age;
        // skip over remaining newline.
    }
    while (again != 'n');
}

```



```

        cin.ignore( );
        cout << "Address line 1: ";
        cin.getline(person.address1, 50);
        cout << "Address line 2: ";
        cin.getline(person.address2, 50);
        cout << "Phone: ";
        cin.getline(person.phone, 14);
        people.write((char *)&person,
        sizeof(person));
        cout << "Do you want to enter another
        record? ";
        cin >> again;
        cin.ignore( );
    }
    while (toupper(again) == 'Y');
    people.close( );
    return 0;
} //The end of main function

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output with Example Input

Enter the following information about a person:

Name: Charlie Baxter

Age: 42

Address line 1: 67 Kennedy Bvd.

Address line 2: Perth,SC 38754

Phone: (803)555-1234

Do you want to enter another record? y

Enter the following information about a person:

Name: Merideth Murney

Age: 22

Address line 1: 487 Lindsay Lane

Address line 2: Hazelwood,NC 28737

Phone: (704)453-9999

Do you want to enter another record? n

Press any key to continue

٧-١٧ - فتح ملف لعمليتي الإدخال والإخراج

Opening a File for Both Input and Output

في هذه الفقرة سنتعرف على كيفية فتح ملف ما من أجل الكتابة (الإدخال) والقراءة (الإخراج) ، ويتم ذلك بفتح الملف أولاً للكتابة ، دون الحاجة لإغلاقه ومن ثم إعادة فتحه من جديد ، ولهذه الغاية لا بد من استخدام العبارة البرمجية التالية ضمن البرنامج المعني بذلك:

```
fstream file(" data.dat ", ios::in | ios::out);
```

مثال (7-20)

البرنامج التالي يوضح لنا كيفية تعريف بنية برمجياً ، ومن ثم استخدام هذه البنية من أجل فتح ملف ما لكتابة المعطيات الخاصة ببنية اسمها Invtry ضمن الملف الذي تم فتحه.

// This program sets up a file of blank inventory records.

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

```
#include <fstream.h>
```

// Declaration of Invtry structure

```
struct Invtry {
```

```
    char desc[30];
```

```
    int qty;
```

```
    float price;
```

```
};
```

```

void main( ) {
    fstream inventory("c:files_c++\\invtry.dat", ios::out |
    ios::binary);
    Invtry record = { "", 0, 0.0 };
    // Now write the blank records
    for (int count = 0; count < 5; count++)
    {
        cout << "Now writing record " << count << endl;
        inventory.write((char *)&record, sizeof(record));
    }
    inventory.close( );
} //The end of main function

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output

Now writing record 0

Now writing record 1

Now writing record 2

Now writing record 3

Now writing record 4

Press any key to continue

مثال (7-21)

البرنامج التالي يوضح لنا كيفية عرض محتويات ملف السجل الذي تم فتحه وكتابة معطياته فيه من خلال المثال (7-20).

// This program displays the contents of the inventory file.

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

```
#include <fstream.h>
```

```
// Declaration of Invtry structure
```

```
struct Invtry
```

```
{
```

```
    char desc[30];
```

```
    int qty;
```

```
    float price;
```

```
};
```

```
void main( ) {
```

```
    fstream inventory("c:files_c++\\invtry.dat",
```

```
    ios::in | ios::binary);
```

```
    Invtry record = { "", 0, 0.0 };
```

```
// Now read and display the records
```

```
    inventory.read((char *)&record, sizeof(record));
```

```
    while (!inventory.eof( )) {
```

```
        cout << "Description: ";
```

```
        cout << record.desc << endl;
```



```

        cout << "Quantity: ";
        cout << record.qty << endl;
        cout << "Price: ";
        cout << record.price << endl << endl;
        inventory.read((char *)&record, sizeof(record));
    }
    inventory.close( );
} //The end of main function

```

بعد تنفيذ البرنامج في المثال (7-20) ، ومن ثم وبشكل مباشر تنفيذ البرنامج في المثال (7-21) ، سنحصل على الخرج التالي (على الشاشة):

Result

Description:
Quantity: 0
Price: 0
Description:
Quantity: 0
Price: 0
Description:
Quantity: 0
Price: 0
Description:
Quantity: 0
Price: 0
Description:
Quantity: 0
Price: 0
Press any key to continue

مثال (7-22)

البرنامج التالي يسمح للمستخدم بتحرير (جلب) سجل محدد من ملف قائمة الجرد Inventory File ، وهذا بالتأكيد بعد تنفيذ البرنامجين الواردين في المثالين (20-7) و (21-7) على الترتيب.

/ This program allows the user to edit a specific record in the inventory file. */*

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

```
#include <fstream.h>
```

```
// Declaration of Invtry structure
```

```
struct Invtry
```

```
{
```

```
    char desc[30];
```

```
    int qty;
```

```
    float price;
```

```
};
```

```
void main( )
```

```
{
```

```
    fstream inventory("c:files_c++\\invtry.dat",
```

```
    ios::in | ios::out | ios::binary);
```

```
    Invtry record;
```

```
    long recNum;
```

```

cout << "Which record do you want to edit?";
cin >> recNum;
inventory.seekg(recNum * sizeof(record), ios::beg);
inventory.read((char *)&record, sizeof(record));
cout << "Description: ";
cout << record.desc << endl;
cout << "Quantity: ";
cout << record.qty << endl;
cout << "Price: ";
cout << record.price << endl;
cout << "Enter the new data:\n";
cout << "Description: ";
cin.ignore( );
cin.getline(record.desc, 31);
cout << "Quantity: ";
cin >> record.qty;
cout << "Price: ";
cin >> record.price;
inventory.seekp(recNum * sizeof(record), ios::beg);
inventory.write((char *)&record, sizeof(record));
inventory.close( );
} //The end of main function

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي (على الشاشة):

Result

//Program Screen Output with Example Input

Which record do you want to edit?2

Description:

Quantity: 0

Price: 0

Enter the new data:

Description: Wrench

Quantity: 10

Price: 4.67

Press any key to continue





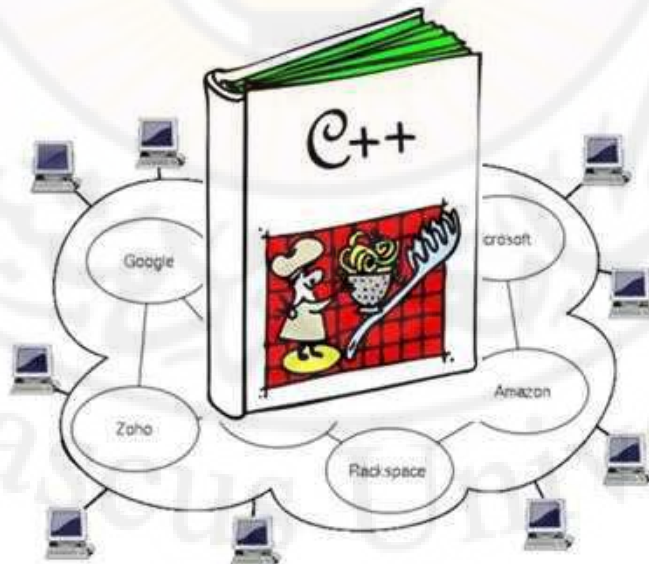
+

المراجع الأجنبية

References

1. BJECTE ORIENTED PROG WITH C++ 4E books 13
Digit ISBN 9780070669079 ,Publication Year 2010 ,
Authors E.BALAGURUSAMY , Book Edition 4 , Book
Binding Type SOFTCOVER , Number of pages N/A
2. DAVID LEVINE.“ **TEACH YOURSELF C++**” .Second
Edition, Greg Wiegand Sams Publishing 201 W. 103rd
Street Indianapolis, IN 46290, 772 p.
3. DEITEL P.J ,DEITEL H.M.[2007]. “ **C HOW TO
PROGRAM**”. Fifth Edition,2007 by Pearson Education,
Inc.Upper Saddle River, New Jersey 07458,1110 p.
4. DEITEL P.J ,DEITEL H.M.[2008]. “ **C++ HOW TO
PROGRAM**”.Sixth Edition,2008 by Pearson Education ,
Inc.Upper Saddle River, New Jersey 07458,1429 p.
5. DOUGLAS C.SCHMIDT . ” **AN OVERVIEW OF C++** ”
Vanderbilt University,www.cs.wustl.edu, September 8,
2003.
6. FRANCOIS FLEURET . “ **C++ LECTURE NOTES** ”.
University of Chicago, 2010.

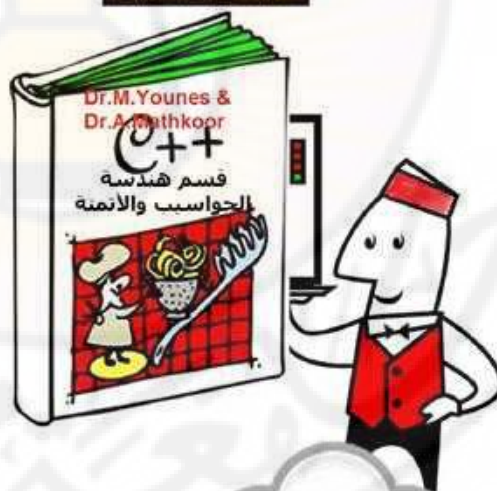
7. GREG PERRY . “C++ By EXAMPLE” . 1992 by Que
Printed in the United States of America 761 p.
8. HORSTMANN C.S.[1999] “COMPUTING CONCEPTS
WITH C++ ESSENTIALS”. Second Edition,1999 by
John Wiley & Sons, Inc. 327 p.
9. Herbert Schildt C++: **The Complete Reference, Fourth
Edition**, Copyright © 2007 by The McGraw-Hill
Companies.
10. The C++ Programming Language ,2012, Third Edition
and Special Edition Addison-Wesley, ISBN 0-201-
88954-4 and 0-201-70073-5
11. SHARAM HEKMAT .”C++ PROGRAMMING ”. 1998
Pragmatics Software Pty. Ltd.



المراجع العربية

١. التفكير والإبداع في لغة C++ - المهندس إحسان أحمد الرفاعي، المهندس ضياء زهير الأدرلي، المهندس ياسر عبد الكريم صبح- شعاع للنشر والعلوم - سورية - حلب - 2005.
٢. الدليل الشامل للاحتراف في لغة C++ - عبد الحافظ ، يمان البني . دار البراق - سورية - 2004 .
٣. كيف تبرمج بلغة C++ - الدكتور صلاح الدوه جي - شعاع للنشر والعلوم - سورية - حلب - 2004.
٤. لغة البرمجة C ، مدخل هام إلى لغة C++ - المهندس فادي حجار - شعاع للنشر والعلوم - سوريا - حلب - 2000.
٥. المرجع الأساسي لمستخدمي C/C++ - المهندس عزب محمد عزب، مجدي محمد أبو العطا - العربية لعلوم الحاسب - مصر - القاهرة - 1998.
٦. المرجع الأساسي في برمجة C++ ، د.عوض منصور وآخرون -الأردن - مؤسسة البشير الأردن - 2000 .





المصطلحات العلمية

إنكليزي - عربي

English – Arabic



A	
Adapters	الموائمات
Address	العنوان
Array Processors	المعالجات المصفوفية
Arrow Operator	العملية سهم (مؤشر)
Assigning Objects	إسناد الأغراض
Arrays of Objects	مصفوفة الأغراض
ANSI C	لغة C القياسية
Analog Signal	الإشارة التمثيلية
Analog Transmission	الإرسال التمثيلي
Add tow objects	اجمع غرضين
Abstract Class	الصف المجرد
Addressing	العنونة
Applying Polymorphism	تطبيقات تعدد الأشكال
Arrays Of Pointers	مصفوفة المؤشرات
ALU(Arithmetic and Logic Unit)	وحدة الحساب والمنطق
Assembly Language	لغة التجميع
ANSI(American National Standards Institute)	المعهد الأمريكي القومي للمقاييس
ASCII(American Standards Code for Information Interchange)	الترميز المعياري الأمريكي لتبادل المعلومات
AC(Accumulator)	المراكم

B	
Bandwidth	عرض النطاق (المجال)
BASIC	لغة بيزك
Bit Manipulation	التعامل على مستوى البت
Binary Operator	عملية ثنائية
Base Class	الصف الأساسي
Base Class Access Control	التحكم بالوصول إلى الصف الأساسي
Bubble Sort	الفرز الفقاعي
Buffer	ذاكرة التخزين المؤقتة (المخزن)
Byte	البايت (8 بت)
C	
Char	محرّف
Class	الصف
CISC(Complex Instruction Set Computers)	الحواسيب ذات مجموعة التعليمات المعقدة
CPU Design	تصميم وحدة المعالجة المركزية
Combined Programming Language	لغة CPL
Command	الأمر
Constant functions	التوابع الثابتة
constant Variables	المتحولات الثابتة

Cache	ذاكرة الكاش
Constructor	الباني
Computer Networks	شبكات الحواسيب
Constant Objects	الأغراض الثابتة
Composition	التركيب
Create an independent reference	إنشاء المراجع المستقلة
Coordinate value	قيمة إحداثيات النقطة
Create a queue	إنشاء رتل
Call – by – Value	الاستدعاء بالقيمة
Call – by – Reference	الاستدعاء بالعنوان
Call – by – Reference with Pointers	الاستدعاء بالعنوان بواسطة المؤشرات
CPU (Central Processing Unit)	وحدة المعالجة المركزية
Communication network	شبكة الاتصال
Computer Performance Evaluation	تقييم أداء الحاسوب
Compiler	مترجم
Control Logic	منطق التحكم
D	
data members	أعضاء المعطيات
Derived Class	الصف المشتق
Destructor	المدمر (الهدام)

Direct Base Class	الصف الأساسي المباشر
DMA (Direct Memory Access)	النفاز المباشر للذاكرة
Double	عدد حقيقي مضاعف
Dot Operator	العملية نقطة
Digital Data	المعطيات الرقمية
Digital Transmission	الإرسال الرقمي
Digital Switch	المبدل الرقمي
Dual Core	المعالجات ثنائية النواة
Data Cache	ذاكرة كاش للمعطيات
DMA (Direct Memory Access)	الوصول المباشر للذاكرة
Direct Addressing	العنونة مباشرة
Default Constructor	الباني الافتراضي
Degree	الدرجة
Distance Learning	التعليم عن بعد
Distributed Operating Systems	نظم التشغيل الموزعة
Distributed Systems	النظم الموزعة
Decrement	النقصان بمقدار واحد
dB (Disable)	الديسيبل
Data Flow Analysis	تحليل تدفق المعطيات
E	
Encapsulation	التغليف

End class	نهاية الصف
End main	نهاية التابع الرئيسي
Early Binding	الربط المبكر
Even	زوجي
Encryption	التشفير
Exception Handling	معالجة الاستثناءات
Exception Handler	معالج الاستثناءات
EBCDIC (Extended Binary Coded Decimal Interchange Code)	الترميز الثنائي الممتد لتبادل الرموز العشرية
External Interfaces	واجهات الترابط الخارجية
Electronic Mail	البريد الإلكتروني
External Cache memory	ذاكرة الكاش الخارجية
F	
Function	تابع
Float	عدد حقيقي
Floppy disk	الأقراص المرنة
Flexibility	المرونة
Failover Manager	مدير تدارك الفشل
Function Templates	قوالب التتابع
Class Templates	قوالب الصفوف
Friend Class	الصف الصديق

Friend Function	التابع الصديق
Friend Operator Functions	التوابع الصديقة للمعاملات
	التسامح بالأخطاء
	التجميع بتقسيم التردد
	النقل ثنائي الاتجاه
Fetch Stage	مرحلة الجلب
File Server	مخدم الملفات
FTP (File Transfer Protocol)	بروتوكول نقل الملفات
Filters	المرشحات
Firewall	الجدار الناري
	تصحيح الخطأ الأمامي
G	
Gbps (Giga a bit per second)	جيجا بيت بالثانية تقاس بها سرعة
Generation4 (G4)	الجيل الرابع
GUI (Graphic User Interface)	واجهة المستخدم الرسومية
Generation3 (G3)	الجيل الثالث
GPR (General Purpose Registers)	سجلات الأغراض العامة
Global Objects	الأغراض العامة
H	
Hardware	العتاد

Hard Disk's Interface	واجهة ترابط الأقراص الصلبة
Harvard Architecture	بنية هارفارد
Host Computer	الحاسوب المضيف
http (Hyper Text Transfer Protocol)	بروتوكول نقل النصوص المتشعبة
I	
I/O Bus	مساري الدخل/الخرج
Inheritance	الوراثة
I/O Functions	توابع الدخل/الخرج
Index of top-of-stack	دليل قمة المكس
Independent Access	النفاذ المستقل
Independent References	المراجع المستقلة
Interface of class	واجهة الصف
Instruction Cache	ذاكرة كاش للتعليمات
Internal Cache memory	ذاكرة كاش الداخلية
Interconnection Network	شبكة الترابط البيئي
Insertion Operator	عامل الإدراج
Istream	مجرى دخل
Interconnection Cost	كلفة الاتصال
Increment	الزيادة بمقدار واحد
ID (Instruction Decode)	فك ترميز التعليمات

IBM (International Business Machine)	شركة لإنتاج الحواسيب
IEEE (Institute of Electrical & Electronic Engineers)	معهد مهندسي الكهرباء والالكترونيات
Indirect Base Class	الصف الأساسي غير المباشر
IR (Instruction Register)	سجل التعليم IR
Infrared Technology	تقنية الأشعة تحت الحمراء
Instruction Fetch Datapath	جلب التعليم من ممر المعطيات
Idealized Memory	الذاكرة المثالية
Index Addressing	العنونة بالدليل
ISP (Internet Service Provider)	مزود خدمة الإنترنت
IBR(Instruction Buffer Register)	سجل مخزن التعليم
IR(Instruction Register)	سجل التعليم
J	
Jumper	موصل
Link	طبقة الربط
Local Bus	المسرى المحلي
Kbps (Kilo bit per second)	كيلو بت /الثانية
L	
Latency Time	زمن التأخير

Loop Buffer	مخزن الحلقة
Local Objects	الأغراض المحلية
Late Binding	الربط المتأخر
Local Loop	الحلقة المحلية
LED (Light Emitting Diodes)	ديودات لإصدار الضوء
Logical Operators	العمليات المنطقية
List	قائمة
Last In First Out	الداخل أخيراً خارج أولاً
LSB (Low Sign Bit)	الخانة ذات الوزن الأدنى
Localizing of Reference	المرجعية المحلية
M	
Master	سيد
Magnetic Tape	الأشرطة الممغنطة
Magnetic disk	الأقراص المغناطيسية
Multiple Inheritance	الوراثة المتعددة
Message	الرسائل
Microcontroller	المتحكم الصغري
Member	عضو
Modes Addressing	أنماط العنوان
Message Passing	تمرير الرسائل

Modem (Modulation and Demodulation)	الموديم
Multiplexing	التجميع أو الانتخاب
Machine Cycle	دورة الآلة
Main memory	الذاكرة الرئيسية
Macintosh Computer	حاسوب ماكنتوش
Mbps(Mega bit per second)	ميغابايت بالثانية (وحدة لقياس سرعة نقل المعلومات)
MMU(Memory Management Unit)	وحدة إدارة الذاكرة
Message Switching Technology	تقنية تبادل الرسائل
Memory card	بطاقة الذاكرة
Machine Language	لغة الآلة
Multitasking	تعدد المهام
Mainframes	الحواسيب الرئيسية (الكبيرة)
Minicomputers	الحواسيب الصغيرة
Microprocessor Speed	سرعة المعالج
MAR(Memory Address Register)	سجل عنوان الذاكرة
Multiprogramming	البرامج المتعددة
N	
Network	الشبكة

Nodes	العقد
Networks Operating Systems	نظم تشغيل الشبكات
Network Management	إدارة الشبكة
Novell Netware	نظام نوفل لتشغيل الشبكات
Network Load Balancing	توازن حمل الشبكة
Null	لا شيء (صفر)
Number	رقم
O	
Optical Disk	الأقراص الضوئية
Operator	عملية حسابية
Odd	فردى
Open Systems	النظم المفتوحة
Overloading	التحميل الزائد
Overloading Unary Operators	التحميل الزائد للعوامل الأحادية
Overloading Binary Operators	التحميل الزائد للعوامل الثنائية
Ordered Delivery	النقل المنظم
Overloading Constructor	التحميل الزائد للبيانات
Overloading Logical Operators	التحميل الزائد للعوامل المنطقية
Overloading The Relational Operators	التحميل الزائد لعوامل المقارنة
Override	التجاوز

Overriding a template function	تجاوز تابع القالب
Ostream	مجرى خرج
P	
Polymorphism	تعدد الأشكال
Privet	المحددات الخاصة
Public	المحددات العامة
Pointers	المؤشرات
Privet function and Variable	المتحولات والتوابع الخاصة
Processor	المعالج
Push character on stack	دفع محرف على المكس
Programming	البرمجة
Physical Layer	الطبقة المادية
PC (Program Counter)	عداد البرنامج PC
Pop character from stack	سحب محرف من المكس
Point	نقطة
Pointer To Objects	مؤشر الأغراض
PU(Process unit)	وحدة المعالجة
Pipelining	التوارد
Passed by reference	التمرير بالمرجع (العنوان)
Passed by Value	التمرير بالقيمة
Passed by reference with Pointer	التمرير بالمرجع بواسطة المؤشرات
Plotter	الراسمة

Performance	الأداء
Public Function	التوابع العامة
Private Function	التوابع الخاصة
Public Variables	المتحولات العامة
Private Variables	المتحولات الخاصة
Public Function	التوابع العامة
PC (Personal Computer)	الحاسوب الشخصي
Performance of CPU	أداء وحدة المعالجة المركزية
Print Hexadecimal value	طبع قيمة بالنظام السداسي عشر
Print Decimal value	طبع قيمة بالنظام العشري
Print Binary value	طبع قيمة بالنظام الثنائي
Print Octal value	طبع قيمة بالنظام الثماني
Parallel instruction executing	التنفيذ المتوازي للتعليمية
Public Inheritance	الوراثة العامة
Protected Inheritance	الوراثة المحمية
Private inheritance	الوراثة الخاصة
Parallel Processing	المعالجة المتوازية
Parallel Transmission	الإرسال المتوازي
Polymorphism Class	صف متعدد الأشكال
Pure Abstract Class	الصف المجرد النقي
Pure virtual functions	التوابع الافتراضية النقية
Pointer to start of list	مؤشر بداية القائمة
Pointer to end of list	مؤشر نهاية القائمة

Pointer to next item	مؤشر العنصر التالي
Pointers & Arrays	المؤشرات والمصفوفات
Private Networks	الشبكات الخاصة
Parallel Processing	المعالجة المتوازية
Protected Members	الأعضاء المحمية
Pointers To Pointers	مؤشرات المؤشرات
Q	
QOS (Quality of Service)	جودة الخدمة
Queue	رتل
R	
Reliability	الوثوقية
Reselection	إعادة الانتخاب
Real Addresses	العناوين الحقيقية
References	المراجع (العنونة)
Registers	السجلات
Random Errors	الأخطاء العشوائية
Retiring References	إعادة المراجع
Return a reference	إرجاع المرجع
RS- 232	دائرة لربط ملحقات الحاسوب عن

	طريق المنفذ التسلسلي
Return null	أرجع صفر
Register Renaming	إعادة تسمية السجل
Resource Conflict	نزاع الموارد
RISC (Reduce Instruction Set Computer)	الحواسيب ذات مجموعة التعليمات المختصرة
Reference Model	النموذج المعياري
Register to Register Operations	عمليات من سجل إلى سجل
Register File Details	مكونات ملف السجل
S	
SATA(Serial ATA)	واجهة الترابط التسلسلية
Serial Interface	واجهة الترابط التسلسلية
Structured Language	لغة بنوية
Software	البرمجيات
Stack	المكدس
Status	الحالة
Sub Channel	قنوات فرعية
Stack is empty	المكدس فارغ
Stack is full	المكدس ممتلئ
Static	ساكن

System Interconnection	الربط البيني للنظام
Static Variables	المتحولات الساكنة
Static Functions	التوابع الساكنة
Static objects	الأغراض الساكنة
Shared Memory	الذاكرة المشتركة
Standard Template Library	مكتبة القوالب القياسية
Simplicity	البساطة
Scaling	التدريج
Strings	السلاسل
Superscalar Processors	المعالجات السلمية الفائقة
Source	المصدر
Scheduling	الجدولة
Synchronization	التزامن
Shared Memory	الذاكرة المشتركة
Single Cycle Processor	المعالج وحيد الدورة
Switch	مبدل
System and Hardware Failures	فشل النظام والعتاديات
Sub tow objects	اطرح غرضين
SPR (Special Purpose Registers)	السجلات ذات الأغراض الخاصة
	شركة تقييم الأداء القياسي
Supercomputers	الحواسيب الفائقة

T	
Tasks	مهام
Target	الهدف
Template	القالب
Terminator	منهيات
Timer	مؤقت
Timing	التوقيت
The binary search	البحث الثنائي
Time sharing	المشاركة بالزمن
Throughput	الإنتاجية
Triangle	مثلث
TCP (Transmission Control Protocol)	بروتوكول التحكم بالنقل
Traffic	الازدحام
Transmitter	المرسل
Types	الأنماط (الأنواع)
U	
USB (Universal Serial Bus)	منافذ المسرى التسلسلي العام
UNIX	نظام تشغيل UNIX
V	
Vector	شعاع

Virtual Circuits	الدارات الافتراضية
Vector Processors	المعالجات الشعاعية
Virtual Functions	التوابع الافتراضية
Virtual Channel	القناة الافتراضية
Virtual Base Class	الصف الأساسي الافتراضي
Von Neumann Computer Model	نموذج حاسوب فان نيومان
Virtual Memory Organization	تنظيم الذاكرة الافتراضية
Visual BASIC	لغة بيزك المرئية
W	
Windows Systems	نظام تشغيل النوافذ
WWW(World Wide Web)	شبكة الويب العالمية
X	



اللجنة العلمية :

الدكتور المهندس جمال الياسين

الدكتور المهندس محمد طاهر الشاذلي

الدكتور المهندس روق خالك

المدقق اللغوي :

الدكتور صباح دلاش

حقوق

الطبع والنشر محفوظة لمديرية الكتب والمطبوعات الجامعية

