بسم الله الرحمن الرحيم

جامعة السّودان المفتوحة

برنامج علوم الحاسوب وتقنية المعلومات

أساليب البرمجة (2)

رمز المقرر ورقمه: حسب 1027

إعداد المادة العلمية: أ. د. محمد أحمد الأفندي

تصمـــيم تعلـــيمي : أ. سلمى مامون عبد النور

: أ. منال محمد بشير التنقاري

التدقيق اللغوي : أ. على محمد أحمد الصديق

التصميم الفني : مـــنى عثـــمان الـــننقة

منشورات جامعة السودان المفتوحة، الطبعة الأولى 2006

جميع الحقوق محفوظة لجامعة السودان المفتوحة، لا يجوز إعادة إنتاج أيّ جزء من هذا الكتاب، وبأيّ وجه من الوجوه، إلاّ بعد الموافقة المكتوبة من الجامعة.

مقدمة المقرر

بسم الله الرحمن الرحيم والصلاة والسلام على خاتم الأنبياء والمرسلين سيدنا محمد وعلى آله وصحبه أجمعين وبعد

عزيزي الدارس في الجزء الأول من هذا الكتاب ذكرنا أن لغة c++ لغة مزدوجة، توفر الأسلوبين: الإجرائي(procedural) والكياني (object-oriented)، وخصصنا **الجزء الأول** بكامله **للأسلوب الإجرائي**. حان الوقت **الآن** لدراسة **الأسلوب الآخر: البرمجة الكيانية** باستخدام لغة **C++**

هذا الجزء من الكتاب يتعرض لموضوع البرمجة الكيانية بشيءٍ من التفصيل. يبدأ الكتاب (بعد هذا الفصل) بدراسة **الفصائل** (classes) في c++ **الفصائل تمثل القوالب الأساسية التي تستخدم في تعريف الكيانات**، وهي **تشبه** إلى حد كبير "**السجلات**" **(structures)** التي استخدمناها في الجزء الأول.

بعد الفصائل نقوم بدراسة **التوارث (inheritance)**، و **التركيب (composition)** ، وهما طريقتان مستخدمتان في بناء الكيانات وتطويرها.

بعد دراسة التوارث و التركيب ننتقل إلى موضوع "**تعدد الأدوار**" **(polymorphism)** **و الدوال الإفتراضية (virtual functions)**، و التي تسمح بالتعديل "التحتي" للكيانات دون الحاجة إلى تغيير الواجهة التي يتعامل معها طلاب خدمة هذه الكيانات.

موضوع "تغيير دور العوامل و الدوال" يشبه إلى حد كبير موضوع تعدد الأدوار ولكنه يختلف عنه اختلافات بسيطة. هذا الموضوع نعرض له بعد دراسة "تعدد الأدوار".

بعد ذلك نعرض لمختلف المواضيع ذات العلاقة المباشرة و غير المباشرة مع البرمجة الكيانية. تشمل هذه المواضيع "**الحالات الاستثنائية**"، "**القوالب**"، "**نطاقات الأسماء**" **و** "**الأنواع المعرفة**".

كل الفصول مزودة بمجموعة من الأمثلة و التدريبات المنوعة التي تساعد في فهم المادة وأسئلة التقويم الذاتي حتى تتأكد من فهمك واستيعابك للمقرر. نتمنى لك عزيزي القارئ

وقتاً طيباً مع هذا الكتاب ونذكرك أن عملية البرمجة و تطوير نظم الحاسوب ماهي إلا عملية ممارسة عملية. احرص على كتابة أكبر قدر من البرامج حتى ترسخ المفاهيم الأساسية في ذهنك و تتهيأ للانطلاق إلى آفاق أوسع.

الأهداف العامة للمقرر

عزيزي الدارس، بعد دراستك لهذا المقرر نتوقع منك أن تكون قادراً على أن:

1. تطّبق المفاهيم الأساسية للبرمجة الكيانية.
2. تشرح الفصائل والكيانات والعلاقات بينها.
3. تقييم إسهام الإرث في التطور البرمجي.
4. تلخص تحميل الدوال والعوامل.
5. تعرف تعدد الادوار والربط المتأخر.
6. تستخدم تقنية الدوال التخيلية.
7. تصمم قوالب عامة للفصائل والدوال.
8. تحتاط للأخطاء المحتملة في البرنامج بإستخدام أوامرالإستثناء.
9. تحل مشكلة تضارب أسماء المتغيرات في البرنامج بإستخدام مجالات الأسماء.

محتويات المقرر

الرقم	الموضوع	الصفحة
1	مفاهيم أساسية في البرمجة الكيانية (الموضوعية)	1
2	الفصائل	9
3	الإرث والتركيب	63
4	تحميل الدوال والعوامل	123
5	تعدد الأدوار والدوال التخيلية	153
6	القوالب	173
7	مجالات الأسماء	205
8	الحالات الاستثنائية	217


الوحدة الأولى
مفاهيم أساسية في
البرمجة الكيانية (الموضوعية)

محتويات الوحدة

الموضوع	الصفحة
المقدمة	3
تمهيد	3
أهداف الوحدة	3
1. تعريف الكيان	4
1.1 خصائص بسيطة أو مركبة	4
2.1 الدوال	5
خلاصة	6
لمحة مسبقة عن الوحدة التالية	6
إجابات التدريبات	7
مسرد المصطلحات	7
المراجع	8

المقدمة

تمهيـــد

عزيزي الدارس، لاحظ الرواد الأوائل أنَّ البرنامج أو النظام الحاسوبي ما هو إلا تفاعل بين مجموعة كيانات مستقلة بصورة تؤدي إلى إنجاز المطلوب، وهذا ما سمي **بالبرمجة الكيانية(Objects Programming)**. إذا نجحنا في **التعرف** على هذه **الكيانات و تصميمها بصورة صحيحة** نكون قد **نجحنا** في **نصميم النظام** ككل . و هذا مابنى عليه الرواد فلسفتهم المبسطة في تصميم نظم الحاسوب.

أهداف الوحدة



بعد فراغك **عزيزي الدارس** من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. **تطبق** المفاهيم الأساسية للبرمجة الكيانية.
2. **تميز** بين الأسلوب الإجرائي والأسلوب الكياني.
3. **تعرّف** الكيان.

1. الكيان :Object

عزيزي الدارس، هب أنَّ مكتبةً من المكتبات طلبت منك تصميم نظام بلغة c++ لإدارة عملية بيع الكتب المدرسية في المكتبة، و أخبروك أنهم يرغبون أنْ يساعدهم البرنامج في الأمور التالية:

1- عند بيع كتاب يقوم النظام بخفض كمية المخزون من الكتاب، إضافة سعره لدخل اليوم، ثم إضافة ربح الكتاب إلى الأرباح.

2- عندما توشك الكمية المتوفرة من كتاب معين على النفاد، يذكر النظام إدارة المكتبة بضرورة طلب كمية إضافية.

.... إلى آخر ما يمكن أن يقوم به نظام من هذا النوع.

الملاحظة أن هذا النظام ينطوي و يتعامل مع **عدة كيانات (objects) فعلية و معنوية، بسيطة و مركبة مثل "كتاب"، "مخزون"، "كمية"، "سعر"، "ربح"** ...الخ. كل من هذه **الكيانات** له **خصائصه الخاصة** به و **يتفاعل مع الكيانات الأخرى لإنجاز المهام المطلوبة من البرنامج.**

حسب هذه الفلسفة يتضح لك أن **الكيان (object) عبارة عن وحدة مستقلة** تتكون من الآتي:

1.1 خصائص بسيطة أو مركبة

وهذه عبارة عن صفات، أو تعريفات تتعلق بالكيان يسهل تمثيلها كمتغيرات عادية أو مركبة. مثال ذلك اسم الكتاب، رقم الإصدار،المخزون، المؤلف، دار النشر ...الخ. كل هذه الخصائص تتعلق بالكتاب. كما تلاحظ أن بعض هذه الخصائص قد يمثل كيانات أخرى، فمثلاً. المؤلف، و دار النشر قد تكون كيانات أخرى مستقلة في البرنامج، ولكنها تستخدم هنا كخصائص للكتاب.

2.1 الدوال Functions

(عمليات و أساليب): يستخدمها الكيان في تغيير وتعديل الخواص، أو التفاعل مع الكيانات الأخرى في البرنامج. مثلاً: تعديل المخزون عند بيع نسخة من الكتاب يمكن أن تقوم به دالة يملكها الكيان (الكتاب في هذه الحالة). كذلك تعديل حجم المبيعات، طباعة الفاتورة، إلى غير ذلك من العمليات المطلوبة.

بناءً على ذلك تكون **الدوال و المتغيرات** في البرنامج موزعة بين الكيانات المكونة للبرنامج. **العبء الأساسي** في تصميم البرنامج هو **التعرف على هذه الكيانات من صيغة المسألة، و تحديد كيفية التفاعل بين هذه الكيانات.**

هذا الأسلوب في تصميم البرامج **يختلف جذرياً عن الأسلوب** "**الإجرائي**" **أو** "**المنهاجي**" **(procedural)** المؤسس على أن البرنامج ما هو إلا مجموعة إجراءات أو دوال تمثل الأوامر والخطوات التي ينبغي تنفيذها لإنجاز هدف البرنامج.

أسئلة تقويم ذاتي



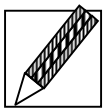
عرف ما يلي:

أ.البرمجة الكيانية؟

ب. الكيان ؟

ج. مما يتكون الكيان؟

تدريب(1)



قارن بين أسلوب البرمجة الكيانية والبرمجة "الإجرائية" أو "المنهاجية"

الخلاصة

عزيزي الدارس، أوضحنا في هذا الفصل مفهوم "الكيان" (object) وذكرنا حينها أن الكيان ينطوي على خواص (attributes) وعمليات (methods) تستخدم في معالجة الخواص. مجموعة الخواص يمكن تمثيلها كمتغيرات عادية، أو مركبة. أما العمليات فيتم تمثيلها كدوال أو إجراءات برمجية داخل الكيان.

لمحة مسبقة عن الوحدة التالية

بعد دراستك للوحدة الأولى وشرح المفاهيم الإساسية في البرمجة الكيانية في الوحدة التالية وهي بعنوان الفصائل سوف نضع النقاط على الحروف و نعرض بشيءٍ من التفصيل لكيفية تعريف الكيانات في لغة سي++ ، تعريف الكيانات باستخدام الفصائل ، مستويات حماية العناصر في الفصائل (protection) وأثرها على استخدام هذه العناصر ،استخدام الكيانات داخل البرنامج ودوال الإنشاء (constructors)، دورها في تعريف الفصائل، كيفية و شروط استخدامها ،دوال الإخلاء (الهدم) Destructors،المؤشرات والكيانات،مصفوفات الكيانات.

إجابات التدريبات

تدريب(1)

البرمجة الكيانية	البرمجة "الإجرائية" أو "المنهاجية"
الدوال و المتغيرات في البرنامج موزعة بين الكيانات المكونة للبرنامج. العبء الأساسي في تصميم البرنامج هو التعرف على هذه الكيانات من صيغة المسألة و تحديد كيفية التفاعل بين هذه الكيانات.	أن البرنامج ما هو إلا مجموعة إجراءات أو دوال تمثل الأوامر والخطوات التي ينبغي تنفيذها لإنجاز هدف البرنامج.

مسرد المصطلحات

- **البرمجة الكيانية(Objects Programming)**

تفاعل بين مجموعة كيانات مستقلة بصورة تؤدي إلى إنجاز المطلوب.

- **الكيان (object)**

عبارة عن و حدة مستقلة.

- **دوال functions**

هي عمليات و أساليب يستخدمها الكيان في تغيير و تعديل الخواص أو التفاعل مع الكيانات الأخرى في البرنامج.

المصادر والمراجع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.



الوحدة الثانية

الفصائل (Classes)

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**11**
تمهيد	**11**
أهداف الوحدة	**12**
1. تعريف الكيانات و الفصائل	**13**
2. مستويات حماية العناصر في الفصائل (protection)	**16**
3. استخدام الكيانات داخل البرنامج.	**17**
4. استخدام دالة الإنشاء	**23**
1.4 الخواص العامة لدوال الإنشاء	27
2.4 اسناد الكيانات بعضها الى بعض	28
5. دوال الإخلاء (الهدم)	**29**
6. المؤشرات و الكيانات	**37**
7. مصفوفات الكيانات	**40**
الخلاصة	**42**
لمحة مسبقة عن الوحدة التالية	**42**
إجابات التدريبات	**43**
مسرد المصطلحات	**60**
المراجع	**61**

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في أولى مراحل البرمجة الكيانية بلغة C++. هذا الفصل يمثل المدخل الأول و الأساس للبرمجة الكيانية، ويركز على كيفية تعريف الكيانات (objects) و كيفية التعامل معها داخل البرامج. كما سيتبين لك من مادة الفصل، أن C++ تستخدم فكرة الفصائل (classes) في تعريف أنواع الكيانات. الفصائل تشبه إلى حد كبير السجلات (structures) التي درسناها في أساليب البرمجة(1) ، فمن منطلق C++ الكيانات ماهي إلا نوع آخر من المتغيرات المركبة التي يختلف نوعها من تطبيق إلى آخر. لكي يتم تعريف هذه الكيانات داخل البرامج لابد من معرفة نوعها. تستخدم لغة C++ فكرة الفصيلة (class) في تعريف نوع الكيان. إذا أردنا أن نعرف كيان و استخدامه في برنامج، لابد أولاً من تعريف فصيلة (class) لهذا الكيان تحتوي على كل الخصائص و العمليات التي يستخدمها الكيان (مواصفات الكيان). بعد ذلك تستخدم هذه الفصيلة في تعريف الكيان مثل ما يتم تعريف المتغيرات العادية، بالإضافة إلى مواضيع جانبية أخرى و مجموعة من الأمثلة التطبيقية.. كذلك داخل هذه الوحدة تجد تدريبات وأسئلة تقويم ذاتي لها إجابات في نهاية الوحدة لتساعدك على الفهم والاستيعاب واسترجاع النقاط الرئيسية.

أهداف الوحدة



بعد فراغك عزيزي الدارس من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. **تعرَّف** الفصائل .
2. **تعرَّف** الكيانات باستخدام الفصائل.
3. **تناقش** مستويات حماية العناصر في الفصائل وأثرها على استخدام هذه العناصر.
4. **توضّح** الاستخدام السليم للكيانات داخل البرنامج.
5. **تستخدم** دوال الإنشاء .
6. **تصف** دور دوال الإنشاء في تعريف الفصائل، وكيفية شروط استخدامها.
7. **تستخدم** دوال الإخلاء (الهدم).
8. **تشرح** المؤشرات و الكيانات.
9. **تنشئ** مصفوفات الكيانات.

1. تعريف الفصائل و الكيانات

تعريف الكيان في لغة C++ يتم على **مرحلتين:**

1- **في المرحلة الأولى** يتم تعريف فصيلة تحتوي على المواصفات الكاملة للكيان.

2- **في المرحلة الثانية** يتم تعريف الكيان نفسه كما تعرف المتغيرات العادية.

المرحلة الأولى

الفصيلة (class) تشبه إلى حد كبير السجل (structure) الذي استخدمناه في الجزء الأول من هذا الكتاب لتعريف المتغيرات المركبة. الشيء الجديد الذي سنضيفه هذه المرة هو إمكانية تعريف دوال (methods) داخل الفصيلة وإمكانية تحديد مستوى الحماية (protection) لعناصر الفصيلة. عموماً يمكن تعريف الفصيلة كما يلي:

```
class class_name
  { [protection_level:]
        type attribute_1;
        type attribute_1;
        method1 defn;
        method2 defn;
          .
          .
          .
  };
```

كما يتضح من الهيكل العام أعلاه، **تعريف الفصيلة يبدأ بالكلمة الخاصة class متبوعة باسم الفصيلة، ثم مجموعة من الخواص attributes و الدوال methods**. مجموعة **الخواص و الدوال** تكون عادة **مسبوقة بكلمة خاصة (اختيارية) تدل على مستوى الحماية**. المقصود ب "**مستوى الحماية**" هنا **إمكانية الوصول إلى هذه الخواص و الدوال من خارج الفصيلة و استخدامها**. إذا كان مستوى **الحماية عالٍ(خاص)، فلا**

يمكن استخدام الخواص و الدوال **من خارج الفصيلة**. أما إذا كان مستوى **الحماية عادي (عام) فيمكن** استخدام الخواص و الدوال المعنية **من خارج الفصيلة**.

لكي تتضح الصورة، ويظهر المقصود من طريقة التعريف أعلاه نعطي مثالاً عاجلاً:

مثال (1)

في هذا المثال نعطي تعريفاً لفصيلة Book يمكن استخدامها لا حقاً في تعريف كيان Mybook:

أولاً: تعريف الفصيلة:

```
Class Book
{ private:
     string Title;
     string author;
     int ISBN;
     float price;
     int Quantity;
   public:
      void UpdateQuantity(int n);
      void PrintInvoice();
      void SellCopy();
};
```

الفصيلة Book أعلاه تنطوي على مجموعة من الخواص، و العمليات. **الخصائص هي: Title, author, ISBN, price, Quantity** و **العمليات هي: UpdateQuantity(), PrintInvice(), SellCopy()**.

لاحظ أن تعريف **الخواص** في الفصيلة مسبوق بالكلمة **private**. هذا يعني أن هذه الخواص **لا يمكن الوصول إليها** و استخدامها من خارج الفصيلة. المستوى **private هو مستوى الحماية الأعلى** في C++ مثلاً **لا يمكن تعديل سعر الكتاب (price) إلا باستخدام دوال معرفة داخل الفصيلة نفسها.**

بعد تعريف الخواص استخدمنا مستوى جديد public لتعريف عمليات الفصيلة.
المستوى **public هو المستوى الأدنى**(عام) من الحماية في C++ و يعني **إمكانية الوصول، و الإستخدام من أي فصيلة أخرى.**
يسري مفعول المستوى private حتى بداية المستوى public ، و يسري مفعول المستوى public حتى نهاية الفصيلة.

ملحوظة

تحديد مستوى الحماية لمجموعة من أعضاء الفصيلة (خواص أو عمليات) يتم باستخدام إحدى **الكلمات المحجوزة private، أو public، أو protected** قبل هذه الأعضاء. لاحظ أن استخدام **النقطة المزدوجة** ":" **بعد هذه الكلمات لازم.** يظل **مستوى الحماية** ساري المفعول حتى **نلغيه** باستخدام **مستوى مغاير.** مثلاً **مستوى الحماية private** في الفصيلة أعلاه **يظل ساري** المفعول **حتى نستخدم كلمة public** لبدء مستوى جديد من الحماية. C++ لا تفرض ترتيباً معيناً لمستويات الحماية داخل الفصيلة. **يمكن تكرار مستوى الحماية الواحد** أكثر من مرة **في الفصيلة.**

ثانياً تعريف الكيان داخل البرنامج:

تعريف الكيان يتم باستخدام **اسم الفصيلة متبوعاً باسم الكيان:**

```
int main()
{ ……
  Book MyBook; // a variable MyBook of type Book
  ……..
  ……..
}
```

في البرنامج أعلاه استخدمنا **الفصيلة Book** في تعريف **كيان (متغير)** اسمه **Mybook**. لاحظ أن **تعريف الكيانات يشبه تعريف المتغيرات العادية.**

2. مستويات حماية العناصر في الفصائل (Pretection)

هناك **ثلاثة مستويات للحماية** في C++سوف نعرض لها في الجدول أدناه:

جدول يوضح مستويات الحماية الثلاثة في C++

مستويات الحماية	الدلالة عليه في تعريف الفصيلة بالكلمة المحجوزة	الوصول للمتغيرات والدوال
المستوى الأعلى (الخاص)	Private	لا يمكن الوصول إليها و استخدامها من خارج الفصيلة إلا في حالات استثنائية
المستوى وسيط	Protected	لا يمكن الوصول لها إلا من فصائل لها علاقة خاصة بالفصيلة المالكة
المستوى الأدنى (العام)	public	مشاعة للجميع ويمكن الوصول إليها من أي فصيلة أخرى.

ملحوظة

إذا لم يذكر مستوى الحماية في تعريف بعض الخواص أو الدوال داخل الفصيلة، فإن مستوى الحماية الافتراضي في C++ هو المستوى الأعلى private.

مثال (2)

في المثال أدناه نقوم بتعريف **فصيلة** اسمها **Rectangle** (مستطيل). تنطوي الفصيلة على **خاصتين: width, height** بالإضافة إلى **دالتين: SetSides(float,float) و area(void).**

```
class Rectangle {
    float width, height;
  public:
    void SetSides (float,float);
    float area (void) {return (width*height);}
};
```

لاحظ أن **مستوى الحماية للدالتين: ()SetSides و ()area هو public (المستوى المشاع)**. هذا يعني أن الدالتين **يمكن استخدامهما من خارج الفصيلة.**

بما أن **مستوى الحماية للخاصتين width و height لم يذكر، فهذا يعني أن مستوى الحماية لهما هو private** وبالتالي **لا يمكن** استخدام هاتين الخاصتين **من خارج الفصيلة.** أي أنه **لا يمكن إسناد قيم** لهما أو **قراءة قيمهما من خارج الفصيلة.**

3. استخدام الكيانات داخل البرامج

مثال(3)

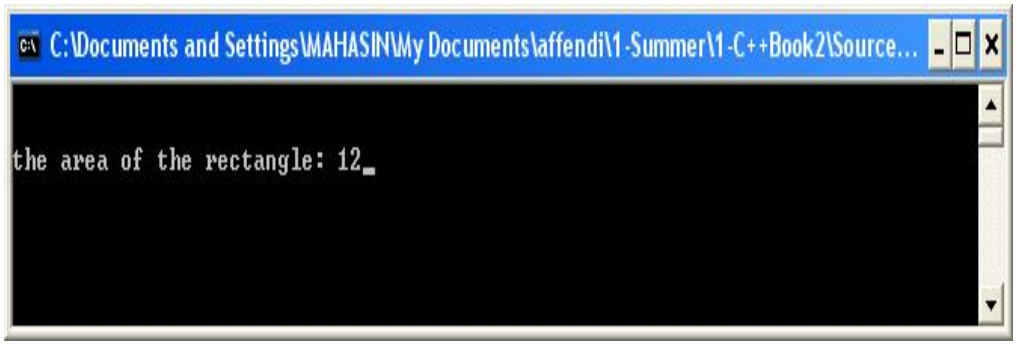
المثال أدناه عبارة عن برنامج صغير **يستخدم الفصيلة Rectangle** في **تعريف كيان MyRect**، ثم **يحسب و يطبع مساحة الكيان:**

برنامج(1)

```
// classes example
#include <iostream.h>
class Rectangle {
    float width, height;
  public:
    void SetSides (float,float);
    float area (void) {return (width*height);}
};
void Rectangle::SetSides (float w, float h)
{
  width = w;
```

```
  height = h;
}
int main ()
{ float RectArea;
  Rectangle MyRect; //declare an object of
type Rectangle
  MyRect.SetSides (3.0,4.0); //set the width
and height values
  RectArea = MyRect.area;
  cout << "area: " << RectArea;  //compute and
print the area
  cin >>r;
}
```

عينة من تنفيذ البرنامج:



ملاحظات على البرنامج

1- الملاحظة الأولى هي أن **الفصيلة تم تعريفها خارج البرنامج ()main**، وهذا هو **الأفضل** و التقليد المتبع في معظم البرامج.

2- **تعريف الفصيلة ليس إلا وصفاً للفصيلة، و لا أثر له على الذاكرة**، مثله مثل الأنواع المستخدمة في تعريف المتغيرات (int, float …etc.). من هذا المنطلق **لا يمكن استخدام اسم الفصيلة في عمليات الحوسبة** (إلا في حالات خاصة نذكرها فيما بعد). **الفصيلة تستخدم في تعريف كيانات (Objects)، و الكيانات هي التي تستخدم في عمليات الحوسبة.**

3- **الدالة ()SetSides** لم يتم تعريفها بالكامل داخل الفصيلة، إنما أعطي **عنوانها فقط داخل الفصيلة**، ثم تم **تعريف الدالة** بكاملها **خارج الفصيلة**. أما **الدالة ()area فقد تم تعريفها بالكامل داخل الفصيلة. لغة C++ تعطي هذين الخيارين**، أي بالإمكان تعريف الدوال الأعضاء داخل الفصيلة بالكامل، أو تعريف العنوان فقط داخل الفصيلة، ثم إعطاء التعريف الكامل خارج الفصيلة. **الخيار الأول** يستحدم عادةً في حالة **الدوال صغيرة الحجم**، و **الخيار الثاني** يستخدم في حالة **الدوال الكبيرة. الدوال المعرفة بالكامل داخل الفصيلة تسمى الدوال الداخلية (Inline Methods)**.

4- عند **تعريف دالة خارج الفصيلة**، لابد من **نسبتها إلى الفصيلة الأم باستخدام علامة النطاق** "::" **(نقطتان مزدوجتان)**. في البرنامج المذكور يبدأ **التعريف الكامل للدالة ()SetSides خارج الفصيلة** بالسطر:

Recatngle::SetSides(float w, float h)

لاحظ استخدام علامة النطاق "::".

5- **يبدأ البرنامج ()main بتعريف كيان MyRect من نوع Rectangle**. طريقة تعريف الكيانات تتبع نفس الأسلوب المستخدم في تعريف المتغيرات العادية، مع اختلافات بسيطة تظهر عما قليل. **تم تعريف الكيان MyRect**

6- **بذكر الفصيلة (النوع) أولاً، ثم ذكر الكيان:**

Rectangle MyRect;

7- بعد تعريف الكيان، قام البرنامج **بإسناد قيم لعرض و طول الكيان (المستطيل) المعرف باستخدام الدالة ()SetSides:**

Rectangle.SetSides(3.0, 4.0);

لاحظ أن البرنامج في هذه الحالة **استخدم النقطة العادية** "." **في نسبة الدالة إلى الكيان المالك**. رمز النسبة إلي الكيان يختلف من رمز النسبة إلي الفصيلة (علامة النطاق). **تنسب العناصر الأعضاء إلى الفصيلة باستخدام علامة**

النطاق "::"، أما في حالة الكيانات، فتنسب العناصر الأعضاء باستخدام النقطة العادية "." .

8- **لا يمكن إسناد قيم لعرض و طول الكيان مباشرةَ داخل البرنامج**، ذلك أن **مستوى الحماية** لهذين العنصرين هو **المستوى الأعلى (private)**. **الطريقة الوحيدة لإسناد قيم لهذين العنصرين هو استخدام الدالة التي وفرتها الفصيلة.**

MyRect.SetSides (3.0,4.0);

لا يمكن استخدام جمل مثل:

MyRect.width=3.0; MyRect.height=4.0;

في البرنامج إلا إذا غيرنا مستوى الحماية إلى المستوى الأدنى (public).

9- بعد ذلك قام البرنامج **بحساب مساحة الكيان** (المستطيل) **باستخدام الدالة العضو area**. وكما هو متوقع، لابد من **نسبة الدالة إلى الكيان باستخدام النقطة "."** كالتالي:

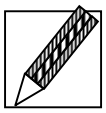
RectArea=MyRect.area();

أسئلة تقويم ذاتي



1- كيف يمكن تعريف الفصائل classes في لغة C++؟
2- هل يمكن تعريف الفصائل داخل البرنامج main()؟
3- ما الفرق بين الفصيلة class و الكيان object؟
4- ماهي مستويات الحماية لعناصر الفصيلة في C++ وما مغزى كل منها؟
5- ما الهدف من إعطاء مستويات مختلفة للحماية داخل الفصيلة؟

تدريب (1)



1- اكتب تعريفاً لفصيلة يمكن استخدامها في حساب مساحة و محيط مثلث' علماً بأن مساحة المثلث يمكن حسابها باستخدام العلاقة:

Area = SQRT(S(S-A)(S-B)(S-C))

حيث S هي:

S = (A+B+C)/2

والعناصر A, B, C هي أطوال المثلث.

2- استخدم الفصيلة المعرفة في السؤال أعلاه في برنامج يقوم بحساب مساحة ومحيط مثلث تؤخذ أطواله من المستخدم.

استخدم مستويات الحماية المناسبة لعناصر الفصيلة.

الفوائد المجنية من استخدام مستويات حماية مختلفة(شكل نقاط)

الهدف من استخدام **مستويات حماية مختلفة** في لغات البرمجة الكيانية هو **الفصل بين الإستخدام و تفاصيل التصميم.** كثير من مستخدمي الكيانات لا يحتاجون إلى معرفة تفاصيل تصميم هذه الكيانات و الخوارزميات المستخدمة في تحقيقها، بل من الأفضل ألا يعرفوها. من هذا المنطلق يمكن استخدام **المستوى العام public** في إعطاء واجهات تسمح لمستخدم الكيان **بالإستفادة التامة من الخدمات المعطاة**، بينما يستخدم **المستوى private** في **إخفاء تفاصيل كيفية تقديم هذه الخدمات.** من **فوائد هذا الفصل** بين الواجهات و تفاصيل التصميم، أن **مطور الكيان قد يغير تفاصيل التصميم و تحقيقها دون أن تتأثر الواجهات.**

فائدة أخرى من فوائد مستويات الحماية، هي أنه بالإمكان **حماية بعض المتغيرات الحساسة في الفصيلة وعدم السماح بالوصول إليها من الخارج الفصيلة.** مثال ذلك **كلمات السر** في نظم التشغيل، **حسابات العملاء** في المصارف، **درجات الطلاب** في الجامعات ...الخ.

أسئلة تقويم ذاتي

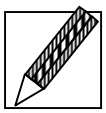


1- ماهي الأخطاء في تعريف الفصائل التالية:

a.
```
class Tes1
{ privae:
    int x,y;
  public
   int Test1( );
    void SetValues(a=0, b=0)
    {x=a; y=b;}
     product ( )
     { return x*y }
  };
```

b.
```
class Test2
{ private:
     float p,q,r;
      int m,n;
   public:
     Test2(float, float);
      Test2(int, int);
     SetFloatValues(float, float, float)
const;
      SetintValues( int, int);
   }
```

تدريب(2)



1.أكمل تطوير الفصيلة Book التي ابتدأنا تعريفها في بداية الفصل:

```
Class Book
{ private:
    string Title;
    string author;
    int ISBN;
    float price;
    int Quantity;
  public:
     void UpdateQuantity(int n);
     void PrintInvoice();
     void SellCopy();
     };
```

2- أكتب تفاصيل الدوال المذكورة في الجزء العام من الفصيلة. أضف دالتي إنشاء لفصيلة Book أعلاه، إحداهما تسند قيم ابتدائية للأعضاء Title, author, ISBN والأخرى تسند قيم ابتدائية للأعضاء Price و Qunatity.

3- أكتب برنامجاً كاملاً تختبر فيه عناصر الفصيلة Book.

4. استخدام دالة الإنشاء Constructor Function

في المثال السابق استخدمنا الدالة SetSides() لإسناد قيم ابتدائية لطول و عرض الكيان المعرّف (المستطيل). **القيم الإبتدائية مطلوبة عادةً في تعريف الكثير من الكيانات وهي ظاهرة عامة في البرمجة الكيانية.** مثلاً عند فتح **حساب جديد** في مصرف تكون **القيمة الإبتدائية** للحساب إما **صفراً أو القيمة التي يودعها الزبون لحظة فتح الحساب.** عند **تسجيل طالب جديد** في الجامعة، تكون **القيمة الإبتدائية للمعدل التراكمي صفراً** ...الخ. من هذا المنطلق **وفرت لغة C++ طريقة أسهل لإسناد القيم الإبتدائية في الكيانات.**

هذه الطريقة ، هي استخدام دالة خاصة تسمى دالة الإنشاء Constructor. دالة الإنشاء تأخذ إسم الفصيلة (اسم الدالة هو اسم الفصيلة المالكة) مع إمكانية تمرير عناصر (اختيارياً).

ملحوظة

C++ تقوم بإدراج **دالة إنشاء إفتراضية** مع كل فصيلة. هذه الدالة الافتراضية لاتقوم بعمل شيء ولاتستقبل أي عناصر ممررة وتستدعى لحظة استخدام الفصيلة في تعريف كيانات. هذه الدالة تستخدم لحفظ المكان فقط، حتى ينتظم نسق البرمجة الكيانية ويمكن إعادة تعريفها وتكرارها في الفصيلة الواحدة بشروط معينة.

دالة الإنشاء الافتراضية يمكن إعادة تعريفها وتكرارها أكثر من مرة داخل الفصيلة الواحدة بنفس الاسم بشرط أن يختلف عدد العناصر الممررة من صورة إلى صورة، وذلك بغرض إعطاء مرونة كافية في استخدام الفصيلة.

دالة الإنشاء الإفتراضية التي **تدرجها C++ مع الفصيلة Rectangle** هي:

```
Rectangle() {}
```

هذه الدالة يمكن إعادة تعريفها لتقوم بعمل الدالة SetSides() كما يلي:

```
class Rectangle {
    float width, height;
  public:
    Rectangle (float,float); //using a
constructor function
    float area (void) {return (width*height);}
};
Rectangle::Rectangle (float w, float h)
//defining the constructor
 {
  width = w;
  height = h;
}
```

الفائدة التي نجنيها في هذه الحالة هي **أن دالة الإنشاء يتم استدعاؤها تلقائياً عند تعريف كيان تابع للفصيلة** وبالتالي **لا نحتاج لاستخدام دالة أخرى. البرنامج** أدناه يوضح ذلك:

مثال (4)



برنامج(2)

```
#include <iostream.h>

class Rectangle {
    float width, height;
  public:
    Rectangle (float,float); //using a
constructor function
    float area (void) {return (width*height);}
};

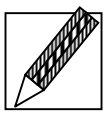
Rectangle::Rectangle (float w, float h)
//defining the constructor
 {
  width = w;
  height = h;
}
int main () {
  int r;
  Rectangle MyRect(3.0,4.0); //declare an
object of type Rectangle
  cout << "area: " << rect.area();  //compute
and print the area
  cin >>r;
}
```

لاحظ أن البرنامج قام بتعريف الفصيلة بالصورة التالية:

Rectangle MyRect(3.0,4.0);

وهذا يعني استدعاء دالة الإنشاء بالصورة Rectangle(3.0,4.0) وبالتالي إسناد القيم 3.0 و 4.0 للعرض و الطول على التوالي دون الحاجة لاستخدام دالة أخرى.

تدريب(3)



ترى ما الذي يحدث إذا تم تعريف الكيان في البرنامج أعلاه بدون تمرير عناصر: Rectangle MyRect; ؟ علل؟ ما الحل؟

تعدد دوال الإنشاء مسموح به في C++ **بحيث تختلف في عدد العناصر الممررة.**

تعريف الفصيلة الجديد يصبح:

```
class Rectangle {
    float width, height;
  public:
    Rectangle() { }  // constructor 1, without
parameters
    Rectangle (float,float); //constructor 2
with parameters
    float area (void) {return (width*height);}
};

Rectangle::Rectangle (float w, float h)
//defining the constructor
 {
  width = w;
  height = h;
}
```

1.4 الخواص العامة لدوال الإنشاء

الخواص العامة التي تميز دوال الإنشاء عن غيرها من الدوال يمكن تلخيصها في الجدول التالي:

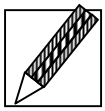
دالة الإنشاء	الخاصية
الاسم	اسم الفصيلة
النوع	دالة الإنشاء **ليس لها نوع** و يمكن أن **توصف بأي صفة**، حتى **كلمة void لا يمكن استخدامها في وصف دوال الإنشاء.**
الاستخدام	يمكن **استخدام أكثر من دالة إنشاء للفصيلة الواحدة، تشترك** كلها في الإسم **(اسم الفصيلة)**، بشرط أن **تختلف** عن بعضها البعض في **عدد العناصر الممررة.**
الاستدعاء	يتم **استدعاء الدالة تلقائياً** عند **تعريف دالة تابعة للفصيلة.** في هذه الحالة يقال أن **الكيان دخل نطاق الفصيلة.**

أسئلة تقويم ذاتي



1- ماهي دالة الإنشاء ومتى يتم استدعاؤها؟
2- مااسم دالة الإنشاء للفصيلة Book التي عرفناها في المثال ؟
3- عدِّل دالة الإنشاء الإفتراضية للفصيلة Book بحيث يصير السعر price صفر، و المخزون صفر لحظة تعريف الكيان.
4- هل يمكن إعطاء أكثر من دالة إنشاء للفصيلة الواحدة؟

تدريب(4)



1. قم بتعريف فصيلة car تحتوي على الخصائص و العمليات التالية:

الخصائص:

نوع السيارة CarType ، الموديل Model ، بلد المنشأ Country

رقم الشاسيه EngineNo ، سنة التصنيع Year

العمليات:

حساب متوسط استهلاك السيارة ()MilesPerGallon

طباعة معلومات السيارة ()PrintInfo

2. اكتب ثلاثة دوال إنشاء للفصيلة car أعلاه، الأولى لاتستقبل أي عناصر و تقوم بإعطاء قيمة افتراضية (لاشيء أو صفر) لكل الخصائص. الثانية تستقبل عنصراً واحداً هو سنة التصنيع وتسند القيمة المعطاة إلى سنة التصنيع. الثالثة تستقبل ثلاثة عناصر: سنة التصنيع، الموديل، رقم الشاسيه وتستخدم العناصر الممررة في تغيير القيم المقابلة.
3. اكتب برنامجاً متكاملاً تختبر فيه الفصيلة التي كتبتها.

2.4 إسناد الكيانات بعضها إلى بعض (object assignment)

العمليات الحسابية العادية لا يمكن تطبيقها مباشرةً على الكيانات، إلا إذا أعيد تعريفها كما سنرى فيما بعد. **العملية الوحيدة التي يمكن تطبيقها مباشرةً على الكيانات هي عملية الإسناد. C++ تسمح بإسناد الكيانات بعضها إلى بعض، إذا كانت من نفس النوع. مثلاً: إذا عرفنا كيانين Rect1 وRect2 فيمكن إسناد بعضهما إلى بعض كما في المثال التالي:**

```
Rectangle Rect1, Rect2;
Rect1.SetSides(5,4);
Rect2 = Rect1;
```

عملية الإسناد تنطبق فقط على الخواص البيانية. في الحالة أعلاه **يتم إسناد طول وعرض الكيان Rect1 إلى مايقابلها في Rect2.**

5. دوال الإخلاء (الهدم) Destructors Functions

الكيانات تستخدم الكثير من **موارد النظام** بعد إنشائها. هذه الموارد قد تكون **مساحات من الذاكرة، ملفات معينة، أجهزة بعينها** ...الخ. وبما أن **البرامج الكيانية قد تنشيء عدداً كبيراً من الكيانات أثناء تنفيذها، فإن أداء هذه البرامج قد يتدنى بصورة كبيرة إذا لم نتبع أسلوباً اقتصادياً في استخدام هذه الكيانات.**

القاعدة العامة الأساسية في استخدام الكيانات:

هي ألا تنشيء كياناً إلا إذا احتجت إليه، وأن تخلي موارد النظام التي يستخدمها هذا الكيان بمجرد الإنتهاء من استخدامه.

عزيزي الدارس: تعرضنا في الفقرات السابقة لعملية إنشاء الكيانات ودوال الإنشاء التي تستدعى لحظة الإنشاء. C++ تسمح باستخدام **دوال إخلاء تستدعى عند الإنتهاء من استخدام الكيان. دالة الإخلاء لأي فصيلة تأخذ إسم الفصيلة مسبوقاً بعلامة التلدا (~). مثلاً، دالة الإخلاء للفصيلة Rectangle هي:**

~Rectangle()

C++ تستخدم دالة إخلاء افتراضية تستدعى تلقائياً عند خروج الكيان من من نطاقه. سنكتفي بهذه الدالة الافتراضية في المرحلة الحالية، على الرغم من أنه بالإمكان تصميم دوال إخلاء خاصة.

مثال (5)

برنامج **يقرأ و يطبع الخواص المتعلقة بشخص معين Person.** يتم تمثيل **الشخص بفصيلة Person** لها عدة **خصائص (attributes).**

برنامج(3)

```
#include <iostream>
#include <stdlib.h>
using namespace std;
class Person
{private:
   string FirstName;
   string LastName;
   string Name;
   int Age;
 public:
   Person();
   void SetFirstName(string);
   void SetLastName(string);
   string GetFirstName(string &);
   string GetLastName(string &);
   void PrintName();
};
Person::Person()
{ FirstName=""; LastName="";
}
void Person::SetFirstName(string Fname)
{ FirstName = Fname;
}
void Person::SetLastName(string Lname)
{ LastName= Lname;
}
string Person::GetFirstName(string &Fname)
{ Fname = FirstName;
  return FirstName;
}
string Person::GetLastName(string &Lname)
{ Lname = LastName;
  return LastName;
}
```

```
void Person::PrintName()
{ cout <<FirstName<<"    "<<LastName<<endl;
}

int main()
{ Person P;
  string FName, LName;
  cout <<"please enter your first name\n?";
  cin >>FName;
  cout <<"Please enter your last name\n?";
  cin >>LName;
    P.SetFirstName(FName);
    P.SetLastName(LName);
    cout<<"Your full name is :";
    P.PrintName();
    system("pause");
    return 0;
}
```

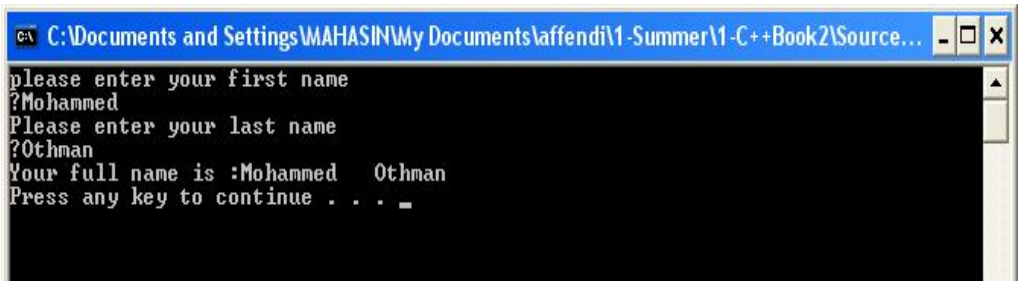
ملاحظات على البرنامج:

1- فصيلة Person **تحتوي** على أربع خواص وخمسة عمليات، بالإضافة لدالة الإنشاء التي أعيد تعريفها.

2- كل **العمليات تم تعريفها خارج الفصيلة، على الرغم من** أنه كان **بالإمكان تعريفها داخل الفصيلة**، و**الهدف** هو توضيح أن هذا **الخيار متاح في لغة** C++

3- استخدمنا **علامة النطاق** "::" **في تعريف الدوال خارج الفصيلة.**

عينة من تنفيذ البرنامج:



مثال(6)

برنامج **يحسب جذور معادلة جبرية من الدرجة الثانية:**

الفرق بين هذا المثال و الأمثلة الأخرى أننا نتعامل في هذه الحالة مع **فصيلة تمثل نوعاً من المعادلات و كيفية حلها. الفصيلة Quadratic في هذه الحالة تحتوي على معاملات المعادلة و الدالة التي تستخدم في حساب جذور المعادلة،** و**هذه الفصيلة** مثال للكثير من مكونات مكتبات لغات البرمجة الكيانية. وتجد معظم الخوارزميات في هذه المكتبات تصمم في شكل فصائل.



برنامج(4)

```
#include <iostream.h>
class Quadratic
{ private:
    double A, B, C;
    double root1, root2;
    double term;
  public:
    Quadratic(); //the constructor;
    void SetCoefs(double, double, double);
    void ComputeRoots();
    void PrintRoots();
 };
  Quadratic::Quadratic()
  {A=0; B=0; C=0; x=0; root1=0; root2=0;}
void Quadratic::SetCoefs(double C1, double C2, double
C3)
{ A=C1; B=C2; C=C3; }
void Quadratic::ComputeRoots()
{ term=sqrt(B*B-4*A*C);
  root1= (-B+term)/2;
  root2= (-B - term)/2;
 }
 void Quadratic::PrintRoots()
 { cout <<"root1 of the equation is "<<root1<<endl;
   cout <<"root2 of the equation is "<<root2<<endl;
 }
```

```
int main(int argc, char* argv[])
{ Quadratic Q;
double A,B,C;
  cout << " Please enter the coeficients of  Q: \n?" ;
  cin >>A>>B>>C;
  Q.SetCoefs(A,B,C);
  Q.ComputeRoots();
  Q.PrintRoots();
  return 0;
}
```

ملاحظات على البرنامج:

ليس في البرنامج شيءٌ جديد **يختلف عن سابقاته**، إلا أن البرنامج يقوم **بإنشاء كيان Q من النوع Quadratic ثم يستخدم دوال هذا الكيان في حساب جذور معادلة معطاة:**

```
Q.SetCoefs(A,B,C);
 Q.ComputeRoots();
 Q.PrintRoots();
```

مثال (7)

في هذا المثال نصمم **فصيلة Timer يمكن استخدامها في توقيت العمليات أو تسجيل الزمن لبعض الأحداث. الفصيلة غنية بالدوال** التي يمكن استخدامها في **تعديل الزمن أو قراءته.**

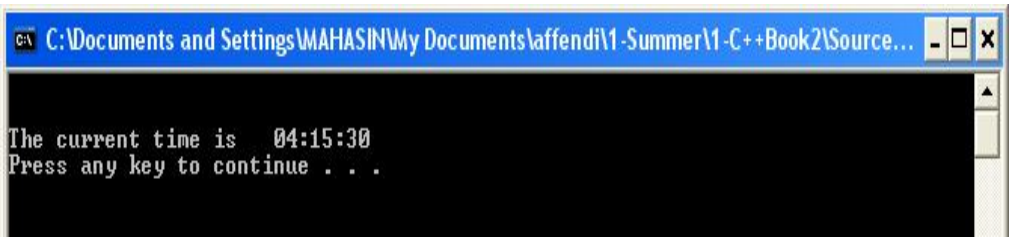
برنامج(5)

```
#include <iostream.h>
class Timer
 { private:
     int hr;
     int min;
     int sec;
   public:
    void SetTime(int, int, int);
    void GetTime( int & , int &, int &) const;
    void PrintTime() const;
    void IncrementSeconds();
    void IncrementMinutes();
```

```
    void IncrementHours();
    void EqualTime(const Timer &OtherTimer)
const;
};
void Timer::SetTime(int hours, int minutes,
int seconds)
{ if (hours>=0 && hours<24)
    hr = hours;
  else hr=0;
  if (minutes>=0 && minutes<60)
    min = minutes;
  else min=0;
  if (seconds >=0 && seconds <60)
     sec = seconds;
  else sec=0;
}
void Timer::GetTime(int &hours, int &minutes,
int &seconds) const
{   hours=hr;  minutes=min; seconds=sec;}
void Timer::PrintTime() const
{ if (hr<10)
    cout <<"0";
  cout <<hr<<":";
  if (min<10)
    cout <<"0";
  cout <<min<<":";
  if (sec<10)
     cout <<"0";
   cout <<sec;
}
void Timer::IncrementHours()
{   hr++;
    if (hr>23) hr=0;
}
void Timer::IncrementMinutes()
```

```
{    min++;
     if (min>59) min=0;
}
void Timer::IncrementSeconds()
{    sec++;
     if (sec>59) sec=0;
}
int main () {
  Timer T;
  T.SetTime(4,15,30);
  cout <<endl<<endl<<"The current time is    ";
  T.PrintTime();
  cout<<endl;
  system("pause");
  return 0;
}
```

عينة من تنفيذ البرنامج:



ملاحظات على البرنامج:

1- لاحظ **أن الدالة GetTime() تستخدم عناصر مرجعية (reference parameters) لإرجاع الناتج إلى المستخدم**. هذا النوع من العناصر عرضنا له في الجزء الأول من الكتاب.

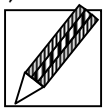
2- **أضفنا وصف const في تعريف الدالة GetTime()**، وهذا يعنى أن **الدالة GetTime() لايسمح** لها **بتعديل** أيٍ من **خواص الفصيلة Timer**. **الدوال الثابتة (const) يسمح لها** فقط بقراءة خواص الفصيلة و **لا يسمح لها** بتعديلها.

نشاط(1)



إذا استخدمنا الفصيلة Timer في المثال ... أعلاه في تعريف كيانين:
Timer Clock1, Clock2; ، عزيزي الدارس فكر واستنبط أثر عملية الإسناد: Clock1=Clock2 على الكيان Clock1؟

تدريب(5)



1. محل تجاري يريد أن يبني تطبيقاً بسيطاً يحسب حجم المبيعات اليومي، و يحسب نسبة الربح في هذه المبيعات. طلب منك أن تصمم فصيلة DailySales تسمح لهذا المحل بتعديل حجم المبيعات اليومي، و حساب نسبة الربح (باعتبار أن سعر الشراء لكل سلعة معروف). قم بتصميم هذه الفصيلة و اختبارها. الفكرة العامة هي إعطاء دوال تقوم بـ**تعديل عناصر الفصيلة** و حساب المطلوب.
2. جامعة معينة تريد أن تبني تطبيقاً بسيطاً يقوم بحساب عدد الفصول المتبقية لتخرج كل طالب، باعتبار أن الحد الأقصى لعدد ساعات المقرر في كل فصل هو 18. طلب منك تصميم فصيلة RMSemesters يمكن استخدامها في تعديل عدد الساعات التي أكمل الطالب دراستها، وذلك بعد ظهور نتائج كل فصل، ثم حساب العدد التقريبي للفصول المتبقية. قم بتصميم هذه الفصيلة و الدوال المطلوبة، ثم اختبارها.

6. المؤشرات و الكيانات

من منطلق C++ **الكيانات ما هي إلا نوع آخر من المتغيرات المركبة.** بناءً على ذلك **يمكن التعامل معها بصورة تشبه تعاملنا مع المتغيرات الأخرى** بما في ذلك **استخدام المؤشرات في تعريفها و معالجتها.** البرنامج التالي يشرح بصورة مبسطة كيف يمكن استخدام المؤشرات في تعريف الكيانات.



برنامج (6)

```
#include <iostream>
using namespace std;
class Rectangle {
    float width, height;
  public:
    void SetSides (float, float);
    float area (void) {return (width * height);}
};
void Rectangle::SetSides (float a, float b)
{
  width = a;
  height = b;
}
int main ()
{
  Rectangle a, *b, *c;
  Rectangle * d = new Rectangle[2];
  b= new Rectangle;
  c= &a;
  a.SetSides (1,2);
  b->SetSides (3,4);
  d->SetSides (5,6); //setting the sides of d[0]
  d[1].SetSides (7,8);
  cout << "a area: " << a.area() << endl;
```

```
  cout << "*b area: " << b->area() << endl;
  cout << "*c area: " << c->area() << endl;
  cout << "d[0] area: " << d[0].area() << endl;
  cout << "d[1] area: " << d[1].area() << endl;
  system("pause");
  return 0;
}
```

ملاحظات عامة على البرنامج:

1- في **السطر الأول** من البرنامج **(بعد عنوان الدالة ()main)** قمنا **بتعريف متغير a من نوع Rectangle و مؤشرين b و c إلى كيانات من نوع Rectangle:**

Rectangle a, *b, *c;

2- في **السطر الثاني** من البرنامج قمنا **بتعريف مؤشر d يشير إلى بداية مصفوفة** من الكيانات تتكون من كيانين وقمنا **بإنشاء المصفوفة لحظة التعريف** (كان بالإمكان إرجاء عملية الإنشاء إلى وقتٍ لاحق):

Rectangle * d = new Rectangle[2];

3- في **السطر الذي يلي ذلك** قمنا **بإنشاء الكيان الذي يشير إليه المؤشر b.** لاحظ أن **تعريف المؤشر لايعني إنشاء الكيان**، و لابد من **استخدام العامل new متبوعاً بنوع الكيان** كما تم في هذا السطر:

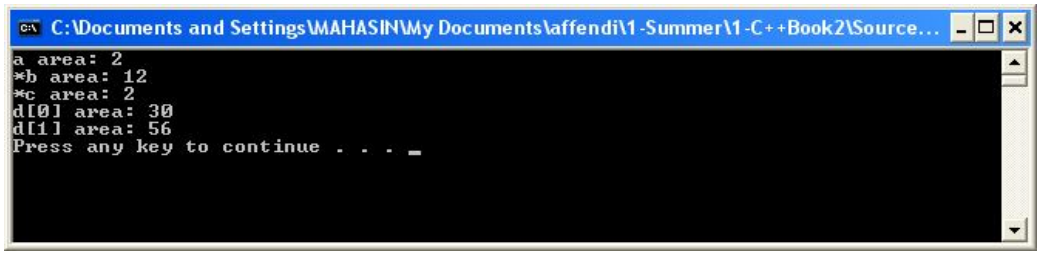
b= new Rectangle;

4- بعد ذلك **أسندنا عنوان الكيان a إلى المؤشر c الذي عرفناه في صدر البرنامج: c=&a;.** تذكر أن العلامة **& هي عامل العنوان.** بموجب هذه العملية أصبح **المؤشر c الآن يشير إلى موقع الكيان a في الذاكرة.**

5- بعد ذلك **أسندنا بعض القيم إلى أبعاد المستطيلات المختلفة**، ثم **حسبنا مساحة بعض هذه المستطيلات.** لاحظ أننا **استخدمنا النقطة العادية في نسبة العناصر إلى الكيان a**، ذلك أنه كيان عادي. أما **في حالة المؤشرات b,c,d** فإننا **استخدمنا المعامل ←** في **التعامل** مع **أعضاء الكيانات المشار إليها:**

```
  a.SetSides (1,2);
   b->SetSides (3,4);
   d->SetSides (5,6); //setting the sides of
d[0]
   d[1].SetSides (7,8);
   cout << "a area: " << a.area() << endl;
   cout << "*b area: " << b->area() << endl;
   cout << "*c area: " << c->area() << endl;
   cout << "d[0] area: " << d[0].area() <<
endl;
   cout << "d[1] area: " << d[1].area() <<
endl;
```

عينة من تنفيذ البرنامج:



7. مصفوفات الكيانات Arrays of objects

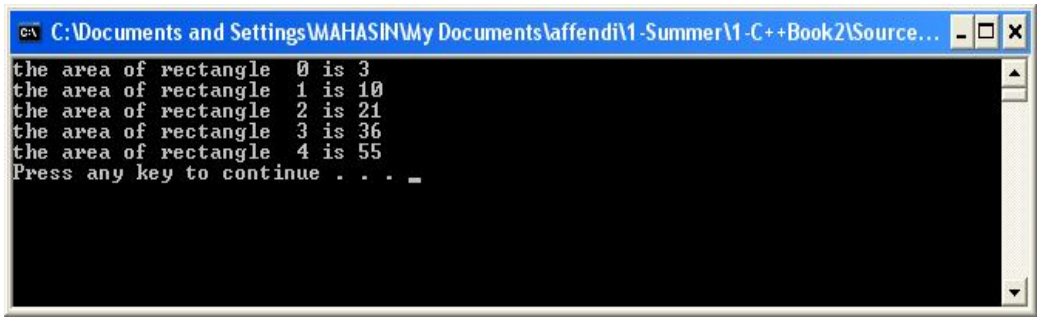
يمكن أيضاً **إنشاء مصفوفات عادية من الكيانات مثلما** نفعل في حالة **المتغيرات العادية. البرنامج المبسط** التالي **يشرح** ذلك بصورة واضحة. **ينشيء البرنامج مصفوفة من المستطيلات** ثم **يستخدم جملة تكرار لحساب مساحة كلٍ منها:**



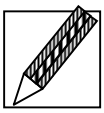
برنامج (7)

```
#include <iostream>
using namespace std;
class Rectangle {
    float width, height;
  public:
    void SetSides (float, float);
    float area (void) {return (width *
height);}
};

void Rectangle::SetSides (float a, float b)
{
  width = a;
  height = b;
}
int main ()
{
  Rectangle a[5];
  for (int i=0; i<5; i++)
  { a[i].SetSides (i+1.5,(i+1)*2);
    cout << "the area of rectangle  " <<i<<"
is "<<a[i].area() << endl;
   }
    system("pause");
  return 0;
}
```



تدريب(6)



1.اكتب يستخدم الفصيلة Timer التي عرفناها في المثال ... أعلاه في تسجيل وطباعة زمن الوصول لمجموعة من الموظفين في مؤسسة معينة. استخدم مصفوفة من الكيانات لتمثيل الموظفين. افترض أن عدد الموظفين يعطى للبرنامج قبل البدأ في تسجيل زمن الوصول.

2.أعد كتابة البرنامج أعلاه باستخدام المؤشرات (pinters). استخدم مؤشر في تعريف المصفوفة و معالجتها.

2.صمم فصيلة Complex تسمح للمستخدم بإجراء عمليات الجمع و الطرح على الأعداد المركبة. تذكر أن العدد المركب يتكون من شقين: حقيقي، و تخيلي.

نشاط(2)



إذا أضفنا صفة const إلى تعريف الدالة ()SetSides في الفصيلة Rectangle التي عرفناها في المثال ... أعلاه:

void SetSides (float, float) const;

عزيزي الدارس فكر واستنبط أثر هذه الصفة على هذه الدالة؟

الخلاصة

ركز هذا الفصل على **كيفية تعريف الفصائل و استخدامها في تعريف الكيانات.** بدأ الفصل **بإعطاء الهيكل العام لتعريف الفصائل،** ثم تبع ذلك **شرحٌ مفصل لكيفية استخدام مستويات الحماية في تعريف مكونات الفصائل.** بعد ذلك قدمنا **شرحاً مفصلاً لدور دوال الإنشاء ودوال الإخلاء في تصميم الفصائل. استخدام المؤشرات في تعريف الكيانات، و التعامل مع مصفوفات الكيانات** هما الموضوعان اللذان ختمنا بهما الفصل.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس هل تعلم أن تصميم الكثير من البرامج المعقدة اليوم لايبدأ من الصفر وإنما يبدأ من حيث انتهى الآخرون. يرجع ذلك كله إلى تقنية "إعادة استخدام البرمجيات" (software reuse) و التي تتبني على أسلوبين مشهورين في البرمجة الكيانية:

- الإرث Inheritance.
- التركيب Composition.

في الوحدة القادمة سوف نعرض لهذين الأسلوبين و المواضيع المتعلقة بهما بشيءٍ من التفصيل.

إجابات التدريبات

تدريب (1)

```
// classes example1
#include <iostream.h>
class Triangle {
    float length1, length2, length3 ;
  public:
    void SetSides (float,float, float);
float surround (void) {return ((length1+
length2+ length3));
float s (void) {return ((length1+ length2+
length3)/2));
float area (void) {return (SQRT(S(S-
length1)(S- length2)(S- length3)));}
};
void Triangle::SetSides (float length1, float
length2, float length3)
{
  length1 = A;
  length2 = B;
  length3 = c;
}
int main ()
{ float TriaArea; float Triasurround;
  Triangle MyTria; //declare an object of type
Triangle
 cout << " surround: " << Triasurround;
//compute and print the surround
 cout << "area: " << TriaArea;  //compute and
print the area
  cin >>A,B.C;
}
```

تدريب(2)

.1

```
Class Book
{ private:
    string Title;
    string author;
    int ISBN;
    float price;
    int Quantity;
  public:
     void UpdateQuantity(int n);
     void PrintInvoice();
     void SellCopy();
    };
    Book:: Book(string Title, string author );
    Book:: Bok(int ISBN, float price, int Quantity);
```

.2

الدالة UpdateQuantity() تستقبل عنصراً صحيحاً وتعدل الكمية الموجودة من الكتاب باستخدام هذا العنصر. الدالة PrintInvoice() تطبع رقماً خاصاً بالفاتورة، متبوعاً بمعلومات الكتاب وسعره. الدالة SellCopy() تعدل مخزون الكتاب (الكمية الموجودة)، تضيف سعر الكتاب إلى مجموع المبيعات وتطبع فاتورة البيع (يمكن لهذه الدالة أن تستخدم الدوال الأخرى في الفصيلة.

.3

```
#include<iostream.h>
#include<conio.h>
class Book
{
char Title;
char Author;
int ISBN,Quantity;
float Price;
public:
Book(char T,char A,int IS)
{
 Title= T;
 Author= A;
 ISBN=IS;
  }
Book(int Q,float P)
{
Quantity= Q;
Price= P;
}
void UpdateQuantity(int n);
void PrintInvoice(){
cout<<"the Book Title is   :"<<Title<<endl;
cout<<"Author Name is  : "<<Author<<endl;
cout<<"ISBN   :"<<ISBN<<endl;
cout<<"the quantity is :"<<Quantity<<endl;
cout<<" the price is : "<<Price<<endl;
 }
void SellCopy()
{
int TotalSell=5;
TotalSell= TotalSell+Price;
cout<<endl;
cout<<"The total sell is :  "<<TotalSell<<endl;
```

```
}
};
void Book :: UpdateQuantity(int n)
{
 Quantity=Quantity+n;
 cout<<"The total quantity is : "<<Quantity;
 }
main()
 {
 Book Mybook('y','al',876);
 Book  Mybook1(20,205);
 Mybook=Mybook1;
 Mybook.PrintInvoice();
 Mybook.UpdateQuantity(3);
 Mybook.SellCopy();
 getch();
    }
```

تدريب(3)

في هذه الحالة ترفض C++ التعريف وتعطي إشعاراً بالخطأ.

السبب في ذلك أننا أعدنا تعريف الدالة الإفتراضية و غيرنا عدد العناصر الممررة لهذه الدالة. عند تغيير عدد العناصر الممررة لدالة الإنشاء الإفتراضية تلغي سي++ الدالة الأصلية.

في المثال الإيضاحي 3 اعلاه، دالة الإنشاء الإفتراضية تركت كما هي (لم يتم إعادة تعريفها)، لذا استخدمتها C++عندما عرفنا المتغير MyRect:

```
Rectangle Myrect;
```

أما في المثال الإيضاحي 4، قمنا بإعادة تعريف دالة الإنشاء وتغيير عدد العناصر الممررة، وعليه قامت C++بإلغاء دالة الإنشاء الإفتراضية الأصلية. عندما حاولنا استخدام الدالة الإفتراضية مرة أخرى، كان من الطبيعي أن تعترض C++و تعطي رسالة الخطأ.

الحل هب أننا نريد تعريف كيان من نـــوع Rectangle دون أن نحـــدد كـــم طولـــه أو عرضه لحظة التعريف، ونريد أن نرجيء ذلك إلى مرحلة متقدمة في البرنـــامج. مـــن الواضح أن التعريف الذي أعطيناه في المثال 4 لايسمح بذلك. ما الحل إذاً؟

الحل في مثل هذه الحالات هو إضافة دالة إنشاء أخرى لا تحتاج إلى تمريـــر عناصـــر، فيصبح لدينا دالتا إنشاء، بدلاً من واحدة. تعدد دوال الإنشاء مسموح به في C++بحيـــث تختلف في عدد العناصر الممررة.

تدريب (1)
بطريقة أخرى

1-

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class Triangle
{
float first,second,third;
public:
void setsides(float a,float b,float c)
{first=a;
second=b;
third=c;
}
float get_s(float a,float b,float c);
float area(void)
{float s=(first+second+third)/2;
return(sqrt(s*(s-first)*(s-second)*(s-third)));}
float rounds(void)
{return(first+second+third); }
};
 main()
 {
```

```
    float triArea;
    Triangle Mytri;
    Mytri.setsides(3.0,4.0,3.0);
    triArea=Mytri.area();
    cout<<"The Area of the Tiangle is   : " <<triArea<<endl  ;
    cout<<"The round of the Triangle is : " <<Mytri.rounds()<<endl;
     getch();
     }
```

2-

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class Triangle
{
float first,second,third;
public:
void setsides()
{
cout<<"please enter the values";
cin>>first>>second>>third;
}
float area(void)
{float s=(first+second+third)/2;
return(sqrt(s*(s-first)*(s-second)*(s-third)));}
float rounds(void)
{return(first+second+third); }
};
 main()
 {

    float triArea;
    Triangle Mytri;
    Mytri.setsides();
```

```
    triArea=Mytri.area();
    cout<<"The Area of the Tiangle is   : " <<triArea<<endl  ;
    cout<<"The round of the Triangle is : " <<Mytri.rounds()<<endl;
     getch();
     }
```

تدريب (4)

1-

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
class Car
{
string CarType;
string Model;
string Country ;
int EngineNo;
int Year ;
public:
Car(void)
{}
Car(int y)
{Year = Y;
}
Car(int Y,char M,int E)
{Year=Y;
Model= M;
EngineNo=E;
}
void MilesPerGalon()
{
```

```
}
void PrintInfo(void)
{
cout<<CarType<<Model<<Country;
cout<<EngineNo<<Year;
}
};
main()
Car myCar(2005)
Car Mycar1(2004,'Nissan',298)
MyCar.PrinInfo();
MyCar1.PrintInfo();
 getch();
}
```

تدريب (5)

1-

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
using namespace std;
class Car
{
string CarType;
string Model;
string Country ;
int EngineNo;
int Year ;
public:
Car(void)
{}
Car(int y)
{Year = Y;
```

```
}
Car(int Y,char M,int E)
{Year=Y;
Model= M;
EngineNo=E;
}
void MilesPerGalon()
{

}
void PrintInfo(void)
{
cout<<CarType<<Model<<Country;
cout<<EngineNo<<Year;
}
};
main()
Car myCar(2005)
Car Mycar1(2004,'Nissan',298)
MyCar.PrinInfo();
MyCar1.PrintInfo();
 getch();
}
```

2-

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
const MaxHours= 18;
class RMSemesters
{
int NSemesters,CurrentSem;
public:
RMSemesters(int NS,int CS)
{
```

```
NSemesters=NS;
CurrentSem=CS;
}
int CompHours(void);
int RMS(void);
};
int RMSemesters::CompHours(void)
{
return(CurrentSem*MaxHours);
}
int RMSemesters::RMS(void)
 {
 return(NSemesters-CurrentSem);
 }

main()
{
RMSemesters RM(8,3);
 cout<<"Te completed Hours are :      "<<RM.CompHours()<<"
Hours"<<endl;
 cout<<"The remaining Semesters are : "<<RM.RMS()<<"
Semesters"<<endl;

 getch();
}
```

تدريب(6)

1-

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class Timer
{
int Hr,Min,Sec;
public:
void SetTime(int , int , int);
void GetTime(int &,Int &, int &) const;
void PrintTime()const;
void IncrementSeconds();
void IncrementMinutes();
void IncrementHouts();
};
void Timer::SetTime(int hours,int minutes,int seconds)
{
if(hours>=0&&hours<24)
hr = hours;
else
hr=0;
if(minutes>=0&&minutes<60)
min=minutes;
else
min=0;
if(seconds>=0&&second<60)
sec=seconds;
else
sec=0;
}
void Timer::GetTime(int &hours,int &minutes,int &seconds) const
{
```

```
hours=hr
minutes=min;
seconds=sec;
}
void Timer::PrintTime()const
{
if(hr<10)
 cout<<"0";
 cout<<hr<<":";
 if(min<10)
  cout<<"0";
  cout<<min<<":";
  if(sec<10)
   cout<<"0";
   cout<<sec;
   }
 void Timer:: IncrementHours()
 {
 hr++;
 if(hr>23)
 hr=0;
 }
 void Timer::IncrementMinutes()
 {
 min++;
 if(min>59)
 min=0;
 }
 void Timer::IncrementSeconds()
 {
 sec++;
 if(sec>59)
 sec=0;
 }
 main()
```

```
 {
 int i;
 Timer *ArrivalTime[5],*CheckOutTime[5];
 for(i=1;i<=5;i++)
 ArrivalTime[i]->SetTime(07,15,45);
 cout<<endl<<endl<<"The Workers check in times are ";
  for(i=1;i<=5;i++)
 ArrivalTime[i]->PrintTime();
 cout<<endl;
 for(i=1;i<=5;i++)
 CheckOutTime[i]->SetTime(2,30,37);
 cout<<"The Workers ckeck-out times are :  " ;
  for(i=1;i<=5;i++)
 CheckOutTime[i]->PrintTime();
 cout<<endl;
 getch();
}
```

2-

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class Timer
{
int Hr,Min,Sec;
public:
void SetTime(int , int , int);
void GetTime(int &,Int &, int &) const;
void PrintTime()const;
void IncrementSeconds();
void IncrementMinutes();
void IncrementHouts();
};
void Timer::SetTime(int hours,int minutes,int seconds)
{
if(hours>=0&&hours<24)
```

```
hr = hours;
else
hr=0;
if(minutes>=0&&minutes<60)
min=minutes;
else
min=0;
if(seconds>=0&&second<60)
sec=seconds;
else
sec=0;
}
void Timer::GetTime(int &hours,int &minutes,int &seconds) const
{
hours=hr
minutes=min;
seconds=sec;
}
void Timer::PrintTime()const
{
if(hr<10)
 cout<<"0";
 cout<<hr<<":";
 if(min<10)
  cout<<"0";
  cout<<min<<":";
  if(sec<10)
   cout<<"0";
   cout<<sec;
   }
 void Timer:: IncrementHours()
 {
 hr++;
 if(hr>23)
 hr=0;
```

```
}
void Timer::IncrementMinutes()
{
min++;
if(min>59)
min=0;
}
void Timer::IncrementSeconds()
{
sec++;
if(sec>59)
sec=0;
}
main()
{
int i;
Timer *ArrivalTime[5],*CheckOutTime[5];
for(i=1;i<=5;i++)
ArrivalTime[i]->SetTime(07,15,45);
cout<<endl<<endl<<"The Workers check in times are ";
 for(i=1;i<=5;i++)
ArrivalTime[i]->PrintTime();
cout<<endl;
for(i=1;i<=5;i++)
CheckOutTime[i]->SetTime(2,30,37);
cout<<"The Workers ckeck-out times are :  " ;
 for(i=1;i<=5;i++)
CheckOutTime[i]->PrintTime();
cout<<endl;
getch();
}
```

3-

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

```
class Complex
{
float real,img  ;
public:
float SetReal(float a)
{
real=a;
}
float SetImg(float b)
{
img=b;
}

float AddReal(void)
{
return(SetReal(real)+SetReal(real));
}
float AddImg(void)
{return(SetImg(img)+SetImg(img));
}
float SubReal(void)
{
return(SetReal(real)-SetReal(real));
}
float SubImg(void)
{return(SetImg(img)-SetImg(img));
}
};
main()
{
Complex x,z;
Complex y;
x.SetReal(3.5);
y.SetReal(4.5);
x.SetImg(5.5);
```

```
y.SetImg(2.5);
cout<<z.AddReal()<<endl;
cout<<z.AddImg()<<endl;
cout<<z.SubReal()<<endl;
cout<<z.SubImg()<<endl;
getch();
}
```

مسرد المصطلحات

- **الفصيلة class**

هو محتوي لكل الخصائص و العمليات التي يستخدمها الكيان (مواصفات الكيان).

- **مستوى الحماية protection**

يسمح للكيان بالوصول إلى الخواص، والدوال من خارج الفصيلة واستخدامها.

- **مستوى الحماية خاص private**

لا يمكن الكيان من استخدام الخواص و الدوال من خارج الفصيلة.

- **مستوى الحماية الوسيط Protected**

لا يمكن أن تستخدمه إلا فصائل لها علاقة خاصة بالفصيلة المالكة

- **مستوى الحماية العام public**

يمكن من استخدام الخواص و الدوال المعنية من خارج الفصيلة.

- **دالة الإنشاء constructor**

دالة خاصة تأخذ اسم الفصيلة مع إمكانية تمرير عناصر (اختيارياً).

- **دالة إنشاء إفتراضية Virtual constructor**

تنشئ مع كل فصيلة ، لا تقوم بعمل شيء، و لا تستقبل أي عناصر ممررة وتستدعى لحظة استخدام الفصيلة في تعريف كيانات.

- **عملية الإسناد object assignment:**

هي العملية الوحيدة التي يمكن تطبيقها مباشرةً على الكيانات حيث تسمح بإسناد الكيانات بعضها إلى بعض إذا كانت من نفس النوع.

- **دوال إخلاء Destructors**

هي دوال تستدعى عند الإنتهاء من استخدام الكيان. دالة الإخلاء لأي فصيلة تأخذ إسم الفصيلة مسبوقاً بعلامة التلدا (~).

المصادر والمراجع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.


الوحدة الثالثة
الإرث و التركيب
(Inheritance and Composition)

محتويات الوحدة

الموضوع	الصفحة
المقدمة	65
تمهيد	65
أهداف الوحدة	67
1. الإرث	68
1.1 إعادة تعريف دالة الإنشاء وأثره على الإرث.	76
2.1 شجرة الإرث و عمقها	78
2. الدوال والفصائل الصديقة	89
1.2 الدوال الصديقة	89
2.2 الفصائل الصديقة	92
3. التركيب	95
خلاصة	104
لمحة مسبقة عن الوحدة التالية	104
إجابات التدريبات	105
مسرد المصطلحات	121
المراجع	122

المقدمة

تمهيد

عزيزي الدارس،

هل تعلم أن نظام الويندوز (Windows) والنظم المشابهة توفر الكثير من الكيانات الجاهزة التي يسهل استخدامها في البرامج المطورة دون الحاجة إلى معرفة تفاصيلها؟ هل تعلم أنه بالإمكان تصميم معالج كلمات يشبه ال(Word) في دقائق؟ هل تعلم أنه بالإمكان تصميم متصفح انترنت خاص بك في وقتٍ قصير؟ ....إلى غير ذلك من النظم التي نراها في السوق اليوم.

واحدة من أهم **فوائد البرمجة الكيانية** عزيزي الدارس هي إمكانية **إعادة استخدام الكثير من الكيانات (software reuse)** التى تم تصميمها من قبل بسهولةٍ كبيرة. عملية إعادة الاستخدام **لا تتطلب** أي معرفة بتفاصيل تحقيق تصميم (implementation) الكيان، إنما **تتطلب** فقط معرفة الواجهة التي يصدّرها الكيان. وقد أسهمت فكرة إعادة استخدام الكيانات في التطور البرمجي الكبير الذي تشهده الساحة، خاصةً في مجال البرمجة المرئية و الرسم بالحاسوب. فمثلاً نظام Windows وشبيهاته من نظم التشغيل توفر كمية كبيرة من الكيانات الجاهزة التي يسهل إعادة استخدامها بسهولة في البرامج العادية. مثال ذلك: الأزرار (buttons)، صناديق الحوار (dialog boxes)، شبكات البيانات (grids)، محررات النصوص الكاملة (full editors) ...إلى آخر ذلك من الخيارات الواسعة.

فكرة إعادة استخدام الكيانات أدت إلى ظهور. انتعاش سوق جديدة لم تكن موجودة من قبل هي سوق "**المكونات المرئية**" **(visual components). المكونات المرئية عبارة عن كيانات متكاملة تقوم بحل مسائل معينة وتستخدم رمزاً مرئياً في الدلالة على نفسها.** إذا أراد المبرمج استخدام مثل هذه المكونات، ماعليه إلا أن يجر رمز المكون ويرمي به على

سطح الشاشة المرئية للبرنامج، مثلما يفعل بالأزرار و صناديق الحوار، و إذا بالمكون يصبح جزءاً فاعلاً من البرنامج.

في هذا الوحدة نعطي أمثلة لتعميق الشرح وتقريب الفهم والإستيعاب لعمليتي الإرث و التركيب، مع مناقشة التبعات العملية لكل منهما وحتى يتمكن الدارس من حل التدريبات وأسئلة التقويم الواردة في الوحدة والتي توجد حلولها في نهاية الوحدة .

أهداف الوحدة



بعد فراغك عزيزي الدارس من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. **تعرف** الإرث.
2. **توضح** أثر إعادة تعريف دالة الإنشاء على الإرث.
3. **تنشئ** شجرة الإرث وتحدد عمقها.
4. **تستخلص** العناصر المشتركة بين فصائل معينة وعزلها في فصيلة واحدة ترثها الفصائل ذات العلاقة.
5. **تعرف** الفصائل و الدوال الصديقة.
6. **تستخدم** عملية التركيب.

لغة C++ وشبيهاتها من **لغات البرمجة الكيانية توفر طريقتين لإعادة استخدام الكيانات:**

1. الإرث (inheritance)

الإرث هي الطريقة الأكثر شيوعاً و استخداماً في الأوساط البرمجية. **يقصد بالإرث هنا إنتماء فصيلة إلى فصيلة، أو فصائل أخرى بغية استخدام ما بداخلها من الخواص، و الدوال.** عملية انتماء الفصائل إلى الفصائل **تختلف** من انتماء الكيانات إلى الفصائل التي رأيناها في الفصل السابق.

المثال التالي يوضح كيف يتم نسبة الفصيلة MyNewClass إلى فصيلة أخرى BaseClass، يفترض أنها موجودة:

```
Class MyNewClass: public BaseClass
{ . . .
   . . .
};
```

في هذه الحالة، **MyNewClass** هي **الفصيلة الوارثة(المنتمية)**، بينما **BaseClass** هي **الفصيلة الموروثة (المنتمى إليها). الفائدة المجنية** في هذه الحالة **أن كل مكونات العناصر الموروثة تصبح جزءاً من الفصيلة الوارثة ويمكن استخدامها كأنها جزء من الفصيلة.** هذا النوع من الإرث **يسمى الإرث الأُحادي (single inheritance).**

لاحظ أننا استخدمنا الكلمة الخاصة public قبل اسم الفصيلة الموروثة، وهذا يعنى أن عناصر الفصيلة الموروثة تنتقل بنفس مستوى حمايتها إلى الفصيلة الوارثة: **ما كان عاماً (public) يصبح عاماً في الفصيلة الوارثة، و ماكان خاصاً (private) يصبح خاصاً في الفصيلة الوارثة،** ... وهكذا.

إذا لم نستخدم كلمة public في عملية الإرث، فإن C++ تفترض أن الإرث يتم بصورة خاصة (private). في هذه الحالة كل مكونات الفصيلة الموروثة تصبح خاصة في الفصيلة الوارثة، و بالتالي لا يمكن استخدامها بصورة مباشرة.

C++ تسمح أيضاً **بالإرث المتعدد (multiple inheritance)**، حيث يمكن **نسبة الفصيلة الوارثة إلى أكثر من فصيلة**. مثلاً:

```
Class MyNewClass: public BaseClass1, BaseClass2
{ . . .
   . . .
};
```

في هذه الحالة تؤول مكونات الفصائل الموروثة بكاملها إلى الفصيلة الوارثة.

ملحوظة

الإرث المتعدد هو نوع من الإرث قد يؤدي إلى بعض الإشكالات إذا تضاربت أسماء المكونات الموروثة، لذا تتجنبه الكثير من لغات البرمجة الكيانية و لا تسمح باستخدامه.

مثال (1)

البرنامج أدناه **يعرِّف فصيلة BaseClass ، و أخرى DerivedClass. الفصيلة الثانية ترث الفصيلة الأولى و تضيف خواص وعمليات أخرى إلى ماورثت.** البرنامج يوضح أن **الخواص و العمليات الموروثة تستخدم كأنها جزء أصلي من الفصيلة الوارثة. الإرث تم** بالصورة العامة **public**.

برنامج(1) 

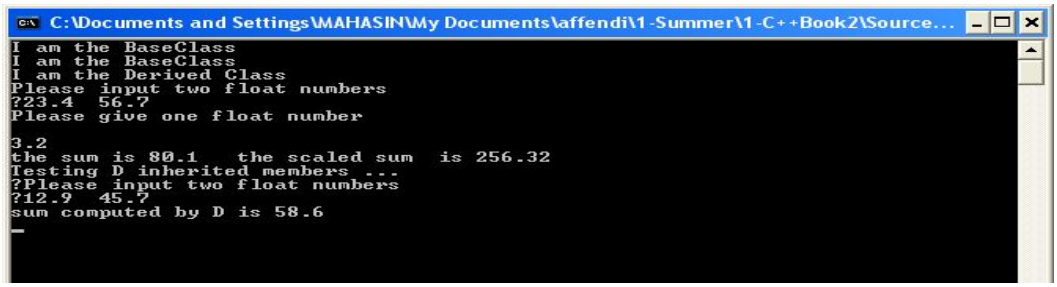
```
#include <iostream.h>
#pragma argsused
class BaseClass
{private:
   float x,y;
 public:
   BaseClass(){ cout<<"I am the
BaseClass"<<endl;
                 x=0.0; y=0.0;}
   void SetValues(float a, float b) {x=a; y=b;}
   float sum()
   { float z=x+y; return z;}
 };
 class DerivedClass: public BaseClass
 { private:
      float r;
   public:
     DerivedClass(){ r=0.0; cout <<"I am the
Derived Class"<<endl;}
     void set(float d) { r= d;}
     float SumAndScale(float a, float b,float c)
     { float s;
       SetValues(a,b); s=sum();
```

```
   cin >>xx>>yy;
   B.SetValues(xx,yy);
   ss=B.sum();
   cout <<"Please give one float
number\n"<<endl;
   cin>>rr;
   pp=D.SumAndScale(xx,yy,rr);
   cout <<"the sum is "<<ss<<"   the scaled sum
is "<<pp<<endl;
```

```
    //now let us use sum as a member of D
    cout <<"Testing D inherited members ...\n?";
    cout <<"Please input two float numbers\n?";
    cin >>xx>>yy;
    D.SetValues(xx,yy);  // inherited SetValues
    ss=D.sum();  //inherited sum
    cout <<"sum computed by D is "<<ss<<endl;

    return 0;
}
```

عينة من تنفيذ البرنامج:



ملاحظات على البرنامج:

1- يبدأ البرنامج **بتعريف الفصيلة الأم:**

```
class BaseClass
{private:
  float x,y;
 public:
  BaseClass(){ cout<<"I am the BaseClass"<<endl;
            x=0.0; y=0.0;}
  void SetValues(float a, float b) {x=a; y=b;}
  float sum()
  { float z=x+y; return z;}
 };
```

تحتوي هذه الفصيلة على **متغيرين** في المستوى الخاص **(private)**، **دالتين** في المستوى العام **(public)**، بالإضافة إلى **دالة الإنشاء** التي **أعيد تعريفها** بحيث **تطبع إشعاراً عند تعريف كيان تابع للفصيلة.**

2- بعد ذلك يقوم البرنامج **بتعريف الفصيلة الوارثة (DerivedClass):**

```
class DerivedClass: public BaseClass
 { private:
    float r;
  public:
   DerivedClass(){ r=0.0; cout <<"I am the Derived Class"<<endl;}
   void set(float d) { r= d;}
   float SumAndScale(float a, float b, float c)
   { float s;
    SetValues(a,b); s=sum();
    set(c);
    float p=r*s; return p; }
 };
```

بالإضافة إلى **العناصر الموروثة** تقوم هذه الفصيلة **بتعريف متغير جديد على المستوى الخاص (r)**، و **دالتين جديدتين على المستوى العام. دالة الإنشاء** أيضاً **تم إعادة تعريفها** بحيث **تطبع إشعاراً عند تعريف كيان تابع للفصيلة.** لاحظ أن **الدالة (SumAndScale() في الفصيلة الوارثة استخدمت الدالتين الموروثتين** SetValues() **و (sum() كأنهما جزء من الفصيلة الوارثة. هذا** إذا كان **الإرث من النوع public أو protected. أما** إذا كان **الإرث من النوع private** فلا بد من **نسبة هذه الدوال إلى الدالة الأم باستخدام علامة النطاق.**

3- **داخل البرنامج، تم تعريف كيانين أحدهما تابع للفصيلة الأم (الكيان B) و الآخر تابع للفصيلة الابن (الكيان D).** لاجديد في استخدام الدوال التابعة للفصيلة الأم، ولكن

يلاحظ أننا **استخدمنا الدوال الموروثة في الفصيلة الإبن، و كأنها جزء أصلي من هذه الفصيلة:**

```
//now let us use sum as a member of D
  cout <<"Testing D inherited members ...\n?";
  cout <<"Please input two float numbers\n?";
  cin >>xx>>yy;
  D.SetValues(xx,yy);  // inherited SetValues
  ss=D.sum();  //inherited sum
  cout <<"sum computed by D is "<<ss<<endl;
```

ملاحظات على الإرث

1- الفصيلة الوارثة (الإبن) **ترث كل شيء من الفصيلة الأم إلا دوال الإنشاء، دوال الإنشاء الناقلة(copy** constructors) و الأصدقاء (friends). (سنتكلم عن الأصدقاء بعد قليل).

2- **دالة الإنشاء الافتراضية للفصيلة الأم** (بالرغم من أنها لاتورث) **يتم استدعاؤها تلقائياً عند تعريف كيان تابع للفصيلة الإبن،** و لعل الدارس الحصيف قد يكون لاحظ ذلك في شاشة تنفيذ البرنامج أعلاه. لاحظ **أن جملة " I am the BaseClass" و التي تتطبعها الدالة الإفتراضية للفصيلة الأم ظهرت مرتين: مرةً عند تعريف الكيان B و مرةً عند تعريف الكيان D.**

3- **الفصيلة الوارثة** تستطيع **استخدام العناصر العامة (public) و العناصر المحمية (protected) من الفصيلة الأم مباشرةً.** أما **العناصر الخاصة في الفصيلة الأم فلايمكن استخدامها مباشرةً في الفصيلة الوارثة.**

4- مستويات الحماية للعناصر الموروثة في الفصيلة الإبن يمكن **تلخيصها في الجدول التالي:**

إمكانية الوصول و الاستخدام من:	public	protected	private
أعضاء الفصيلة نفسها (الأم)	نعم	نعم	نعم
أعضاء الفصائل الوارثة	نعم	نعم	لا
أعضاء الفصائل الأخرى التي لا تربطها علاقة بالفصيلة	نعم	لا	لا

من الجدول أعلاه نخلص **للقاعدة العامة** التالية:

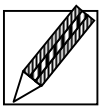
أن العناصر العامة (public)في الفصيلة الأم يمكن الوصول إليها و استخدامها في الفصيلة نفسها، أو من أي فصيلة أخرى. العناصر المحمية (protected) يمكن الوصول إليها، و استخدامها من أعضاء الفصيلة الأم و أعضاء الفصائل الوارثة. أما العناصر الخاصة(private) فلا يمكن الوصول إليها إلا من أعضاء الفصيلة الأم نفسها.

أسئلة تقويم ذاتي



1- كيف تتم عملية الإرث في C++
2- ما الأشياء التى لاترثها الفصيلة الوارثة؟
3- ما الفرق بين الإرث العام (public)، الإرث المحمي (protected) و الإرث الخاص (private)؟
4- في حالة الإرث العام، هل يمكن للفصيلة الوارثة أن تستخدم العناصر الخاصة في الفصيلة الأم؟
5- ماهو الفرق بين العناصر المحمية (protected) و العناصر العامة (public) في حالة الإرث العام؟

تدريب(1)



1.اكتب فصيلة أم CircularShape تحتوي على عناصر تتعلق بخواص الدائرة العادية وعمليات حساب المساحة، و المحيط. صمم فصيلة أخرى Cylinder ترث الفصيلة CircularShape، و تضيف إليها الخواص المتعلقة بالشكل الأسطواني (مثل إرتفاع الإسطوانة) وعمليات حساب المساحة و المحيط للشكل الأسطواني. اكتب برنامجاً تختبر فيه صحة تصميماتك.

2. في حالة الإرث الخاص(private)، كيف تستخدم الفصيلة الوارثة عناصر الفصيلة الأم وكيف تستفيد منها؟ إذا غيرنا طريقة الإرث في البرنامج أعلاه إلى إرث خاص: class DerivedClass: private BaseClass

```
{ private:
    float r;
  public:
   DerivedClass(){ r=0.0; cout <<"I am the Derived
Class"<<endl;}
   void set(float d) { r= d;}
   float SumAndScale(float a, float b, float c)
   { float s;
    SetValues(a,b); s=sum();
    set(c);
    float p=r*s; return p; }
 };
```

هل يعمل البرنامج في هذه الحالة؟ و إذا كانت الإجابة لا: كيف يمكن تعديل البرنامج ليؤدي المطلوب؟

1.1 إعادة تعريف دالة الإنشاء وأثر ذلك على عملية الإرث

ذكرنا في الفقرات السابقة أن **دوال الإنشاء ليست ضمن الموروثات** و لا تصير بحال عضواً في الفصيلة الوارثة، إلا أن دالة الإنشاء الافتراضية في الفصيلة الأم تستدعى تلقائياً عند تعريف كيان تابع للفصيلة الإبن. إذا أعدنا تعريف دالة الإنشاء الإفتراضية دون استخدام عناصر ممررة، فإن ذلك لا يؤثر على عملية الإستدعاء المذكورة أعلاه: أي أن **دالة الإنشاء الإفتراضية في الفصيلة الأم يتم استدعاؤها عند تعريف كيانات تابعة للفصائل الوارثة، حتى بعد إعادة تعريف هذه الدالة، ما لم نستخدم عناصر ممررة عند إعادة التعريف.**

في المثال السابق أعدنا تعريف دالة الإنشاء BaseClass() غير أننا لم نستخدم عناصر ممررة عند إعادة التعريف، لذا تم استدعاء هذه الدالة عند تعريف الكيان D التابع للفصيلة الوارثة DerivedClass.

أما إذا استخدمنا عناصر ممررة عند إعادة تعريف دالة الإنشاء فإن ذلك يلغي دالة الإنشاء الافتراضية، كما ذكرنا في الفصل الثاني، و بالتالي لايمكن استدعاؤها عند تعريف كيانات وارثة. مثلاً إذا غيرنا تعريف دالة الإنشاء BaseClass() في المثال أعلاه بحيث تصبح:

```
BaseClass(float a, float b){ cout<<"I am the BaseClass"<<endl;
                x=b; y=b;}
```

فإن هذا يؤدي إلى إلغاء دالة الإنشاء الإفتراضية، و بالتالي عدم استدعائها عند تعريف الكيان D.

إذا كنا حريصين على استدعاء دالة الإنشاء للفصيلة الأم في هذه الحالة، فلابد من النص على ذلك في تعريف الفصيلة الوارثة. C++ تسمح بذلك على النحو التالي:

```
class DerivedClass: public BaseClass
 { private:
    float r;
  public:
```

```
  DerivedClass():BaseClass(float a, float b)
 { r=0.0; cout <<"I am the Derived Class"<<endl;}
   void set(float d) { r= d;}
  float SumAndScale(float a, float b, float c)
  { float s;
    SetValues(a,b); s=sum();
    set(c);
    float p=r*s; return p; }
};
```

قاعدة عامة

إذا أردنا استدعاء دالة إنشاء من الفصيلة الأم عند تعريف كيان تابع للفصيلة الإبن نضيف إسم دالة الإنشاء للفصيلة الأم إلى اسم دالة الإنشاء للفصيلة الوارثة بعد النقطة المزدوجة:

DerivedClass():BaseClass(float a, float b)

تدريب(2)

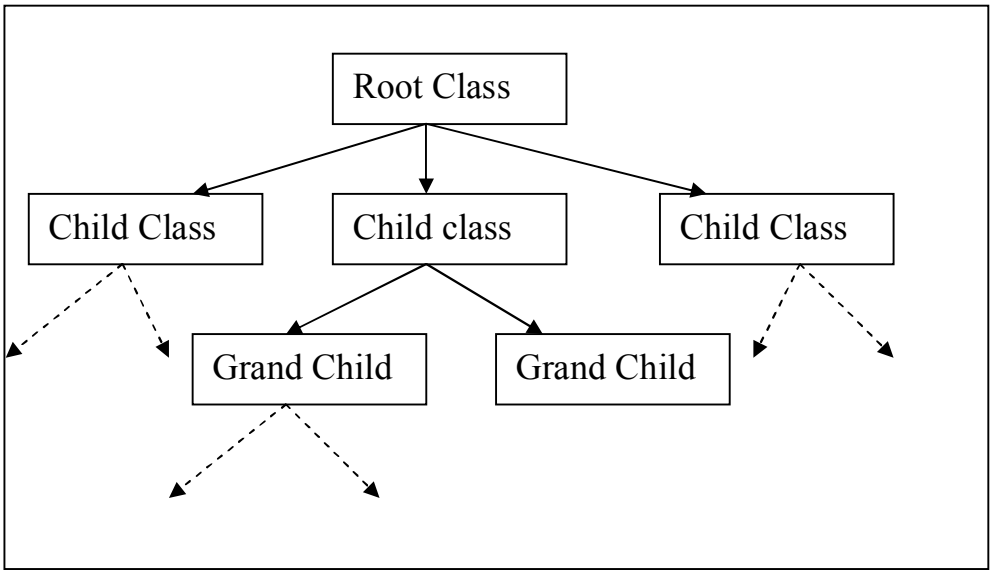


1. حسب قواعد الهندسة التحليلية، اي نقطة هندسية تمثل بإحداثيين: الإحداثي السيني و الإحداث الصادي. صمم فصيلة `Point` تمثل نقطة هندسية. الفصيلة ينبغي أن تسمح للمستخدم بإسناد قيم للإحداثيين، أحدهما أو كلاهما، طباعة الإحداثيين ام إسنادهما إلى متغيرات أخرى.

2. صمم فصيلة Cylinder ترث الفصيلة Circle وتستخدم خواص هذه الفصيلة في تمثيل قاعدة الأسطوانة وحساب محيط القاعدة و مساحتها. الفصيلة Cylinder ينبغي أن تحتوي على دوال تحسب مساحة و حجم الأسطوانة، باعتبار أن ارتفاع الأسطوانة معروف.

2.1 شجرة الإرث و عمقها

في المثال البسيط أعلاه **استخدمنا فصيلتين فقط: إحداهما تمثل الأم والأخرى تمثل الابن.** عموماً **يمكن أن تكون هناك أكثر من فصيلة وارثة، الفصائل الوارثة يمكن أيضاً أن ترثها فصائل أخرى إلى أي عمق.** من هذا المنطلق **يمكن تمثيل علاقة الإرث كشجرة، جذرها الفصيلة الأم وفروعها الفصائل الوارثة:**



في المثال التالي **نستخدم شجرة إرث تحتوي على فرعين عمق أحدهما 2 و عمق الآخر (1)**

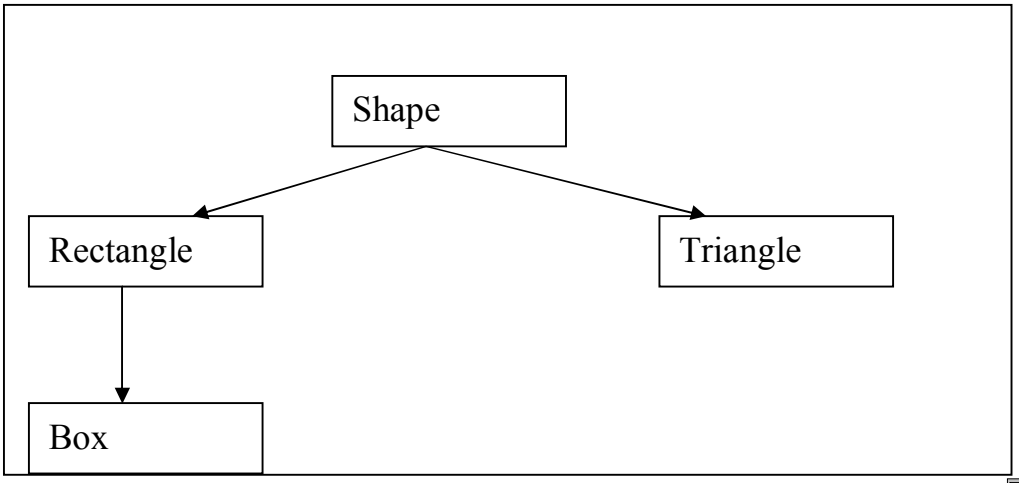
مثال (2)

الفصيلة الأم في هذا المثال هي **Shape** و التي تحتوي على **متغيرين (العرض و الإرتفاع)** و **دالتان: ()SetValues و ()Area.** فصيلة **Shape ترثها** فصيلتان، هما **Rectangle و Triangle.** الفصيلتان الوارثتان لايضيفان جديداً، و لكنهما **يعيدا تعريف الدالة ()Area لتحسب مساحة الشكل حسب القاعدة الخاصة بكل شكل. رأينا** من قبل **كيف تتم**

إعادة تعريف دالة الإنشاء، و الآن نرى صورة أخرى من عملية إعادة التعريف لدالة عادية. عملية إعادة تعريف الدوال و تغيير مقاصدها تقليدٌ متبع في البرمجة الكيانية و له أوجه مختلفة سنتطرق إليها في بعض الفصول القادمة.

البرنامج يحتوي على **درجة أخرى من الإرث** حيث **ترث** الفصيلة **Box** الفصيلة **Rectangle**. هذا يعني أن الفصيلة **Box ترث** أيضاً الفصيلة **Shape بصورة غير مباشرة.**

الشجرة التالية توضح العلاقة بين هذه الفصائل:



برنامج(2)

```
#include <iostream>
#include <stdlib.h>
using namespace std;
// derived classes
class Shape {
  protected:
    float width, height;
  public:
    void set_values (float a, float b)
```

```
      { width=a; height=b;}
    float area(void) {}
  };
class Rectangle: public Shape {
  public:
    float area (void)
      { return (width * height); }
  };
class Triangle: public Shape {
  public:
    float area (void)
      { return (width * height / 2); }
  };
class Box: public Rectangle
{ private:
     float BoxHeight;
  public:
      Box(float a) {BoxHeight = a;}
      float Volume (void)
      { return area()*BoxHeight; }
};
int main () {
  Rectangle rect;
  Triangle trgl;
  rect.SetValues (4,5);
  trgl. SetValues (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  Box B(3.4);
  B.SetValues (7.2, 4.8);
  cout <<B.Volume()<<endl;
  return 0;
}
```

لاحظ أن **الكيان B** داخل البرنامج **يستخدم الدوال و المتغيرات المعرفة في الفصيلة (الجدّة) Shape، و كأنها جزء منها**، وهذا **طبيعي جداً** في البرمجة الكيانية: بغض النظر عن عمق العلاقة بين الفصائل الوارثة و الموروثة، **فكلما يحتويه الآباء يؤول إلى الأبناء و الأحفاد الوارثين.**

عينة من تنفيذ البرنامج:



حسب قواعد الهندسة التحليلية، أيّ نقطة هندسية تمثل بإحداثيين: الإحداثي السيني و الإحداث الصادي. صمم فصيلة Point تمثل نقطة هندسية. الفصيلة ينبغي أن تسمح للمستخدم بإسناد قيم للإحداثيين، أحدهما، أو كلاهما، طباعة الإحداثيين أم إسنادهما إلى متغيرات أخرى.

مثال(3)

في هذا المثال التطبيقي، الفصيلة **PartTimeEmployee ترث** الفصيلة **Person** التي استخدمناها من قبل في الفصل الثاني **(إرث عام)**. من هذا المنطلق يمكن للفصيلة **PartTimeEmployee** أن **تستخدم الخواص و الدوال** التي توفرها الفصيلة **Person مباشرةً** كما هو **واضح** من تفاصيل البرنامج. الفصيلة **PartTimeEmployee أضافت بعض الخواص و العمليات الخاصة بها.**

برنامج (3)

```
#include <iostream>
#include <stdlib.h>
using namespace std;
class Person
{private:
   string FirstName;
   string LastName;
   string Name;
   int Age;
 public:
   Person();
   void SetFirstName(string);
   void SetLastName(string);
   string GetFirstName(string &);
   string GetLastName(string &);
   void PrintName();
};
Person::Person()
{ FirstName=""; LastName="";
}
void Person::SetFirstName(string Fname)
{ FirstName = Fname;
}
void Person::SetLastName(string Lname)
{ LastName= Lname;
}
string Person::GetFirstName(string &Fname)
{ Fname = FirstName;
  return FirstName;
}
string Person::GetLastName(string &Lname)
{ Lname = LastName;
```

```
  return LastName;
}
void Person::PrintName()
{ cout <<FirstName<<"   "<<LastName<<endl;
}
class PartTimeEmployee: public Person
{private:
    float Salary, Rate;
    int hours;
 public:
     PartTimeEmployee()
     { Rate=0.0; hours=0; Salary=0;}
     void SetValues(float r, int h)
     { Rate=r; hours=h; }
     void ComputeSalary()
     { Salary = hours * Rate;}
     float GetSalary()
     {return Salary; }
};
int main()
{ PartTimeEmployee EMP;
  string FName, LName;
  cout <<"please enter employee first name\n?";
  cin >>FName;
  cout <<"Please enter employee last name\n?";
  cin >>LName;
    EMP.SetFirstName(FName);
    EMP.SetLastName(LName);
    cout<<"Employee full name is :";
    EMP.PrintName();
    //Now enter Salary rate and hours worked
    int R,H;
    cout
```

```
     <<"Please enter rate per hour and hours
worked\n?";
    cin >>R>>H;
    EMP.SetValues(R,H);
    EMP.ComputeSalary();
    float S=EMP.GetSalary();
    cout <<"The Employee Salary is "<<S<<endl;
    return 0;
}
```

عينة من تنفيذ البرنامج:



مثال (4)

في هذا المثال **نستخدم** الفصيلة **الأم LibItem لاحتواء الخواص و العمليات المشتركة بين أنواع المعروضات في مكتبة عامة** (في هذه الحالة الكتب و الدوريات العلمية). بعد ذلك **نعرِّف فصيلتين وارثتين، تقتبسان مواصفات الفصيلة الأم.** هاتان **الفصيلتان هما Book و Journal.** لاحظ أننا استخدمنا **مستوى الحماية protected للخواص المشتركة في الفصيلة الأم، وذلك حتى يتسنى للفصائل الوارثة التعامل مع هذه الخواص و تعديلها إذا لزم.**

برنامج(4)

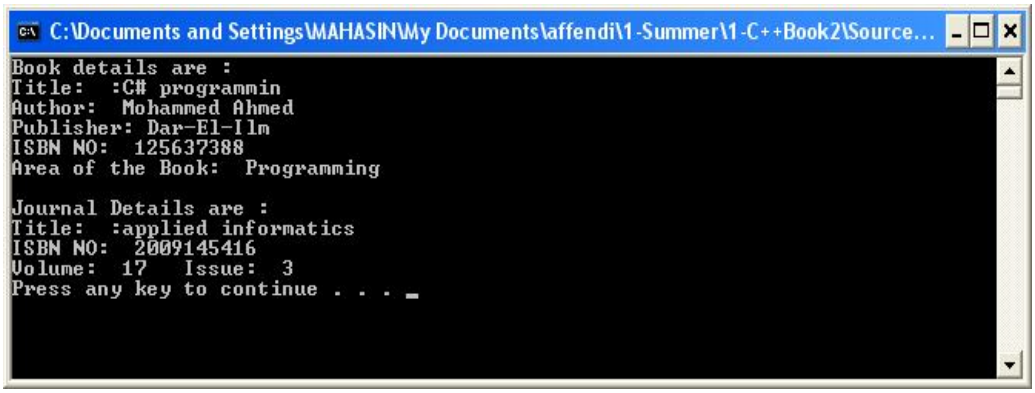
```
#include <iostream>
using namespace std;
class LibItem
{ protected:
    string index;
    string Title;
    long ISBN;
  public:
    LibItem(){}
    void SetIndex(string s) {index =s;}
    void SetTitle(string s) {Title = s;}
    void SetISBN(long x) {ISBN = x;}
    void PrintDetails()
    { cout<<"Tiltle is "<<Title<<endl;
      cout <<"ISBN is "<<ISBN<<endl;
      cout <<"Index is "<<index<<endl;
     }
  };
  class Book: public LibItem
  {private:
      string author;
      string publisher;
      int YearOfPub;
      string area;
    public:
      Book(){}
      void SetValues(string au, string pub,
string ar, int Year)
      {author=au; publisher=pub; area=ar;
YearOfPub=Year; }
      void PrintDetails()
      {cout<<"Title:  :"<<Title<<endl;
```

```
       cout <<"Author:  "<<author<<endl;
       cout <<"Publisher: "<<publisher<<endl;
       cout <<"ISBN NO:  "<<ISBN<<endl;
       cout <<"Area of the Book:  "<<area<<endl;
       }
    };
    class Journal: public LibItem
    { private:
          int VolNo;
          int IssueNo;
      public:
        Journal(){};
        void SetValues (int V, int No) {VolNo=V;
IssueNo=No;}
        void PrintDetails()
        {cout<<"Title:  :"<<Title<<endl;
         cout <<"ISBN NO:  "<<ISBN<<endl;
         cout <<"Volume:  "<<VolNo<<"   Issue:
"<<IssueNo<<endl;
      }

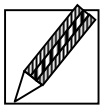
     };
int main(int argc, char* argv[])
{       Book B;
        Journal J;
        B.SetTitle("C# programmin");
        B.SetISBN(125637388);
        B.SetValues("Mohammed Ahmed", "Dar-El-
Ilm","Programming", 2005);
        J.SetTitle("applied informatics");
        J.SetValues(17,3);
        cout<<"Book details are :"<<endl;
        B.PrintDetails();
```

```
        cout <<endl<<"Journal Details are :
"<<endl;
        J.PrintDetails();
        system("pause");
        return 0;
}
```

عينة من تنفيذ البرنامج:



تدريب(3)



1- الكثير من الجامعات تفتح أكثر من شعبة للمقرر الواحد. كل الشعب تشترك في اسم المقرر، عدد الساعات المعتمدة، الكتاب المقرر، وتختلف في رقم القاعة، وقت المحاضرات، مدرس المادة، عدد الطلاب المسجلين. صمم فصيلة مشتركة (الفصيلة الأم) اسمها course ثم صمم فصيلة أخر اسمها section ترث الفصيلة course وتضيف إليها التفاصيل الخاصة بالشُعب. الفصيلة الأم ينبغي أن تحتوي على الدوال اللازمة لتغيير أو قراءة الخواص، ودالة لطباعة المقرر. الفصيلة section ينبغي أن تحتوي على الدوال اللازمة لتعديل الخواص الإضافية، و دالة لتعديل عدد الطلاب المسجلين. لاحظ أنه ينبغي حماية الخواص البيانية. مستوى الحماية للخواص البيانية ينبغي أن تكون private أو protected.

2- هب أن هناك فصيلة أخرى Professor تمثل أستاذ في الجامعة. افترض أن هناك خاصية في هذه الفصيلة إسمها Sections تحتوي على قائمة بالشعب التي ينيغي أن يدرِّسها الأستاذ. اكتب دالة تطبع عدد الطلاب في كل شعبة. لاحظ أن مستوى الحماية للمتغير "عدد الطلاب" في الفصيلة Section لايسمح للفصيلة Professor بالإستخدام المباشر، و عليه لابد من أن تكون الفصيلة Professor صديقة للفصيلة Section. عدِّل الفصيلة Section بحيث تسمح للفصيلة Professor بالوصول المباشر إلى عناصرها الخاصة.

2. الدوال و الفصائل الصديقة (friend functions and classes)

ذكرنا في الفقرات السابقة **أن العناصر الخاصة (private) لايمكن الوصول إليها إلا من داخل الفصيلة، و العناصر المحمية (protected) لايمكن الوصول إليها إلا من داخل الفصيلة أو الفصائل الوارثة. هذه القاعدة يمكن تجاوزها باستخدام** فكرة **الدوال و الفصائل الصديقة (friend functions and classes)**.

قاعدة هامة:

C++ **تسمح** لأي فصيلة بتعريف دوال وفصائل صديقة. هذه الفصائل والدوال الصديقة يجوز لها الوصول إلى العناصر الخاصة في الفصيلة و استخدامها.

1.2 الدوال الصديقة Friend Function

يتم تعريف الدوال الصديقة داخل الفصيلة بذكر كلمة friend متبوعة بعنوان الدالة.

التعريف الكامل للدالة يمكن أن يعطى داخل الفصيلة أو خارجها. مثلاً:

```
friend int MyFunction(int, int);
```

لاحظ أن الدالة الصديقة لا تعتبر عضواً في الفصيلة، و تستخدم كدالة مستقلة. من هذا المنطلق، **الدوال الصديقة قليلة الإستخدام** في أوساط البرمجة الكيانية، و **تستخدم في البيئات الهجينة التي تجمع بين البرمجة التقليدية و البرمجة الكيانية.**

عزيزي الدارس : المثال التالي يشرح كيف يتم تعريف الدوال الصديقة

مثال تطبيقي (5)

في المثال التالي، **الفصيلة Rectangle** تعرف **دالة صديقة ()CopyAndDouble:**

```
friend Rectangle CopyAndDouble (Rectangle);
```

عنوان الفصيلة يعطى داخل الفصيلة، أما التعريف الكامل للدالة فيعطى خارج الفصيلة:

```
Rectangle CopyAndDouble (Rectangle RectParam)
{
 Rectangle RectTarget;
 RectTarget.width = RectParam.width*2;
 RectTarget.height = RectParam.height*2;
 return (RectTarget);
}
```

الملفت للنظر أن الدالة CopyandDouble() تستقبل كياناً (كعنصر ممرر) وترجع كياناً من النوع Rectangle.



برنامج(5)

```
// friend functions
#include <iostream>
using namespace std;
class Rectangle {
    float width, height;
  public:
    void SetValues (float, float);
    float area (void) {return (width * height);}
    friend Rectangle CopyAndDouble (Rectangle);
};

void Rectangle::SetValues(float a, float b) {
  width = a;
  height = b;
}

Rectangle CopyAndDouble (Rectangle RectParam)
{
  Rectangle RectTarget;
  RectTarget.width = RectParam.width*2;
  RectTarget.height = RectParam.height*2;
  return (RectTarget);
```

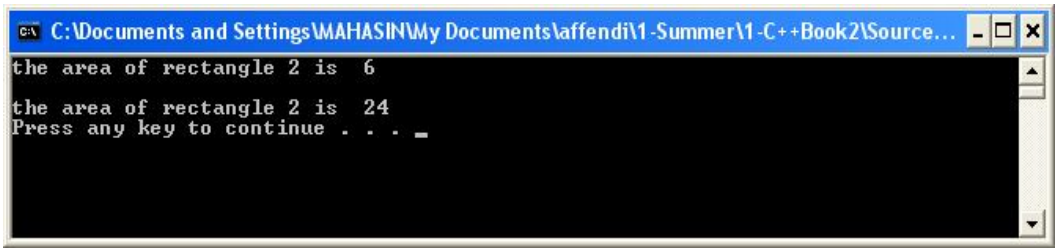
```
}

int main () {
  Rectangle Rect1, Rect2;
  Rect1.SetValues (2,3);
  cout <<"the area of rectangle 2 is  "<<
Rect1.area()<<endl;
  Rect2 = CopyAndDouble (Rect1);
  cout <<endl<<"the area of rectangle 2 is  "<<
Rect2.area()<<endl;
  system("pause");
  return 0;
}
```

ملاحظات على البرنامج:

1- لاحظ أن **الدالة CopyAndDouble()** تقوم **بتعديل العنصرين width و height للفصيلة Rectangle، على الرغم** من أن مستوى **الحماية** لهذين العنصرين هو **المستوى الأعلى (private)** وهذه الدالة **ليست عضواً** في الفصيلة.

2- **الدالة CopyAndDouble() استخدمت** داخل البرنامج **دون نسبتها** إلى أيّ فصيلة.



2.2 الفصائل الصديقة Friend Classes

الفصائل الصديقة أكثر استخداماً في أوساط البرمجة الكيانية.

تعريف الفصائل الصديقة داخل فصيلة معينة يتم بذكر الكلمة friend متبوعة بالكلمة الخاصة class ثم اسم الفصيلة الصديقة. مثلاً:

friend class MyClass;

المثال التالي يشرح كيفية تعريف و استخدام الفصائل الصديقة:

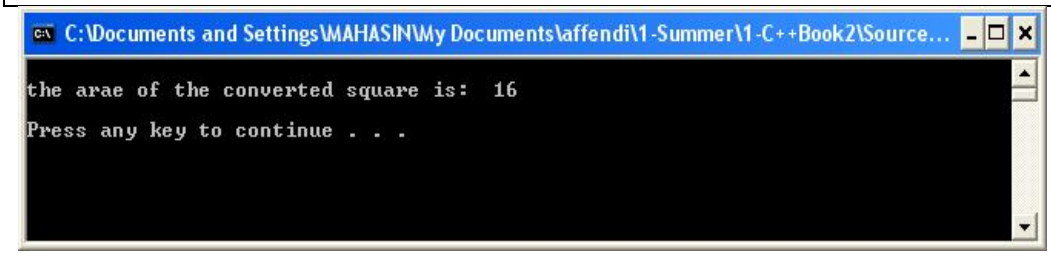
مثال (6)

في هذا المثال **الفصيلة Square تعلن** أن الفصيلة **Rectangle فصيلة صديقة.** بناءً على ذلك **تستطيع الدوال الأعضاء في الفصيلة Rectangle الوصول إلى العناصر الخاصة في الفصيلة Square.** مثلاً **الدالة convert** في **الفصيلة Rectangle أمكنها الوصول إلى العنصر side،** و هو **عنصر خاص في الفصيلة Square.**


برنامج(6)

```
// friend class
#include <iostream>
using namespace std;
class Square;
class Rectangle {
    float width, height;
  public:
    float area (void)
      {return (width * height);}
    void convert (Square a);
};
class Square {
  private:
    float side;
```

```
  public:
    void SetSide (float a)
      {side=a;}
    friend class Rectangle;
};
void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
int main () {
  Square sqr;
  Rectangle rect;
  sqr.SetSide(4);
  rect.convert(sqr);
  cout << endl<<"the arae of the converted
square is:  "
        <<rect.area()<<endl<<endl;
  system("pause");
  return 0;
}
```



ملاحظات على البرنامج:

1- **يبدأ** البرنامج **بتعريف عنوان الفصيلة Square دون إعطاء مواصفاتها بالكامل، وذلك لتمكين الفصيلة Rectangle من استخدام اسم الفصيلة Square في تعريف عناصرها.**

كان بالإمكان إعطاء التعريف الكامل للفصيلة Square قبل Rectangle، ولكن Square أيضاً تحتاج أن تستخدم Rectangle، و عليه لابد من إعطاء اسم أحدهما قبل الآخر.

2- **الدالة ()convert في Rectangle تسقبل كياناً من النوع Square و تحوله إلى مستطيل.**

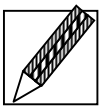
3- **الفصيلة Square تعلن أن الفصيلة Rectangle فصيلة صديقة حتى يتسنى للدالة convert الوصول إلى العناصر الخاصة في Square و القيام بمهمتها.**

أسئلة تقويم ذاتي



1. ما الداعي لاستخدام الفصائل و الدوال الصديقة؟
2. كيف يتم تعريف الفصائل و الدوال الصديقة؟
3. ما الذي يميز الفصائل و الدوال الصديقة عن غيرها من الفصائل و الدوال؟

تدريب(4)



تحتفظ وزارة التربية بسجلات للمدارس العامة و الخاصة. تشترك هذه المدارس في الإسم، تاريخ الإنشاء، المكان الجغرافي، عدد الطلاب. المدارس الخاصة تضيف إلى هذه المعلومات: رقم الترخيص، تاريخ الترخيص، متوسط المصاريف الدراسية السنوية، صاحب المدرسة. المدارس المهنية عبارة عن مدارس عامة تغطي تخصصات مختلفة وتعطي أنواع مختلفة من الشهادات. في هذه الحالة، المعلومات الإضافية المطلوبة هي: نوع تخصص المدرسة المهنية، مستوى الشهادة المعطاة. صمم شجرة الإرث لمدارس وزارة التربية، ثم صمم كل الفصائل المعنية و اختبرها في برنامج.

3. التركيب Composition

الطريقة الأخرى **لإعادة استخدام الكيانات** هي **التركيب**، و **يعني إدراج فصيلة داخل فصيلة أخرى.**

عملياً، يتم ذلك **بتعريف كيان تابع للفصيلة المدرجة داخل الفصيلة المعنية. مثلاً**، في المثال التالي، **ندرج الفصيلة Date في الفصيلة Person بتعريف كيان BirthDate** كمايلي:

```
Date BirthDate;  // an example of composition
```

انظر تفاصيل المثال أدناه.

مثال(7)



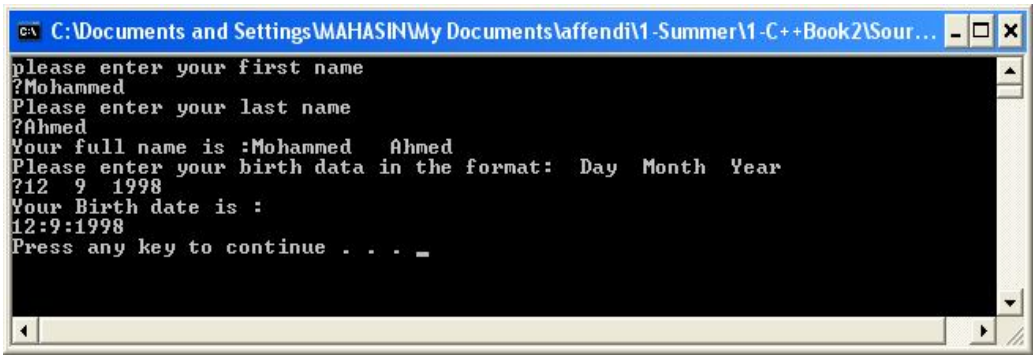
برنامج(7)

```
// Composition Example
#include <iostream>
#include <stdlib.h>
using namespace std;
class Date
{private:
    int Day;
    int Month;
    int Year;
 public:
     Date(){Day=0; Month=0; Year=0;}
     void SetDate(int d, int m, int y)
     { Day=d; Month=m; Year=y;}
     void PrintDate(void)
    { cout <<Day<<":"<<Month<<":"<<Year<<endl; }
};
class Person
{private:
```

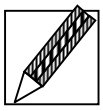
```
   string FirstName;
   string LastName;
   string Name;
   int Age;
 public:
   Date BirthDate;  // an example of composition
   Person();
   void SetFirstName(string);
   void SetLastName(string);
   string GetFirstName(string &);
   string GetLastName(string &);
   void PrintName();
};
Person::Person()
{ FirstName=""; LastName="";
}
void Person::SetFirstName(string Fname)
{ FirstName = Fname;
}
void Person::SetLastName(string Lname)
{ LastName= Lname;
}
string Person::GetFirstName(string &Fname)
{ Fname = FirstName;
  return FirstName;
}
string Person::GetLastName(string &Lname)
{ Lname = LastName;
  return LastName;
}
void Person::PrintName()
{ cout <<FirstName<<"   "<<LastName<<endl;
}
```

```
int main()
{ Person P;
  string FName, LName;
  cout <<"please enter your first name\n?";
  cin >>FName;
  cout <<"Please enter your last name\n?";
  cin >>LName;
    P.SetFirstName(FName);
    P.SetLastName(LName);
    cout<<"Your full name is :";
    P.PrintName();
    //Now testing the Birthdate Subcomponent
    int D,M,Y;
    cout
     <<"Please enter your birth data in the
format:  Day  Month  Year\n?";
    cin >>D>>M>>Y;
    P.BirthDate.SetDate(D,M,Y);
    cout <<"Your Birth date is :\n";
    P.BirthDate.PrintDate();
    system("pause");
    return 0;
}
```

عينة من تنفيذ البرنامج:



تدريب(5)



صمم فصيلة Circle تستخدم الفصيلة Point التي صممتها في السؤال السابق بطريقة التركيب (composition)، و تستخدم دوال الفصيلة المدرجة في تمثيل و تعديل مركز الدائرة الذي هو نقطة هندسية. بالإضافة إلى ذلك، تحتوي الفصيلة Circle على دوال تحسب مساحة و محيط الدائرة. هذا بالطبع يتطلب تمثيل نصف القطر في الفصيلة.

مثال(8)

هذا **المثال يجمع بين الإرث و التركيب**. الفصيلة **Worker ترث** الفصيلة **Person**. **داخل** الفصيلة **Worker استخدمنا** الفصيلة **Timer لتعريف زمن الوصول والخروج للعمال: ArrivalTime و CheckOutTime**. بهذا نكون **أدرجنا** الفصيلة **Timer داخل** الفصيلة **worker وأعدنا استخدمناها بطريقة التركيب.**



برنامج(8)

```
//Composition - Timer
#include <iostream>
#include <stdlib.h>
using namespace std;
class Person
{private:
   string FirstName;
   string LastName;
   string Name;
   int Age;
 public:
   Person();
   void SetFirstName(string);
   void SetLastName(string);
```

```
   string GetFirstName(string &);
   string GetLastName(string &);
   void PrintName();
};
Person::Person()
{ FirstName=""; LastName="";
}
void Person::SetFirstName(string Fname)
{ FirstName = Fname;
}
void Person::SetLastName(string Lname)
{ LastName= Lname;
}
string Person::GetFirstName(string &Fname)
{ Fname = FirstName;
  return FirstName;
}
string Person::GetLastName(string &Lname)
{ Lname = LastName;
  return LastName;
}
void Person::PrintName()
{ cout <<FirstName<<"   "<<LastName<<endl;
}

class Timer
 { private:
     int hr;
     int min;
     int sec;
   public:
    void SetTime(int, int, int);
    void GetTime( int & , int &, int &) const;
```

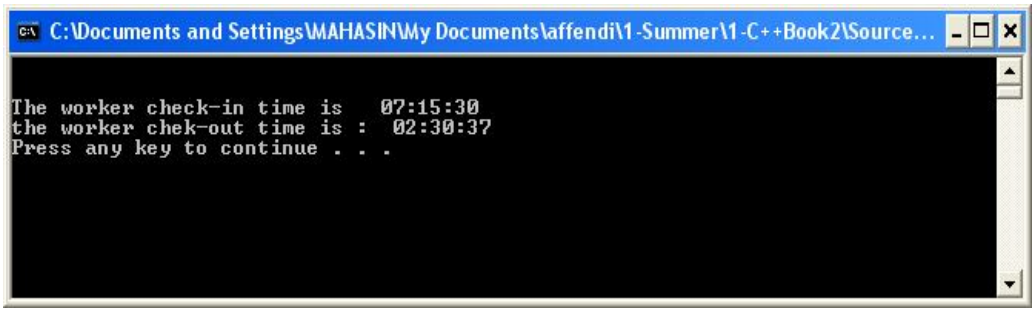
```
    void PrintTime() const;
    void IncrementSeconds();
    void IncrementMinutes();
    void IncrementHours();
    void EqualTime(const Timer &OtherTimer)
const;
};
void Timer::SetTime(int hours, int minutes, int
seconds)
{ if (hours>=0 && hours<24)
    hr = hours;
  else hr=0;
  if (minutes>=0 && minutes<60)
    min = minutes;
  else min=0;
  if (seconds >=0 && seconds <60)
     sec = seconds;
  else sec=0;
}
void Timer::GetTime(int &hours, int &minutes,
int &seconds) const
{   hours=hr;  minutes=min; seconds=sec;}

void Timer::PrintTime() const
{ if (hr<10)
    cout <<"0";
  cout <<hr<<":";
  if (min<10)
    cout <<"0";
  cout <<min<<":";
  if (sec<10)
     cout <<"0";
   cout <<sec;
```

```
}
void Timer::IncrementHours()
{   hr++;
    if (hr>23) hr=0;
}
void Timer::IncrementMinutes()
{   min++;
    if (min>59) min=0;
}
void Timer::IncrementSeconds()
{   sec++;
    if (sec>59) sec=0;
}
class Worker: public Person
{ private:
   float Rate,Hours,Salary;
  public:
    Timer ArrivalTime, CheckOutTime;
    void SetValues(float r, int h)
    { Rate =r; Hours=r; }
    float ComputeSalary()
    { Salary=Rate * Hours;
      return Salary;
    }
 };
int main ()
{
  Worker W;
  W.ArrivalTime.SetTime(07,15,30);
  cout <<endl<<endl<<"The worker check-in time
is   ";
  W.ArrivalTime.PrintTime();
  cout<<endl;
```

```
  W.CheckOutTime.SetTime(2,30,37);
  cout <<"the worker chek-out time is :  ";
  W.CheckOutTime.PrintTime();
  cout<<endl;
  system("pause");
  return 0;
}
```

عينة من تنفيذ البرنامج:



أسئلة تقويم ذاتي

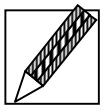


1- ما الفرق بين الإرث و التركيب؟

3- ما الفوائد العامة للإرث و التركيب؟

4- ما الداعي لاستخدام الفصائل و الدوال الصديقة؟ كيف يتم تعريف الفصائل و الدوال الصديقة؟ و مالذي يميزها عن غيرها من الفصائل و الدوال.

تدريب(6)



صمم فصيلة Address تحتوي على تفاصيل العنوان البريدي لأي شخص أو مؤسسة مع الدوال اللازمة لتعديل، أو قراءة هذه التفاصيل. صمم فصيلة أخرى تستخدم Person وتستخدم الفصيلة Address بطريق التركيب لإضافة خاصية العنوان لأي شخص. بالإضافة إلى ذلك، الفصيلة Person تحتوي على التفاصيل الأخرى

الخلاصة

عملية الإرث هي إحدى الطرق لإعادة استخدام الكيانات في لغات البرمجة الكيانية تعني انتساب أو انتماء فصيلة إلى فصيلة، أو إلى مجموعة فصائل أخرى بهدف امتلاك ما بداخل هذه الفصائل من الخواص، والعمليات، دون الحاجة إلى إعادة تعريف أو حتى معرفة كيفية عمل هذه الفصائل الأخرى .

مستويات الحماية للعناصر الموروثة في الفصيلة الابن تشمل العناصر العامة في الفصيلة الأم ويمكن الوصول إليها و استخدامها في الفصيلة نفسها أو من أي فصيلة أخرى. العناصر المحمية (protected) و يمكن الوصول إليها و استخدامها من أعضاء الفصيلة الأم، و أعضاء الفصائل الوارثة. أما العناصر الخاصة فلا يمكن الوصول إليها إلا من أعضاء الفصيلة الأم نفسها، يمكن تجاوزهذا باستخدام فكرة الدوال، و الفصائل الصديقة (friend functions and classes) ،حيث تسمح C++لأي فصيلة بتعريف دوال و فصائل صديقة. هذه الفصائل و الدوال الصديقة يجوز لها الوصول إلى العناصر الخاصة في الفصيلة و استخدامها.

أما الطريقة الأخرى لإعادة استخدام الكيانات هي التركيب، و يعني إدراج فصيلة داخل فصيلة أخرى.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس في الوحدة التالية سنستعرض موضوعين هامين يزيدان من متعة البرمجة الكيانية هما تحميل الدوال وتحميل العوامل.

إجابات التدريبات

تدريب (1)

1-

```
#include<iostream.h>
#include<conio.h>
const pi=3.14 ;
class CircularShape
{
float radius;
public:
CircularShape(float r)
{
radius= r;
}
float CicularArea(void)
{
return(pi*radius*radius);
}
CircularSurround(void)
{
return(2*pi*radius);
}
};
class Cylinder :public CircularShape
{
float hieght;
public:
float SetHieght(float h)
{
hieght= h;
}
```

```
float CylinderArea(void)
{
return(CicularArea()*hieght);
}
float CircularSurround(void)
{
return(2*(CircularSurround())+hieght);
}
};
main()
{
Cylinder  MyCil;
MyCil.SetHieght(4);
MyCil.CicularArea();
MyCil.CircularSurround();
cout<<MyCil.CylinderArea();
cout<<MyCil.CylinderArea();
getch();
}
```

تدريب (2)

1-

```
#include<iostream.h>
#include<conio.h>
class Point
{
float X_Axis,Y_Axis;
public:
Point(float x,float y)
{
X_Axis= x;
Y_Axis= y;
}
```

```
float PrintPoint(void)
{
cout<<"the x Axis  is :  "<< X_Axis<<endl;
cout<<"the y Axis  is :  "<< Y_Axis<<endl;
}
};
main()
{
Point P(4.5,9.5);
P.PrintPoint();
 getch();
}
```

2-

```
#include<iostream.h>
#include<conio.h>
const pi=3.14 ;
class Circle
{
float radius;
public:
float SetRadius(float r)
{
radius= r;
}
float CircularArea(void)
{
return(pi*radius*radius);
}
CircularSurround(void)
{
return(2*pi*radius);
}
};
```

```
class Cylinder :public Circle
{
float hieght;
public:
float SetHieght(float h)
{
hieght= h;
}
float CylinderArea(void)
{
return(CircularArea()*hieght);
}
float CircularSurround(void)
{
return(2*(CircularSurround())+hieght);
}
};
main()
{
Cylinder  MyCil;
MyCil.SetRadius(5);
MyCil.SetHieght(4);
MyCil.CircularArea();
MyCil.CircularSurround();
cout<<MyCil.CylinderArea();
cout<<MyCil.CylinderArea();
getch();
}
```

تدريب (3)

1-

```
#iclude<iostream.h>
#include<conio.h>
class Course
{
string CourseName,BookName;
int CreditHours;
public:
void GetCourse(string CN ,string  BN)
{
CourseName= CN;
BookName=BN;
}
void GetHours(int CHs)
{
CreditHours = CHs;
}
void ShowDetails(void)
{
cout<<CourseName<<endl;
cout<<BookName<<endl;
cout<<CreditHours<<endl;
}
};

class Section:public Course
{
string  SectionName;
int RoomNo, LectTime, No_of_Students  ;
public:
void GetSection(string SN)
```

```
{
SectionName = SN;
}
void GetData(int RN,int NS)
{
RoomNo= RN;
 No_of_Students= Ns;
 }
 void GetTime(int Time)
 {
  LectTime = Time;
  }
  void ShowData(void)
{
cout<<SectionName<<endl;
cout<<RooNo<<endl;
cout<<LectTime<<endl;
cout<<No_of_Students<<endl;

}
};
main()
{
Section S;
S.GetCourse('C++' ,'Programming');
S.GetHours(24) ;
S.ShowDetails();
S.GetSection('Computer');
S.GetData(5,45);
S.GetTime();
s.ShowData();
getch()
}
```

تدريب(4)

```
#include<iostream.h>
#include<conio.h>
class School
{
string Name;
string Location;
int EstablishDate;
int No_of_Students;
Public:
void GetNameLoc(string N,string Loc);
{
Name N;
Location= Loc;
}
void GetDate_No(int D,int No)
{
EstablishDate =D ;
No_of_Students= No;
}
void ShowDetails(void)
{
cout<<Name<<endl;
cout<<Location<<endl;
cout<<EstablishDate<<endl;
cout<<No_of_Students;
}
class privateSchool : public School
{
int AuthNo;
int AuthDate;
float AvgAnnFees;
```

```
string Woner;
public:
void GetPrivateData(int ANo,int ADate)
{
AuthNo = ANo;
AuthDate= ADate;
}
void GetFees(flost AvgF)
{
 AvgAnnFees = AvgF;
 }
void Show privateData(void)
{
cout<<AuthNo<<endl;
cout<<AuthDate<<endl;
cout<<AvgAnnFees<<endl;
cout<<Woner<<endl;
}
class TechSchool : public School
{
string Trade;
string CertificateLevel;
public:
void GetTechData(string Tr,CL)
{
Trade = Tr;
CertificateLevel = CL;

  }
void ShowTechData(void)
{
cout<<Trade<<endl;
cout<<CertificateLevel<<endl;
```

```
}
main()
PrivateSchool Pr;
TechSchool Tch;
Pr.GetNameLoc('omer','Khartoum');
Pr.GetDate_No(1999,330);
Pr.GetPrivateData(112,1998);
Pr.GetFees(20000);
Pr.ShowDetails();
Pr.Show privateData();
Tch.GetNameLoc('khalid','Omdurman');
Tch.GetDate_No(1978,330);
Tch.GetTechData('Electric','high Level');
Tch.ShowDetails();
Tch.ShowTechData();
}
```

تدريب (5)

```
#include<iostream.h>
#include<conio.h>
const Pi=3.14;
class Point
{
float X_Axis,Y_Axis;
public:
void SetPointX(float x)
{
X_Axis= x;

}
void SetPointY(float y)
{
Y_Axis= y;
}
float GetPointX(void)
{
return X_Axis ;

}
float GetPointY(void)
{
return Y_Axis ;
}
};
class Circle
 {
float Radius;
public:
Point P;
```

```
};
main()
{
float X,Y,R;
Circle C;
C.P.SetPointX(3);
C.P.SetPointX(5) ;
C.P.SetPointY(4)  ;
C.P.SetPointY(2)   ;
cout<<"X  = "<< 0.5*((C.P.GetPointX())+(C.P.GetPointX()))<<endl;
cout<<"Y  = "<< 0.5*(C.P.GetPointY())+((C.P.GetPointY()))<<endl ;
R= (X+Y)/2;
cout<<"Radius = " <<R<<endl;
cout<<"Area = "<<Pi*(R*R)<<endl;
cout<<"Surround = " <<2*Pi*R<<endl;
 getch();
}
```

تدريب(6)

```
#include<iostream.h>
#include<stdlib,h>
using namespace std;
class Address
{
string Street;
int BuilNo;
int PoBox,ZIP;
public:
void SetStreet(string Str)
{
Street = Str;
```

```
}
void SetDetails(int BNo,int Pb,int Z)
{
BuildNo= BNo;
PoBox= PB;
Zip= Z;
}
void Print Address(void)
{
cout<<"your addres is : "<<endl;
cout<<Street<<BuildNo<<PoBox<<Zip <<endl;
}
 };
class Person
string FirstName(string);
string LastName(string);
string Name ;
int Age;
public:
person();
Address Add;
void SetFirstName(string);
void SetLastName(string);
string GetFirstName(string &);
string GetLatName(string &);
void PrintName();
};
Person::Person()
{
FirstName= "";
LastName= "";
}
void Person::GetFirstName(string Fname)
```

```
{
FirstName= Fname;
return Fname;
}
void Person::LastFirstName(string lname)
{
LastName= Lname;
return Lname;
}
void Person::PrintName()
{
cout<<FirstName <<"  "<<LastName<<endl;
int main()
{
Person P;
string FName,LName;
cout<<"please enter your First name \n ?";
cin>>FName;
cout<<"please enter your Lasst name \n ?";
cin>>LName;
P.Add.SetStreet('Jamaa');
P.Add.GetStreet();
P.Add.GetDetails();
P.Add.SetDetails(23,45,22);
P.SetFirstName(FName);
P.SetLastName(Lname);
cout<<" Your full Name is :";
P.PrintName();
cout<<"Your Address is : " <<endl;
P.Add.PrintAddress();
getch();
 }
```

```
#include<iostream.h>
#include<stdlib,h>
using namespace std;
class Address
{
string Street;
int BuilNo;
int PoBox,ZIP;
public:
void SetStreet(string Str)
{
Street = Str;
}
void SetDetails(int BNo,int Pb,int Z)
{
BuildNo= BNo;
PoBox= PB;
Zip= Z;
}
void Print Address(void)
{
cout<<"your addres is : "<<endl;
cout<<Street<<BuildNo<<PoBox<<Zip <<endl;
}
 };
class Person
string FirstName(string);
string LastName(string);
string Name ;
int Age;
public:
person();
Address Add;
```

```
void SetFirstName(string);
void SetLastName(string);
string GetFirstName(string &);
string GetLatName(string &);
void PrintName();
};
Person::Person()
{
FirstName= "";
LastName= "";
}
void Person::GetFirstName(string Fname)
{
FirstName= Fname;
return Fname;
}
void Person::LastFirstName(string lname)
{
LastName= Lname;
return Lname;
}
void Person::PrintName()
{
cout<<FirstName <<"  "<<LastName<<endl;
int main()
{
Person P;
string FName,LName;
cout<<"please enter your First name \n ?";
cin>>FName;
cout<<"please enter your Lasst name \n ?";
cin>>LName;
P.Add.SetStreet('Jamaa');
```

```
P.Add.GetStreet();
P.Add.GetDetails();
P.Add.SetDetails(23,45,22);
P.SetFirstName(FName);
P.SetLastName(Lname);
cout<<" Your full Name is :";
P.PrintName();
cout<<"Your Address is : " <<endl;
P.Add.PrintAddress();
getch();
 }
```

مسرد المصطلحات

- **المكونات المرئية (visual components)**

المكونات المرئية عبارة عن كيانات متكاملة تقوم بحل مسائل معينة وتستخدم رمزاً مرئياً في الدلالة على نفسها.

- **الإرث Inheritance**

إنتماء فصيلة إلى فصيلة أو فصائل أخرى بغية استخدام ما بداخلها من الخواص و الدوال.

- **الإرث الأُحادي Single Inheritance**

هذا النوع من الإرث كل مكونات العناصر الموروثة تصبح جزءً من الفصيلة الوارثة ويمكن استخدامها كأنها جزء من الفصيلة.

- **الإرث المتعِدد Multiple Inheritance**

هذا النوع من الإرث يمكن **نسبة الفصيلة الوارثة إلى أكثر من.**

- **شجرة الأرث** Inheritance Tree

هي شجرة، جذرها الفصيلة الأم وفروعها الفصائل الوارثة.

- **الدوال الصديقة Friend Functions:**

يتم تعريفها داخل الفصيلة بذكر كلمة friend متبوعة بعنوان الدالة. التعريف الكامل للدالة يمكن أن يعطى داخل الفصيلة أو خارجها.

- **الفصائل الصديقة Friend Classes**

يتم تعريفها داخل فصيلة معينة يتم بذكر الكلمة friend متبوعة بالكلمة الخاصة class ثم اسم الفصيلة الصديقة.

- **التركيب Composition**

يعني إدراج فصيلة داخل فصيلة أخرى.

المصادر والمراجع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.


الوحدة الرابعة
تحميل الدوال والعوامل
Functions and Operators
Overloading

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**125**
تمهيد	**125**
أهداف الوحدة	**125**
1. تحميل الدوال	**127**
1.1 تحميل الدوال واعادة تعريف الدوال	128
2. تحميل العوامل	**129**
1.2 تحميل العوامل الثنائية	129
3. المؤشر this	**137**
4. تحميل العوامل الأحادية	**139**
5. المتغيرات الدائمة في الفصائل	**144**
الخلاصة	**144**
لمحة مسبقة عن الوحدة التالية	**144**
إجابات التدريبات	**145**
مسرد المصطلحات	**151**
المراجع	**152**

المقدمة

تمهيد

عزيزي الدارس، ذكرنا في الوحدة الماضية أنه يمكن أن يكون للفصيلة الواحدة أكثر من دالة إنشاء، بشرط أن تختلف هذه الدوال في عدد العناصر الممرة. كل دوال الإنشاء تشترك في الاسم (الذي هو إسم الفصيلة) و تختلف في عدد العناصر الممررة. تعدد دوال الإنشاء في فصيلة معينة و اشتراكها في الإسم تعتبر صورة من صور تحميل الدوال (function overloading)، التي تسمح بها C++ تسهيلاً على المطورين. كلمة "تحميل" هنا تعني إعطاء أكثر من معنى للدالة، أو العامل المعين داخل نطاق معين. الفائدة التي نجنيها في حالة تحميل الدوال هي إمكانية استخدام نفس الدالة للقيام بأغراض مختلفة، وهذا يريح المصمم و المستخدم من تشابك الدوال المختلفة. أما في حالة تحميل العوامل فالفائدة أعم و أعظم. تحميل العوامل يعني أنه بالإمكان إعادة برمجة الكثير من رموز العمليات كعامل الجمع "+" وعامل الضرب "*" مثلاً، بحيث يمكن استخدامها مباشرة في جمع و ضرب المصفوفات، الأعداد المركبة و غيرها من الكيانات. بالإضافة إلى عملية التحميل نغطي مواضيع جانبية أخرى هذا بالإضافة إلى ما تعودنا عليه من تدريبات و أسئلة تقويم ذاتي تعينك على فهم الدرس وبلوغ الأهداف المرجوة منه.

أهداف الوحدة



بعد فراغك عزيزي الدارس من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تستخدم تحميل الدوال.
2. تستخدم تحميل العوامل.
3. تشرح إستخدام المؤشر this.
4. توضح المتغيرات الدائمة في الفصائل.

1. تحميل الدوال Functions Overloading

تحميل الدوال يعني **استخدام** اسم واحد **لتعريف** أكثر من دالة في النطاق الواحد، **بشرط** إختلاف هذه الدوال في عدد العناصر، أو نوعها. **ترجمان** C++ **يميز** بين هذه الدوال **بعدد العناصر، أو نوعها.** المثال التالي يشرح عملية تحميل الدوال:

مثال(1)

هذا المثال **يحتوي على حالتين توضحان عملية تحميل الدوال بصورتيها: اختلاف عدد العناصر و اختلاف الأنواع. الدالة ()divide تستخدم** مرةً **لقسمة الأعداد الصحيحة،** ومرة أخرى **لقسمة الأعداد الحقيقية. الترجمان يستخدم** إحدى **الدالتين** بناءً على **نوع العناصر الممررة. الدالة ()MyMax** تستخدم مرةً **لحساب الأكبر** بين **عددين،** و تستخدم مرةً أخرى **لحساب الأكبر** بين **ثلاثة أعداد. الترجمان** ينظر في هذه الحالة إلى **عدد العناصر الممررة،** ثم يستخدم الأنسب منهما.

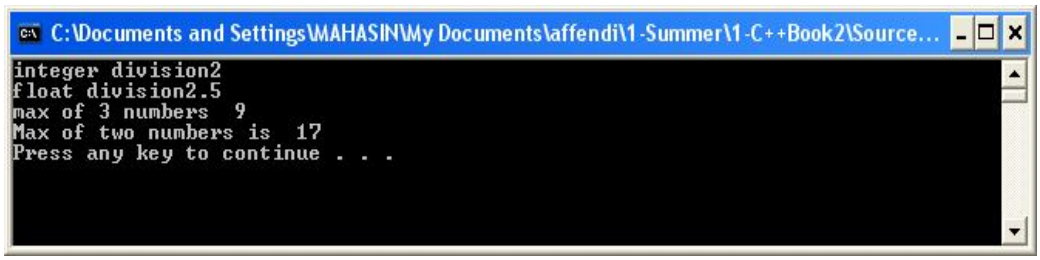
برنامج(1)

```
// overloaded functions
#include <iostream>
using namespace std;
int divide (int a, int b)
{
  return (a/b);
}

float divide (float a, float b)
{
  return (a/b);
}
int MyMax(int a, int b)
{ if (a>b) return (a); else return(b);}
int MyMax(int a, int b, int c)
{int m;
 if (a>b) m=a; else m=b;
 if (c>m) m=c;
```

```
 return(m);
}
int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout <<"integer division"<< divide (x,y)<<endl;
  cout << "float division"<<divide (n,m)<<endl;
  cout <<"max of 3 numbers
"<<MyMax(9,4,7)<<endl;
  cout <<"Max of two numbers is
"<<MyMax(17,3)<<endl;
  system("pause");
  return 0;
}
```

عينة من تنفيذ البرنامج:



1.1 تحميل الدوال و إعادة تعريف الدوال

لاحظ أن **تحميل الدوال حالة خاصة** من **إعادة تعريف الدوال** وتعني **استخدام أكثر من صورة للدالة في نطاق واحد** (same scope) من البرنامج (اسم واحد وأدوار مختلفة). باختصار، **تحميل الدوال** نعني به **اختلاف أدوار الدالة في نطاق واحد** (فصيلة واحدة أو مستوى واحد من البرنامج)، **ويشترط** فيه **اختلاف العناصر و الأنواع**.

الصورة الأخرى من **إعادة التعريف** هي أن تقوم **فصيلة وارثة بإعادة تعريف دالة موروثة**. في هذا الحالة **لا يشترط** اختلاف عدد العناصر و لا اختلاف الأنواع، ذلك أن النطاق مختلف (فصيلة أخرى). **هذا النوع** من إعادة التعريف **يعني اختلاف أدوار الدالة**

من نطاق إلى نطاق (من فصيلة إلى أخرى) **و لا يشترط** فيه **اختلاف العناصر و لا الأنواع.**

أسئلة تقويم ذاتي



1- ماذا نعني بتحميل الدوال؟

2- ما الفرق بين تحميل الدوال و إعادة تعريف الدوال في الكيانات الوارثة؟

2. تحميل العوامل Operator Overloading

من المعلوم أنّ C++ توفر الكثير من العوامل الحسابية، و عوامل المقارنة التي تستخدم مع البيانات العادية. قد يتساءل المرء: هل بالإمكان إعادة برمجة هذه العوامل بحيث تستخدم مع الكيانات، و الأنواع المركبة؟ الإجابة، لحسن الحظ، نعم. **C++ تسمح بتحميل العوامل مثلما تسمح بتحميل الدوال. كل العوامل الثنائية و الأحادية يمكن تحميلها بسهولة.** مثلاً يمكن **تحميل عامل الجمع** "+" **بحيث يستخدم في جمع عددين مركبين، أو مصفوفتين** كما سنرى في بعض الأمثلة أدناه.

1.2 تحميل العوامل الثنائية

الصورة العامة لتحميل العوامل الثنائية هي:

```
return-type operator op-sign (const ClassName&
OtherObject) const
{ ……
   algorithm
   return value;
}
```

حيث return-type هو نوع القيمة التي ترجعها الدالة، operator كلمة خاصة تستخدم في تعريف العوامل، op-sign هو رمز العملية المراد إجراؤها، ClassName هو اسم الفصيلة التي يتم التحميل من أجلها.

مثلاً، بالنسبة للمستطيل أدناه، يمكن تحميل عامل الجمع كما يلي:

```
Rectangle operator+ (const Rectangle Rect) const
   {Rectangle temp;
    temp.width=width+ Rect.width;
    temp.height=height+Rect.height;
    return temp;
     }
```

في هذه الحالة نوع القيمة التي ترجعها الدالة (return-type) هو Rectangle، رمز العملية (op-sign) هو "+"، و اسم الفصيلة التي يتم التحميل من أجلها هو Rectangle.

المغزى العام للتعريف أعلاه هو ان الفصيلة ستقوم باستخدام عامل الجمع الثنائي "+" لإجراء عملية جمع من نوع خاص. الطرف الأول في هذه العملية هو الفصيلة المالكة (التي قامت بالتعريف). الطرف الثاني هو العنصر المعطى بين قوسين بعد رمز العملية (في هذه الحالة Rect). عملية الجمع يتم إجراؤها بالصورة المذكورة بين المعرجتين {}(صلب الدالة).

بناءً على ذلك، يمكن استخدام العامل المحمل داخل البرنامج بالصورة التالية:

```
Rectangle Rect1(3,4), Rect2(6,7);
Rectangle Rect3;
Rect3 = Rect1 + Rect2;  //استخدام عامل الجمع المحمل
```

المثال التالي يوضح كيفية التحميل الثنائية بشيءٍ من التفصيل:

مثال (2)

في هذا المثال **نقوم بتحميل عامل الجمع** "+"، **عامل الضرب** "*" **و عامل المقارنة** "==" بحيث **تستخدم في معالجة المستطيلات**. كما هو معروف، **المستطيل له خاصتان أساسيتان:** الطول و العرض. عملية **جمع مستطيلين ينتج** منها **مستطيل ثالث، طوله** حاصل **جمع طولي المستطيلين**، و **عرضه** حاصل **جمع عرضي المستطيلين. عملية الضرب تعرَّف بنفس الطريقة، وكذلك عملية التساوي.**



برنامج(2)

```
// operator overloading
#include <iostream>
using namespace std;
class Rectangle {
    float width, height;
  public:
    Rectangle (){}
    Rectangle (float,float); //using a
constructor function
    Rectangle operator+ (const Rectangle Rect)
const
    {Rectangle temp;
      temp.width=width+ Rect.width;
      temp.height=height+Rect.height;
      return temp;
    }
    Rectangle operator* (const Rectangle Rect)
const
    {Rectangle temp;
      temp.width=width*Rect.width;
      temp.height=height*Rect.height;
      return temp;
    }
    bool operator== (const Rectangle Rect) const
    {return (width==Rect.width) && (height ==
Rect.height); }
    float area (void)
```

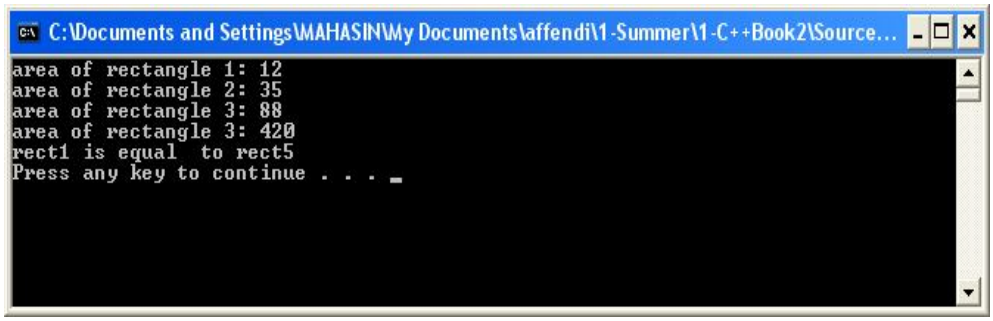
```
     {float x=width*height; return x;}
};
Rectangle::Rectangle (float w, float h)
//defining the constructor
 {
  width = w;
  height = h;
}
//Rectangle Rectangle::operator+ (const Rectangle
Rect) const
int main () {
  Rectangle rect1(3.0,4.0); //declare an object
of type Rectangle
  Rectangle rect2(5.0, 7.0); //another rectangle
  Rectangle rect3, rect4;
  rect3=rect1+rect2;
  rect4=rect1*rect2;
  Rectangle rect5(3.0,4.0);
  cout << "area of rectangle 1: " <<
rect1.area()<<endl;  //compute and print the area
  cout << "area of rectangle 2: " <<
rect2.area()<<endl;
  cout << "area of rectangle 3: " <<
rect3.area()<<endl;
  cout << "area of rectangle 3: " <<
rect4.area()<<endl;
  if (rect1==rect5)
      cout<<"rect1 is equal  to rect5 "<<endl;
  system("pause");
  return 0;
}
```

لاحظ أن **نوع القيمة المرجعة للعامل المحمل "==" هو bool. لاحظ ايضاً استخدام عاملي الجمع و الضرب داخل البرنامج:**

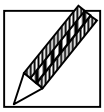
```
Rectangle rect3, rect4;
 rect3=rect1+rect2;
 rect4=rect1*rect2;
```

عامل المقارنة (التساوي) استخدمناه في الجملة الشرطية:

```
if (rect1= =rect5)
    cout<<"rect1 is equal  to rect5 "<<endl;
```



تدريب (1)



صمم دالة محمّلة Volume() يمكن استخدامها في حساب حجم صندوق أو أسطوانة. الصندوق قد يكون مكعب، أو متوازي مستطيلات. في حالة الأسطوانة، العناصر المطلوبة هي نصف قطر القاعدة، وطول الأسطوانة. في حالة المكعب هناك عنصر واحد هو طول ضلع المكعب، أما في حالة متوازي المستطيلات فهناك ثلاثة عناصر: طول، وعرض القاعدة، ثم ارتفاع متوازي المستطيلات. لاحظ أنه ينبغي تصميم ثلاثة صور للدالة Volume().

مثال(3)

في هذا المثال نقوم **بتحميل عامل الجمع** بحيث يمكن **استخدامه في جمع عددين مركبين. عملية جمع الأعداد المركبة يمكن تعريفها كما يلي:**

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

حيث a,c هي **الأجزاء الحقيقية** من **الأعداد المركبة**، b,d هي **الأجزاء التخيلية.**



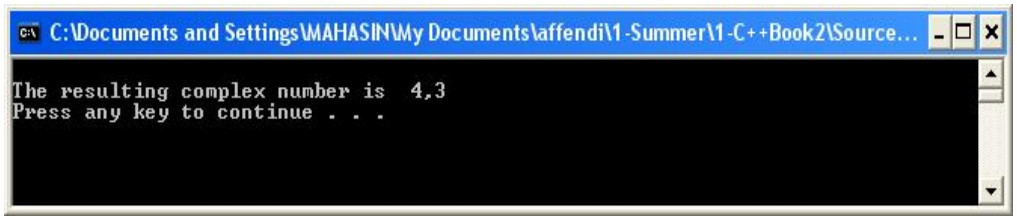
برنامج(3)

```
// Complex Nos: overloading operators example
#include <iostream>
using namespace std;
class Complex {
  public:
    float real,img;
    Complex () {};
    Complex (float,float);
    Complex operator + (Complex); // عنوان العامل
المحمل
};

Complex::Complex (float a, float b) {
  real = a;
  img = b;
}

Complex Complex::operator+ (Complex param)
//تعريف العامل المحمل
 {
  Complex temp;
  temp.real = real + param.real;
  temp.img = img + param.img;
  return (temp);
```

```
}
int main () {
  Complex a (3,1);
  Complex b (1,2);
  Complex c;
  c = a + b;   //استخدام العامل المحمل
  cout << endl<<"The resulting complex number
is  "
       <<c.real << "," << c.img<<endl;
  system("pause");
  return 0;
}
```



ملاحظات على البرنامج:

هذه المرة **تم تحميل العامل على مرحلتين:** في **المرحلة الأولى** أعطينا **عنوان التعريف فقط.** في **المرحلة الثانية** أعطينا **تفصيل التعريف.** لاحظ أنه **لابد من نسبة العامل إلى الفصيلة الأم** إذا عرفنا العامل خارج الفصيلة:

```
Complex Complex::operator+ (Complex param) //تعريف العامل المحمل
 {
  Complex temp;
  temp.real = real + param.real;
  temp.img = img + param.img;
  return (temp);
}
```

هنا **Complex الأولى** هي **نوع الناتج،** و **Complex الثانية** هي **إسم الفصيلة صاحبة العامل.**

أسئلة تقويم ذاتي



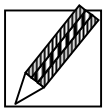
1- ماهي الصورة العامة لتحميل العوامل الثنائية في++C

2- حمِّل عامل الإسناد "=" بحيث يمكن استخدامه في إسناد مصفوفة Vector إلى أخرى.

3- أضف إلى الفصيلة complex أعلاه عامل الضرب المحمل "*" بحيث يمكن استخدامه مباشرةً في ضرب عددين مركبين. عملية ضرب عددين مركبين تعرّف كمايلي:

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

تدريب(2)



1. جمع المصفوفات عملية روتينية نقوم بها في كثيرٍ من التطبيقات وتتطلب نوعاً من أوامر التكرار. هذه العملية يمكن اختصارها بتحميل عامل الجمع "+" ليقوم بجمع مصفوفتين. صمم فصيلة Vector تقوم مقام المصفوفة. داخل هذه المصفوفة حمِّل عامل الجمع بحيث يقوم بجمع مصفوفتين.

2. بافتراض وجود فصيلة clock تشير إلى الزمن بالساعة، الدقيقة والثانية، قم بتحميل عامل الجمع "+"، بحيث يمكن استخدامه في جمع كيانين ينتميان إلى الفصيلة clock. مثلاً، إذا عرفنا كيانين:

```
clock  a(03,15,30),b(12,45,12);
```

فيمكن استخدام عامل الجمع لجمعهما في كيان ثالث كايلي:

```
c = a+b;
```

3. المؤشر "this"

ذكرنا في الوحدات الماضية **أن الدوال الأعضاء في فصيلة معينة يسمح لها بالوصول إلى العناصر الخاصة في الكيان** C++ **توفر مؤشراً خاصاً خفياً اسمه this مع كل كيان. هذا المؤشر يمكن استخدامه في أيٍّ من الدوال الأعضاء** من أجل ذلك توفير **احتياجات الدوال** التى تتلخص في:

- الإشارة إلى الكيان الذي يحتويها بكامله. **يحدث ذلك مثلاً عند** استخدام مؤشر عام في استدعاء الدالة العضو بدلاً من اسم الكيان المباشر.
- أن تنظر الدالة العضو إلى الكيان الذي تم الإستدعاء في نطاقه: هل هو الكيان المالك أم كيان آخر؟
- أن ترجع مؤشراً إلى الدالة المالكة.

المثال التالي يشرح كيفية استخدام المؤشر this للتأكد من هوية الكيان المستدعي:

مثال (4)

في هذا المثال **نعرِّف فصيلة Test تحتوي** على **دالة واحدة IsItMe()**. هذه **الدالة تستقبل عنصراً واحداً من النوع Test** و**تنظر** إذا ما كان **هذا العنصر هو الكيان المالك أم لا**. إذا كان **العنصر الممرر هو الكيان المالك، ترجع الدالة القيمة 1، و إلا فترجع القيمة صفر.**

برنامج(4)

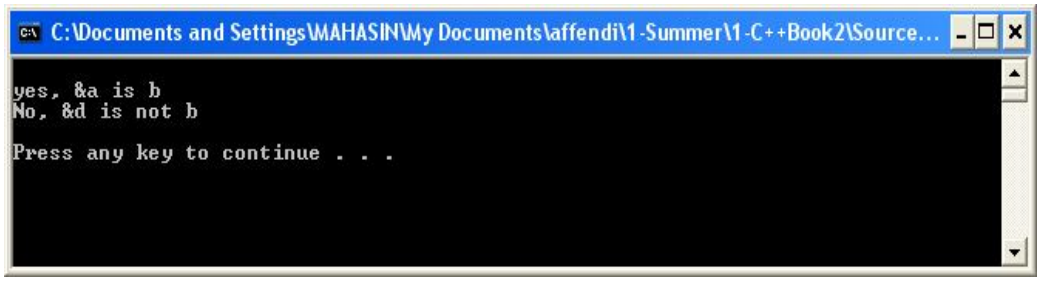
```
// this pointer
#include <iostream>
using namespace std;
class Test {
  public:
    int IsItMe (Test & param);
};
int Test::IsItMe (Test& param)
{
```

```
  if (&param == this) return 1;
  else return 0;
}
int main () {
  Test a;
  Test* b = &a;
  if ( b->IsItMe(a) )
    cout <<endl<< "yes, &a is b"<<endl;
  Test d;
  if ( b->IsItMe(d) )
    cout <<endl<< "yes, &d is b"<<endl;
  else cout<<"No, &d is not b"<<endl<<endl;
    system("pause");
  return 0;
}
```

لاحظ أننا **استخدمنا المؤشر b في الإشارة إلى الكيان a** (وضعنا عنوان a في المؤشر b):

```
Test a;
Test* b = &a;
```

بعد ذلك **استخدمنا المؤشر b في استدعاء الدالة IsItMe()**. في **الحالة الأولى** كانت **الإجابة** "**نعم**"، لأن **العنصر الممرر هو a**. أما في **الحالة الثانية**، فكانت **الإجابة لا** لأن **العنصر الممرر هو d**.



4. تحميل العوامل الأحادية

تحميل العوامل الأحادية أسهل من تحميل العوامل الثنائية.

الصورة العامة لتحميل العوامل الأحادية هي:

```
ClassName operator op-sign ( )
{ ……
   algorithm
   return *this;
}
```

في هذه الحالة **ClassName** هو **إسم الفصيلة** التي يتم التحميل فيها. لاحظ أننا **لانحتاج** إلى **عنصر بين القوسين (بعد op-sign) لأن العملية أحادية ولاتحتاج** إلى **طرف آخر. الدالة ترجع مؤشراً للفصيلة المالكة باستخدام المؤشر this.**

المثال التالي يوضح كيف يتم تحميل العامل الأحادي ++ في الفصيلة Rectangle:

مثال (5)

في هذا المثال **قمنا بتحميل العامل الأحادي ++ في الفصيلة Rectangle:**

```
Rectangle operator++ ()
{ ++width;
  ++height;
  return *this;
}
```

داخل البرنامج استخدمنا هذا العامل الأحادي مع كيان من النوع Rectangle:

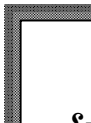
```
++rect1;
```

برنامج(5)

```
// unary operator overloading
#include <iostream>
using namespace std;
class Rectangle {
    float width, height;
  public:
    Rectangle (){}
    Rectangle (float,float); //using a
constructor function
    Rectangle operator++ ()
    { ++width;
      ++height;
      return *this;
    }
    void PrintDims()
    {cout <<width<<"   "<<height<<endl;}
};
Rectangle::Rectangle (float w, float h)
//defining the constructor
 {
  width = w;
  height = h;
}
//Rectangle Rectangle::operator+ (const
Rectangle Rect) const

int main () {
  Rectangle rect1(3.0,4.0); //declare an
object of type Rectangle
  ++rect1;
  rect1.PrintDims();
  system("pause");
  return 0;
}
```

أسئلة تقويم ذاتي



1. أشرح عملية تحميل الدوال؟
2. ما الصورة العامة لتحميل العوامل الثنائية و الأحادية في سي++؟
3. لماذا نستخدم **المؤشر this**؟

1- المتغيرات الدائمة في الفصائل (Static Variables)

أساليب البرمجة(1) ذكرنا **أن المتغيرات الدائمة عبارة عن نوع خاص من المتغيرات المحلية تحتفظ بقيمتها حتى بعد الخروج من الدالة أو المنطقة التي تحتويها في البرنامج. هذا النوع من المتغيرات يمكن استخدامه في الكيانات** كذلك. **المتغير الدائم في كيان معين يحتفظ بآخر قيم اسندت له حتى بعد الخروج من نطاق الكيان الذي يحويه.**

المثال التالي يوضح كيف يحتفظ هذا النوع من المتغيرات بقيمته عبر عدة كيانات.

مثال(6)

الفصيلة Test2 في هذا المثال **تحتوي على متغير دائم n. دالة الإنشاء** في الفصيلة **ترفع** قيمة n بواحد. **دالة الإخلاء تخفض** قيمة n بواحد. هذا **يعنى** أننا كلما **دخلنا نطاق** كيان من النوع **Test2 زادت** قيمة n بواحد، وكلما **خرجنا من نطاق** كيان من النوع **Test2 نقصت** قيمة n بواحد.

برنامج(6)

```
// static members in classes
#include <iostream>
using namespace std;
class Test2 {
  public:
    static int n;
    Test2 () { n++; };
    ~Test2 () { n--; };
};

int Test2::n=0;

int main () {
  Test2 a;
  Test2 b[5];
  Test2 * c = new Test2;
  cout <<endl<<"The value of n now is "<< a.n
<< endl;
  delete c;
  cout << "The value of n after deleting c is
"<<Test2::n << endl;
  system("pause");
  return 0;
}
```

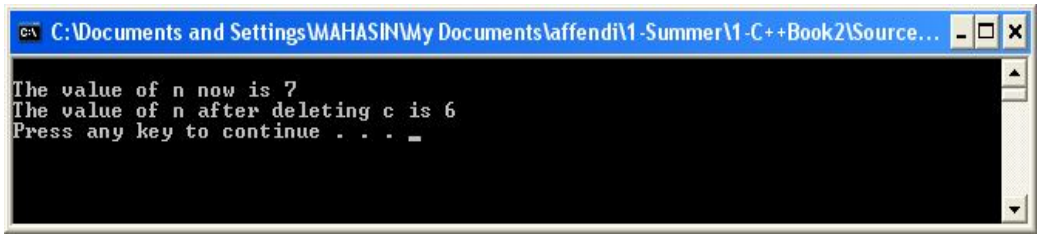
داخل البرنامج **أنشأنا كيان a من النوع Test2**، ثم **خمسة كيانات b** من النوع **Test2**، ثم **كياناً ديناميكياً** من **نفس النوع**. **مجموع الكيانات** التي أنشأناها **سبعة**، لذا يتوقع أن تصبح **قيمة المتغير n** بعد هذه التعريفات **سبعة، وهو ما حصلنا عليه** عندما طبعنا قيمة n في هذه المرحلة (انظر شاشة تنفيذ البرنامج أدناه).

بعد ذلك قمنا **بإزالة الكيان الديناميكي c**. يتوقع في هذه الحالة أن **تنقص قيمة n إلى ستة**، ذلك أن **دالة الإخلاء** التي **تستدعى** في هذه الحالة **تنقص قيمة n بواحد**. **شاشة تنفيذ** البرنامج **تدل** على **أن قيمة n** أصبحت **فعلاً ستة**.

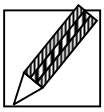
ملحوظة:

ربما كانت هذه المرة الأولى التي نستغل فيها دالة الإخلاء استغلالاً مباشراً. في كل الأمثلة السابقة كنا نعتمد على النسخة الإفتراضية من هذه الدالة.

لاحظ أننا استخدمنا اسم الفصيلة و علامة النطاق لطباعة قيمة n في نهاية البرنامج أعلاه: Test2::n وهذا مسموحٌ به.



تدريب(3)



حمِّل عامل التزويد الأحادي "++" بحيث يمكن استخدامه في تزويد التاريخ التي استخدمناها في الوحدة الماضية.Dateبيوم في فصيلة التاريخ

الخلاصة

C++ تسمح باستخدام اسم واحد لعدة دوال في نطاق واحد، بشرط اختلاف هذه الدوال في عدد العناصر الممررة أو نوعها. من وجهة نظر أخرى، يمكن أن يكون للدالة الواحدة أكثر من دور. تختلف الأدوار باختلاف عدد العناصر أو اختلاف أنواع العناصر. بالإمكان إعادة برمجة العوامل المختلفة في C++ بحيث تستخدم لإجراء عمليات خاصة على الكيانات و الأنواع المركبة.يمكن استخدام المؤشر this داخل دالة للإشارة إلى الكيان الذي يحويها، وذلك لأغراض مختلفة.بالإمكان استخدام المتغيرات الدائمة في الكيانات. تحتفظ هذه المتغيرات بقيمتها حتى بعد الخروج من نطاق الكيان المالك.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس، في الوحدة التالية سوف نتناول تعريف الدوال التخيلية وأدوارها ومفهومها، كما سنعرف حل مشكلة عدم تمييز المترجم للفصيلة الأساسية باستخدام تعدد الأدوار والربط المتأخر، فهيا الي وحدتنا التالية.

إجابات التدريبات

تدريب1

```
#include<iostream.h<
#include<conio.h<
float Volume(float Radius,float Length) ;
int Volume(int Length);
int Volume(int Length, int Width,int Hieght) ;
main()
{
Volume(4.5,6) ;
Volume(5) ;
Volume(4,5,6) ;
getch();
}
float Volume(float Radius,float Length)
}
cout<<(Length*Radius)<<endl) ;
}
int Volume(int Length(
{
cout<<(Length*3)<<endl;
}
int Volume(int Length, int Width,int Hieght(
{
cout<<(Length*Width*Hieght)<<endl;
}
```

تدريب (2)

1-

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

```
class Vector
{
int arr1[5],arr2[5] ;
public:
Vector(){}
Vector(int &a,int &b)
{
for(int i = 1;i<=5;i++)
{
arr1[i]= a    ;
arr2[i]= b;
}
}
void show()
{
for(int i= 1;i<=5;i++)
{
cout<<arr1[i]<<endl;
cout<<arr2[i]<<endl;
}
}
 Vector operator +(Vector op2);
};
Vector Vector::operator+(Vector op2)
{
Vector T;
for(int i=1;i<=5;i++)
{
T.arr1[i]=op2.arr1[i]+arr1[i];
T.arr2[i]=op2.arr2[i]+arr2[2];
}
return T;
}
main()
{
```

```
Vector ob1[5],ob2[5];
for(int i=1;i<=5;i++)
cin>>ob1[i]>>ob2[i];
ob1.show();
ob2.show();
ob1=ob1+ob2;
ob1.show();
getch();
}
```

2-

```
#include<iostream.h>
#include<conio.h>
class clock
{
int hour,minute,second ;
public:
clock(){}
clock(int h,int m , int s)
{
hour = h;
minute = m;
second = s;
}
void show()
{
cout<<hour<<endl ;
cout<<minute<<endl;
cout<<second<<endl;
}
 clock operator +(clock op2);
};
clock clock::operator+(clock op2)
{
clock T;
T.hour=op2.hour+hour;
```

```
T.minute=op2.minute+minute;
T.second= op2.second+second;
return T;
}
main()
{
clock ob1(3,30,20),ob2(2,23,12);
ob1.show();
ob2.show();
ob1=ob1+ob2;
ob1.show();
getch();
}
```

3-

```
#include<iostream.h>
class Date
{
public:
Date(int m=1,int d=1, int y= 1900);
void SetDate(int ,int,int);
Date operator++();
Date operator++(int);
const Date &operator+=(int);
int LeapYear(int);
int EndOfMonth(int);
private:
int Month;
int Day;
int Year;
static int days[];
void Increment();
};
int Date::days[]={0, 31, 28,31,30,31,30,31,31,30,31,30,31};
Date::Date(int m,int d, int y )
{
```

```
SetDate(m,d,y);}
void Date::SetDate(int mm,int dd,int yy)
{ Month=(mm>=1&&mm<=12)? mm : 1;
Year = (yy>=1900&&yy<=2100) ? yy: 1900;
if(Month==2 && LeapYear(Year))
Day= (dd>=1 && dd<= 29) ? dd : 1;
else
Day = (dd>=1 && dd<= days[Month]) ? dd : 1;
}
Date::Date operator++()
{
Increment();
return *this;
}
Date Date::operator++(int)
{
Date temp  = *this;
Increment();
return temp
return
}
const Date &Date::operator+=(int AditionalDays)
{
 for(int i= 1;i<=AdditionalDays;i++)
 Increment();
 return *this;
 }
 int Date::LeapYear(int y)
 {
 if(y%400= 0||(Y%100!=0 && y%4==0))
 return 1; //leap year
 else
 return 0; // not leap year
 }
 int Date::EndOfMonth(int d)
```

```
 {
 if(Month==2 && LeapYear(year))
 return d==29; //last day of Feb. in leap year
 else
 return d== days[Month];
 }
 void Date::Increment()
 {
 if(Endofmonth(Day) && Month == 12) //end of year
 {
 Day=1;
 Month = 1
 ++year;
 }
 else
 if(EndOfMonth(day))   // end of month
 {
 day = 1;
 ++Month;
 }
 else
 ++Day
 }
main()
{
d1,d2(12,27,1998),d3(0,99,8045);
cout<<"d1 is :  "<<d1<<endl
    <<"d2 is :  "<<d2<<endl
    <<"d3 is :  "<<d3<,endl<<endl;
cout<<"d2+=7 is "<<(d2+=7)<<endl<<endl;
d3.SetDate(2,28,1998);
cout<<"d3 is : " <<d3<<endl;
cout<<"++d3 is : " <<++d3<<endl<<endl;
return 0
}
```

مسرد المصطلحات

- **تحميل الدوال Function Overloading**

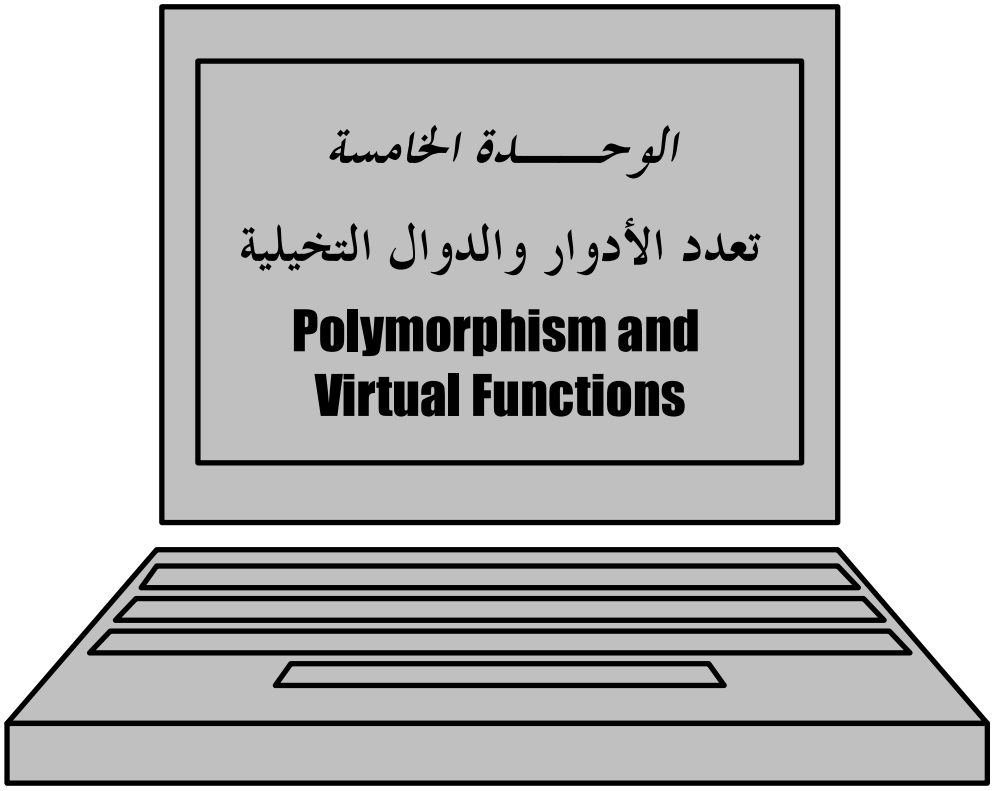
يعني استخدام إسم واحد لتعريف أكثر من دالة في النطاق الواحد، **بشرط** إختلاف هذه الدوال في عدد العناصر أو نوعها.

- **تحميل العوامل Operator Overloading**

يعني إعادة برمجة كل العوامل الحسابية و كل عوامل المقارنة التي تستخدم مع البيانات العادية بحيث تستخدم مع الكيانات و الأنواع المركبة .

المصادر والمراجع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.


الوحـــــدة الخامسة
تعدد الأدوار والدوال التخيلية
Polymorphism and
Virtual Functions

محتويات الوحدة

الموضوع	الصفحة
المقدمة	155
تمهيد	155
أهداف الوحدة	155
1. إعادة تعريف الدوال	156
2. تحديد المشكلة	159
3. الدوال التخيلية والربط المتأخر	162
4. تمرير الكيانات كعناصر إلى الدوال	164
الخلاصة	169
لمحة مسبقة عن الوحدة التالية	169
إجابات التدريبات	170
مسرد المصطلحات	171
المراجع	172

المقدمة

تمهيد

عزيزي الدارس، مرحباً بك في الوحدة الخامسة من هذا المقرر والتي وردت بعنوان تعدد الأدوار والدوال التخيلية، تناولت الوحدة تعريف الدوال التخيلية وأدوارها ومفهومها. كما تضمنت الوحدة أسئلة تقويم ذاتي وتدريبات لتساعدك في فهمها وأستيعابها.

C++ تعطي نوعاً من المرونة في إستخدام مؤشرات الكيانات (object pointers)، حيث يمكن استخدام مؤشرات الفصائل الأساس (Base Class Pointers) في معالجة الكيانات التابعة للفصائل الوارثة. المشكلة التي تنشأ في هذه الحالة أن ترجمان اللغة لايستطيع أن يفرق بين الحالات التي يستخدم فيها مؤشر الفصيلة الأساس للإشارة إلى كيانات وارثة و الحالات التي يستخدم فيها المؤشر في الإشارة إلى كيانات تابعة للفصيلة الأساس. C++ تحل المشكلة باستخدام تقنية مبسطة تعرف في أوساط الحاسوبيين ب"تعدد الأدوار" (polymorphism). في هذا الفصل نشرح المشكلة أولاً، ثم نعطي حلها.

أهداف الوحدة

عزيزي الدارس، عند انتهائك من دراسة هذه الوحدة اتوقع أن تكون قادراً على أن:

1. **تعرف** المشكلة التي تنشأ من عدم تمييز المترجم للفصيلة الأساسية ووارثيه.
2. **تعرف** حل المشكلة باستخدام تعدد الأدوار والربط المتأخر.
3. **تستخدم** تقنية الدوال التخيلية في حل المشكلة.

يميل الكثير من مطوري البرامج الكيانية إلى استخدام مؤشرات كيانات (object pointers) في تطبيقاتهم بدلاً من أسماء الكيانات المباشرة، خاصةً عند تمرير الكيانات كعناصر للدوال. السبب في ذلك عمومية المؤشرات و مرونتها في التعامل مع الكيانات. ومما يزيد في هذه المرونة أن مؤشرات الفصيلة الأساس (Base Class Pointers) في شجرة الإرث يمكن استخدامها في الإشارة إلى الكيانات التي تنتمي إلى كل أنواع الفصائل الوارثة. مثلاً: إذا كان نوع العنصر الممرر إلى دالة معينة مؤشراً إلى كيان ينتمي إلى الفصيلة الأساس Base Class، فيمكن استخدامه لتمرير كيانات تنتمي إلى كل الفصائل التي ترث Base Class.

المشكلة التي تنشأ في هذه الحالة هي أن ترجمان C++ **لايستطيع أن يميز بين ال Base Class ووارثيه** (الأبناء و الأحفاد)، ويحل المشكلة بأن يفترض أن المؤشر **يشير إلى كيان تابع لل Base Class**. بالطبع قد يكون هذا الإفتراض خاطئاً في كثيرٍ من الأحيان.

C++ تحل هذه المشكلة باستخدام فكرة تعدد الأدوار (polymorphism) و الربط المتأخر (late binding). التقنية في غاية البساطة وتعتمد على استخدام نوع خاص من الدوال يسمى الدوال التخيلية (virtual functions).

في هذه الوحدة نعرض المشكلة، ثم نعرض حلها.

1. زيارة أخرى لإعادة تعريف الدوال

لكي نشرح المشكلة بوضوح نعود مرة أخرى لمثال إعادة تعريف الدوال الذي عرضناه في الوحدة الثانية (المثال أدناه). لو تذكر، أن شجرة الإرث في هذا المثال تتكون من ثلاثة مستويات. كل الفصائل في شجرة الإرث تستخدم دالة ()area لحساب مساحة الشكل المقابل. هذه الدالة تم تعريفها أولاً في الفصيلة الأم Shape ثم أعيد تعريفها في كل الفصائل الوارثة.

مثال (1)

برنامج(1)

```
#include <iostream>
#include <stdlib.h>
using namespace std;
// derived classes
class Shape {
  protected:
    float width, height;
  public:
    void SetValues (float a, float b)
      { width=a; height=b;}
      float area(void) {}
  };
class Rectangle: public Shape {
  public:
    float area (void)
      { return (width * height); }
  };
class Triangle: public Shape {
  public:
    float area (void)
      { return (width * height / 2); }
  };
class Box: public Rectangle
{ private:
     float BoxHeight;
  public:
      Box(float a) {BoxHeight = a;}
      float area(void)
      {     return     (2*width+     2*height     +
2*BoxHeight);}
      float Volume (void)
      { return area()*BoxHeight; }
};
int main () {
```

```
  Rectangle rect;
  Triangle trgl;
  rect.SetValues (4,5);
  trgl.SetValues (4,5);
  cout << "The arae of the rectangle is "
<<rect.area() << endl;
  cout << "The arae of the rectangle is "
<<trgl.area() << endl;
  Box B(3.4);
  B.SetValues(7.2, 4.8);
  cout <<"The volume of the Box B is
"<<B.Volume()<<endl;
  cout <<"The area of the Box B is "<<B.area()<<endl;
  system("PAUSE");
  return 0;
}
```

لاحظ تعريف الدالة **(area)** في الفصائل المختلفة:

class	**The area() function**
Shape	float area(void) {}
Rectangle	float area (void) { return (width * height); }
Triangle	float area (void) { return (width * height / 2); }
Box	float area(void) { return (2*width+ 2*height + 2*BoxHeight);}

في هذا المثال استخدمنا أسماء الفصائل مباشرةً، لذا لم يكن هناك مشكلة في انتقاء الصورة المناسبة من الدالة area(). بناءً على إسم الفصيلة ننتقي الصورة المناسبة من الدالة.

2. ماالمشكلة إذاً؟

المشكلة تنشأ إذا استخدمنا مؤشراً إلى الفصيلة الأم Shape ثم بعد ذلك حاولنا استخدامه لمعالجة الفصائل الوارثة. المثال أدناه يشرح تفاصيل المشكلة:

مثال (2)



برنامج(2)

```
#include <iostream>
#include <stdlib.h>
using namespace std;
// derived classes
class Shape {
  protected:
    float width, height;
  public:
    void SetValues (float a, float b)
      { width=a; height=b;}
      float  area(void)  {cout<<"BaseClass"<<endl;
return 0;}
  };
class Rectangle: public Shape {
  public:
    float area (void)
      { cout<<"Rectangle"<<endl;
          return (width * height); }
  };
class Triangle: public Shape {
  public:
    float area (void)
      { cout <<"triangle"<<endl;
          return (width * height / 2); }
  };
class Box: public Rectangle
{ private:
     float BoxHeight;
  public:
```

```
      Box(){BoxHeight=0;}
      Box(float a) {BoxHeight = a;}
      float area(void)
      { cout <<"Box"<<endl;
            return     (2*width+     2*height     +
2*BoxHeight);}
      float Volume (void)
      { return area()*BoxHeight; }
};
int main () {
  Shape *p1;
  Shape *p2;
  Shape *p3;
  Rectangle rect;
  Triangle trgl;
  Box B;
  p1 = &rect;
  p2 = &trgl;
  p3 = &B;
  p1->SetValues (4,5);
  p2->SetValues (4,5);
  p3->SetValues(7.2, 4.8);
  cout << "The arae of the rectangle is " <<p1-
>area() << endl;
  cout << "The arae of the rectangle is " <<p2-
>area() << endl;
  //cout <<"The volume of the Box B is "<<p3-
>Volume()<<endl;
  cout <<"The area of the Box B is "<<p3-
>area()<<endl;
  system("PAUSE");
  return 0;
}
```

كما هو واضح من المثال، يبدأ البرنامج بتعريف ثلاثة مؤشرات من النوع Shape:

```
Shape *p1;
Shape *p2;
Shape *p3;
```

بعد ذلك يعرف البرنامج ثلاثة كيانات من فصائل مختلفة كما يلي:

```
Rectangle rect;
Triangle trgl;
Box B;
```

بعد ذلك يسند البرنامج عناوين هذه الكيانات إلى المؤشرات التي تم تعريفها:

```
p1 = &rect;
p2 = &trgl;
p3 = &B;
```

لاحظ أن C++ تسمح بذلك حسب القاعدة التي ذكرناها في المقدمة. مؤشر الفصيلة الأساس (base class pointer) يمكن استخدامه في الإشارة إلى الكيانات التي تنتمي إلى أي من الفصائل الوارثة.

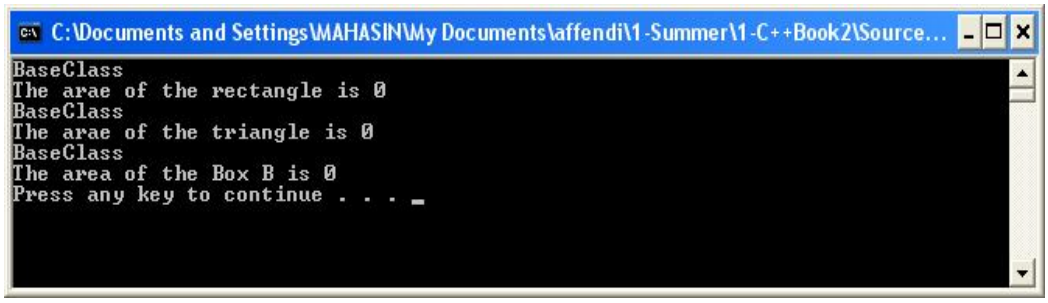
الخطوة التالية هي إستخدام المؤشرات في معالجة الفصائل المقابلة و طباعة النتائج.

```
p1->SetValues (4,5);
p2->SetValues (4,5);
p3->SetValues(7.2, 4.8);
cout << "The arae of the rectangle is " <<p1->area() << endl;
cout << "The arae of the rectangle is " <<p2->area() << endl;
```

لاحظ أننا قمنا بتعديل الدالة area() بحيث تطبع إشعاراً يدل على المكان الذي أستدعيت منه. مثلاً في الفصيلة الأم Shape أصبحت الدالة كما يلي:

```
float area(void) {cout<<"BaseClass"<<endl;  return 0;}
```

انظر الآن إلى الناتج:



المفاجأة في شاشة الناتج أعلاه هي أن البرنامج استخدم في الحالات الثلاث دالة الفصيلة الأم Shape في حساب مساحة الأشكال المختلفة. يدل على ذلك الإشعار الذي تم طباعته قبل كل نتيجة (Base Class) و القيمة المطبوعة للناتج، و التي هي صفر لكل الأشكال.

هذا بالرغم من أننا أخبرنا البرنامج أن p1 يشير إلى المستطيل (p1->& rect)، وأن p2 يشير إلى المثلث (p2->& trgl) وp3 يشير إلى متوازي المستطيلات (p3->& B). كل ذلك لم يجدِ.

السبب في ذلك أن ترجمان C++ لاحظ أن كل المؤشرات p1,p2,p3 هي من النوع Shape، واتخذ قراراً بأنه ينبغي استخدام هذه المؤشرات للوصول إلى عناصر هذه الفصيلة فقط. أما عناوين الكيانات التي أسندت إلى هذه المؤشرات فلاأثر لها على قرار الترجمان، ذلك أنها لايمكن معرفتها بالفعل إلا لحظة تنفيذ البرنامج. حينها لايمكن تعديل قرار الترجمان الذي يُتخذ عادةً قبل التنفيذ.

3. الحل: الدوال التخيلية و الربط المتأخر

Virtual Functions and Late Binding

الحل هو أن نقنع الترجمان بتأجيل اتخاذ قراره فيما يتعلق بمؤشرات الكيانات حتى وقت تنفيذ البرنامج. إذا رأى الترجمان مؤشراً إلى الفصيلة الأم ينبغي ألا يستعجل في اتخاذ القرار بأن كل مايشير إليه هذا المؤشر تابع فعلاً للفصيلة الأم، فقد يكون الأمر غير ذلك.

الطريقة التي تتبعها C++ لإقناع الترجمان بإرجاء قراره في غاية البساطة، وهي استخدام كلمة virtual في تعريف الدوال التى يتوقع إعادة تعريفها. مثلاً، بالنسبة للفصيلة area() أعلاه يتم تعريفها كالتالي في الفصيلة الأم:

```
virtual float area() { return 0; }
```

عندما يرى الترجمان كلمة virtual أمام دالة فإنه يرجيء قراره إلى وقت تنفيذ البرنامج. عند تنفيذ البرنامج يقوم نظام C++ بإنشاء جدول خاص بالدوال التخيلية حسب

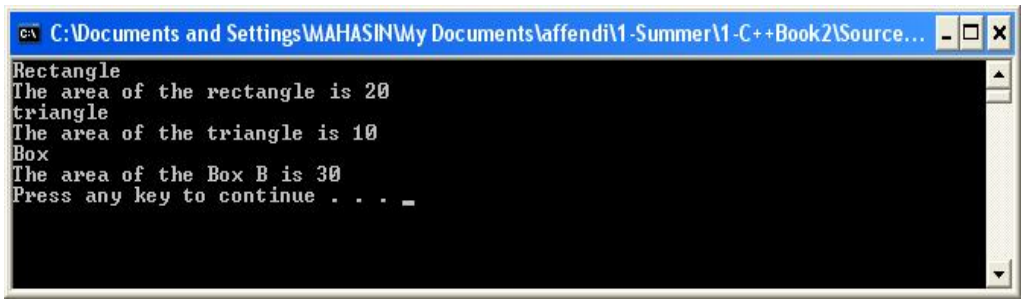
العناوين الفعلية للكيانات. تستخدم قيمة المؤشرات في الوصول إلى الدالة المناسبة في كل حالة. الفصيلة Shape الجديدة تصبح:

```
// new Shape class with virtual function area
class Shape {
  protected:
    float width, height;
  public:
    void SetValues (float a, float b)
      { width=a; height=b;}
      virtual          float          area(void)
{cout<<"BaseClass"<<endl; return 0;}

  };
```

لاحظ استخدام الكلمة الخاصة virtual قبل اسم الدالة ()area.

البرنامج في هذه الحالة يعطي النتائج المتوقعة:



هذا الأسلوب في تعريف الدوال يعرف عند العاملين في هذا المجال ب"تعدد الأدوار" (polymorphism)، ذلك أن نوع واحد من المؤشرات يمكن أن يستخدم في الوصول إلى دالة تتعدد أدوارها وتختلف حسب قيمة المؤشر وقت التنفيذ.

4. تمرير الكيانات كعناصر إلى الدوال

المثال التالي يشرح استخدام تقنية تعدد الأدوار في حالة تمرير الكيانات كعناصر للدوال:

مثال (3)

هذا المثال يحتوي على فصيلتين:الفصيلة الأساس Base Class و الفصيلة الوارثة Derived. الفصيلة الوارثة تعيد تعريف الدالة ()print الموروثة من Base Class. بالإضافة إلى ذلك يحتوي المثال على الدالة ()Test Print التي تستخدم في استدعاء الدالة ()print التابعة للعنصر الذي يمرر لها. العنصر الذي تستقبله الدالة عبارة عن مرجع (reference) إلى كيان من النوع Base Class.

ملحوظة

> الكيانات يمكن تمريرها للدوال بالقيمة (by value) او باستخدام مرجع إلى الكيان (by reference). في حالة التمرير بالقيمة' فإن النظام يمرر نسخة من الكيان إلى الدالة، كما هو الحال في المتغيرات العادية (اي أن النسخة الأصلية من الكيان لاتتأثر بالتغييرات التي تحدث داخل الدالة). في حالة التمرير بالمرجع، فإن الكيان الممرر يتأثر بالتغييرات التي تحدث داخل الدالة.

في المثال أدناه نفترض أن الكيان p يمرر بالمرجع أو باستخدام مؤشر. لاحظ أن الدوال التخيلية لامعنى لها إذا كان التمرير بالقيمة.

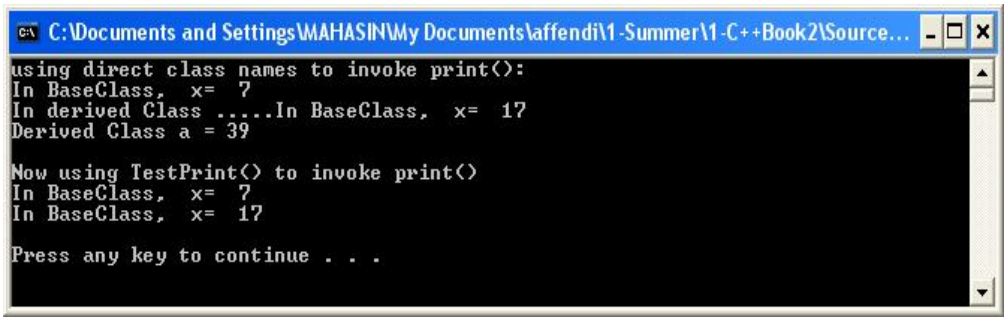
مثال (3)

برنامج(3)

```
#include <iostream>
using namespace std;
class BaseClass
{private:
    int x;
 public:
     BaseClass(int u=0)
     {x=u;}
     void print()
     { cout <<"In BaseClass,  x=  "<<x<<endl;
}
};
class Derived: public BaseClass
{ private:
    int a;
  public:
    Derived(int u=0, int v=0):BaseClass(u)
    {a =v; }
    void print()
    { cout <<"In derived Class .....";
      BaseClass::print();
      cout <<"Derived Class a = "<<a<<endl;
    }
};
void TestPrint(BaseClass &p)
{ p.print();    }
int main()
{ BaseClass First(7);
  Derived Other(17,39);
  //testing
  cout <<"using direct class names to invoke
print(): "<<endl;
```

```
  First.print();
  Other.print();
  //testing TestPrint()
  cout <<endl<<"Now using TestPrint() to
invoke print()"<<endl;
  TestPrint(First);
  TestPrint(Other);
  cout <<endl;
  system("pause");
  return 0;
}
```

في هذا النسخة من البرنامج، الدالة print() لم تعرَّف كدالة تخيلية (virtual)، لذا يتوقع أن يتم استدعاء صورة الدالة التابعة للBaseClass في كل الأحوال. العينة أدناه من تنفيذ البرنامج تشير إلى أن هذا هو الذي حدث فعلاً.

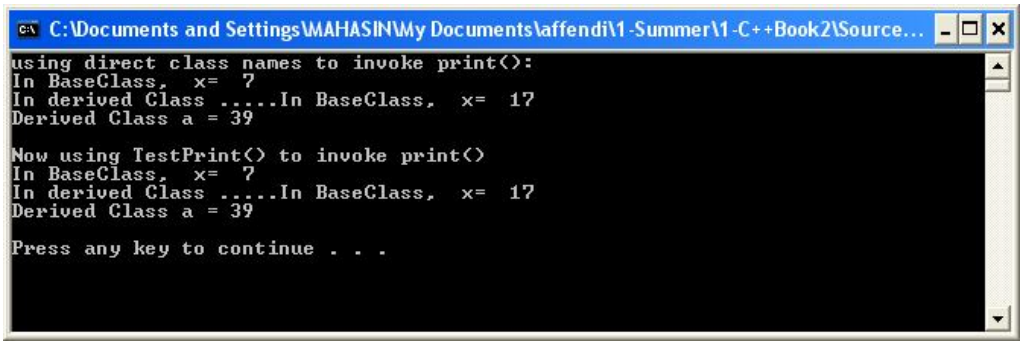


لكي نحل هذه المشكلة، ماعلينا إلا أن نضفي صبغة "التخيلية" (virtual) على الدالة print() في الفصيلة Base Class:

```
class BaseClass
{private:
    int x;
 public:
     BaseClass(int u=0)
     {x=u;}
     virtual void print()
     { cout <<"In BaseClass,  x=  "<<x<<endl;
}
};
```

انظر استخدام كلمة virtual قبل اسم الدالة ()print.

ناتج البرنامج في هذه الحالة يكون كما يلي:



لاحظ أن الدالة ()Test Print في هذه الحالة استطاعت أن تستدعي الدالة ()print التابعة للفصيلة Derived عندما مررنا إليها مرجع إلى الكيان Other، الشيء الذي عجزت عنه في المثال السابق.

أسئلة تقويم ذاتي

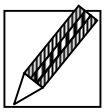


1- ما المشكلة التي تنشأ عند استخدام مؤشر أو مرجع من نوع الفصيلة الأساس لمعالجة كيانات تابعة للفصائل الوارثة؟

2- وضح كيف يمكن حل المشكلة أعلاه في C++.

3- عرِّف باختصار مايلي: الربط المتأخر (late binding)، الدوال التخيلية (virtual functions)، تعدد الأدوار (polymorphism).

تدريب (1)



1- صمم فصيلة أساس اسمها Person تحتوي على دالة اسمها Print Details() تطبع التفاصيل المتعلقة بشخص معين. صمم دالة أخرى Student ترث الدالة Person و تضيف إليها معلومات أخرى مثل الرقم الجامعي، الأستاذ المشرف، المستوى الدراسي ...الخ. عدّل الدالة Print Details() في الفصيلة Student لتطبع التفاصيل الكاملة المتعلقة بـStudent. استخدم هاتين الفصيلتين في شرح فكرة تعدد الأدوار و الدوال التخيلية. اكتب برنامجاً كاملاً ونفذه.

2- هل يمكن استخدام فكرة تعدد الأدوار في الوصول إلى الدالة المناسبة في حالة تمرير الكيانات بالقيمة إلى الدوال؟

الخلاصة

عزيزي الدارس حاول أن تتأكد من أنك قد حققت الأهداف الخاصة بالوحدة، وتستطيع بعد ما ناقشناه في هذه الوحدة أن تخلص إلى النقاط التالية:

- تُستخدم المؤشرت كثيراً في معالجة الكيانات وتمريرها إلى الدوال.
- مؤشرات الفصائل الأساس (Base Class Pointers) تتوافق نوعياً مع كل الفصائل الوارثة. هذا يعني أن مؤشرات الفصائل الأساس يمكن استخدامها في معالجة الكيانات التابعة للفصائل الوارثة.
- الإشكال الذي ينجم من الحقيقية أعلاه هو أن الترجمان لايستطيع استدعاء الدوال التابعة للفصائل الوارثة.
- C++ وشبيهاتها من لغات البرمجة الكيانية تحل المشكلة باستخدام فكرة تعدد الأدوار (polymorphism) و الدوال التخيلية (virtual functions).

حاول أن تعزز فهمك لهذه الوحدة بحل عدد من التدريبات ويمكنك الاستعانه بالمراجع في هذا المنهج.

لمحة مسبقة عن الوحدة الدراسية التالية

نتناول في الوحدة الدراسية القادمة موضوع القوالب وهى تقنية هامة تستخدم في تصميم مكتبة قياسية كبيرة، وإدراجها مع ترجمانات C++، ستفتح شهيتك للبرمجة عندما تستخدم تقنية القوالب لأنها ستوفر عليك الكثير من التكرار الممل وتوفر وقتك وجهدك لأجزاء أخرى من البرنامج، وفقك الله وأعانك في طريق العلم والتعلم.

إجابات التدريبات

```
class Person
{
        private:
                string  Name;
                int  DOB;
        public:
                void PrintDetails()
                {
                        //Print Person Details
                        cout << "I am a Person, my Name : " << m_Name;
                        cout << "Date of Birth: ";
                        m_DOB.Print();
};
class Student : public Person
{
        private:
                int No;
        public:
                void PrintDetails()
                {
                        //Print Student Details
                        cout << "I am a student, my No is: " << No;
        }
};

void main()
{
         Person P1("Ali", CDate(11,10,1980));
         Student S1("Ali", (11,10,1980), 101);
         cout << "P1:"; P1.Print();
         cout << "E1:"; S1.Print();
        cout << "ME1:"; ME1.Print();
        Person P2 = (Person) S1;
        Person P3 = (Person) ME1;
        Student S2 = (Student) ME1;
        Cout << "P2:"; P2.Print();
        Cout << "P3:"; P3.Print();
        Cout << "S2:"; S2.Print();
}
```

مسرد المصطلحات

- **تعدد الأدوار Polymorphism**

تقنية برمجية تستخدم في حل مشكلة عدم تمييز المترجم للفصيلة الأساسية ووارثيها.

- **الربط المتأخر Late Binding**

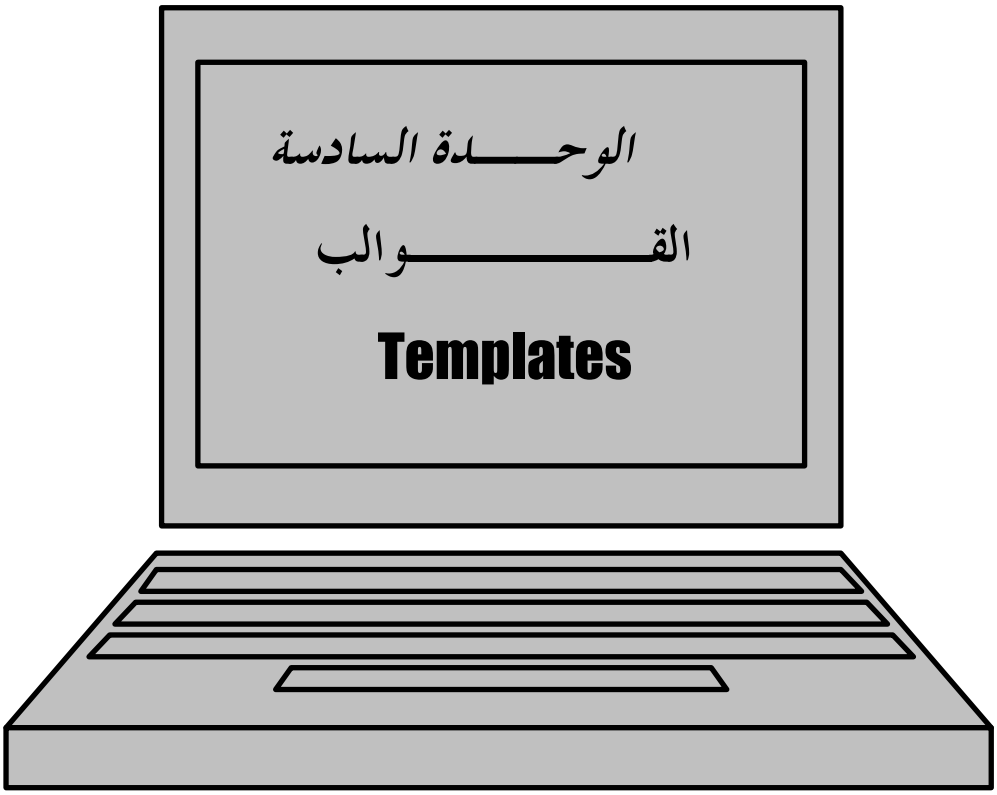
تقنية برمجة تستخدم في حل المشكلة السابقة الذكر.

- **الدوال التخيلية Virtual Functions**

نوع خاص من الدوال يستخدم في تقنيتى تعدد الأدوار والربط المتأخر.

المصادر والمـــراجـــع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.


الوحـــــدة السادسة
القـــــــــوالب
Templates

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**175**
تمهيد	**175**
أهداف الوحدة	**176**
1. قوالب الدوال	**177**
1.1 الشكل العام لتعريف قوالب الدوال	**177**
2.1 استخدام أكثر من نوع مطلق في تعريف القوالب	181
2. قوالب الفصائل	**182**
2.1 الشكل العام لتعريف قوالب الفصائل	182
2.2 تخصيص القوالب	184
2.3 استخدام عناصر إضافية في تعريف القوالب	186
الخلاصة	**195**
لمحة مسبقة عن الوحدة التالية	**196**
إجابات التدريبات	**197**
مسرد المصطلحات	**202**
المراجع	**203**

المقدمة

تمهيد

عزيزي الدارس، مرحباً بك في الوحدة السادسة من هذا المقرر والتي وردت بعنوان القوالب، تناولت الوحدة قوالب الدوال و قوالب الفصائل، وكيفية استخدام أكثر من نوع مطلق في تعريفها، والعمل على تخصص قالب عام للفصيلة وكذلك كيفية استخدام عناصر إضافية في تعريف القوالب.

عزيزي الدارس: لعلك لاحظت أن الكثير من الخوارزميات تستخدم نفس المنطق و العمليات بغض النظر عن نوع البيانات الذي تتعامل معه. مثلاً خوارزمية البحث عن عنصر في مصفوفة تستخدم نفس الطريقة للبحث بغض النظر عن نوع البيانات: أعداد صحيحة، أعداد حقيقية، متغيرات ألفبائية ...الخ. يكون من المناسب في هذه الحالة لو صممنا الدوال والفصائل المعنية بصورة معممة، نغض فيها الطرف عن نوع البيانات حتى وقت الإستخدام. مصممو لغة C++ أدركوا ذلك ووفروا تقنية تمكن المطورين من تصميم قوالب عامة للفصائل و الدوال. وقد استخدمت هذه التقنية فعلاً في تصميم مكتبة قياسية كبيرة من القوالب أدرجت مع الكثير من ترجمانات C++. في هذا الوحدة نناقش هذه التقنية و نعطي أمثلة على استخدامها. المواضيع التي نعرض لها تشمل:

- قوالب الدوال.
- قوالب الفصائل.
- تخصيص قوالب الفصائل.
- استخدام أكثر من عنصر عام في تصميم قوالب الفصائل.

كما تضمنت الوحدة أسئلة تقويم ذاتي وتدريبات لتساعدك في فهمها وأستيعابها.

أهداف الوحدة



عزيزي الدارس عند فراغك من دراسة هذه الوحدة اتوقع أن تكون قادراً على أن:

1. **تتعرف** على تقنية تصميم القوالب العامة للفصائل والدوال.
2. **تستخدم** أكثر من نوع مطلق في تعريف القوالب.
3. **تخصص** قالب عام للفصيلة.
4. **تستخدم** عناصر إضافية في تعريف القوالب.

الكثير من الخوارزميات تستخدم نفس المنطق والعمليات في أداء عملها، بغض النظر عن نوع المتغيرات الذي تتعامل معه. مثلاً خوارزميات الترتيب (sorting) التي تعاملنا معها في الجزء الأول من هذا الكتاب تستخدم سلسلة من عمليات المقارنة و التبديل أو التحريك للوصول للهدف النهائي. على الرغم من أن المنطق لايتغير، إلا أننا نضطر في الحالات التقليدية لكتابة صورة من الدالة خاصة بالأعداد الحقيقية، و أخرى خاصة بالأعداد الصحيحة، ثم أخرى خاصة بالسجلات ...الخ. الوضع يزداد سوءً في حالة البرمجة الكيانية، ذلك أننا سنضطر لكتابة صورة مختلفة من الفصيلة لكل نوع من الكيانات.

إدراكاً لحجم المشكلة، أدخلت C++ فكرة القوالب (templates). القالب عبارة عن نوع عام (مطلق)، لا لون له ولا طعم، ولارائحة، يستخدم مؤقتاً في تصميم دوال معممة تتغاضى عن ماهية المتغيرات و أنواعها في مرحلة التصميم. القوالب في C++ نوعان: قوالب الدوال وقوالب الفصائل. نبدأ أولاً بقوالب الدوال:

1. قوالب الدوال (Function Templates)

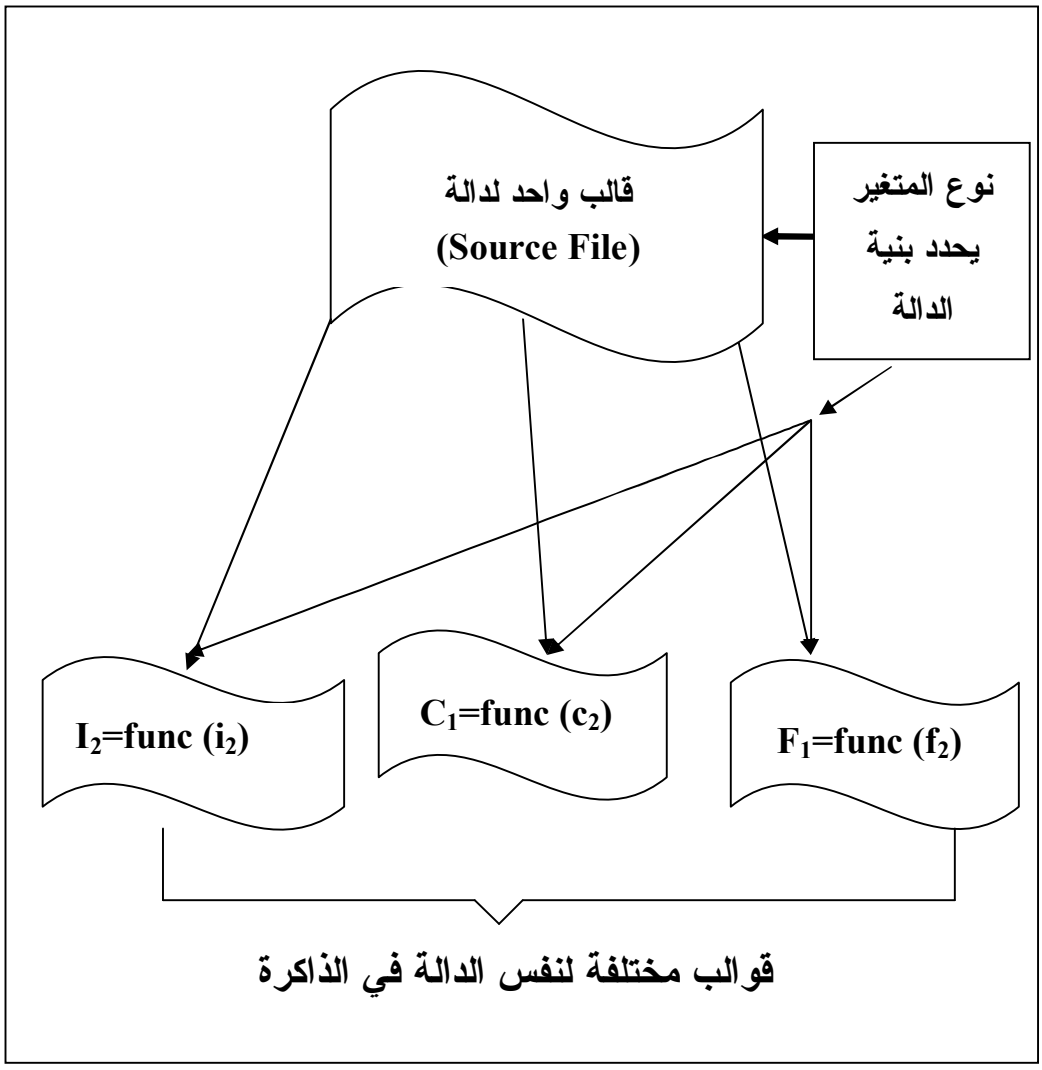
1.1 الشكل العام لتعريف قوالب الدوال

قوالب الدوال يتم تعريفها بالصورة التالية:

```
template  <class template-name> function definition
                                                    أو
template  < typename template-name> function definition
```

كلمة classtemplate و typename كلمات خاصة في C++. أما template-name فهو الاسم الذي يطلقه المستخدم على القالب المعني. بعد اسم القالب يأتي تعريف الدالة.

قد يكون من الجميل أننا نستطيع كتابة دالة معينة مرة واحدة فقط، هذه الدالة يمكنها العمل مع أنواع مختلفة من البيانات، وهذا بالضبط ماتفعله قوالب الدالة ويوضحة الشكل (1).



قالـــــب الدالـــــة Function Template

الشكل (1)

مثال (1)

المثال التالي يوضح بشيءٍ من التفصيل كيفية تعريف قوالب الدوال. يبدأ البرنامج بتعريف دالة معممة ()GetMax، تتعامل مع قالب عام أسميناه GType. لاحظ أننا استخدمنا GType في تعريف العناصر الممررة (parameters) للدالة (a و b) و تعريف الناتج (result). داخل البرنامج استخدمنا الدالة عدة مرات باستخدام أنواع مختلفة من المتغيرات، دون الحاجة إلى إعادة تعريف الدالة.



برنامج (1)

```
// function template
#include <iostream.h>

template <class GType>// function template
GType GetMax (GType a, GType b) {
  GType result;
  result = (a>b)? a : b;
  return (result);
}
int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  float p=5.7, q=8.9, r;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  r=GetMax<float>(p,q);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

ملاحظات على البرنامج

1- يبدأ البرنامج بتعريف قالب الدالة

```
template <class GType>
GType GetMax (GType a, GType b) {
  GType result;
  result = (a>b)? a : b;
  return (result);
}
```

لاحظ أننا استخدمنا الاسم GType لنرمز به للنوع المطلق المعمم. هذا الاسم يستخدم في تعريف كل البيانات التي تتعامل معها الدالة كما تلاحظ.

2- داخل البرنامج، الدالة GetMax() تعاملت مع ثلاثة أنواع من المتغيرات

```
k=GetMax<int>(i,j);
n=GetMax<long>(l,m);
r=GetMax<float>(p,q);
```

في كل حالة يتم تخصيص النوع GType بذكر النوع الفعلي بعد اسم الدالة و قبل العناصر.

أسئلة تقويم ذاتي



1- القوالب طريقة جيدة لتكوين عائلة من:

أ- المتغيرات ب- الدوال

ج- الفصائل د- البرامج

2- ماهو الشكل العام لتعريف قوالب الدوال؟

3- عناصر القوالب تعرف في البرنامج باستخدام الكلمة المحفوظة.....

4- صح أم خطأ:

تنشئ القوالب أوتوماتيكياً إصدارات مختلفة من الدوال إعتماداً على مدخلات المستخدم.

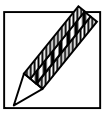
5- اكتب القالب للدالة التي ترجع دائماً العنصر مضروباً في 2 ؟

2.1 استخدام أكثر من نوع مطلق في تعريف القوالب

الأمثلة أعلاه تفترض أن الدالة تستخدم نوع معمم واحد في تعريف العناصر الممررة و الناتج. أحياناً، تحتاج بعض الدوال إلى أكثر من نوع في تعريف عناصرها. C++ تسمح باستخدام أكثر من نوع في تعريف قوالب الدوال. المثال البسيط أدناه يوضح كيف يتم ذلك:

```
template <class T, class U>
T My Function (T  a, U  b)
{  …….
   ……..
   return (……);
}
```

تدريب (1)



1- صمم قالباً عاماً لدالة تقوم بالبحث عن عنصر معين في قائمة معطاة. الدالة ينبغي أن ترجع موقع العنصر في القائمة إذا كان العنصر موجوداً في القائمة، و إلا فترجع القيمة 1- جرب القالب لمتغيرات من نوع integer، ثم متغيرات من نوع string.

2- عدّل القالب أعلاه بحيث يحسب تكرار عنصر معين في القائمة المعطاة.

2. قوالب الفصائل Class Templates

مثلما استخدمنا قوالب الدوال، نستطيع أن نستخدم قوالب فصائل لاتلتزم بنوع معين للبيانات في تعريف الخواص و الدوال المدرجة داخلها. طريقة تعريف قوالب الفصائل تشبه إلى حد كبير طريقة تعريف قوالب الدوال. نبدأ بتعريف اسم القالب، ثم بعد ذلك نستخدم هذا الإسم في تعريف المتغيرات و الدوال داخل الفصيلة.

1.2 الشكل العام لتعريف قوالب الفصائل

```
template <class T>
class ClassName {
   private:
     private members…
     ……
  public:
     public members….
    ……
};
```

مثال (2)

المثال التالي يوضح هذه الطريقة. يبدأ البرنامج بتعريف قالب فصيلة Test Class ثم يستخدم هذه الفصيلة في تعريف كيان داخل البرنامج بعد تخصيص النوع العام إلى int.



برنامج (2)

```
// class templates
#include <iostream.h>

template <class T>
class TestClass {
    T value1, value2;
  public:
    TestClass (T first, T second)
      {value1=first; value2=second;}
    T getmax ();
};

template <class T>
T TestClass<T>::getmax ()
{
  T retval;
  retval = value1>value2? value1 : value2;
  return retval;
}

int main () {
  TstClass <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

ملاحظات على البرنامج

1- يبدأ البرنامج بإعطاء اسم للنوع المعمم T، ثم يستخدم هذا النوع في تعريف متغيريين ودالتين (الأولى منهم دالة الإنشاء).

2- دالة الإنشاء تعرف بالكامل داخل الفصيلة (inline)، أما الدالة الأخرى GetMax() فيعطى تعريفها الكامل خارج الفصيلة.

3- عند تعريف GetMax خارج الفصيلة أعدنا تعريف اسم النوع المعمم T قبل تعريف الفصيلة، واستخدمنا هذا النوع المعمم قبل اسم الفصيلة وقبل اسم الدالة (في الواقع قبل علامة النطاق)، وهذه قاعدة عامة:

```
template <class T>
T  TestClass<T>::getmax ()
{
  T  retval;
  retval = value1>value2? value1 : value2;
  return retval;
}
```

2.2 تخصيص القوالب

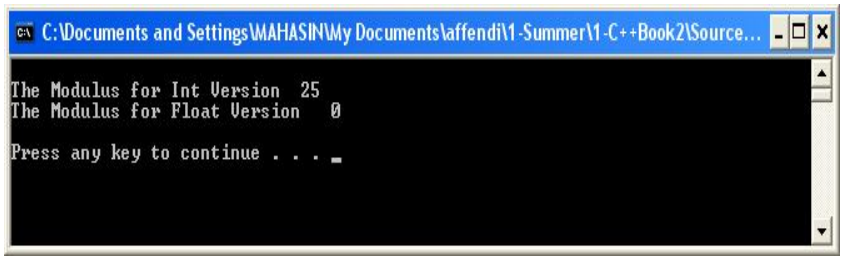
أحياناً يتطلب المقام التعامل بصورة خاصة مع بعض الأنواع الفعلية بعد التعريف العام لقالب الفصيلة. مثلاً، قد نقوم بتعريف قالب عام لفصيلة، ثم نغير طريقة تنفيذ بعض العمليات في حالة الأعداد الصحيحة. C++ تسمح في هذه الحالة بتخصيص القالب العام للفصيلة وتصميم نسخة مخصصة من الفصيلة على النحو الذي يفي بالغرض.

مثال (3)

في المثال أدناه، مثلاً، العملية Modulus() لامعنى لها، إلا إذا كان النوع الذي نتعامل معه هو الأعداد الصحيحة. لذا تم تصميم نسخة مخصصة من الفصيلة العامة لإعطاء طريقة مختلفة لإجراء الدالة Modulus:

برنامج(3)

```
// Template specialization
#include <iostream>
using namespace std;
template <class T>
class XYPoint {
    T value1, value2;
  public:
    XYPoint (T first, T second)
      {value1=first; value2=second;}
    T modulus () {return 0;}
};
template <> // الصورة المخصصة من الفصيلة
class XYPoint <int>
{
    int value1, value2;
  public:
    XYPoint (int first, int second)
      {value1=first; value2=second;}
    int modulus (){   return value1%value2; }
};
int main () {
  XYPoint <int> MyIntPoint (100,75);
  XYPoint <float> MyFloatPoint (100.0,75.0);
  cout << "The Modulus for Int Version  "
       <<MyIntPoint.modulus() << '\n';
  cout << "The Modulus for Float Version   "
       <<MyFloatPoint.modulus() << '\n';
  system("pause");
  return 0;
}
```



لاحظ أننا استخدمنا القالب الشاغر template < > في تعريف الصورة المخصصة من الفصيلة وإعطاء الصورة الجديدة لإجراء الدالة Moduls():

```
template <>
class XYPoint <int> {
   int value1, value2;
 public:
   XYPoint (int first, int second)
     {value1=first; value2=second;}
   int modulus (){  return value1%value2; }
};
```

3.2 استخدام عناصر إضافية في تعريف القوالب

C++ تسمح باستخدام عناصر إضافية في تعريف قوالب الفصائل. المثال التالي يوضح كيف يمكن استخدام عنصر إضافي في تعريف القالب لتوفير المزيد من المرونة.

مثال إيضاحي (4)

هذا المثال يعطي قالباً لتعريف مصفوفات من أي نوع. التعريف يستخدم عنصراً إضافياً يمثل حجم المصفوفة. بهذه الطريقة يستطيع المستخدم أن يتحكم في حجم المصفوفة.

```
// array template
// array template
#include <iostream>
using namespace std;
template <class T, int N>
class array {
    T MBlock [N];
  public:
    void SetMember (int x, T value);
    T GetMember (int x);
};
template <class T, int N>
void array<T,N>::SetMember (int x, T value)
{   MBlock[x]=value; }

template <class T, int N>
T array<T,N>::GetMember (int x) {
  return MBlock[x];
}
int main () {
  array <int,5> IntArray;
  array <float,5> FloatArray;
  IntArray.SetMember (0,100);
  FloatArray.SetMember (3,3.1416);
  cout << IntArray.GetMember(0) << '\n';
  cout << FloatArray.GetMember(3) << '\n';
  system("pause");
  return 0;
}
```

لاحظ استخدام العنصر الإضافي في تعريف القالب:

```
template <class T, int N>
class array {
   T MBlock [N];
  public:
   void SetMember (int x, T value);
   T GetMember (int x);
};
```

داخل البرنامج، استخدمنا القالب في تعريف مصفوفتين' إحداهما صحيحة و الأخرى حقيقية:

```
array <int,5> IntArray;
array <float,5> FloatArray;
```

مثال تطبيقي (5)

في هذا المثال نصمم قالباً لنوع خاص من تراكيب البيانات يسمى "المِرَصَّة" (stack). المرصة عبارة عن نوع خاص من القوائم يقبل الحذف و الإضافة من جهة واحدة فقط هي أعلى القائمة. تخيل أن لديك صندوق ضيق، عميق ترص فيه مجموعة من الكتب. بما أن الصندوق ضيق، فلا يمكن الإضافة إلا من فوهته العليا. إذا اردنا كتاباً في أسفل الصندوق، فلابد من إزاحة ما وضعنا فوقه أولاً. القاعدة العامة هنا هي "مايدخل آخراً، يخرج أولاً" (Last-in-first-out).

هناك عمليتان أساسيتان على المرصة: الإضافة (push()) و الحذف (pop()). المثال أدناه يعطي قالباً عاماً للمرصة يمكن استخدامه مع أي نوع من البيانات:

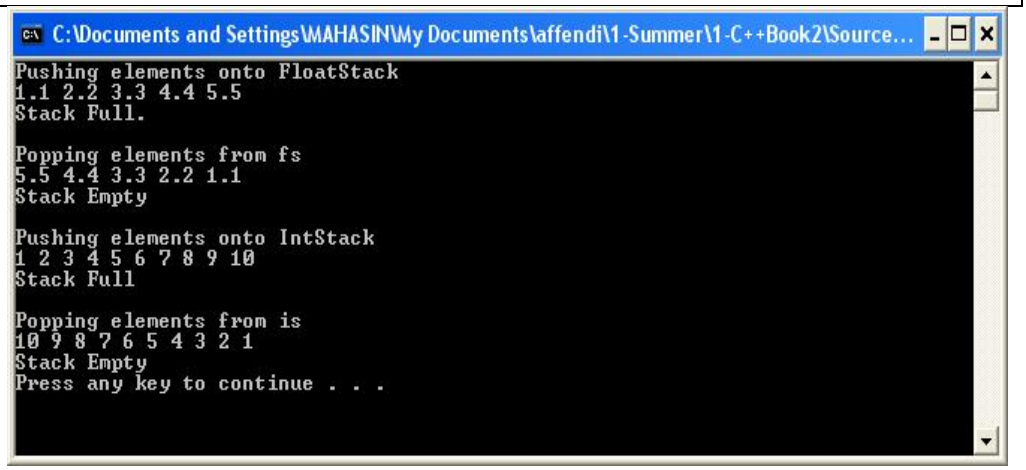
برنامج (5)

```
//stack template
#include <iostream>
template <class T>
class Stack
{
public:
     Stack(int = 10) ;
     ~Stack() { delete [] stackPtr ; }
     int push(const T&);
     int pop(T&) ;   // pop an element off the
stack
     int isEmpty()const { return top == -1 ; }
     int isFull() const { return top == size - 1 ;
}
private:
     int size ;  // Number of elements on Stack
     int top ;
     T* stackPtr ;
} ;
//constructor with the default size 10
template <class T>
Stack<T>::Stack(int s)
{
     size = s > 0 && s < 1000 ? s : 10 ;
     top = -1 ;  // initialize stack
     stackPtr = new T[size] ;
}
 // push an element onto the Stack
template <class T>
int Stack<T>::push(const T& item)
{
     if (!isFull())
     {
          stackPtr[++top] = item ;
          return 1 ;  // push successful
     }
```

```
     return 0 ;  // push unsuccessful
}
// pop an element off the Stack
template <class T>
int Stack<T>::pop(T& popValue)
{
     if (!isEmpty())
     {
          popValue = stackPtr[top--] ;
          return 1 ;  // pop successful
     }
     return 0 ;  // pop unsuccessful
}
using namespace std ;
int main()
{
     Stack<float> FloatStack(5) ;
     float f = 1.1 ;
     cout << "Pushing elements onto FloatStack" <<
endl ;
     while (FloatStack.push(f))
     {
          cout << f << ' ' ;
          f += 1.1 ;
     }
     cout << endl << "Stack Full." << endl
     << endl << "Popping elements from fs" << endl
;
     while (FloatStack.pop(f))
          cout << f << ' ' ;
     cout << endl << "Stack Empty" << endl ;
     cout << endl ;

     Stack<int> IntStack ;
     int i = 1 ;
     cout << "Pushing elements onto IntStack" <<
endl ;
     while (IntStack.push(i))
```

```
    {
        cout << i << ' ' ;
        i += 1 ;
    }
    cout << endl << "Stack Full" << endl
    << endl << "Popping elements from is" << endl
;
    while (IntStack.pop(i))
            cout << i << ' ' ;
    cout << endl << "Stack Empty" << endl ;
    system("pause");
    return 0;
}
```



ملاحظات على البرنامج

1- دالة الإنشاء (Stack()) تقوم بتحديد حجم المرصة حسب العنصر الممرر و تجعل مؤشر المرصة stack Ptr يشير إلى أعلى المرصة:

```
template <class T>
Stack<T>::Stack(int s)
{
     size = s > 0 && s < 1000 ? s : 10 ;
     top = -1 ;  // initialize stack
     stackPtr = new T[size] ;
}
```

2- الدالة ()push تضيف عنصراً إلى المرصة من أعلاها بعد التأكد من أن المرصة غير مملوءة:

```
template <class T>
int Stack<T>::push(const T& item)
{
     if (!isFull())
     {
          stackPtr[++top] = item ;//عملية الإضافة
          return 1 ;  // push successful
     }
     return 0 ;  // push unsuccessful
}
```

3- عملية الحذف ()pop تقوم بحذف عنصر من أعلى القائمة بعد التأكد من أن المرصة ليست فارغة:

```
template <class T>
int Stack<T>::pop(T& popValue)
{
     if (!isEmpty())
     {
          popValue =stackPtr[top--] ; //عملية الحذف
          return 1 ;  // pop successful
     }
     return 0 ;  // pop unsuccessful
}
```

أسئلة تقويم ذاتي



1- ماهو الشكل العام لتعريف قوالب الفصائل؟

2- قالب الفصيلة:

أ- يصمم لحفظ في اماكن مختلفة ب- يعمل مع أنواع مختلفة من البيانات

ج- يتعامل مع كل كيان بشكل مفرد

د- يتعامل مع الفصائل التي لها أعداد مختلفة من الدوال الفصوية (member function).

3- صمم قالباً عاماً لدالة تقوم بالبحث عن عنصر معين في قائمة معطاة. الدالة ينبغي أن ترجع موقع العنصر في القائمة إذا كان العنصر موجوداً في القائمة، و إلا فترجع القيمة -1. جرب القالب لمتغيرات من نوع integer، ثم متغيرات من نوع string.

4- عدّل القالب أعلاه بحيث يحسب تكرار عنصر معين في القائمة المعطاة.

تدريب (2)



1- اكتب برنامج قالب لدالة تبحث في مصفوفة عن قيمة معينة (Specific Value)
ترجع الدالة رتبة القيمة المعينة في المصفوفة؟

2- وضح كيف يقوم المترجم (Compiler) بالتعامل مع القالب في السؤال الأول.

3- صمم قالباً لفصيلة اسمها List Search تحتوي على مصفوفة ودالة ()Search تبحث عن عنصر معين في القائمة. اختبر التصميم في برنامج كامل.

4- صمم قالباً لفصيلة Array Sort يحتوي على مصفوفة ودالة ()sort تقوم بترتيب محتويات المصفوفة. يمكن استخدام خوارزمية الترتيب التي عرضنا لها في الجزء الأول من الكتاب.

5- أعد حل السؤال 2 في التدريب (1) باستخدام قالب فصيلة بدلاً من قالب دالة.

الخلاصة

لنقم عزيزي الدارس بتلخيص ما تعلمناه في هذه الوحدة في نقاط هامة وهى:

- تسمح C++ باستخدام قوالب عامة للدوال والفصائل لاتتقيد بأنواع معينة للبيانات وقت التصميم. هذا يعطي قدراً كبيراً من المرونة ويريح المطورين من إعادة المجهود.
- يمكن استخدام أكثر من عنصرعام في تصميم قوالب الفصائل والدوال.
- بالإمكان تخصيص قوالب الفصائل العامة للتعامل مع بعض الحالات الإستثنائية.
- C++ توفر مكتبة قياسية للقوالب العامة تشبه إلى حد كبير مكتبة الدوال القياسية، ولكنها بالطبع أكثر مرونة.

أرجو أن تكون استوعبت هذه النقاط العامة وأنصحك أن لا تتردد في مراجعة الوحدة أكثر من مرة إذا رأيت ذلك، ويمكنك طلب المساعدة من مشرفك الأكاديمي إذا صعب عليك جزء منها.

لمحة مسبقة عن الوحدة الدراسية التالية

عزيزي الدارس: في هذا الوحدة القادمة نغطي موضوعاً بسيطاً، ولكنه ذو أهمية كبيرة في تصميم نظم البرامج الحديثة، ذلك هو موضوع مجالات الأسماء (namespaces). المقصود هنا مجالات أسماء المتغيرات. C^{++} تسمح باستخدام مجالات أسماء مختلفة للتمييز بين اسماء المتغيرات، ولإعطاء المطورين حرية أكبر في استخدام اسماء المتغيرات دون وجلٍ من إمكانية تضارب هذه الأسماء.

إجابات التدريبات

تدريب (2)

1- // template use for function that finds number in array #

```
#include   < iostream>
using namespace std;
//……
// function retyrns index number of item, or if not found
template <class a type>
int find (a type* array, a type value, int size)
{
for (int j=o; j++)
if (array [j]= =value)
  return j ;
return-1;
}
//……
Char chr Arr [ ]= {1,3,5,9,11,13}; // array
Char Ch = s;
int  intArr [ ] = {1,3,5,9,11,13};
int   in = b;
long  lognArr [ ] = {1L,3L,5L,9L,11L,13L},
long  lo = 11L;
double  dubArr [ ] = {1.0,3.0,5.0,9.0,11.0,13.0};
double   db = 4.0;
int  main ( )
{
cout <<"/In 5 in chrArray: index= " << find (chrArr,ch,6);
cout <<"In 6 in intArray: index=" << find (intArr,int,6);
cout <<"In 11 in longArray: index="<<  find longArr,10,6;
cout <<"In 4 in dubArray: index=" << find (dubArr,db,6);
```

```
    cout << end L;
    return Q;
    }
```

2- تكون مخرجات البرنامج كالآتي:

```
 5 in chrArray: index = 2
 6 in intArray: index = -1
11 in lonArray: index = 4
 4 in dubArray: index = -1
```

يتعامل المترجم مع أربعة إصدرات مختلفة من الدالة كل واحدة لنوع مختلف من البيانات.

1- **بطريقة اخرى**

```
#include<iostream.h>
template<class T<
int ArraySearch(T *array,int, int)
{
const ArraySize=100؛
int a[ArraySize],SearchKey,Element;
for(int x=0;x<SearchKey;x(++)
cout<<"enter integer Search Key :"<<endl;
Element=ArraySearch(a,SearchKey,ArraySize);
if(Element!= -1( )
cout<<"Found value in Element : " <<Element<<endl<<endl;
else
cout<<"Vlue not found"<<endl;
return 0;
int ArraySearch(int array[],int Key,int SizeOfArray( )
for(int n=0;n<SizeOfArray;n(++ )
if(array[n]== Key(  )
return n;
```

```
return -1
}
main()
{
int SearchKeyA = 3;
float SearchKeyB= 2.2;
char SearchKeyC= 'A;
int arrA[3]={2,3,5 ; }
float arrB[3]={1.,2.2,3.3 ;
char arrC[3]= "abc  ";
}
3−
#include<iostream.h>
template<class T>
int ArraySearch(T *array,int, int( )
{
const ArraySize=10 ;
int a[ArraySize],SearchKey,Element ;
for(int x=0;x<SearchKey;x(++)
cout<<"enter integer Search Key :"<<endl ;
Element=ArraySearch(a,SearchKey,ArraySize) ;
if(Element!= -1)
cout<<"Found value in Element : " <<Element<<endl<<endl؛
else
cout<<"Vlue not found"<<endl ;
return 0 ;
int ArraySearch(int array[],int Key,int SizeOfArray)
for(int n=0;n<SizeOfArray;n (++)
if(array[n]== Key)
return n ;
return -1
```

```
}
main()
{
int SearchKeyA = 3 ;
float SearchKeyB= 2.2 ;
char SearchKeyC= 'A ;
int arrA[3]={2,3,5 } ;
float arrB[3]={1.,2.2,3.3} ;
char arrC[3]= "abc ;
  }
4-
#include<iostream.h >
#include<iomanip.h >
template<class T>
class ArraySort
{
int ArraySize,hold ;
public:
Sort(T *array,int)
{
cout<<"Data Items in Orginal Order :"<<endl ;
)for(int i=0;i<=ArraySize;i(++)
cout<<setw(4)<<a} ;
for(int pass= 1;pass<ArraySize;pass(++)
for(i=0;i< ArraySize-1;i(++)
if(a[i]>a[i+1   ;
hold=a[I ;
a[i]=a[i+1 ] ;
a[i+1]=hold ;
}
cout<<endl<<"Data Items in Ascending Order : " <<end ;
```

```
for(i=0;i<ArraySize;i(++)
cout<<setw(4)<<a[I؛ ] ;
cout<<endl;
return n ;
return -1  ;
{
}
main()
{
ArraySort<int> SearchKeyA = 3  ;
ArraySort<float> SearchKeyB= 2.2 ;
ArraySort<char> SearchKeyC= 'A ;
int arrA[3]={2,3,5   } ;
float arrB[3]={1.,2.2,3.3 };
char arrC[3]= "abc  " ;
}
```

مسرد المصطلحات

*** قوالب الدوال Functions Templates**

هى نوع عام مطلق يصمم لدالة معمم تتغاضى عن ماهية المتغيرات وأنواعها.

*** قوالب الفواصل Class Templates**

تصميم لفصائل عامة لا تلتزم بنوع معين للبيانات في تعريف الخواص والدوال المدرجة

داخلها.

*** المِرصّة (المكدسة) Stack**

عبارة عن نوع خاص من القوائم يقبل الحذف والإضافة من جهة واحدة فقط هى أعلىالقائمة.

المصادر والمـــراجـــع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.


الوحدة السابعة
مجالات الأسماء
Namespaces

محتويات الوحدة

الموضوع	الصفحة
المقدمة	207
تمهيد	207
أهداف الوحدة	207
1. مجال الأسم	208
2. استخدام Using	210
الخلاصة	214
مسرد المصطلحات	214
لمحة مسبقة عن الوحدة التالية	215
المراجع	216

المقدمة

تمهيد

مرحباً بك عزيزي الدارس، في هذه الوحدة السابعة من هذا المقرر، في مجال البرمجة فأن واحدة من المشاكل التي تواجه مطوري البرامج الكبيرة هي إمكانية تضارب أسماء المتغيرات في البرنامج، ذلك أن البرنامج الواحد قد يشترك في تطويره أكثر من شخص. وقد أصبحت هذه المشكلة أكثر وضوحاً بعد انتشار التطبيقات الموزعة ونظم الويب. في مثل هذه الحالات يتركب البرنامج من أجزاء كثيرة، تأتي من مصادر مختلفة، وربما تشابكت أسماء المتغيرات و الدوال. لحل هذه المشكلة، تسمح لغة C++ للمطورين باستخدام ما يعرف ب "مجال الإسم" (Namespace).

أهداف الوحدة



بعد فراغك عزيزي الدارس من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تعرف مجال الاسم
2. Using في اضفاء مجال الاسم في مستوى معين في البرنامج

1. مجال الاسم Namespace

مجال الاسم عبارة عن **منطقة في البرنامج تحتوي على عدة متغيرات.** إذا أردنا **استخدام هذه المتغيرات خارج مجالها،** فلابد من **نسبتها إلى المجال الذي يحويها باستخدام رمز النطاق** "::".

المثال التالي يوضح ذلك:

مثال (1)



برنامج(1)

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

namespace SpaceOne
{
  int MyVar = 27;
  float x=4.5;
}
namespace SpaceTwo
{
  double MyVar = 79.3654;
  int y=3;
}
int main () {
  cout << SpaceOne::MyVar << endl;
  cout << SpaceTwo::MyVar << endl;
  return 0;
}
```

البرنامج أعلاه يحتوي على **مجالي أسماء. المجال الأول** يحتوي على **متغيرين MyVar و x. المجال الثاني** يحتوي على **متغير اسمه MyVar** ايضاً، ولكنه **من نوع مختلف (double):**

المجال الأول:

```
namespace SpaceOne
{
  int MyVar = 27;
  float x=4.5;
}
```

المجال الثاني:

```
namespace SpaceTwo
{
  double MyVar = 79.3654;
  int y=3;
}
```

عند **استخدام هذه المتغيرات** في البرنامج لابد من **نسبتها إلى المجال الذي عرِّفت فيه** لإزالة اللبس:

```
cout << SpaceOne::MyVar << endl;
cout << SpaceTwo::MyVar << endl;
```

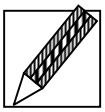
لاحظ استخدام **رمز النطاق** "::" في نسبة المتغير إلى مجاله.

أسئلة تقويم ذاتي



1. هل يمكن في المثال أعلاه **استخدام** المتغير x و المتغير y **من غير** إسنادهما إلى مجاليهما؟ (لاحظ أن x غير موجود في المجال الثاني، y غير موجود في المجال الأول).

تدريب(1)



1. **أكتب برنامجاً** يحتوي على مجال أسماء university **وعرف فصيلتين** داخل هذا المجال: student و course. **ضع** أي محتويات داخل الفصيلتين و**أطبعهما**.

2. استخدام using

إذا **تكرر استخدام مجال إسم في منطقة معينة من البرنامج** فيمكن **استخدام الكلمة الخاصة using مشفوعةً باسم المجال، بدلاً من تكرار اسم المجال داخل البرنامج.**
الأمثلة التالية توضح **كيف تستخدم جملة using:**

مثال (2)



برنامج(2)

```
// using namespaces
#include <iostream.h>

namespace SpaceOne
{
  int MyVar = 27;
  float x=4.5;
}

namespace SpaceTwo
{
  double MyVar = 79.3654;
  int y=3;
}

int main ()
{ using namespace SpaceOne;
  cout << MyVar << endl;
  cout << MyVar*x << endl;
  return 0;
}
```

لاحظ أنه إذا **استخدمنا جملة using لإضفاء مجال اسم في مستوى معين من البرنامج**، فإنه **لايمكن إضفاء مجال إسم آخر في نفس المستوى من البرنامج باستخدام جملة using**. مثلاً، البرنامج التالي سيعطي **رسالة خطأ:**

```
namespace SpaceOne
{
  int MyVar = 27;
  float x=4.5;
}
namespace SpaceTwo
{
  double MyVar = 79.3654;
  int y=3;
}
int main ()
{using namespace SpaceOne;
  cout << MyVar << endl;
  cout << MyVar*x << endl;
  using namespace SpaceTwo; //خطأ
  cout << MyVar << endl;
  cout << MyVar*y<< endl;

  return 0;
}
```

الحل في هذه الحالة هو **استخدام مستوى مختلف لكل مجال إسم**. البرنامج التالي يحدث مستويين مختلفين **باستخدام الأقواس {} ويستخدم مجال إسم مختلف في كل مستوى:**

مثال(3)



برنامج(3)

```
namespace SpaceOne
{
  int MyVar = 27;
  float x=4.5;
}

namespace SpaceTwo
{
  double MyVar = 79.3654;
  int y=3;
}

int main ()
{ // مستوى 1
   {using namespace SpaceOne;  //2 مستوى
    cout << MyVar << endl;
    cout << MyVar*x << endl;
  }
  { using namespace SpaceTwo; //3 مستوى
    cout << MyVar << endl;
    cout << MyVar*y<< endl;
  }
  return 0;
}
```

نشاط (1)



ياترى هل تعمل هذه البرنامج؟

```
namespace SpaceOne
{
  int MyVar = 27;
  float x=4.5;
}
namespace SpaceTwo
{
  double MyVar = 79.3654;
  int y=3;
}
int main ()
{ //1 مستوى
  using namespace SpaceOne;
  cout << MyVar << endl;
  cout << MyVar*x << endl;
  { using namespace SpaceTwo; //2 مستوى
    cout << MyVar << endl;
    cout << MyVar*y<< endl;
  }
  return 0;
}
```

لاحظ أن **الكثير من نظم C++ الحديثة تتطلب نسبة دوال مكتبة C++القياسية (standard C++ library) إلى مجال الأسماء std.** مثلاً، **الدالة cout** التي أكثرنا من استخدامها في الأمثلة السابقة **تنتمي إلى هذا المجال، وقد يتطلب المقام استخدامها بالصورة التالية:**

```
std::cout << MyVar << endl;
```

أو استخدام الجملة:

```
using namespace std;
```

في صدر البرنامج.

نظام Dev-C++ يتطلب استخدام نطاق الإسم std مع cout و شبيهاتها من الدوال.

الخلاصة

ناقشنا في هذه الوحدة واحدة من المشاكل التي تواجه مطوري البرامج الكبيرة وهـــي إمكانية تضارب أسماء المتغيرات في البرنامج، الكبيرة وهي إمكانية تضـــارب اســـماء المتغيرات في البرنامج، لحل هذه المشكلة تسمح لغة C++للمطورين باستخدام مايعرف بـــ مجال الاسم من أهم النقاط:

- **استخدام رمز ( ) المتغير للمجال الذي ينتمي له.**
- **يمكن استخدام الكلمة مصفوعة باسم المجال بدلاً من تكرار اسم المجال داخل البرنامج.**

مسرد المصطلحات

- **مجال الاسم Name space**

منطقة في البرنامج تحتوي على عدة متغيرات.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس، البحث عن أسباب الأخطاء المنطقية في البرامج (debugging) يستغرق وقتاً طويلاً. في كثيرٍ من الأحيان يشعر المبرمج أن ثمة خللٌ ما في البرنامج، ولكن يصعب عليه أن يدرك أين الخطأ. سي++ تسمح للمبرمج أن يحتاط لمثل هذه الحالات باستخدام أوامر الإستثناء (Exceptions) ، هذا ما سنتناوله في الوحدة التالية.

المصادر والمراجع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.

الوحدة الثامنة

الحالات الإستثنائية

Exceptions



محتويات الوحدة

الموضوع	الصفحة
المقدمة	219
تمهيد	219
أهداف الوحدة	220
1. الأستثناء	221
1.1 أساليب معالجة الاستثناءات	221
2. فصائل الاستثناءات و كيفية استخداماتها.	228
خلاصة	236
المراجع	237

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في هذه الوحدة

يستطيع المبرمج أن يحيط أماكن الشك التي يتوقع أن تحوي أخطاء منطقية في البرامج (debugging) بأوامر خاصة تترقب الأخطاء المحتملة وتقوم بمعالجتها حين وقوعها. في هذا الفصل نتناول أوامر الاستثناء (Exceptions) وكيفية إستخدامها في البرامج.

في هذه الوحدة ادرجنا تدريبات واسئلة تقويم ذاتي حتى تتمكن من استيعابها وفهمها.

هناك نوعان من الأخطاء البرمجية:

أ- **أخطاء تركيبية** تتعلق بقواعد اللغة، و هذه عادةً **يكتشفها ترجمان** اللغة **قبل** البدء في تشغيل البرنامج.

ب-**أخطاء منطقية لا يتم اكتشافها** إلا **بعد** البدء في تشغيل البرنامج.

الأخطاء المنطقية أصعب اكتشافاً من الأخطاء التركيبية اللغوية وقد يكون لها **نتائج وخيمة على أداء البرنامج**، وربما يتأخر اكتشافها حتى بعد إصدار البرنامج.

بعض الأخطاء المنطقية يتسبب في

1. تعطيل البرنامج،
2. يؤدي إلى نتائج خاطئة .

إدراكاً لخطورة هذا النوع من الأخطاء، أصبحت الكثير من لغات البرمجة الحديثة **توفر بعض الأساليب التي تساعد في اكتشافها و معالجتها بما يناسب المقام.**

الأخطاء المنطقية (التي تحدث أثناء تشغيل البرنامج) يطلق عليها "استثناءات" (Exceptions) في مصطلح العاملين في مجال هندسة البرمجيات.

أهداف الوحدة



بعد فراغك عزيزي الدارس من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. **تعرف** أنواع الأخطاء البرمجية وتميز الاستثناءات.
2. **تستخدم** أساليب معالجة الاستثناءات.

1. الاستثناء (exception)

يوصف بأنه حالة شاذة تحدث أثناء تنفيذ البرنامج.

مثل:

- محاولة القسمة على صفر.
- عدم وجود فراغ كافي في الذاكرة لاستيعاب قيمة متغير معين.

1.1 أساليب معالجة الاستنثاءات

Exception Handling Techniques

نحاول التنبؤ بهذه الاستنثاءات، و معالجتها بالصورة المناسبة.

1. أحياناً يكون من المناسب طباعة رسالة تدل على نوع الاستثناء الذي حدث و الخروج من البرنامج.
2. أحياناً أخرى يكون الأنسب معالجة الاستثناء و الاستمرار في البرنامج، وكأن شيئاً لم يحدث.

C++ توفر ثلاثة عوامل لمعالجة الاستثناءات هي: try و throw و catch. هذه العوامل لابد أن تكون متتالية في البرنامج، لايفصل بينها شيء، ويكون استخدامها عادةً بالصورة التالية:

```
try {
  // code to be tried
  throw exception;
}
catch (type of exception)
{
  // code to be executed in case of
exception
}
```

من الهيكل العام أعلاه **يتضح** أن :

كلمة try

- تكون **متبوعة** بفقرة برمجية تحتوي على **الأوامر المراد محاولة تنفيذها** (منطقة الشك) بالإضافة إلى **جملة throw.**
- عادةً تحاول **جملة try تنفيذ الأوامر المدرجة داخلها،** فإذا **فشلت استخدمت جملة throw** في إثارة الاستثناء.

جملة throw

- تكون متبوعة بالاستثناء (Exception) الذي يكون عادة:

- متغيراً من أيَّ نوع.

-جملة عادية تعطي نوع الخطأ.

- رقماً صحيحاً يدل على الخطأ .

- كياناً متكاملاً يعطي نوع الخطأ .

بعض الطرق التي يمكن أن تستخدم في معالجة الخطأ:

جملة catch التي ينبغي أن **تأتي مباشرةً بعد جملة try، مهمتها استقبال الاستثناءات التي تثيرها جملة try ومعالجتها.**

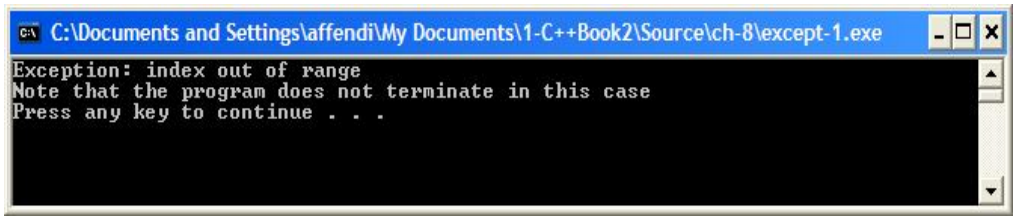
لاحظ أن جملة catch **تستخدم** عنصر ممرر. **الشرط الأساسي** في استقبال الاستثناءات المثارة هو أن **يتوافق نوع الاستثناء المثار مع نوع العنصر الممرر لجملة catch. كلما** أثارت جملة **try** استثناءً، **تنظر** جملة **catch** في نوع الاستثناء المثار، **فإذا كان متوافقاً** مع نوع العنصر الممرر، **استقبلته و عالجته، و إلا تجاهلته.**

مثال (1)

المثال التالي **يوضح عملياً كيف تثار الإستثناءات و كيف تستقبل و تعالج. يبدأ** البرنامج **بتعريف مصفوفة بسيطة ومتغيراً حرفياً OutOfRange يمثل نوع الاستثناء. بعد ذلك** يحاول البرنامج **تنفيذ أوامر تقوم بتعبئة المصفوفة** (داخل جملة try). **إذا واجه البرنامج حالة** تكون فيها **n أكبر من حجم المصفوفة** يقوم **بإثارة**

استثناء من النوع OutOfRange. جملة catch تتلقف هذا الإستثناء وتقوم **بمعالجته لأنه متوافق** مع نوع العنصرالممرر e. المعالجة في هذه الحالة **تقتصر على طباعة رسالة تبين نوع الخطأ الذي حدث.**

```
// exceptions
#include <iostream>
using namespace std;
int main () {
  int IntArray[10];
  string OutOfRange ="index out of range";
  try
  {
    for (int n=0; n<=10; n++)
    {
      if (n>9) throw OutOfRange;   //إثارة الاستثناء
      IntArray[n]=n*n;
    }
  }
  catch (string e)
  {
 cout << "Exception: " << e << endl;// معالجة
الاستثناء
  }
  cout <<"Note that the program does not
terminate in this case"<<endl;
  system("pause");
  return 0;
}
```



لاحظ أن البرنامج **يستمر في التنفيذ بعد** جملة catch، **رغم إثارة الإستثناء. السبب** في ذلك أن **جملة catch اختارت** هذا الأسلوب في معالجة الخطأ **ولم تقم** بإيقاف البرنامج. **أحياناً** تستدعي المعالجة إيقاف البرنامج أو تعليقه.

نشاط(1)



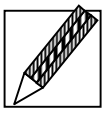
أعط بعض الأمثلة للأخطاء التركيبية و المنطقية في برامج C++ ونفذها في جهازك.

أسئلة تقويم ذاتي



1- ما الفرق بين الأخطاء التركيبية و الأخطاء المنطقية؟
2- ما الاستثناءات؟ وكيف يتم علاجها في لغة C++
3- ما الهيكل العام لجملة try…catch؟

التدريب(1)



1- نفذ البرنامج المعطى في مثال(1) من غير استخدام جملة try…catch و انظر ماذا يحدث. استخدم 100 بدلاً من 10 كحد أقصى للمتغير n في جملة for، ثم كرر تنفيذ البرنامج من غير جملة try…catch و انظر مايحدث.
2- اكتب برنامجاً يحسب رواتب العمال في مؤسسة معينة (عدد الساعات مضروباً في أجر الساعة الواحدة) ويثير استثناءً إذا كان الناتج (الراتب) أعلى من 10000.

1.1.1 استخدام جمل catch المتعددة

أحياناً يتطلب المقام إثارة أكثر من استثناء في جملة try واحدة، وبالتالي **استخدام أكثر من جملة catch واحدة بعد جملة try. C++ تسمح باستخدام جملة throw أكثر من مرة داخل جملة try لإثارة أكثر من استثناء.** المثال التالي يوضح كيف يتم ذلك:

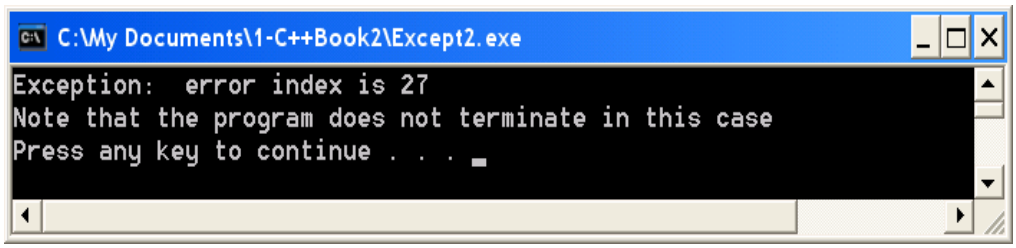
مثال (2)

في هذا المثال **جملة try تستخدم جملتي throw، إحداهما تثير استثناءً من النوع OutOfRange** و **الأخرى تثير استثناءً من النوع OverFlow**. هناك **جملتا catch بعد جملة try إحداهما لمعالجة الاستثناء الأول** (العنصر الممرر متغير ألفبائي) **والأخرى لمعالجة الإستثناء OverFlow** (العنصر الممرر متغير صحيح).

```
// exceptions: multiple catch blocks
#include <iostream>
using namespace std;
int main () {
  int IntArray[10];
  string OutOfRange ="index out of range";
  int OverFlow=27;
  int sum;
  try
  {
    for (int n=0; n<=10; n++)
    {
      if (n>9) throw OutOfRange; //إثارة الإستثناء الأول
      {IntArray[n]=n*n;
    if (sum>200) throw OverFlow;//إقارة الإستثناء الثاني
        sum=sum+IntArray[n];
      }
    }
  }
```

```
  catch (string e) //معالجة الإستثناء الأول
  {
    cout << "Exception: " << e << endl;
  }
  catch (int e)//معالجة الإستثناء الثاني
  {
    cout << "Exception:  error index is " << e
<< endl;
  }

  cout <<"Note that the program does not
terminate in this case"<<endl;
  system("pause");
  return 0;
}
```



2.1.1 استخدام جملة catch عامة

أحياناً، **يصعب** على المبرمج **التنبؤ** بنوع الخطأ الذي سيحدث، **ويهمه فقط** أن يعرف **أن خطأً من نوعٍ ما قد حدث في منطقة معينة من البرنامج، أو يفضل** أن **يعالج كل الأخطاء المحتملة** باستخدام **جملة catch واحدة. C++ تسمح** باستخدام **جملة catch معممة** يمكن استخدامها في مثل هذه الحالات. **جملة catch المعممة تستخدم ثلاثة نقاط أفقية متتالية بدلاً من العنصر الممرر، كنايةً عن** أنها **لا يهمها نوع الإستثناء.**

الشكل العام لجملة catch المعممة هو:

```
try {
  // code here
}
catch (...) //لاحظ استخدام الثلاث نقاط
{
  cout << "Exception occurred";
}
```

مثال(3)

في هذا المثال **نضيف جملة catch معممة** للبرنامج الذي قدمناه في **المثال السابق**.

جملة catch المعممة تستخدم في التعامل مع الاستثناءات غير المعروفة:

```
// exceptions: multiple catch blocks
#include <iostream>
using namespace std;
int main () {
  int IntArray[10];
  string OutOfRange ="index out of range";
  int OverFlow=27;
  int sum;
  try
  {
    for (int n=0; n<=10; n++)
    {
      if (n>9) throw OutOfRange; //إثارة الإستثناء الأول
      {IntArray[n]=n*n;
        if (sum>200) throw OverFlow;// إقارة الإستثناء
الثاني
        sum=sum+IntArray[n];
      }
    }
  }
  catch (string e) //معالجة الإستثناء الأول
```

```
  {
    cout << "Exception: " << e << endl;
  }
  catch (int e)//معالجة الإستثناء الثاني
  {
    cout << "Exception:  error index is " << e
<< endl;
  }
catch (...)//معالجة الإستثناءات غير المعروفة
    {
        cout << "An exception of unknown type
occurred" << endl;
    }

  cout <<"Note that the program does not
terminate in this case"<<endl;
  system("pause");
  return 0;
}
```

2. فصائل الاستثناءات و استخداماتها Exception classes

في الأمثلة السابقة استخدمنا أنواعاً عادية من الفصائل لتعريف الاستثناءات. في **التطبيقات العملية الواقعية يفضل استخدام فصائل (classes) كاملة لتعريف الاستثناءات. هذا يساعد كثيراً في معالجة الإستثناءات و إعطاء حلول متكاملة في حالة حدوثها.** المثال التالي يوضح كيف يمكن تصميم و إستخدام فصائل كاملة للاستثناءات.

مثال(4)

في هذا المثال **نصمم فصيلة استثناء اسمها DivideByZero تستخدم** في حالة **محاولة القسمة على صفر. هذه الفصيلة تستخدم** داخل البرنامج **في إثارة** استثناء عند محاولة القسمة على صفر **و تعطي** معلومات مفصلة عن حالة البرنامج لحظة حدوث الإستثناء (لا تكتفي بمجرد رسالة خطأ).

برنامج(4)

```
#include <iostream>
#include <string>
using namespace std;

class DivideByZero {
public:
    DivideByZero(int n, int d)
    {  num=n;  denom=d;    message="Divide  by
zero";}
    ~DivideByZero() {}

    int getNumerator() {return num;}
    int getDenominator() {return denom;}
    string getMessage() {return message;}

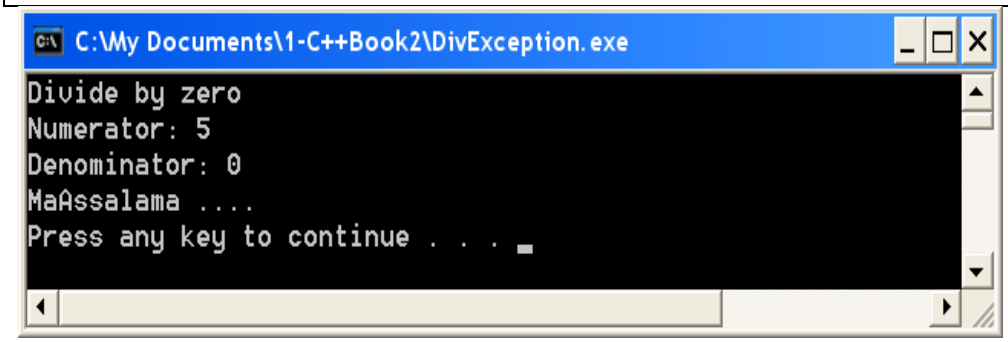
private:
    int num;
    int denom;
    string message;
};
int main()
{
    int x = 5;
    int y = 0;
    int result;
```

```
    try {
        if (y == 0) {
            throw DivideByZero(x, y);
        }

        result = x/y;
    }
    catch (DivideByZero e) {
        cout << e.getMessage() << endl;
        cout     <<     "Numerator:     "     <<
e.getNumerator() << endl;
        cout    <<    "Denominator:    "    <<
e.getDenominator() << endl;
    }

    cout << "MaAssalama ...." << endl;
    system("pause");
    return 0;

}
```



البرنامج أعلاه استخدم الفصيلة DivideByZero لمعالجة الاستثناء المتوقع. الشكل العام لهذه الفصيلة هو:

```
class DivideByZero {
public:
    DivideByZero(int n, int d)
    {  num=n;  denom=d;    message="Divide  by
zero";}
    ~DivideByZero() {}

    int getNumerator() {return num;}
    int getDenominator() {return denom;}
    string getMessage() {return message;}

private:
    int num;
    int denom;
    string message;
};
```

المفيد في هذه الفصيلة أنها **تحتوي على معلومات متكاملة عن حالة البرنامج لحظة حدوث الإستثناء.** المعلومات في هذه الحالة تتمثل في قيمة البسط و المقام. **هذا يسهل على المبرمج معالجة الخطأ.**

لاحظ استخدام الفصيلة DivideByZero في جملة throw:

throw DivideByZero(x,y);

جملة throw في هذه الحالة **تثير** استثناءً من النوع DivideByZero. **جملة catch تستقبل** هذا الإستثناء و **تستخلص** المعلومات اللازمة من الكيان الممرر.

مثال(5)

في هذا المثال **نصمم فصيلة Stack تستخدم فصيلتي استثناء: StackOverflow و StackEmpty.** كما سبق أن أوضحنا في الوحدة السابعة أن المرصة (stack) عبارة عن نوع خاص من القوائم يمكن التعامل معه من جهة واحدة فقط هي أعلى القائمة.

الإضافة و الحذف تتم من أعلى القائمة فقط. عملية الإضافة في هذا المثال تقوم بها الدالة ()push وعملية الحذف تقوم بها الدالة ()pop.

الدالة ()push تستخدم فصيلة **StackOverflow لإثارة** استثناء عند محاولة **إضافة** عنصر إلى مرصة ممتلئة، بينما **تستخدم الدالة ()push** الفصيلة **StackEmpty لإثارة** استثناء عند محاولة **حذف** عنصر من مرصة خالية.

```
#include <iostream>
#include <string>
using namespace std;
class StackOverflow {
public:
    StackOverflow() {}
    ~StackOverflow() {}
    void getMessage()
    {
     cout << "Exception: Stack Full" << endl;
    }
};
class StackEmpty {
public:
    StackEmpty() {}
    ~StackEmpty() {}
    void getMessage()
    {
        cout << "Exception: Stack Empty" <<
endl;
    }
};
// A stack class
class Stack {
public:
    Stack() : index(-1) {}
    ~Stack() {}
```

```
    void push(int item)
    {
        index++;

        if (index >= 20)
        {
            throw StackOverflow();
        }

        data[index] = item;
    }
    int pop()
    {
        if (index < 0)
        {
            throw StackEmpty();
        }
        int item = data[index];
        index--;
        return item;
    }
private:
    int data[20];
    int index;
};

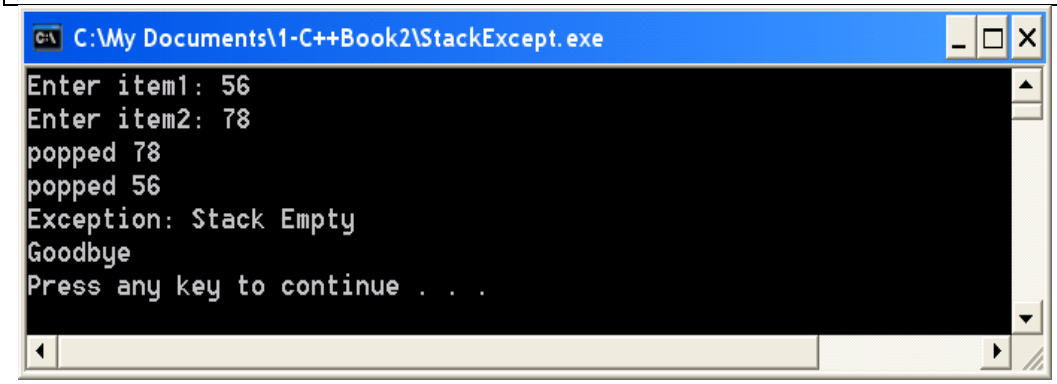
int main()
{
    Stack stk;
    int item1,item2, val;
    cout << "Enter item1: ";
    cin >> item1 ;
    cout << "Enter item2: ";
    cin >> item2;
    try {
```

```
        stk.push(item1);
        stk.push(item2);

        val = stk.pop();
        cout << "popped " << val << endl;
        val = stk.pop();
        cout << "popped " << val << endl;
        val = stk.pop();
        cout << "popped " << val << endl;
    }
    catch (StackOverflow e) {
        e.getMessage();
    }
    catch (StackEmpty e) {
        e.getMessage();
    }
    catch (...)
    {
        cout << "An exception of unknown type
occurred" << endl;
    }

    cout << "Goodbye" << endl;
    system("pause");
    return 0;

}
```



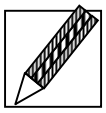
لاحظ أننا استخدمنا **ثلاثة جمل catch** في البرنامج أعلاه:

واحدة : لمعالجة الخطأ الناجم من محاولة الإضافة إلى مرصة مملوءة.

الثانية: لمعالجة الخطأ الناجم من محاولة الحذف من مرصة فارغة.

الثالثة: عامة للإخبار بحدوث خطأ غير معروف السبب.

تدريب (2)



1- اكتب برنامجاً يحسب متوسط الدرجات التي حصل عليها الطلاب في مادة معينة. استخدم أوامر الاستثناء في معالجة الحالات التي تكون فيها الدرجات المعطاة أكثر من 100 أو أقل من صفر.

2- أعد كتابة البرنامج أعلاه باستخدام فصيلة استثناء اسمها GradeAverage. الفصيلة GradeAverage يمكن استخدامها في إثارة استثناء في حالة أن الدرجة أكبر من 100 أو أقل من صفر مع الإستمرار في حساب المتوسط. في النهاية تعطي الفصيلة معلومات عن عدد الدرجات التي استخدمت في حساب المتوسط، مع قائمة بالدرجات التي لم تستخدم في حساب المتوسط (الدرجات التي تكون أكبر من مائة أو أقل من صفر).

3- اكتب فصيلة SafeArray تمنع التخزين خارج نطاق المصفوفة. إستخدم أوامر الاستثناء في تصميم هذه الفصيلة.

4- بعض دوال مكتبة C++القياسية تثير استثناءات معرفة سلفاً في فصائل استثناء جاهزة. هذه الفصائل مبنية على فصيلة استثناء قياسية: std::exception. حاول أن تكتشف الهيكل العام لفصائل استثناء الدوال الجاهزة و كيف يمكن استخدامها في البرامج العادية.

الخلاصة

- الأخطاء المنطقية التي تحدث بعد البدء في تشغيل البرنامج تسبب مشاكل كبيرة للمطورين وتطوّل زمن التطوير، وبالتالي تزيد تكلفة تطوير البرامج.
- يمكن تقليل مخاطر الأخطاء المنطقية باستخدام أوامر الإستثناء في لغات البرمجة.
- الأخطاء المنطقية تسمى "الإستثناءات" (exceptions) في عرف مهندسي البرمجيات.
- سي++ توفر ثلاثة أوامر لمعالجة الإستثناءات هي: try catch throw.
- الأمثلة التي قدمناها تشرح كيف يمكن استخدام هذه الأوامر.
- يمكن استخدام فصائل متكاملة لمعالجة الإستثناءات. الفائدة التي نجنيها في هذه الحالة هي إعطاء المطور معلومات متكاملة عن حالة البرنامج لحظة حدوث الإستثناء، بالإضافة إلى طرق لمعالجة الإستثناء.

المصادر والمراجع

1- Deictl, H. & Deitel, P., **C++ How to Program.** Third Edition, Prentice Hall: 2001.
2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
4- Lippman, S. lajoie, J. **C++ Primer,** (3rd Edition). Addison Wesley: 1998.
5- Horstmann, C. **Computing Concepts With C++ Essentials.** John Wiley: 1997.
6- Stroustrup, B. **The C++ Programming Language.** (3 rd edition), Addison Wesley: 1990.
7- Ellis, M. and Bjarnc Stroupstrup, **The Annotated Reference Manual.** Addison Wesley: 1990.
8- Tcale, S. C++ IOS treams Handbook. Addison Wesley: 1993.
9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs.** Addison Wesley. 1996.
11- Schildt, H**. C++ From the Ground up.** Mc Graw Hill. 2003.
12- Savitch, W. **Absolute C++.** Addison Wesley: 2002.
13- Cogswell, J. **C++ all – in – one Desk Refcrence for Dummies.** 2002.