
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

تمهيد

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



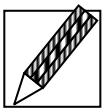
عرف ما يلي:

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)

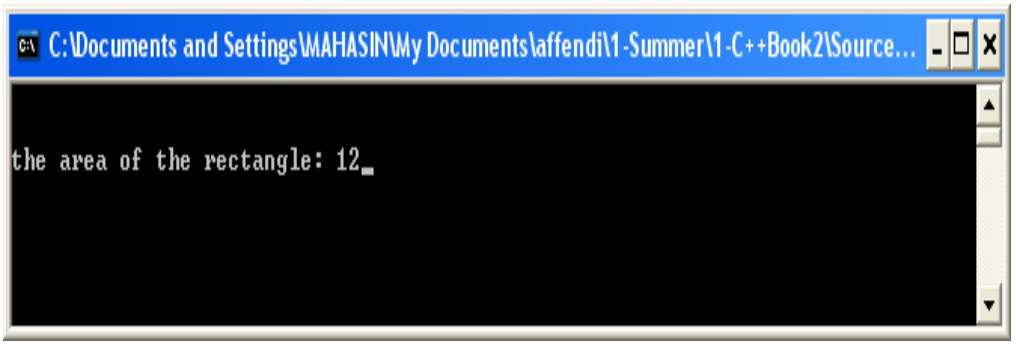
المثال أدناه عبارة عن برنامج صغير **يستخدم الفصيلة Rectangle** في **تعريف كيان MyRect**، ثم **يحسب و يطبع مساحة الكيان:**

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 مباشرةً 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:

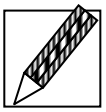
RectArea=MyRect.area();

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:

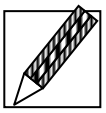
دالة الإنشاء	الخاصية
الاسم	اسم الفصيلة
النوع	دالة الإنشاء **ليس لها نوع** و يمكن أن **توصف بأي صفة**، حتى **كلمة void لا يمكن استخدامها في وصف دوال الإنشاء.**
الاستخدام	يمكن **استخدام أكثر من دالة إنشاء للفصيلة الواحدة، تشترك** كلها في الإسم **(اسم الفصيلة)**، بشرط أن **تختلف** عن بعضها البعض في **عدد 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;
}
```

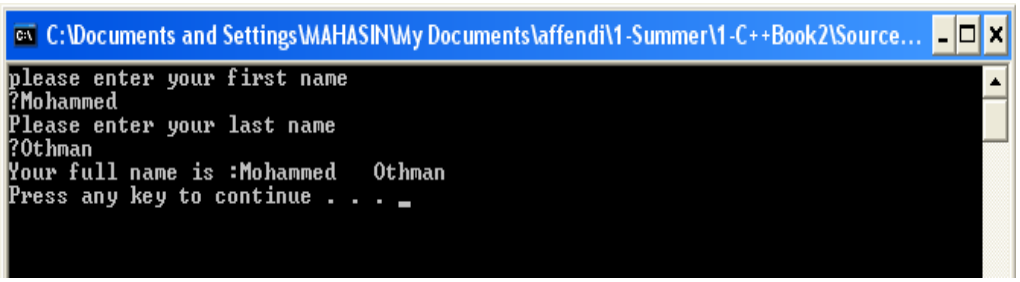
ملاحظات على البرنامج:

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، وهذه 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)

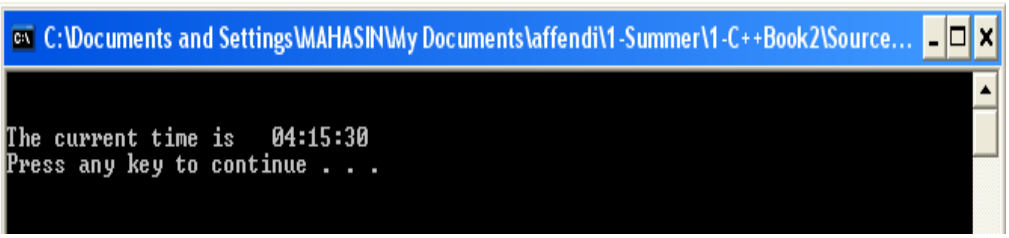
في هذا المثال نصمم **فصيلة Timer يمكن استخدامها في توقيت العمليات أو تسجيل الزمن لبعض الأحداث. الفصيلة غنية بالدوال** التي يمكن استخدامها في **تعديل الزمن أو قراءته.**

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.

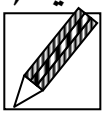
2- **أضفنا وصف const في تعريف الدالة GetTime()**، وهذا يعنى أن **الدالة GetTime() لايسمح** لها **بتعديل** أيٍ من **خواص الفصيلة Timer**. **الدوال الثابتة (const) يسمح لها** فقط بقراءة خواص الفصيلة و **لا يسمح لها** بتعديلها.

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 بـتعديل 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;
```

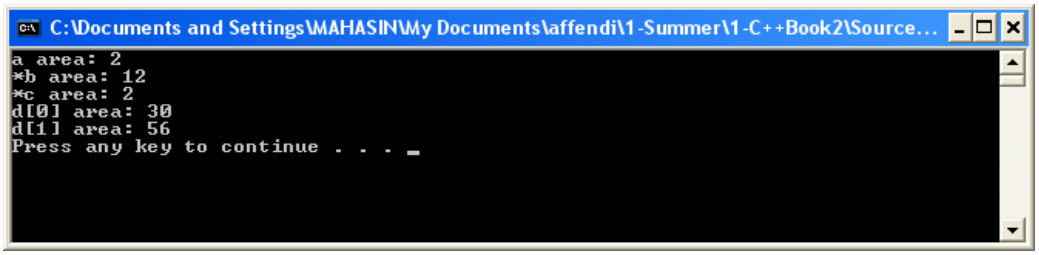
4- بعد ذلك **أسندنا عنوان الكيان a إلى المؤشر c الذي عرفناه في صدر البرنامج: c=&a;**. تذكر أن العلامة **& هي عامل العنوان**. بموجب هذه العملية أصبح **المؤشر c الآن يشير إلى موقع الكيان a في الذاكرة.**

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 أيصاً **إنشاء مصفوفات عادية من الكيانات مثلما** نفعل في حالة **المتغيرات العادية.**

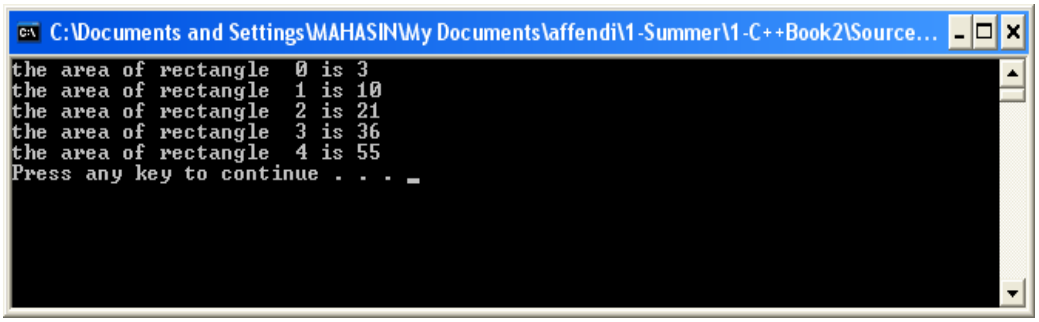
البرنامج المبسط التالي **يشرح** ذلك بصورة واضحة. **ينشيء البرنامج مصفوفة من المستطيلات** ثم **يستخدم جملة تكرار لحساب مساحة كلٍ منها:**



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 تنبني 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 نوع Rectangle دون أن نحدد كم طوله أو عرضه لحظة التعريف، ونريد أن نرجيء ذلك إلى مرحلة متقدمة في البرنامج. من 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 تمرير عناصر، فيصبح لدينا دالتا إنشاء، بدلاً من واحدة. تعدد دوال الإنشاء مسموح به في C++بحيث 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.