



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) التي تم تصميمها من قبل بسهولة كبيرة. عملية إعادة الاستخدام لا تتطلب أي معرفة بتفاصيل تحقيق تصميم (implementation) الكيان، إنما تتطلب فقط معرفة الواجهة التي يصدرها 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، ما عليه 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++ وشببيهاتها 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(المنتمة)، 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 الأحادي (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 يعرّف 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`.



```
#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:

```

C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...
I am the BaseClass
I am the BaseClass
I am the Derived Class
Please input two float numbers
?23.4 56.7
Please give one float number
3.2
the sum is 80.1 the scaled sum is 256.32
Testing D inherited members ...
?Please input two float numbers
?12.9 45.7
sum computed by D is 58.6

```

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 جزءاً 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 الابن، 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 (الابن) 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 الابن، و لعل الدارس الحصيف قد يكون لاحظ ذلك في شاشة تنفيذ البرنامج أعلاه. لاحظ أن جملة " I am the BaseClass" و التي تنطبقها الدالة الافتراضية للفصيلة الأم ظهرت مرتين: مرة عند تعريف الكيان B و مرة 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 فلا يمكن استخدامها مباشرةً في الفصيلة الوارثة.
- 4- مستويات الحماية للعناصر الموروثة في الفصيلة الابن يمكن تلخيصها في الجدول التالي:

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

من الجدول أعلاه نخلص للقاعدة العامة التالية:

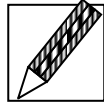
أن العناصر العامة (public) في الفصيلة الأم يمكن الوصول إليها و استخدامها في الفصيلة نفسها، أو من أي فصيلة أخرى. العناصر المحمية (protected) يمكن الوصول إليها، و استخدامها من أعضاء الفصيلة الأم و أعضاء الفصائل الوارثة. أما العناصر الخاصة (private) فلا يمكن الوصول إليها إلا من أعضاء الفصيلة الأم نفسها.

أسئلة تقويم ذاتي



- 1- كيف تتم عملية الإرث في C++
- 2- ما الأشياء التي لا ترثها 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- ما هو 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 الأسطوانة) 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 الابن. إذا أعدنا تعريف دالة الإنشاء الافتراضية دون استخدام عناصر ممررة، فإن ذلك لا يؤثر على عملية الاستدعاء المذكورة أعلاه: أي أن دالة الإنشاء الافتراضية 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 لا يمكن استدعاؤها عند تعريف كيانات واردة. 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 الافتراضية، و بالتالي عدم استدعائها عند تعريف الكيان `D`.

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

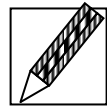
```
class DerivedClass: public BaseClass
{ private:
    float r;
public:
    DerivedClass():BaseClass(float a, float b)
```

```
{ 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)

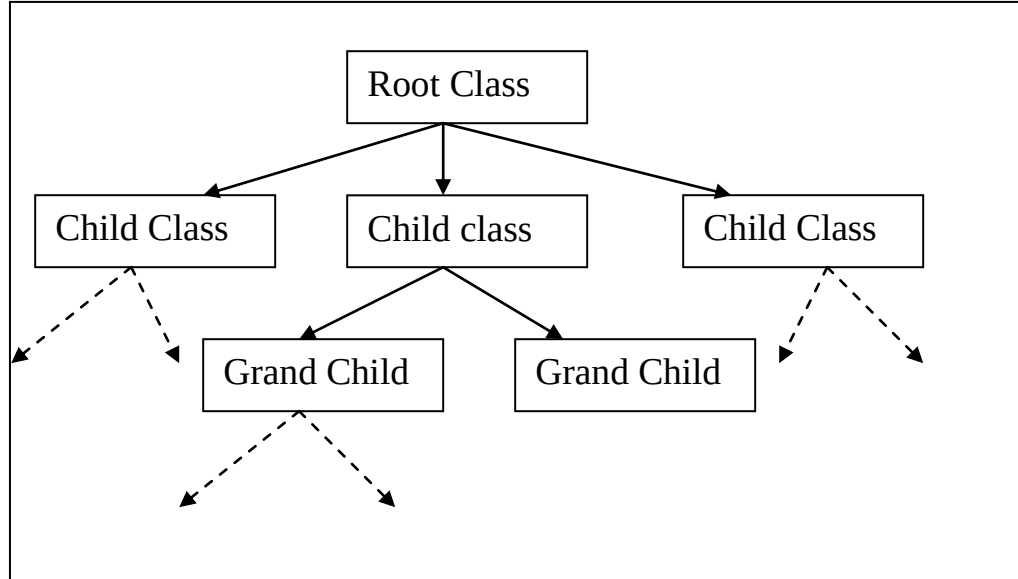
تدريب (2)



1. حسب قواعد الهندسة التحليلية، اي نقطة هندسية تمثل بإحداثيين: الإحداثي السيني و الإحداثي 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 واثرة، 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:

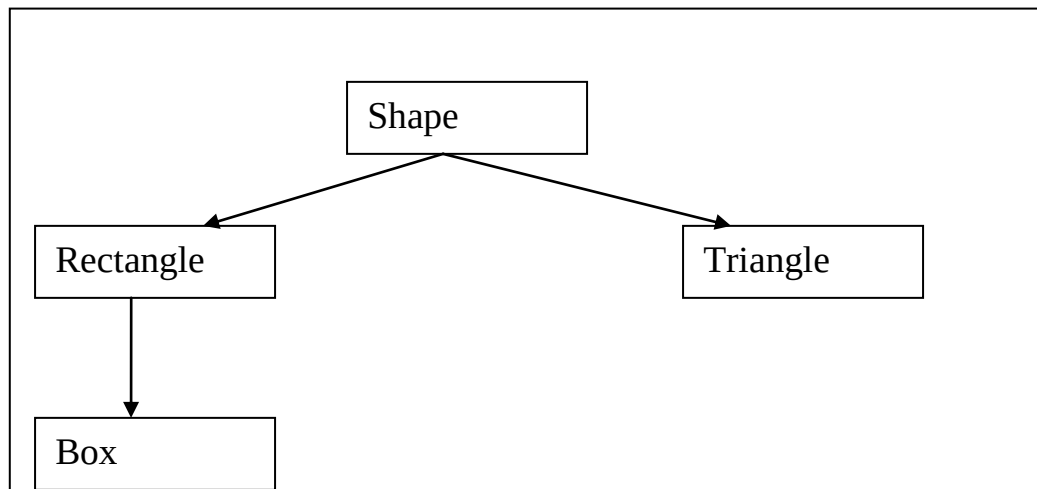


في المثال التالي نستخدم شجرة إرث تحتوي على فرعين عمق أحدهما 2 و عمق الآخر (1) مثال (2)

الفصيلة الأم في هذا المثال هي **Shape** والتي 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 تقليد 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) {}
};
```

```

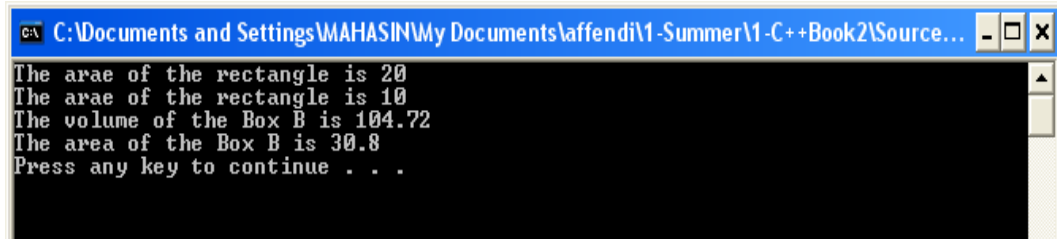
class Rectangle: public Shape {
public:
    float area (void)
        { return (width * height); }
};
class Triangle: public Shape {
public:
    float area (void)
        { return (width * height / 2); }
};
class Box: public Rectangle
{ private:
    float BoxHeight;
public:
    Box(float a) {BoxHeight = 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 (الجدة) **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:



```
C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...
The area of the rectangle is 20
The area of the rectangle is 10
The volume of the Box B is 104.72
The area of the Box B is 30.8
Press any key to continue . . .
```

حسب قواعد الهندسة التحليلية، أي نقطة هندسية تمثل بإحداثيين: الإحداثي السيني و الإحداثي 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** مباشرة 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:

```

C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...
please enter employee first name
?Saeed
Please enter employee last name
?Muhyiddin
Employee full name is :Saeed  Muhyiddin
Please enter rate per hour and hours worked
?175  123
The Employee Salary is 21525
Press any key to continue . . .

```

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 نعرّف 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<<"Title 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;
}

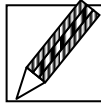
```

عينة من تنفيذ البرنامج:

```
C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++\Book2\Source...
Book details are :
Title: :C# programmin
Author: Mohammed Ahmed
Publisher: Dar-El-Ilm
ISBN NO: 125637388
Area of the Book: Programming

Journal Details are :
Title: :applied informatics
ISBN NO: 2009145416
Volume: 17 Issue: 3
Press any key to continue . . . _
```

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 آخر اسمها section ترث الفصيلة course وتضيف إليها التفاصيل الخاصة بالشعب. 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 ينبغي أن يدرّسها 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. عدّل 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) لا يمكن الوصول إليها إلا من داخل الفصيلة، و العناصر المحمية (protected) لا يمكن 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 كيانياً (كعنصر ممرر) وترجع كيانياً

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.

```
C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...
the area of rectangle 2 is 6
the area of rectangle 2 is 24
Press any key to continue . . . _
```

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 area of the converted
square is:  "
        <<rect.area()<<endl<<endl;
    system("pause");
    return 0;
}

```

```
C:\Documents and Settings\MAHASIN\My Documents\laffendi\1-Summer\1-C++\Book2\Source...
the area of the converted square is: 16
Press any key to continue . . .
```

ملاحظات على البرنامج:

- 1- يبدأ البرنامج بتعريف عنوان الفسيطة Square دون إعطاء مواصفاتها بالكامل، وذلك لتمكين الفسيطة Rectangle من استخدام اسم الفسيطة Square في تعريف عناصرها. كان بالإمكان إعطاء التعريف الكامل للفسيطة 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.

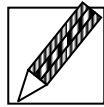
3- الفسيطة Square تعلن أن الفسيطة Rectangle فسيطة صديقة حتى يتسنى للدالة convert الوصول إلى العناصر الخاصة في Square و القيام بمهمتها.

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 الاسم، 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();
}

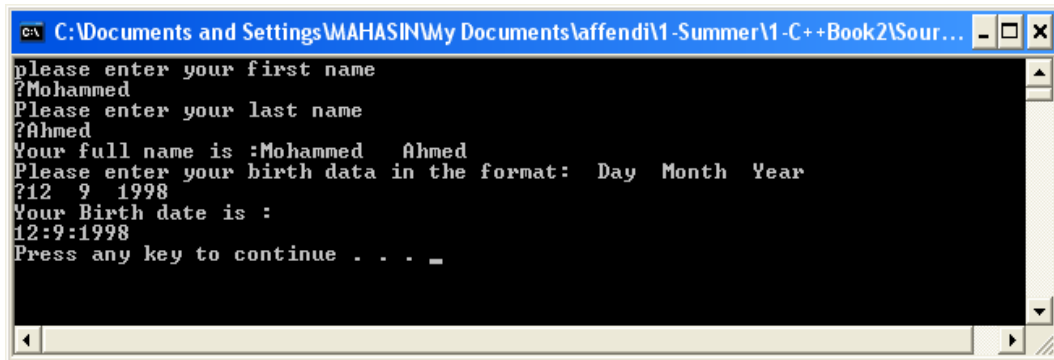
```

```

system("pause");
return 0;
}

```

عينة من تنفيذ البرنامج:



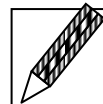
```

C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Sour...
please enter your first name
?Mohammed
Please enter your last name
?Ahmed
Your full name is :Mohammed  Ahmed
Please enter your birth data in the format: Day Month Year
?12 9 1998
Your Birth date is :
12:9:1998
Press any key to continue . . . _

```

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)

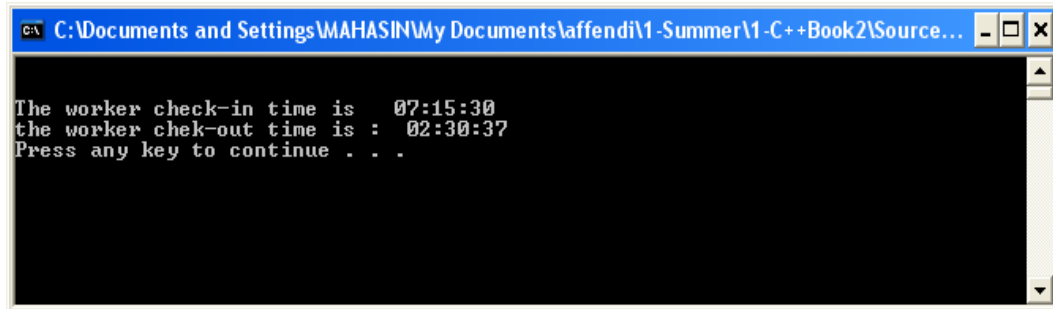
```

```

    { 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:



```

C:\Documents and Settings\MAHASIN\My Documents\laffendi\1-Summer\1-C++Book2\Source...
The worker check-in time is 07:15:30
the worker chek-out time is : 02:30:37
Press any key to continue . . .

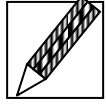
```

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 فصول أخرى بهدف امتلاك ما بداخل هذه الفصول من الخواص، والعمليات، دون الحاجة إلى إعادة تعريف أو حتى معرفة كيفية عمل هذه الفصول 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 الفصول الورثة. أما العناصر الخاصة فلا يمكن الوصول إليها إلا من أعضاء الفصيلة الأم نفسها، يمكن تجاوز هذا باستخدام فكرة الدوال، و الفصول الصديقة (friend functions and classes)، حيث تسمح ++C لأي فصيلة بتعريف دوال و فصول صديقة. هذه الفصول 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 CircularArea(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-
#include<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();
}

```

تدريـب (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();
}

```

تدريـب (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 الأحادي Single Inheritance

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

➡ الإرث المتعدد Multiple Inheritance

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

➡ شجرة الأثر 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. Iajoie, 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 Bjarne Stroustrup, **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 Refrence for Dummies**. 2002.



محتويات الوحدة

الموضوع	الصفحة
المقدمة	125
تمهيد	125
أهداف الوحدة	125
1. تحميل الدوال	127
1.1 تحميل الدوال وإعادة 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 الممررة. 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++ تسهياً 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 استخدام 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 بصورتها: إختلاف عدد العناصر و إختلاف الأنواع. الدالة `divide()` تستخدم مرةً لقسمة الأعداد الصحيحة، ومرةً 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)
```

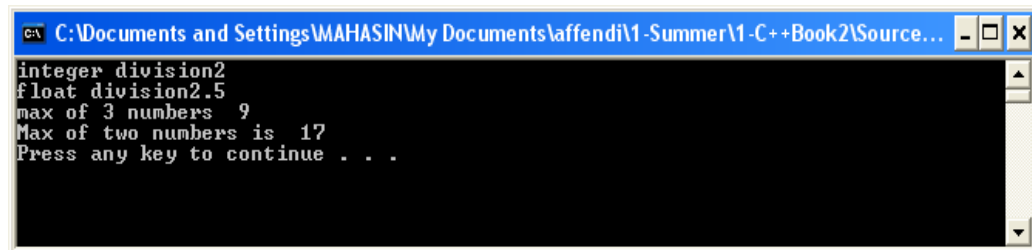
```

{ 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 واثرة 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 تعرّف 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 أيضاً استخدام عامل الجمع و الضرب داخل البرنامج:

```

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

```

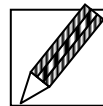
عامل المقارنة (التساوي) استخدمناه في الجملة الشرطية:

```

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;
}

```

```

C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...
The resulting complex number is 4,3
Press any key to continue . . .

```

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 لا بد من نسبة العامل إلى الفصيلة الأم إذا عرفنا العامل خارج الفصيلة:

```
تعريف العامل المحمل // Complex Complex::operator+ (Complex param)
{
    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.

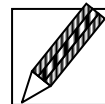
أسئلة تقويم ذاتي



- 1- ماهي الصورة العامة لتحميل العوامل الثنائية في C++
- 2- حمّل عامل الإسناد "=" بحيث يمكن استخدامه في إسناد مصفوفة Vector إلى أخرى.
- 3- أضف إلى الفصيلة complex أعلاه عامل الضرب المحمل "*" بحيث يمكن استخدامه مباشرة 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 كثير 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 حمّل 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);

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 أي من الدوال الأعضاء من أجل ذلك توفير احتياجات الدوال التي 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 تنتظر 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 نعرّف فصيلة `Test` تحتوي على دالة واحدة `IsItMe()`. هذه الدالة تستقبل عنصراً واحداً من النوع `Test` وتنظر إذا ما كان هذا العنصر هو الكيان المالك أم لا. إذا كان العنصر الممرر هو الكيان المالك، ترجع الدالة القيمة 1، وإلا فترجع القيمة صفر.

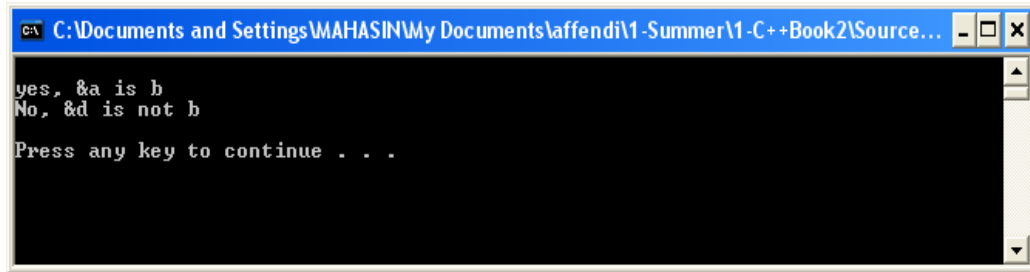
برنامج (4) 

```
// 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**.



```
C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...  
yes, &a is b  
No, &d is not b  
Press any key to continue . . .
```

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 الفصيطة التي يتم التحميل فيها. لاحظ أننا لا نحتاج إلى عنصر بين القوسين (بعد **op-sign**) لأن العملية أحادية ولا تحتاج إلى طرف آخر. الدالة ترجع مؤشراً للفصيطة المالكة باستخدام المؤشر **this**.

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

مثال (5)

في هذا المثال قمنا بتحميل العامل الأحادي ++ في الفسيطة **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)

الفصلة Test2 في هذا المثال تحتوي على متغير دائم n. دالة الإنشاء في الفصلة ترفع قيمة n بواحد. دالة الإخلاء تخفض قيمة n بواحد. هذا يعني 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).

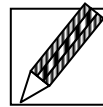
بعد ذلك قمنا بإزالة الكيان الديناميكي **c**. يتوقع في هذه الحالة أن تنقص قيمة **n** إلى ستة، ذلك أن دالة الإخلاء التي تستدعى في هذه الحالة تنقص قيمة **n** بواحد. شاشة تنفيذ البرنامج تدل على أن قيمة **n** أصبحت فعلاً ستة.

ملحوظة:

ربما كانت هذه المرة الأولى التي نستغل فيها دالة الإخلاء استغلالاً مباشراً. في كل الأمثلة السابقة كنا نعتمد على النسخة الافتراضية من هذه الدالة. لاحظ أننا استخدمنا اسم الفصيلة و علامة النطاق لطباعة قيمة **n** في نهاية البرنامج أعلاه: **Test2::n** وهذا مسموح به.

```
C:\Documents and Settings\MAHASIN\My Documents\affendi\1-Summer\1-C++Book2\Source...
The value of n now is 7
The value of n after deleting c is 6
Press any key to continue . . . _
```

تدريب (3)



حَمِّل عامل التزويد الأحادي "++" بحيث يمكن استخدامه في تزويد التاريخ التي استخدمناها في الوحدة الماضية. Date. بيوم 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

تدريب 1

```
#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 AdditionalDays)
{
    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

- 14- Deictl, H. & Deitel, P., **C++ How to Program**. Third Edition, Prentice Hall: 2001.
- 15- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
- 16- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
- 17- Lippman, S. Iajoie, J. **C++ Primer**, (3rd Edition). Addison Wesley: 1998.
- 18- Horstmann, C. **Computing Concepts With C++ Essentials**. John Wiley: 1997.
- 19- Stroustrup, B. **The C++ Programming Language**. (3 rd edition), Addison Wesley: 1990.
- 20- Ellis, M. and Bjarne Stroustrup, **The Annotated Reference Manual**. Addison Wesley: 1990.
- 21- Tcale, S. **C++ IOS treams Handbook**. Addison Wesley: 1993.
- 22- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
- 23- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs**. Addison Wesley. 1996.
- 24- Schildt, H. **C++ From the Ground up**. Mc Graw Hill. 2003.
- 25- Savitch, W. **Absolute C++**. Addison Wesley: 2002.
- 26- Cogswell, J. **C++ all – in – one Desk Reference for Dummies**. 2002.