الباب الثالث

استخدام الفصائل

Classes

(٣-١) الفصائل Classes

إعلان الفصيلة (Class Declaration)

يتم إعلان الفصيلة باستخدام أحد الكلمات المفتاحية الآتية:

- class (الفصيلة)
- struct (المنشأ)
- union (المنشأ المشترك)

وعادة نضع إعلان الفصيلة فى ملف العناوين (Header File) كتنظيم متعارف عليه ، ولكنه ليس بملزم.

أنظر المثال الآتى لإعلان فصيلة النقطة:

```
// Class declaration: ---------- إعلان الفصيلة
class point {
//Data members: ---------- البيانات الأعضاء (الخصائص)
private:
      int x,y
// Member functions: ---------- الدوال الأعضاء (السلوك)
public:
      void set_position(int p1, int p2)
      int access_x(void) {return x;}
      int access_y(void) {return y;}
};
```

أعضاء الفصيلة (Member Functions and Data Members)

كما نرى فى البرنامج السابق ان تعريف الفصيلة يحتوى على:

- إعلان البيانات الأعضاء التى تمثل الخصائص (مثل إحداثى النقطة x, y)

- إعلان الدوال الأعضاء التى تمثل السلوك (مثل تحديد الإحداثى بالدالة set_position())

استخدام الدوال الخطية

يمكنك تعريف بعض الدوال الصغيرة كدوال خطية بداخل إعلان الفصيلة (مثل دوال التوصل إلى إحداثيات النقطة access_x() و access_y() فى هذا المثال). ونلاحظ هنا أننا لم نستخدم الكلمة inline ، فهى اختيارية إذا كانت تعريف الدالة الخطية يقع بداخل إعلان الفصيلة.

تطبيق الفصيلة (Class Implementation)

باستثناء الدوال الخطية ، فإن دوال الفصيلة يتم الإعلان عنها بداخل الفصيلة (ويسمى هذا الإعلان عينة الدالة Prototype) ثم يتم تعريفها فى ملف البرنامج وتسمى هذه التعريفات فى مجموعها بتطبيق الفصيلة (Class Implementation). وهذا هو المثال لتطبيق الدالة set_position().

```
void point::set_position(int p1, int p2)
{
   x = p1;
   y = p2;
}
```

ونلاحظ فى تطبيق الدالة ضرورة تحديد تبعية هذه الدالة إلى الفصيلة point وذلك باستخدام مؤثر النطاق (::). وتنتفى الحاجة إلى هذا المؤثر إذا تم الإعلان بداخل الفصيلة نفسها.

بعد إعلان الفصيلة وتطبيقها يمكنك أن تخلق الأهداف بالأسلوب التالى:

```
point MyPoint, YourPoint;
```

إن هذا الإعلان يؤدى إلى خلق مثالين من فصيلة النقطة بالأسماء MyPoint و YourPoint. وفى المثال التالى نعرض كيفية شحن هذه الأهداف بالقيم العددية (الإحداثيات) وطباعتها باستخدام دوال التوصل access_x و access_y. ونلاحظ عند استخدام الدوال الأعضاء أو البيانات الأعضاء أنها لابد أن تظهر مرتبطة باسم الهدف الذى تتبعه وذلك باستخدام مؤثر النقطة (.) مثل MyPoint.access_x().

```
#include <iostream.h>
...
...
void main(void)
{
// Declare objects: ---------------------------  الإعلانات
   point MyPoint, YourPoint;
// Initialize: ----------------------- شحن الأهداف بالبيانات
  MyPoint.set_position(3,4);
  YourPoint.set_position(5,9);
// Print: ---------------------------------- الطباعة
  cout << "\nThe position of my point is: "
       << MyPoint.access_x() << ", "
       << MyPoint.access_y();
  cout << "\nThe position of your point is: "
       << YourPoint.access_x() << ", "
       << YourPoint.access_y();
}
```

درجات الحماية للفصائل

هناك ثلاث درجات لحماية أعضاء الفصيلة:

درجة الحماية	معناها
public	متاحة للجميع (أى للدوال غير الأعضاء).
protected	يمكن توريثها (أى تصبح أعضاؤها متاحة للفصيلة المشتقة).
private	خاصة ولا يمكن التوصل إلى أعضائها إلاّ عن طريق الدوال الأعضاء ، كما لا يمكن توريثها.

وعند إنشاء الفصائل باستخدام الأنماط class و struct أو union فإن هناك درجة حماية سابقة التعريف مرتبطة بكل نوع كالآتى:

درجة الحماية	النوع
private	class
public	struct
public	union

أما الدرجة private فهى تختص بعمليات الوراثة كما سيلى.

(٣-٢) دوال البناء Constructors

يمكنك شحن الهدف بالبيانات عند الإعلان عنه باستخدام دوال البناء ، التى تعتبر من الدوال الأعضاء بالفصيلة ؛ ويجوز تعريفها بداخل أو خارج الفصيلة. وتحمل دالة البناء نفس اسم الفصيلة.

وتوجد ثلاثة أنواع من دوال البناء نعرضها فيما يلـــى بالاســتعانة بفصيلة النقطة (point) كمثال.

نوع دالة البناء	طريقة إعلان الهدف

١. دوال البناء "المعتادة":
مثال:
point(int p1, int p2)
{
x=p1; y=p2;
}
	مثال:
	point MyPoint(1,1);
	يؤدى هذا الإعلان إلى خلق هـــدف عن الإحداثى (1,1).

٢. دوال البنـــاء ذات البـــارامترات سابقة التعريـــف (Constructors with Default Parameters):
مثال:
point(int p1=0, int p2=0)
{
x=p1; y=p2;
}
	مثال:
	point MyPoint;
	يؤدى هذا الإعلان إلى خلق هـــدف عند الإحداثى (0,0). ومـــن الجـــائز ركوب البارامترات سابقة التعريـــف وتحديد الإحداثيات صراحة باستخدام إعلان مثل point MyPoint(4,5);

٣. دوال البنـــاء ســـابقة التعريـــف (Default Constructors):
مثال:
point(void) {
x=0; y=0;
}
ولا يجوز استخدام أيـــة بـــارامترات عند استخدام مثل هذه الدوال فى بناء الأهـــداف ، حيـــث تســـتخدم القيـــم الموجودة فى تعريف الدالـــة (0,0) فى هذا المثال).
	مثال:
	point MyPoint;
	يؤدى هذا الإعلان إلى خلق هـــدف عند الإحداثى (0,0).

ملاحظات على دوال البناء

- دالة البناء ليس لها نمط (مثل .. ,int, void).
- إذا تم الإعلان عن دالة البناء خارج جسم الفصيلة ، فيلزم استخدام مؤثر النطاق :: لتحديد تبعية الدالة للفصيلة كالمثال التالى:

```
point::point(int p1, int2) { ....
```

- لا يجوز الإعلان عن هدف بدون شحنه بقيم ابتدائية (بدون بنائه) طالما أن الفصيلة تحتوى على دالة بناء.
- من الجائز الجمع بين أكثر من نوع من دوال البناء للفصيلة الواحدة ، فيتم بناء بعض الأهداف بطريقة ما وبناء البعض الآخر بطريقة أخرى ، مع ملاحظة أن النوع الثانى والثالث لا يجتمعان.

دوال البناء للمنشئات المشتركة (Union Constructors)

يتم شحن المنشئات المشتركة (unions) بدوال بناء خاصة تستخدم خاصية التحميل الزائد (Overloading). وفى المثال التالى نشحن المنشأ المسترك ID الذى يحتوى على اسم الموظف ورقمه باستخدام دالتين للبناء ، واحدة للاسم والأخرى للرقم:

```
union ID {
        int number;
        char *name;
        ID(int n) {number=n;}  ------------------ الرقم
        ID(char *alpha) {name= alpha;} -------- الاسم
};
```

ويتم الإعلان عن هدف الموظف وشحنه بداخل الدالـــة الرئيســية للبرنامج بالأسلوب التالى:

```
ID emp1(435); ------------------ إعلان وشحن الرقم
ID emp2("Craig Combel"); ------- إعلان وشحن الاسم
```

وفى شريحة البرنامج التالية نطبع محتويات هذا الهدف بعد الإعلان عنه:

```
#include <iostream.h>

...

...
void main(void)
{
        ID emp1(435);
        ID emp2("Craig Combel");
        cout << endl << emp1.number;
        cout << endl << emp2.name;
        cout << endl;
}
```

مثال (٣-١): تطبيق على دوال البناء

```
// ******************** Example 3-1 ********************
// Constructors
#include <iostream.h>
class point {
private:
        int x, y;
public:
        int access_x(void) {return(x);}
        int access_y(void) {return(y);}
        void setpoint(int p1, int p2);

// Constructor:
        point(int p1, int p2) {x=p1;y=p2;}

// Default constructor:
        point(void) {x=1;y=2;}
};
```

```
// Class implementation
void point::setpoint(int p1, int p2)
{
      x=p1;
      y=p2;
}

// Main function:
void main(void)
{
// Declare object:
        point p1(5,3);
// Declare an object using def. constructor:
        point p2;
// Print results:
        cout <<endl<<"x1="<<p1.access_x()
        <<", "<<"y1="<<p1.access_y();
        cout <<endl<<"x2="<<p2.access_x()
        << ", "<<"y2="<<p2.access_y()<<endl;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-1.cpp

تنفيذ البرنامج:

عند تنفيذ هذا البرنامج فإنه يطبع الآتى على الشاشة:

```
x1=5, y1=3
x2=1, y2=2
Press any key to continue
```

دوال البناء الناسخة Copy Constructors

يمكنك نسخ الأهداف المبنية كالمثال الآتى ، حيث نبنى الهدف MyPoint ثم ننسخه إلى هدف جديد YourPoint فيتم خلقه بنفس القيم الابتدائية:

```
point MyPoint(5,5);
point YourPoint=MyPoint;
```

وفى حالة احتواء الفصيلة على مراجع أو مؤشرات فإنه يلزم استخدام دالة بناء ناسخة محتوية على مرجع إلى الفصيلة ، مثل:

```
point::point(point &MyClass)
{
...
...
}
```

ومن الجائز أن تحتوى دالة البناء الناسخة على أية عمليات خاصة بالنسخ وفى هذه الحالة فقد يتطلب الأمر تمرير بارامترات أخرى إلى الدالة ، ولكن البارامتر الأول لابد وأن يكون مرجعاً إلى الفصيلة. وعند عدم استخدام دالة بناء ناسخة ، فإن المترجم يستدعى دالة بناء ناسخة سابقة التعريف. وبطبيعة الحال فإن هذه الدالة تصلح فقط فى نسخ الأعضاء المباشر كما فى مثال فصيلة النقطة.

ومن المآزق التى قد يقع فيها المبرمج ــ عند نسخ فصيلة محتوية على مؤشر ــ أن تصبح كل من الفصيلة الأصلية والنسخة محتوية على مؤشر يشير إلى نفس الخانة من الذاكرة. فإذا مسحنا المؤشر الأول (بالأمر delete) فإن المؤشر الثانى يمسح تلقائياً. ولذلك يلزم الحذر مع هذه النوعية من الفصائل ، وذلك بتخصيص حيز مستقل من الذاكرة (بالأمر new) لكل مؤشر بكل فصيلة (وهذا يعنى عدم الاكتفاء بدالة النسخ سابقة التعريف التى يمدك بها المترجم).

وفى المثال التالى قد استخدمنا الإحداثى y كمؤشر بدلاً من المتغير الصريح (لمجرد ضرب المثال) وقد أجرينا عملية النسخ باستخدام دالة بناء ناسخة. وهذه هى أحداث البرنامج:

- إعلان الفصيلة والدالة الناسخة.
- إجراء عملية النسخ التى تتضمن حجز خانة ذاكرة للمؤسر الجديد.
- تغيير قيم أعضاء الفصيلة الجديدة للتأكد من أنها لا تؤثر على الفصيلة الأصلية ، ثم طباعة أعضاء الفصيلتين.
- مسح المؤشر بالفصيلة الأولى ، ثم طباعة الفصيلتين مرة أخرى.
- نلاحظ أن القيمة التى يشير إليها المؤشر الممسوح قيمة عشوائية لا معنى لها (عدد كبير جداً أو سالب).

مثال (٣-٢): تطبيق على دوال البناء الناسخة

```
// ****************** Example 3-2 ***********************
// Copy constructors
// Using a new memory location for the copy
#include <iostream.h>
class point {
public:
        int x;
        int *y;
public:
        point(point &);              //عينة دالة البناء الناسخة
        point(void) {x=0; y=new(int);} //Memory Allocation
        void setpoint(int p1, int p2);
};

point::point(point &NewClass)  // تعريف دالة البناء الناسخة
{
        y = new(int);
        *y = *(NewClass.y);
        x = NewClass.x;
}

void point::setpoint(int p1, int p2)
```

```
{
        x = p1;
        *y = p2;
}

void main()
{
        point p;
        p.setpoint(10,15);
// Copy p to p1:
        point p1=p;                                    // عملية النسخ

// Change value of p1 members:        تغيير قيم الفصيلة النسخة
        p1.x = p1.x+5;
        *(p1.y) = *(p1.y)+5;
// Display p and p1 members:              طباعة أعضاء الفصيلتين
        cout << "\nx="<<p.x <<", y="<< *(p.y);
        cout << "\nx="<<p1.x <<", y="<< *(p1.y) << endl;
// Now deallocate p.y memory and see what happens:
        delete p.y;                        // مسح المؤشر الفصيلة الأولى

        cout << "After deleting p.y:";     // الطباعة بعد المسح
        cout << "\nx="<<p.x <<", y="<< *(p.y);
        cout << "\nx="<<p1.x <<", y="<< *(p1.y) << endl;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-2.cpp

تنفيذ البرنامج:

عند تنفيذ هذا البرنامج فإنه يطبع الآتى على الشاشة:

```
x=10, y=15    ---------------  القيم قبل مسح المؤشر بالفصيلة الأولى
x=15, y=20

After deleting p.y:
                                القيم بعد مسح المؤشر بالفصيلة الأولى:
x=10, y=-572662307  --------  القيمة المشار إليها بالمؤشر الممسوح
x=15, y=20
Press any key to continue
```

(٣-٣) دوال الهدم Destructors

تستخدم دالة الهدم أساساً فى إتاحة الذاكرة التى كانت تشغلها الأهداف علاوة على أية وظائف أخرى قد يراها المبرمج. وتأخذ دالة الهدم نفس اسم الفصيلة كما هو الحال مع دالة البناء مع إضافة العلامة ~ كبادئة ، فدالة هدم الفصيلة point تأخذ الاسم ~point وتتمتع دوال الهدم بالخصائص التالية:

- دالة الهدم ليس لها نمط (مثل .. ,int, void)
- لا تأخذ دالة الهدم أية بارامترات.
- لا يجوز استخدام أكثر من دالة هدم واحدة للفصيلة.
- لا تحتاج الدالة إلى استدعاء فهى تستدعى تلقائياً متى وجدت فى الفصيلة.

وعلى سبيل المثال ، فلو انك قد حجزت بعض خلايا الذاكرة للاسم name بعبارة مثل:

```
name = new char[40];
```

فإنه من المناسب استعادة هذا الجزء من الذاكرة باستخدام دالة هدم كالمثال التالى الذى يوضح دالة هدم فصيلة بالاسم Employee:

```
Employee::~ Employee(void)
{
  delete [] name;
}
```

(٣-٤) شحن مصفوفات الأهداف

فى حالة شحن مصفوفة من الأهداف فإننا نستخدم طريقة الشـــحن بالقائمة. وفى المثال التالى نشحن مصفوفة تتكون من أربعة عناصر كل منها يمثل هدفا من فصيلة النقطة point.

```
// Object Declaration:
Point MyPointArrray[4] = {{10,10},{5,15},{4,2},{6,19};
```

وفى حالة وجود دالة بناء أو أكثر بالفصيلة ، فإنه يمكن اســتخدامها كالمثال التالى. ولنفرض أن لدينا دالتين للبناء كالآتى:

```
// point constructors:
        point(int p1, int p2) { x=p1; y=p2; } ----- دالة ذات بارامترات
        point(void) { x=1; y=1; } ----------------- دالة سابقة التعريف
```

من الجائز شحن المصفوفة بمزيج من هذه الدوال كالآتى:

```
// Object Declaration:
        point My_PointArray[4] = {point(10,10), point(15,20)};
```

فى هذا المثال قد تم شحن العنصرين الأول والثانى مـــن عنـــاصر المصفوفة بالنقط (10,10) و (15,20) أما بقية العناصر فقـــد تولـــت الدالة سابقة التعريف شحنها بالقيم (1,1). ومن البديهى أن العنـــاصر التى تشحن بالقيم سابقة التعريف تأتى فى المؤخرة.

مثال (٣-٣): تطبيق على المصفوفات

```
// ******************** Example 3-3 ********************
// Object Aarrays
#include <iostream.h>
// Class declaration
class point {
// Data members:
private:
    int x, y;
// Member functions:
```

```
public:
    point(int p1, int p2) { x=p1; y=p2; }
    point(void) { x=1; y=1; }
    void set_position(int p1, int p2);
    int access_x(void) { return (x); }
    int access_y(void) { return (y); }
};
// Class implementation
void point::set_position(int p1, int p2)
{
  x = p1;
  y = p2;
}
int main()
{
// Object Declaration:
  point My_Point[4]={point(10,10),point(15,20)};
// Display results:
  for(int i=0; i<=3; i++) {
    cout << "\nThe position of the point #" << i << " is: "
         << My_Point[i].access_x() << ", "
         << My_Point[i].access_y();
    }
  return 0;
}
// ****************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-3.cpp

تنفيذ البرنامج:

عند تنفيذ هذا البرنامج يطبع الآتى على الشاشة:

```
The position of the point #0 is: 10, 10
The position of the point #1 is: 15, 20
The position of the point #2 is: 1, 1
The position of the point #3 is: 1, 1
Press any key to continue
```

(٣-٥) استخدام المراجع والمؤشرات إلى الفصائل

يمكنك التوصل إلى أحد أعضاء الفصيلة بمؤشر مثل:

```
Class_Pointer->Class_Member
```

كما يجوز استخدام المراجع فى التوصل إلى الأعضاء كالمثال:

```
Class_Reference.Class_Member
```

وسوف تحتاج إلى هذه الإمكانية عند استخدام أحد أعضاء الفصيلة بداخل دالة غير عضوة ، حيث يتم فى هذه الحالة تمرير مؤشر أو مرجع إلى الفصيلة كبارامتر لهذه الدالة. فعلى سبيل المثال يمكنك إعلان مؤشر إلى فصيلة النقطة كالآتى:

```
point *Ptr;
```

كما يجوز تخصيص هدف سبق إعلانه إلى المؤشر كالآتى:

```
point *Ptr = MyPoint;
```

ثم يتم تمرير هذا المؤشر إلى دالة غير عضوة (مثل الدالة Display) كبارامتر ، كالآتى:

```
Display(Ptr);
```

بهذا تصبح أعضاء الهدف MyPoint متاحة للدالة Display من خلال المؤشر المُمرّر إليها. فيمكنك مثلاً طباعة إحداثيات الهدف MyPoint من داخل الدالة كالآتى:

```
void Display(point *P)
{
    cout << "\nThe position of the point is: "
         << P -> access_x() << "," --------- استخدام المؤشر
         << P -> access_y();
}
```

ولا يفوتنا أن صيغة عينة الدالة تسمح لها باستخدام الفصيلة كبارامتر ، أى:

```
void Display(point *);
```

ولو أردت استخدام المراجع ، فإن عينة الدالة تصبح:

```
void Dispaly(point &);
```

وفى هذه الحالة يتم تخصيص الهدف إلى مرجع إلى الفصيلة عند الإعلان عنه كالآتى:

```
point &Ref=MyPoint;
```

ثم تستدعى الدالة بتمرير المرجع:

```
Display(Rref);
```

أما بداخل الدالة نفسها فيستخدم مؤثر النقطة للتوصل إلى الأعضاء كالآتى:

```
void Display(point &R)
{
    cout << "\nThe position of the point is: "
         << R.access_x() << ","  ---------  استخدام المرجع
         << R.ccess_y();
}
```

أما الدوال الأعضاء التى تتوصل إلى البيانات مباشرة ، فهى فى الحقيقة تستخدم المؤشر الضمنى this إلى البيان ، أى أن عبارة مثل:

```
x=p1;
```

هى مجرد اختصار للعبارة:

```
this->x=p1;
```

(٦-٣) شحن العناصر بطريقة القائمة Initialization List

استخدمنا من قبل دالة بناء بالصورة التالية:

```
point(int p1, int p2)
{
  x=p1;
  y=p2;
}
```

من الجائز أيضاً كتابة نفس الدالة بالصورة الآتية:

```
point(int p1, int p2) : x(p1), y(p2)

{
}
```

إن هاتين الدالتين متكافئتان. فالتعبير x(p1) يكافئ التعبير x=p1 ، والتعبير y(p2) يكافئ التعبير y=p2. ولكن طريقة التنفيذ تختلف. ففى الدالة الأولى يتم تنفيذ جسم الدالة ثم تخصيص البارامترات الممررة إلى القيم x, y. أما فى الدالة الثانية فيتم تجهيز قيم المتغيرات قبل الدخول فى جسم الدالة. إن هذه الطريقة فى شحن المتغيرات ، وهى تسمى الشحن بطريقة القائمة ، تلعب دوراً أساسياً فى توريث الفصائل.

(٣-٧) الوراثة Inheritance

حتى الآن فإننا قد تعاملنا مع فصيلة واحدة ، هى فصيلة النقطة. فإذا أردنا إنشاء فصيلة أخرى مثل الدائرة أو المستطيل ، فإنه من المناسب أن نستخدم الوراثة حتى نستفيد من الجهد المبذول فى إنشاء فصيلة النقطة. وذلك لأن الدائرة ترث من النقطة خاصية المركز. كذلك فإن المستطيل يحتوى على أربعة أركان كل منها عبارة عن نقطة ذات إحداثى معين. وتسمى الفصيلة الوارثة بالفصيلة المشتقة (Derived Class) وتسمى الفصيلة الموروثة بفصيلة الأساس (Base Class). ومن البديهى أن الفصيلة المشتقة ترث كل ما عند فصيلة الأساس وتضيف إليه أعضاء جديدة ، فالدائرة تحتوى على النقطة (المركز) وتحتوى أيضاً على عضو جديد هو نصف القطر. وهذا هو الفلسفة الأساسية فى عملية الوراثة. لأن المبرمج لا يحتاج إلى ابتكار فصائل جديدة (إلاّ نادراً) وكل ما عليه أن يرث ما هو موجود من الفصائل ثم يضيف إليها أعضاء جديدة.

إعلان الفصيلة المشتقة

لتوريث فصيلة النقطة (point) إلى فصيلة جديدة (ولتكن فصيلة الدائرة) بالاسم circle نتبع الآتى:

```
class circle : public point { ---------- إعلان الوراثة
private:
    float radius;
public:
   void set_r(float r) {radius=r;}
// ---------------------------------------- دالة بناء الدائرة
   circle(int x1,int x2, float r): point(x1,x2) {set_r(r);}
};
```

نلاحظ فى الإعلان السابق مايلى:

- تم توريث فصيلة النقطة بالإعلان:

```
class circle : public point
```

أما كلمة public فهى تسمى معدّل التوصل (Access Modifier) للفصيلة المشتقة وسوف نناقشه فى الفقرة التالية.

- نلاحظ أيضاً استخدام طريقة الشحن بالقائمة فى دالة بناء الدائرة. حيث يتطلب الأمر تحديد إحداثيات المركز الموروث من فصيلة النقطة ، وذلك باستدعاء دالة بناء النقطة ، قبل تنفيذ دالة بناء الدائرة.

درجات ومعدّلات التوصّل Access Levels/Modifiers

يوضح الجدول التالى التوليفات المختلفة التى يمكن استخدامها كدرجات للتوصل أو معدلات التوصل لكل من فصائل الأساس والفصائل المشتقة.

	درجة التوصل بفصيلة الأساس	معدل التوصل بالفصيلة المشتقة	درجة توصل الفصيلة المشتقة إلى أعضاء فصيلة الأساس
١	public	public	public
٢	private	public	لا يمكن التوصل إليها
٣	protected	public	protected
٤	public	private	private
٥	private	private	لا يمكن التوصل إليها
٦	protected	private	private

جدول درجات ومعدلات التوصل في عمليات الوراثة

ولفهم هذا الجدول دعنا نستعرض بعض حالات الوراثة المحتملة من فصيلة النقطة إلى فصيلة الدائرة. ولتحقيق ذلك فلنستخدم المثال التالى لفصيلة الدائرة الذى يحتوى على دالة طباعة للمركز ونصف القطر والمحيط:

مثال (٣-٤): تطبيق على الوراثة: فصيلة الدائرة

```
// ***************** Example 3-4.cpp ********************
// Inheritance
//
        #include <iostream.h>
        double const PI=3.14159;
//
        class point {
        private:
          int x, y;
        public:
// point constructor:
          point(int p1=0, int p2=0) { x=p1; y=p2; }
          void set_position(int p1, int p2) { x=p1; y=p2; }
          int access_x(void) { return (x); }
          int access_y(void) { return (y); }
        };
//
```

```
        class circle : public point {
        private:
          float radius;
        public:
          void set_radius(float r) { radius=r; };
          double access_r(void) { return radius; };
          double area(void) { return PI*radius*radius; };
          double perimeter(void) { return 2*PI*radius; };
// circle constructor:
          circle(int x1, int x2, float r)
          : point(x1, x2) { set_radius(r); };
          void print_circle(void);
        };
//
        void circle::print_circle(void)
        {
          cout << "\nCircle center is: "
               << access_x() << "," << access_y();
          cout << "\nCircle radius is: "
               << access_r();
          cout << "\nCircle perimeter is: "
               << perimeter();
          cout << "\nCircle area is: "
               << area();
        }
//
        main()
        {
          circle My_Circle(10,15,3.0);
          My_Circle.print_circle();
          cout << endl;
          return 0;
        }
// ***************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-4.cpp

تنفيذ البرنامج:

عند تنفيذ هذا البرنامج تحصل على النتيجة الآتية:

```
Circle center is: 10,15
Circle radius is: 3
Circle perimeter is: 18.8495
Circle area is: 28.2743
Press any key to continue
```

- إذا كانت درجة التوصل إلى أعضاء فصيلة النقطة public وكان معدل التوصل لفصيلة الدائرة public (التوليفة ١ بالجدول) فإن أعضاء الفصيلة تصبح عامة ويجوز التوصل مباشرة كالمثال:

```
cout << x <<", "<<y;
```

وهذا لا يوصى به بالطبع لأنه يفسد مبدأ حماية البيانات.

- أما مع التوليفة رقم ٣ ، فإن فصيلة الدائرة يمكنها التوصل إلى الأعضاء x, y بفصيلة النقطة. بمعنى أنك تستطيع أن تتعامل مع هذه الإحداثيات مباشرة من داخل أى دالة عضوة بفصيلة الدائرة ، كالمثال التالى:

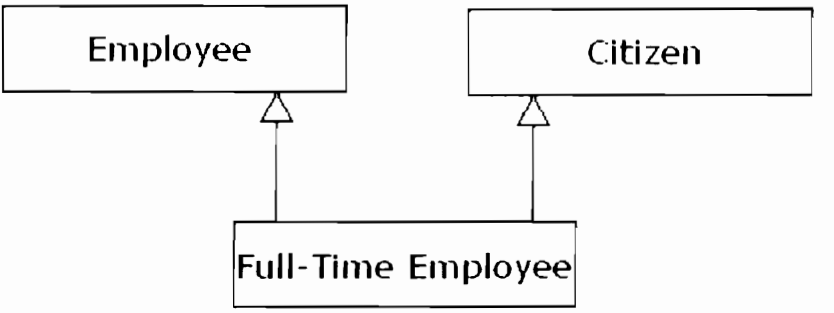
```
void circle::Display(void)
{
        cout << x <<", "<<y;
        ...
```

- أما التوليفتان رقم ٤ ورقم ٦ فهما متشابهتان ، حيث يمكن معهما التوصل إلى الفصيلة المشتقة مباشرة ، ولكن الفصيلة المشتقة تصبح خاصة ولا يمكن توريثها مرة أخرى.
- أما التوليفتان رقم ٢ و رقم ٥ فإن استخدام أى منهما يؤدى إلى استحالة التوصل إلى فصيلة الأساس (إلاّ من خلال دوال التوصل).

استخدم البرنامج السابق لتغيير معدلات التوصل ودرجات التوصل ثم شاهد النتائج ، وبطبيعة الحال سوف تحتاج إلى إضافة بعض العبارات مثل "cout << x" لتجربة النتائج.

(٣-٨) الوراثة المتعددة Multiple Inheritance

يجوز للفصيلة أن ترث من أكثر من فصيلة أساس. وفي الحياة العملية نرى أمثلة للوراثة المتعددة ، فالموظف قد يعمل بمرتب شهرى فى شركة ما (ولنطلق عليه الموظف المتفرغ Full Time Employee) وقد يعمل بالساعة (ولنطلق عليه الموظف المؤقت Temporary Employee) وكلاهما يرث من فصيلتين: فصيلة المواطن (Citizen) وفصيلة الموظف عموماً (Employee). أنظر الشكل التالى:



شكل (٣-١) الوراثة المتعددة

ولنفترض أن الفصائل تحتوى على البيانات الأعضاء الآتية:

- فصيلة المواطن: Citizen

```
class Citizen {
protected:
        char SSN[20]; --- ----رقم التأمين الاجتماعى
        char Nationality[20]; ---------- الجنسية
...
...
};
```

- فصيلة الموظف: Employee

```
class Employee {
```

```
protected:
        char ID[20];  ---------- الرقم الكودى
        char Name[20];  ---------- الاسم
...
...
};
```

أما فصيلة الموظف المتفرغ FTEmployee فسوف ترث من هـــاتين الفصيلتين كل ما بهما من أعضاء ، ويتم الإعلان عنها كالآتى:

```
class FTEmployee: public Citizen, public Employee {
...
...
};
```

وفى المثال التالى نطبق هذا المفهوم.

مثال (٣-٥) تطبيق على الوراثة المتعددة

```
//****************** Example 3-5 **********************
// Multiple Inheritance
#include <stdio.h>
#include <iostream.h>
#include <string.h>
//
// The base classes:
class Employee {                                    فصيلة الموظف //
protected:
        char ID[20];
        char Name[20];
public:                          دالة بناء ذات بارامترات سابقة التعريف //
        Employee (char *id="***-****", char *name="TBN")
        {
                strcpy(ID, id);
                strcpy(Name, name);
        }
};

class Citizen {                                     فصيلة المواطن //
protected:
        char SSN[20];
        char Nationality[20];
```

```
public:                                    // دالة بناء ذات بارامترات سابقة التعريف
        Citizen (char *ssn="000-00-0000", char *na="Egyptian")
        {
                strcpy(SSN, ssn);
                strcpy(Nationality, na);
        }
};
//                                         فصيلة الموظف المتفرغ
class FTEmployee: public Citizen, public Employee {
public:
//                                         دالة بناء سابقة التعريف
        FTEmployee(void) :Employee (), Citizen() {}
//                                         دالة بناء ذات بارامترات
        FTEmployee(char *id, char *name, char *ssn, char *na)
        :Employee (id, name), Citizen(ssn, na) {}
//                                         دوال التوصل إلى البيانات الأعضاء
        char *getID(void) {return ID;}
        char *getName(void) {return Name;}
        char *getSSN(void) {return SSN;}
        char *getNationality (void) {return Nationality;}

};

int main(void)
{
//                                         هدف موظف سابق التعريف
        FTEmployee FTEmployee1;
        cout << endl <<"First FTEmployee:"
                << endl <<FTEmployee1.getID()
                << endl <<FTEmployee1.getName()
                << endl <<FTEmployee1.getNationality ()
                << endl <<FTEmployee1.getSSN()
                << endl;
//                                         هدف موظف ذى بيانات متفردة
        FTEmployee FTEmployee2("MC9-9912","Michael
Cane","435-23-3333", "Irish");
        cout << endl <<"Second FTEmployee:"
                << endl <<FTEmployee2.getID()
                << endl <<FTEmployee2.getName()
                << endl <<FTEmployee2.getNationality ()
                << endl <<FTEmployee2.getSSN()
           << endl;
return 0;
}
```

```
//*************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-5.cpp

ويحتوى البرنامج السابق على الآتى:

- إعلان فصيلة الموظف Employee وبها دالة بناء ذات بارامترات سابقة التعريف تؤدى إلى تخصيص القيم الآتية:
  - Name="TBN" (اختصار العبارة "To Be Named" بمعنى لا يوجد اسم حالياً)
  - ID="***-****".

ويجوز ركوب هذه البارامترات كالمعتاد عند إعلان الهدف.

- إعلان فصيلة المواطن Citizen وبها وبها دالة بناء ذات بارامترات سابقة التعريف تؤدى إلى تخصيص القيم الآتية:
  - SSN= "000-00-0000"
  - Nationality="Egyptian"
- إعلان فصيلة الموظف المتفرغ FTEmployee وبها دالتا بناء ، واحدة سابقة التعريف تؤدى إلى الوراثة الكاملة للفصيلتين. والأخرى ذات بارامترات يمكن استخدامها لتحديد البيانات المميزة للموظف.
- تم إعلان هدفين للموظف المتفرغ: الأول يرث فصيلتى الموظف والمواطن ، والثانى له بياناته الخاصة التى تم إدخالها فى الدالة الرئيسية (main).

تنفيذ البرنامج:

عند تنفيذ البرنامج نحصل على الآتى:

```
First FTEmployee:
***-****
TBN
Egyptian
000-00-0000

Second FTEmployee:
MC9-9912
Michael Cane
Irish
435-23-3333

Press any key to continue
```

ملاحظة:

إن عملية هدم الفصائل المشتقة تتم بصورة عكسية لعملية البنـــاء ، فما يبنى أولاً يهدم أخيراً ، أى أن فصيلة الأساس دائماً تهدم قبـــل الفصائل المشتقة منها. وفى حالة الوراثة المتعددة ، فـــإن إعـــلان الوراثة هو الذى يحدد هذه الأولية. ففى المثال السابق:

```
class FTEmployee: public Citizen, public Employee {...
```

يتم بناء فصيلة المواطن (Citizen) يليـــها بنـــاء فصيلـــة الموظـــف (Employee) ولذلك فإن الأخيرة تهدم أولاً.

ملاحظة:

(٣-٩) تعدد الأطوار Polymorphism

إن تعدد الأطوار (أو تعدد الأشكال) يتحقق بأحد طريقتين:

- التحميل الزائد للدوال (Function Overloading) ــ وهو يستخدم خاصية الربط الاستاتيكى (Static Binding).
- الدوال الافتراضية (Virtual Functions) ــ وهو يستخدم خاصية الربط الديناميكى (Dynamic Binding).

استخدام التحميل الزائد للدوال Function Overloading

فلنفرض أن لدينا ثلاث فصائل بالأسماء A, B, C. من الجائز أن تتضمن جميع هذه الفصائل دالة بنفس الاسم مثل Func ، ولكن وظيفة كل دالة قد تختلف تماماً عن الأخرى. فى إمكانك أن تستدعى هذه الدوال كالآتى:

```
A.Func();
B.Func();
C.Func();
```

ولا يتطلب التحميل الزائد فى هذه الحالة أن تختلف بارامترات الدوال عن بعضها البعض لأن كل دالة ترتبط بهدف معين ولا سبيل إلى الوقوع فى الإبهام. والارتبط بين الدالة وبين الهدف ارتباط إستاتيكى يحدث أثناء الترجمة ولا يمكن تغييره. ويسمى هذا النوع من تعدد الأطوار بتعدد الأطوار الاستاتيكى أو تعدد الأطوار أثناء الترجمة.

استخدام الدوال الافتراضية Virtual Functions

إن الدالة الافتراضية هى الدالة التى تركبها دوال أخرى تحمل نفس الاسم ، فتغير من وظيفتها. ولإعلان دالة افتراضية فإننا نسبقها بالكلمة virtual عند إعلان عينة الدالة ، مثل:

```
virtual double Area(void) { return x*x;}
```

ومن الجائز أن تكون **الدالة الافتراضية نقية** (Pure Virtual Function) بمعنى أن الدالة لا تؤدى أية وظيفة سوى أن تمتطيها الدوال الأخرى. ونعلن عن مثل هذه الدالة كالآتى:

```
virtual double Area(void)=0;
```

ويطلق على الفصيلة المحتوية على الدالة الافتراضية النقية **بالفصيلة المجردة** (Abstract Class).

أما تعدد الأطوار الديناميكى فإنه يتحقق باستخدام الدوال الافتراضية مع المؤشرات كالمثال التالى.

فى هذا المثال سوف ننشئ الفصائل الآتية:

- فصيلة الأبعاد الفراغية DIMENSIONS (وهى فصيلة عامة تتضمن الأبعاد الثلاثة X, Y, Z) ، ومن هذه الفصيلة سوف تأتى بقية الفصائل التى تمثل الأشكال الهندسية المختلفة.
- فصيلة النقطة point
- فصيلة الدائرة circle
- فصيلة الكرة sphere
- فصيلة الاسطوانة cylinder

وسوف نعلن دالة افتراضية لحساب المساحة بالاسم Area() فى فصيلة الأبعاد ، وبالطبع سوف نعلن عن دالة (عادية) بنفس الاسم

بكل من الفصائل الأخرى. وباستخدام مؤشر واحد يشير إلى فصيلة الأساس (DIMENSIONS) ، سوف نستدعى الدالة Area() بكل فصيلة من الفصائل الأخرى لحساب مساحة الشكل (الدائرة أو الكرة.. ألخ). ويتميز استخدام المؤشر مع الدوال الافتراضية بأننا نستطيع أن نغير من اتجاهه بحيث يشير إلى أى هدف من الأهداف أثناء تنفيذ البرنامج (أى بدون ارتباط إستاتيكى بينه وبين أى هدف). وهذا هو البرنامج:

مثال (٣-٦) تطبيق على تعدد الأطوار

```
// ****************** Example 3-6 ********************
// Virtual Functions and Dynamic Linking
#include <iostream.h>
//
double const PI = 3.14159;
//
// A generic dimensions class                    فصيلة الأبعاد الثلاثة
class DIMENSIONS {
protected:
  double X, Y, Z;
public:
  DIMENSIONS(double D1=0, double D2=0, double D3=0);
  double ShowX(void) { return X; }
  double ShowY(void) { return Y; }
  double ShowZ(void) { return Z; }
  virtual double Area(void) { return 0; } //  دالة المساحة الافتراضية
};
//                                                فصيلة الدائرة
class Circle:public DIMENSIONS {
public:
  Circle(double R):DIMENSIONS(R) {}
  double Area(void) { return PI * X * X; }
};
//                                                فصيلة الكرة
class Sphere:public DIMENSIONS {
```

```
public:
   Sphere(double R):DIMENSIONS(R) {}
   double Area(void) { return 4 * PI * X * X; }
};
//                                                    فصيلة الاسطوانة
class Cylinder:public DIMENSIONS {
public:
   Cylinder(double R, double H):DIMENSIONS(R,H) {}
   double Area(void) { return 2*PI*X*X + 2*PI*X*Y; }
};
//                                                    فصيلة النقطة
class Point:public DIMENSIONS {
};
//                                                    دالة بناء فصيلة الأبعاد
DIMENSIONS::DIMENSIONS(double D1, double D2, double D3)
{
   X = D1;
   Y = D2;
   Z = D3;
}
//
int main()
{
   DIMENSIONS *ptr;  // base pointer          مؤشر إلى فصيلة الأساس
// Object declarations:                  إعلان أهداف الأشكال الهندسية المختلفة
   Point P;
   Cylinder CY(2.5,3.0);
   Circle C(3.0);
   Sphere S(4.0);
// Point to a point:                          تحويل المؤشر إلى هدف النقطة
   ptr = &P;
   cout << "\nArea of a point= " << ptr -> Area();
// Point to a circle:                         تحويل المؤشر إلى هدف الدائرة
   ptr = &C;
   cout << "\nArea of a circle with a radius "
        << ptr -> ShowX() << "= " << ptr -> Area();
// Point to a sphere:                         تحويل المؤشر إلى هدف الكرة
   ptr = &S;
   cout << "\nArea of a sphere with a radius "
        << ptr -> ShowX() << "= " << ptr -> Area();
// Point to a cylinder:                       تحويل المؤشر إلى هدف الاسطوانة
   ptr = &CY;
```

```
    cout << "\nArea of a cylinder with a radius "
         << ptr -> ShowX() << " and a height " << ptr -> ShowY()
         << "= " << ptr -> Area() << endl;
  return 0;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-6.cpp

لاحظ الآتى فى البرنامج السابق:

- لا تحتوى فصيلة النقطة على دالة المساحة ، وهذا يؤدى إلى استخدام الدالة الافتراضية Area() الموجودة بفصيلة الأساس. وهى ترجع القيمة 0 على أى حال (أى أنها القيمة المناسبة لمساحة النقطة).
- تم الإعلان عن المؤشر بحيث يشير مبدئياً إلى فصيلة الأساس بالصورة:

```
DIMENSIONS *ptr;
```

- يتم تحويل اتجاه المؤشر بين الأهداف بتخصيص عنوان الهدف الجديد إليه ، مثل:

```
ptr = &C;
```

حيث يدل التعبير &C على عنوان هدف الدائرة فى الذاكرة.

تنفيذ البرنامج

عند تنفيذ هذا البرنامج نحصل على النتيجة الآتية:

```
Area of a point= 0
Area of a circle with a radius 3= 28.2743
Area of a sphere with a radius 4= 201.062
Area of a cylinder with a radius 2.5 and a height 3=
86.3937
Press any key to continue
```

ملاحظة:

لدينا فى البرنامج السابق عدة فصائل ، بكل منها دالة بالاسم Area(). ماذا لو أنك أردت ــ من داخل فصيلة الاسطوانة ــ استدعاء دالة المساحة الموجودة بفصيلة الدائرة (لحساب مساحة سطح الاسطوانة مثلاً)؟ يمكنك تحقيق ذلك بالاستعانة بمؤثر النطاق كالآتى:

```
Circle::Area();
```

ملاحظة:

خصائص الدوال الافتراضية

تخضع الدوال الافتراضية للقيود الآتية:

١. لابد أن تتطابق الدوال فى نوع البارامترات ، والقيمة المرتجعة ، وإلاّ فإنها تصبح دالة محملة تحميلاً زائداً.

٢. لابد أن تكون الدالة الافتراضية عضوة فى الفصيلة (أو صديقة للفصيلة ــ سيلى شرح الصداقة).

٣. يجوز تحويل اتجاه مؤشر يشير إلى فصيلة الأساس لكى يشير إلى أى فصيلة مشتقة ، ولكن العكس غير جائز.

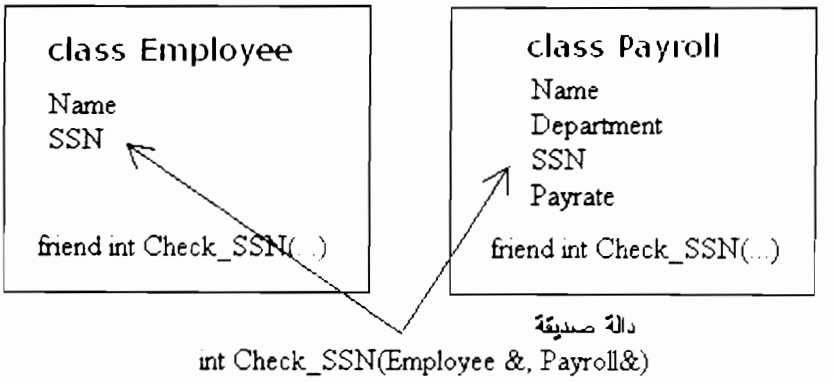
٤. يجوز أن تكون دوال البناء افتراضية ، ولكن لا يجوز أن تكون دوال الهدم افتراضية.

٥. إن الوظيفة الوحيدة للدالة الافتراضية النقية (Pure Virtual Functions) هى أن تركبها الدوال الأخرى ، والوظيفة الوحيدة للفصيلة المجردة (Abstract Class) هى أن تورث.

(٣-١٠) الأصدقاء Friends

الصداقة عبارة عن علاقة بين الفصائل وبعضها ، أو بين الـــدوال والفصائل. وهى تؤدى إلى تسهيلات فى معالجة البيانات ، حيـــث تتمكن الدالة الصديقة لفصيلة ما (أو دوال الفصيلة الصديقـــة) مـــن التوصل للأعضاء الخاصة لهذه الفصيلة.

أولاً: الصداقة بين دالة وفصيلة



شكل (٣-٢) الصداقة بين دالة وفصيلتين

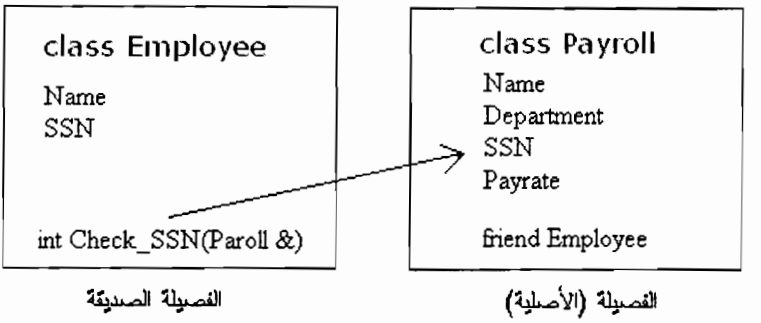
يوضح الشكل السابق علاقة صداقة بين الدالة Check_SSN() وبيـــن كل من الفصيلتين Employee و Payroll ، حيث تتمكن هذه الدالة مـــن التوصل إلى الأعضاء الخاصة مثل SSN بكل من الفصيلتين. وفـــى هذا المثال نرى إحدى فوائد الصداقة ، فالدالة الصديقـــة تســـتطيع التوصل إلى رقم التأمين الاجتماعى (SSN) لموظف معين بقـــاعدة بيانات الموظفين ، وتقارنه بنظيره فى قاعدة بيانات الأجور. إن هذه العملية أساسية للتحقق من هوية الموظـــف كجـــزء مـــن برنـــامج

الأجور. ويتم الإعلان عن هذه الدالة باستخدام كلمة friend بداخل كلٍ من الفصيلتين ، كالآتى:

```
friend int Check_SSN(Payroll &, dBase &);
```

أما تعريف الدالة نفسه فيأتى فى نطاق الملف ، أى خارج إعلانات الفصائل.

ثانياً: الصداقة بين فصيلتين



شكل (٣-٣) الصداقة بين فصيلتين

من الجائز ــ فى نفس المثال المطروح ــ تحقيق المهمة بإنشاء علاقة صداقة بين فصيلة الأجور (Payroll) وفصيلة الموظفين (Employee) كما بالشكل السابق. وفى هذه الحالة فإننا نجعل الدالة Check_SSN عضوة فى أحد الفصيلتين. وتسمى الفصيلة المحتوية على هذه الدالة الفصيلة الصديقة ، حيث أن جميع دوالها تتمكن من التوصل إلى أعضاء الفصيلة الأخرى (الأصلية). يتم إعلان الصداقة باستخدام الإعلان التالى بداخل الفصيلة الأصلية:

```
friend Employee;
```

فى المثال التالى نعرض تطبيقا كاملا للصداقة بين الدالة Check_SSN وبين كل من فصيلتى الأجور والموظفين.

مثال (٣-٧): تطبيق على الصداقة بين دالة وفصيلتين

```
// ******************** Example 3-7 ********************
// Friend Functions
#include <iostream.h>
#include <string.h>
// Forward declaration: ---------- إعلان مسبق عن فصيلة الموظفين
class Employee;
// Class #1: -------------------------- إعلان الفصيلة الأولى
class Payroll {
private:
   char Name[30];
   char Department[4];
   char SSN[12];
   float Payrate;
public:
   Payroll(char *n, char *d, char *ssn, float p);
   friend int Check_SSN(Payroll &, Employee &);
};
// Class #2: -------------------------- إعلان الفصيلة الثانية
class Employee {
private:
   char Name[30];
   char SSN[12];
public:
   Employee(char *n, char *ssn);
   friend int Check_SSN(Payroll &, Employee &);
};

// Class #1 constructor: ---------------------  بناء الفصيلة الأولى
Payroll::Payroll(char *n, char *d, char *ssn, float p)
{
   strcpy(Name,n);
   strcpy(Department,d);
   strcpy(SSN,ssn);
   Payrate=p;
}
```

```
// Class #2 constructor: ---------- بناء الفصيلة الثانية
Employee::Employee(char *n, char *ssn)
{
  strcpy(Name,n);
  strcpy(SSN,ssn);
}
//                                    تعريف الدالة الصديقة (غير العضوة):
// Definition of the non-member friend function:
int Check_SSN(Payroll &x, Employee &y)
{
  if( strcmp(x.SSN, y.SSN) == 0)
    return (1);
  else
    return (0);
}

// The main function: ---------- الدالة الرئيسية
int main()
{
  Payroll person1("Mohamed A. Al-Husseiny","EES","434-68-
6141",34.5);
  Employee person2("Mohamad A. Al Husseiny","434-68-6141");
  if(Check_SSN(person1, person2))
    cout << "\nThe same employee" << endl;
  else
    cout << "\nNot the same employee" << endl;
  return(0);
}
//*************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-7.cpp

تنفيذ البرنامج

عند تنفيذ هذا البرنامج نحصل على النتيجة الآتية:

```
The same employee  نفس الموظف
Press any key to continue
```

ونلاحظ أن نتيجة المقارنة كانت صحيحة نظرا لأن رقم التأمين الاجتماعى (SSN) واحد بكل من الفصيلتين بالرغم من اختلاف طريقة كتابة الاسم ، وهذا هو الهدف من البرنامج. نلاحظ أيضا

ضرورة الإعلان المسبق عن الفصيلة Employee لأن الدالة الصديقة تستخدم أسماء الفصائل كبارامترات لها.

(٣-١١) البيانات الاستاتيكية الأعضاء
Static Data Members

تستخدم الكلمة static فى لغة سى فى الأغراض الآتية:

- لجعل المتغير المحلى (Local Variable) يحتفظ بقيمته بين الاستدعاءات المختلفة للدالة التى يتبعها.
- لجعل نطاق المتغير العام (Global Variable) يقتصر على الملف الموجود به.

أما الاستخدام الثالث الذى قدمته سى++ فهو جعل العضو مشاعا بين جميع أهداف الفصيلة ، ويسمى فى هذه الحالة بالعضو الاستاتيكى أو العضو العام. وعلى سبيل المثال فلو أنك أعلنت هدفين من فصيلة النقطة ، فإن كلا منهما يصبح لديه نسخته الخاصة من الإحداثيات x و y. أما لو كان العضو x عضوا إستاتيكيا فإن جميع الأهداف المعلنة تشترك فيه. ويتم الإعلان عن العضو الاستاتيكى باستخدام بكلمة static داخل الفصيلة ، علاوة على إعلانه مرة أخرى بنطاق الملف بدون كلمة static ، كالمثال التالى:

```
class MyClass {
public:
        static int x; ---------- الإعلان الأول ، بداخل الفصيلة
        ...
};
```

```
int MyClass::x = 5;  ---------- الإعلان الثانى ، بنطاق الملف
```

ويجوز شحن العضو بقيمة ابتدائية فى نطاق الملف ، وإلا فإنه يأخذ القيمة "صفر". ويتم التوصل إلى العضو الاستاتيكى بإحدى طريقتين:

- بالطريقة المعتادة: بربطه بأى هدف (أو دالة عضوة) مثل:

```
obj1.x = 55;
obj1.Set_x(55) ;
```

- بربطه باسم الفصيلة باستخدام مؤثر النطاق (بشرط أن يكون عضوا عاما (public)) ، مثل:

```
MyClass::x = 55;
```

ويعتبر البيان الاستاتيكى العضو مناظرا للمتغيرات العامة (Global Variables).

(٣-١٢) الدوال الاستاتيكية الأعضاء
Static Member Functions

تستخدم هذه الدوال فى التوصل إلى الأعضاء الاستاتيكية وهى تتميز بعدم احتوائها على المؤشر this ، ولذلك فهى لا تصلح للتوصل إلى الأعضاء الأخرى بخلاف الأعضاء الاستاتيكية. ويعلن عنها كالمثال التالى:

```
class MyClass {
public:
static int x;
static void MyStaticFunction(void) { cout << x; }
...
};
```

وتستخدم الدوال الاستاتيكية الأعضاء بطريقتين:

- بالطريقة المعتادة ، أى بربطها بأحد الأهداف مثل:

```
obj1. MyStaticFunction();
```

- بربطها باسم الفصيلة باستخدام مؤثر النطاق ، مثل:

```
MyClass:: MyStaticFunction();
```

وفى المثال التالى نعرض تطبيق لكل من البيانات والدوال الاستاتيكية الأعضاء. وتضمن البرنامج نتائج امتحانات ثلاث مواد (الأهداف: A, B, C) تشترك جميعا فى رقم الامتحان الممثل بالعضو الاستاتيكى TestNumber. كما تتولى الدالة الاستاتيكية العضوة ShowTest() طباعة محتويات هذا العضو.

مثال (٣-٨): تطبيق على البيانات والدوال الاستاتيكية الأعضاء

```
// ****************** Example 3-8.cpp ********************
// Static data members and static member functions
#include <iostream.h>
//
class Test {
private:
  int Subject;
public:
  static unsigned TestNumber;          //        إعلان بيان استاتيكى عضو
  void SetSubject(unsigned j) { Subject = j; }
  void ShowSubject(void)
       { cout << "\nSubject Code: " << Subject; }
// Definition of a static data function:
static void ShowTest(void)                    //   إعلان دالة استاتيكية عضوة
       { cout << "\nTest #: " << TestNumber; }
};
// Definition of a static data member:
unsigned Test::TestNumber;  //        إعلان البيان الاستاتيكى بنطاق الملف
// ------------------------------------------------------ الدالة الرئيسية
int main()
{
  Test A, B , C;
  Test::TestNumber=1; // ------------------- استخدام البيان الاستاتيكى
```

```
// Initialize objects:
  A.SetSubject(343);
  B.SetSubject(877);
  C.SetSubject(922);
// Show objects:
  Test::ShowTest(); // ---------------------- استخدام الدالة الاستاتيكية
  A.ShowSubject();
  B.ShowSubject();
  C.ShowSubject();
  return 0;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-8.cpp

تنفيذ البرنامج

عند تنفيذ هذا البرنامج نحصل على النتيجة الآتية:

```
Test #: 1
Subject Code: 343
Subject Code: 877
Subject Code: 922
Press any key to continue
```

(٣-١٣) دوال التحويل Conversion Functions

تستخدم دوال التحويل فى تحويل الأهداف إلى أنماط أخرى من أنماط البيانات. وتأخذ الدالة الصورة الآتية:

```
operator Type(void)  { return Value; }
```

حيث Type هو النمط المطلوب التحويل إليه.

فمن الجائز مثلا تحويل هدف النقطة إلى عدد صحيح (int). أما قيمة هذا العدد فيحددها منطق برنامجك. ولنفرض أن قيمة العدد الصحيح المطلوب هى x+y أى مجموع الإحداثيين. فى هذه الحالة نكتب الدالة كالآتى:

```
operator int(void)  { return x+y; }
```

فإذا كانت أضفنا الدالة كعضوة فى فصيلة النقطة ، فيمكنك استخدام تعبيرات مثل:

```
MyPoint + YourPoint
MyPoint  + 10
YourPoint + 100
```

فإذا كانت إحداثيات النقطة MyPoint هى 3,4 ، وإحداثيات النقطة YourPoint هى 2,3 ، فإن:

- التعبير الأول يسفر عن القيمة 12 وهى عبارة عن مجموع الإحداثيات كلها (3+4+4+3).
- التعبير الثانى يسفر عن القيمة 17 أى 3+7+10.
- التعبير الثالث يسفر عن القيمة 105 أى 2+3+100.

(٣-١٤) التحميل الزائد للمؤثرات
Operator Overloading

يهدف التحميل الزائد للمؤثرات إلى تعريف وظائف جديدة لها عند استخدامها مع الأهداف. وباستخدام التحميل الزائد تستطيع أن تكتب تعبيرات وعبارات مثل:

```
Object1 > Object2
MyObject++;
YourObject += 3;
```

أما معنى هذه التعبيرات والعبارات فيعتمد على برنامجك. وتستخدم دوال المؤثرات (Operator Functions) فى تحميل المؤثرات وتعتمد صيغة الدالة على إذا ما كان المؤثر ثنائيا (Binary) مثل مؤثرات الجمع والطرح ، أو فرديا (Unary) مثل المؤثر ++ ، كما تعتمد

على كون الدالة عضوة فى الفصيلة ، أو مستقلة. وتأخذ الدالة الصورة العامة:

الدالة العضوة	Type Class::operator op(arg1, arg2,...)
الدالة غير العضوة	Type operator op(Class &, arg1, arg2, ...)

حيث:

- op: رمز المؤثر المراد تحميله.
- Class: اسم الفصيلة.
- arg1, 2, ..: بارامترات الدالة.

وكما نرى أن الفارق بين الصيغتين ، هو المرجع إلى الفصيلة (Class &) الذى يستخدم فى حالة الدالة غير العضوة.

وفى الجدول التالى نلخص صيغات دوال المؤثرات وذلك باستخدام المؤثر =+ كمثال للمؤثرات الثنائية ، والمؤثر ++ كمثال للمؤثرات الفردية ، كما نستخدم فصيلة النقطة (point) كمثال.

	المؤثرات الثنائية (المثال المستخدم =+)	المؤثرات الفردية (المثال المستخدم ++)
الدالة العضوة	void point::operator +=(int r);	void point::operator ++(void)
الدالة غير العضوة	void operator +=(point &p, int r);	void operator ++(point &p)

صيغات دوال تحميل المؤثرات

والدالة التالية تستخدم لتعريف المؤثر =+ بحيث يستخدم فى زيادة إحداثيات النقطة بمقدار r:

```
void point::operator+=(int r)
{
        x += r;
```

```
        y += r;
}
```

أما فى هذا المثال فإننا نعيد تعريف المؤثر ++ لكى يستخدم فى زيادة إحداثيات النقطة بمقدار 1:

```
void point::operator++(void)
{
        x++;
        y++;
}
```

وقد افترضنا هنا أن دالة المؤثر عضوة فى الفصيلة ، وأن لها الحق فى التعامل مع الأعضاء x, y مباشرة. فإذا كانت الدالة غير عضوة فإن هذا يستلزم بعض خطوات المعالجة الإضافية ، فالدالة الأخيرة ، مثلا ، تصبح كالآتى:

```
void operator++(point &p)
{
        int a, b;
        a = p.access_x();
        b = p.access_y();
        a++;
        b++;
        p.set_position(a,b);
}
```

وفى المثال التالى نقوم بتعريف المؤثر =+ بحيث يزيد من نصف قطر الدائرة بمقدار x (قيمة يمكن تحديدها فى الدالة الرئيسية للبرنامج) وذلك باستخدام التعبير:

```
My_Circle +=x;
```

مثال(٣-٩) تطبيق على تحميل المؤثرات

```
// ****************** Example 3-9.cpp ********************
// Overloading operators when used with objects
#include <iostream.h>
double const PI=3.14159;
class point {                                    // فصيلة النقطة
protected:
  int x, y;
```

```
public:
// point constructor:
   point(int p1=0, int p2=0) { x=p1; y=p2; }
   void set_position(int p1, int p2) { x=p1; y=p2; };
   int access_x(void) { return (x); }
   int access_y(void) { return (y); }
};
class circle : public point {                          // فصيلة الدائرة
protected:
   double radius;
public:
   void set_radius(double r) { radius=r; };
   double access_r(void) { return radius; };
   double area(void) { return PI*radius*radius; };
   double perimeter(void) { return 2*PI*radius; };
// circle constructor:
   circle(int x1, int x2, double r)
   : point(x1, x2) { set_radius(r); };
   void print_circle(void);
};
// Operator function:          تعريف دالة تحميل المؤثر += مع فصيلة الدائرة
void operator+=(circle &ref, int x)
{
   ref.set_radius(ref.access_r()+x);       // زيادة نصف القطر بمقدار x
}
// Print results:
void circle::print_circle(void)
{
   cout << "\nThe circle radius is: "
        << access_r();
}
int main()
{
   circle My_Circle(10,15,3.0);
   for(int i=1; i<= 10; i++) {
      My_Circle.print_circle();
      My_Circle += 10;
   }
   return 0;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-9.cpp

تنفيذ البرنامج

عند تنفيذ هذا البرنامج نحصل على النتيجة الآتية:

```
The circle radius is: 3
The circle radius is: 13
The circle radius is: 23
The circle radius is: 33
The circle radius is: 43
The circle radius is: 53
The circle radius is: 63
The circle radius is: 73
The circle radius is: 83
The circle radius is: 93
Press any key to continue
```

القيود التى تخضع لها دوال تحميل المؤثرات

١.لا يمكنك ابتكار مؤثر جديد باستخدام التحميل لمؤثرات مختلفة.

٢.لا يمكنك تغيير معنى مؤثر مبنى فى اللغة مثل توظيف مؤثر الجمع لإجراء عملية قسمة.

٣.لا يمكنك تغيير أولوية المؤثرات (Operator Precedence).

٤.لتحميل المؤثرات الآتية يلزم إعلان الدالة كدالة إستاتيكية:

=	[]	()	->

٥.لا يمكنك تحميل المؤثرات الآتية:

?	.	::	.*	->*

٦.من الشائع أن تكون دوال تحميل المؤثرات من نفس نمط الفصيلة ، وهذا يعنى أن القيمة المرتجعة من الدالة عبارة عن هدف ، مثل:

```
point operator +=(point &MyPoint, int x)
{
...
...
return MyPoint;
}
```

وإذا كانت الدالة لا تستخدم بارامترات فإنها ترجـــع المؤشـــر this كالمثال التالى:

```
point point::operator ++(void)
{
...
...
return this;
}
```

(٣-١٥) تحميل مؤثرات الدخل والخرج
Overloading I/O Operators

كما رأينا أن تحميل المؤثرات بصفة عامة يكسبها تعريفـــا خاصـــا يتوقف على برنامجك. أما مؤثرات الدخل والخرج فإن الهدف العام من تحميلها هو أن تتمكن من استخدامها فى طباعـــة الأهـــداف أو تخزينها فى ملف ، كالمثال:

```
cout << Employee1;
```

إن هذه العبارة تؤدى إلى طباعة الهدف Employee1 بكل محتوياتـــه من الأعضاء. وبنفس المنطق يمكنك استقبال محتويات هدف بأكمله من لوحة الأزرار (أو من قناة الدخل بصفة عامة) بعبارة مثل:

```
cin >> Employee2;
```

ولإعادة تعريف المؤثر >> استخدم الدالة الآتية:

```
ostream &operator<<(ostream &MyStream, MyClass &MyObject);
```

حيث:

- MyStream: مرجع إلى فصيلة الخرج **ostream**.
- MyObject: مرجع إلى الهدف المراد حشره فى قناة الخـــرج ، (ويجوز أن يكون هدفا فى حالة الخرج).

- MyClass: الفصيلة التى يتبعها الهدف MyObject.

ولإعادة تعريف المؤثر << استخدم الدالة الآتية:

```
istream &operator<<(istream &MyStream, MyClass &MyObject);
```

حيث:

- MyStream: مرجع إلى فصيلة الدخل **istream**.
- MyObject: مرجع إلى الهدف المراد استخراجه من قناة الدخل.
- MyClass: الفصيلة التى يتبعها الهدف MyObject.

وكما نرى أن دالة تحميل المؤثر >> ترجع مرجعا إلى فصيلة قناة الخرج **ostream** (أى هدف من النمط **ostream**) ، أما دالة تحميل المؤثر << فهى ترجع مرجعا إلى فصيلة قناة الدخل **istream** (أى هدف من النمط **istream**).

ولأنه مطلوب من هذه الدوال أن تتوصل إلى الأعضاء الخاصة للفصيلة MyClass ، فإنها لابد وأن تكون دوالا صديقة لها.

وفى المثال التالى نستقبل بيانات اثنين من الموظفين Employee1 و Employee2 من لوحة الأزرار ثم نطبعها كأهداف مجملة باستخدام مؤثرات الدخل والخرج بعد إعادة تعريفها.

مثال (٣-١٠): تطبيق على تحميل مؤثرات الدخل والخرج

```
// ****************** Example 3-10.cpp ******************
#include <iostream.h>
#include <iomanip.h>
#include <string.h>
class Payroll {                                    // فصيلة الأجور
private:
```

```
  char name[30];
  char dept_code[4];
  char SSN[12];
  float payrate;
public:                                          // عينات دوال المؤثرات الصديقة
  friend ostream &operator<<(ostream &Strm, Payroll obj);
  friend istream &operator>>(istream &Strm, Payroll &obj);
};                                               //  تعريف دالة المؤثر >> الصديقة:
ostream &operator<<(ostream &Strm, Payroll obj)
{
  Strm << endl << setiosflags(ios::left)
       << setw(20) << obj.name
       << setw(5) << obj.dept_code
       << setw(14) << obj.SSN
       << "  $" << setiosflags(ios::showpoint)
       << setprecision(2) << obj.payrate;
  return Strm;
}                                                //  تعريف دالة المؤثر << الصديقة:
istream &operator>>(istream &Strm, Payroll &obj)
{
  cout << endl << "Please enter employee information:";
  cout << endl << "Name, Department, SSN, and Payrate.";
  cout << endl << "Press <Enter> after each entry:" << endl;
  Strm.get(obj.name,20);
  Strm >> obj.dept_code >> obj.SSN >> obj.payrate;
// Ignore the new_line character, which remains in the buffer:
  Strm.ignore(1);
  return Strm;
}
int main()
{
  Payroll employee1, employee2;                  // إعلان الأهداف
  cin  >> employee1 >> employee2;                // إدخال بيانات الأهداف
  cout << endl << "Employee information:";       // الطباعة
  cout << employee1 << employee2 << endl;
  return(0);
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-10.cpp

تنفيذ البرنامج

الآتى بعد مثال لتشغيل البرنامج واستقبال البيانات ثـــم طباعـــة المعلومات:

```
Please enter employee information: -------- تعليمات التشغيل
Name, Department, SSN, and Payrate.
Press <Enter> after each entry:
Aly AboSamra  ---------------------------- البيانات المدخلة
OCR
768-12-9876
88

Please enter employee information: ------ تعليمات التشغيل
---
Name, Department, SSN, and Payrate.
Press <Enter> after each entry:
Shaimaa Baraka ---------------------------- البيانات المدخلة
ODB
876-77-6543
78

Employee information: ---------------------- طباعة النتائج
Aly AboSamra          OCR 768-12-9876      $88.
Shaimaa Baraka        ODB 876-77-6543      $78.
Press any key to continue
```

ملاحظات على البرنامج:

١. استخدمنا فى هذا البرنامج بعـــض أدوات الفورمـــات (I/O Manipulators) والروتينات منخفضة المســتوى (I/O Low Level Routines) وذلك لتسهيل عملية الدخل ، ولتهذيب شكل الخرج. ولم نتعرض فى هذا الكتاب لهذه الأدوات حيث أنها تخـــرج عن موضوعنا الرئيسى. ويمكنك مراجعتها بكتاب "ثلاث قمم فى كتاب واحد".

٢.تستخدم نفس الفكرة فى حشر الأهداف فى قناة خرج متصلة بملف فى حالة كتابة الأهداف فى ملفات. كما يتم استخراج الأهداف من قناة الدخل المتصلة بالملف فى حالة قراءتها من الملف. وللاطلاع على تفصيلات موضوع الملفات أيضا ، راجع الجزء الأول من كتاب "ثلاث قمم فى كتاب واحد".

(٣-١٦) نماذج الدوال Function Templates

ذكرنا من قبل خاصية التحميل الزائد (راجع مثال ٢-١) التى تمكننا من استخدام اسم واحد للدالة وتعريفها عدة مرات باستخدام بارامترات ذات أنماط مختلفة مثل:

```
double SquareIt(double x) {return x*x;}
int SquareIt(int x) {return x*x;}
```

ومن عيوب هذه الطريقة أنك يجب أن تتأكد من أن البارامترات تختلف عن بعضها بطريقة لا تحتمل الإبهام. هناك طريقة أخرى أكثر سهولة وأقل خطورة وهى استخدام نموذج للدالة يحتوى على نمط عام مثل "MyType" فى هذا المثال:

```
template <class MyType> MyType SquareIt(MyType x)
{ return x*x; }
```

فإذا استدعيت هذه الدالة باستخدام عدد حقيقى (float) كالآتى:

```
SquareIt(2.5)
```

فإن المترجم يتعرف على نوع البارامتر الممرر ، ويحول النمط MyType إلى هذا النمط. ولو أنك استدعيت الدالة باستخدام عدد صحيح مثل SquareIt(5) فإن النمط MyType يصبح int ، وهكذا.

ويجوز أن تعلن عن دالة تستخدم بارامترين من النوع MyType مثل:

```
template <class MyType> MyType MultiplyEm(MyType x, MyType y)
{ return x*y; }
```

وعند استدعاء هذه الدالة يلزم أن يكون كل من البارامترين من نفس النمط ، مثل:

```
MultiplyEm(5,5)
MultiplyEm(5.0,5.0)  أو
```

جرب المثال التالى ، وغير أنواع البارامترات الممررة وفقا للقواعد وشاهد النتائج.

مثال (٣-١١): استخدام نماذج الدوال

```
// ******************** Example 3-11 ******************
// Function Templates
#include <iostream.h>
//
template <class MyType> MyType MultiplyEm(MyType x, MyType
y)
{
        return x*y;
}
//
template <class MyType> MyType SquareIt(MyType x)
{
        return x*x;
}
//
void main()
{
        cout << MultiplyEm(5,5) << endl;
        cout << SquareIt(2.5) << endl;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-11.cpp

تنفيذ البرنامج

عند تنفيذ هذا البرنامج نحصل على النتيجة التالية:

```
25
6.25
Press any key to continue
```

إذا أردت أن تستخدم بارامترات من أنماط مختلفة فعليك أن تعلنها فى نموذج الدالة كالمثال التالى:

```
template <class MyType1, class MyType2>
MyType2 MultiplyEm(MyType1 x, MyType2 y) { return x*y;}
```

فى هذه الدالة استخدمنا نمطين MyType1 و MyType2. ولذلك أصبح من الممكن استخدام استدعاءات مثل:

MultiplyEm(2.51, 2)

أو MultiplyEm(2, 2.51)

أما النمط الذى ترجعه الدالة فهو النمط MyType2 كما جاء فى تعريف الدالة ، وهو نفسه نمط البارامتر الثانى. أى أن الاستدعاء MultiplyEm(2.51, 2) سوف يرجع قيمة صحيحة ، والاستدعاء MultiplyEm(2, 2.51) سوف يرجع قيمة حقيقية. ولو أنك غيرت صيغة تعريف الدالة بحيث ترجع النمط MyType1 فسوف يحدث العكس. يمكنك تجربة هذه التباديل فى المثال التالى.

مثال (٣-١٢): نماذج الدوال باستخدام أنماط بارامترات مختلفة

```
// ******************** Example 3-12 ********************
// Function Templates:
#include <iostream.h>
template <class MyType1, class MyType2>
MyType2 MultiplyEm(MyType1 x, MyType2 y)
{
```

```
        return x*y;
}
//
void main()
{
        cout << MultiplyEm(2.51, 2) << endl;
        cout << MultiplyEm(2, 2.51) << endl;
}
// ***********************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-12.cpp

تنفيذ البرنامج

عند تنفيذ هذا البرنامج نحصل على النتيجة التالية:

```
5
5.02
Press any key to continue
```

(٣-١٧) نماذج الفصائل Class Templates

كما مع نماذج الدوال ، يمكنك أن تنشئ نموذجا عاما لفصيلة مـــا ، ثم تحدد الأنماط المستخدمة فى هذه الفصيلة من داخل التطبيق نفسه عندما تخلق الأهداف المبنية على الفصيلة. ولنفرض أننا نرغب فى إنشاء فصيلة النقطة التى تستخدم أرقاما صحيحة لإحداثيات النقـط. ثم نشأت الحاجة فى وقت لاحق لاستخدام أرقام حقيقية لإحداثيـــات النقط. إن الفصيلة التى تحتوى على أعضاء صحيحة لن تصلح لهذا الغرض. والحل الوحيد هو إنشاء فصيلة جديدة تســـتخدم البيانـــات الحقيقية للإحداثيات. أما استخدام نموذج الفصيلة فهو يجنبنـــا مثـــل هذه المواقف حيث أنه يمنحنا الفرصة أن نحدد الأنماط عند إنشـــاء

الأهداف. وهذا هو مثال لنموذج فصيلة النقطة فى أبسط صـــوره ، حيث يحتوى فقط على البيانات الأعضاء (x, y):

```
template <class T> class point
{
public:
        T x;
        T y;
};
```

إن النمط T فى هذا المثال نمط عام (يماثل النمط MyType فى الأمثلة السابقة). وعندما تعلن هدف نقطة بداخل برنامجك فكل ما عليك أن تستبدل النمط T بالنمط المناسب ، فالهدف الآتى يستخدم إحداثيـــات صحيحة (int):

```
point <int> p1;
```

أما الهدف الآتى فيستخدم إحداثيات حقيقية (double):

```
point <double> p2;
```

ولو أنك أردت إعلان بعض الدوال الأعضاء بداخل إعلان الفصيلة ، فكل ما عليك أن تستبدل الأنماط العددية بالنمط العام T كالأمثلـــة الآتية:

```
T access_x(void) { return(x); };
void setpoint(T p1, T p2) { x=p1; y=p2; }
```

أما لو جاء تعريف الدالة خارج جسم الفصيلة فعليـــك أن تســـتخدم التوصيف الكامل لعناوين الدوال . وهذه أمثلة للدالة setpoint() ودالة البناء point():

```
template <class T> void point<T>::setpoint(T p1, T p2)
{
 ...
}
template <class T> point<T>::point(T p1, T p2)
{
 ...
}
```

مثال (٣-١٣) استخدام نماذج الفصائل مع الدوال الأعضاء

فى هذا المثال استخدمنا فصيلة النقطة محتوية على مجموعة من الدوال الأعضاء ، علاوة على دالتين للبناء ، واحدة ذات بارامترات ، والأخرى سابقة التعريف ، لبناء النقط بإحداثيات صحيحة (1,2). وفى الدالة الرئيسية قد تم الإعلان عن ثلاثة أهداف: صحيح ، وحقيقى ، وسابق التعريف.

```
// ******************** Example 3-13 ********************
// Class Templates
#include <iostream.h>
//
template <class T> class point
{
protected:
        T x;
        T y;
public:
        T access_x(void) {return(x);}
        T access_y(void) {return(y);}
        void setpoint(T p1, T p2);
// Constructor:
        point(T p1, T p2);
// Default constructor:
        point(void) {x=1; y=2;}
};

// Class implementation
template <class T> void point<T>::setpoint(T p1, T p2)
{
        x = p1;
   y = p2;
}
// Class Constructor
template <class T> point<T>::point(T p1, T p2)
{
        x = p1;
        y = p2;
}

// Main function:
void main(void)
{
```

```
// Declare objects:
        point<int> p1(0,0);
        p1.setpoint(10,20);
        point <double> p2 (1.0, 1.0);
        p2.setpoint(1.5,2.5);
// Declare an object using def. constructor:
        point <int> p3;
// Print results:
        cout << "x1=" <<p1.access_x()
        <<", "<<"y1="<<p1.access_y();
        cout <<endl<<"x2="<<p2.access_x()
        << ", "<<"y2="<<p2.access_y();
        cout <<endl<<"x3="<<p3.access_x()
        << ", "<<"y3="<<p3.access_y()
        << endl;
}
// ************************************************
```

يوجد البرنامج على القرص تحت الاسم Ex3-13.cpp

تنفيذ البرنامج

عند تنفيذ هذا البرنامج يعطى النتائج الآتية الممثلة لإحداثيات النقط الثلاثة p1, p2, p3:

```
x1=10, y1=20
x2=1.5, y2=2.5
x3=1, y3=2
Press any key to continue
```

(٣-١٨) اختبر مستواك فى لغة سى++

لا تمثل هذه الأسئلة مراجعة للغة سى ++ التى عرضناها فى هذا الكتاب ، ولكنها أسئلة عامة من المفضل أن تجيب عليها قبل أن تخطو نحو برمجة النوافذ:

١. إذا أعطيت قيم المتغيرات الآتية:

```
a = 1;
b = 2;
c = 3;
```

فما هى قيم التعبيرات الآتية:

```
1.  a <= c || b > c
2.  a < b && c < b
3.  (a += 3) < c
```

٢. إذا أعطيت الإعلان الآتي:

```
int n, i = 100, j = 20, x = 3, y = 100;
```

فما هى قيم n الناتجة من العبارات الآتية:

```
1. n = (i < j) && (x < ++y);
1. n = (j - i) && (x < y++);
2. n = (i < j) || (y += i);
```

٣. إذا أعطيت الإعلان الآتي:

```
int x = 5;
int y = 7;
```

فما هى قيم y الناتجة من العبارات الآتية:

```
1.  y  +=  x++;
2.  y  +=  ++x;
3.  y  +=  (x = x + 2);
```

١. عند قيم معينة للمتغيرات x فإن الماكرو الآتى يعطى نتائج غير صحيحة .اضرب مثالا لهذه القيم وبين طريقة تلافى هذا الخطأ:

```
#define SQUARE(X) X * X
```

٢. ما هو الخطأ الذى يحتوى عليه البرنامج الآتى:

```
void main(void)
{
        int *ptr;
        ptr =10;
}
```

٣. احجز ١٠٠ مكانا فى الذاكرة لتخزين متغيرات صحيحة بحيث يعمل البرنامج تحت أى نظام تشغيل.

٤. ماذا تعنى هذه الإعلانات ؟

```
1.  int *foo(float, int, char *);
2.  int a[5];
3.  int *a[5];
4.  int (*a)[5];
5.  char **a;
```

٥. اعتبر الإعلان التالي:

```
int *a;
```

كم خلية من خلايا الذاكرة (مقاسة بالبايت) يتقدمها المؤشر a نتيجة العبارة a++

٦. كيف تضرب عددا فى ١٥ بدون استخدام علامة الضرب؟ وكيف تضرب عددا فى ١٦ بنفس الطريقة؟

٧. ما معنى الإعلانات التالية:

```
1. const int *i;
2. int *const p;
3. const int *const m;
```

٨. ما هو الحيز الذى تشغله فصيلة النقطة الآتية:

```
class point {
        int x;
        int y;
        int accessX(void) { return x; }
        int accessY(void) { return y; }
        point (int p1=10, int p2=20) {x = p1; y = p2;}
};
```

٩. فى فصيلة النقطة السابقة ، استبدل إعلان المتغير y بالإعلان:

```
int const y;
```

ثم أكتب دالة بناء الفصيلة.

توجد الإجابات بالملحق (أ)

تذكر هذه المصطلحات

Access Level	درجة التوصل
Access Modifier	معدل التوصل
Base Class	الفصيلة الأساسية /فصيلة الأساس
Class Declaration	إعلان الفصيلة
Class Implementation	تطبيق الفصيلة
Constructors	دوال البناء
Constructors with default parameters	دوال البناء ذات البارامترات سابقة التعريف
Conversion Functions	دوال التحويل
Copy Constructors	دوال البناء الناسخة
Data Members	البيانات الأعضاء
Default Constructors	دوال البناء سابقة التعريف
Derived Class	الفصيلة المشتقة
Destructors	دوال الهدم
Friends	الأصدقاء
Initialization List	قائمة الشحن
Member Functions	الدوال الأعضاء
Static Data Members	البيانات الاستاتيكية الأعضاء
Static Member Functions	الدوال الاستاتيكية الأعضاء
Union Constructors	دوال البناء للمنشئات المشتركة
Virtual Functions	الدوال الافتراضية
Function Templates	نماذج الدوال
Class Templates	نماذج الفصائل

الجزء الثالث

برمجة النوافذ

Windows Programming

فى هذا الجزء تبدأ رحلتنا مع برمجة النوافذ ، وهى تتطلب خلفية راسخة عن قواعد لغة سى++ ، وهذا هو السبب فى أننا قدمنا الجزء الأول والثانى فى بداية الكتاب حتى تكون المعارف كلها تحت إصبعك فى كتاب واحد. وبرمجة النوافذ لها أكثر من مدخل ، لكنها جميعا تهدف إلى خلق التطبيق النوافذى بخصائصه المعروفة من القوائم وأدوات التحكم وصناديق الحوار والقوائم. وهذا هو تسلسل الموضوعات فى الأبواب التالية:

الباب الرابع: سوف نبدأ بوصف بيئة النوافذ من وجهة نظر المبرمج.

الباب الخامس: نقدم بعض الأمثلة العامة التى يساعدنا فيها الساحر (Wizard) والتى تمكننا من فهم تصميم البرنامج النوافذى من أقصر طريق.

الأبواب التالية حتى نهاية الكتاب: سوف نستخدم فيها مؤسسة الفصائل MFC باعتبارها الموضوع الرئيسى لهذا الكتاب.

ملاحظة:

فى البرامج التالية بفصول الكتاب سوف نستخدم البنط الثقيل (Bold) لتمييز الآتى:

- دوال الوصلة API مثل **WinMaun()**
- دوال المؤسسة MFC مثل **OnCmdMsg()**
- فصائل المؤسسة MFC مثل **CDocument**. لاحظ أن دوال البناء تحمل اسم الفصائل ولكنها تتميز عنها بوجود الأقواس (). ومن الجائز أن تكون الدالة ذات بارامترات ولكننا سوف نغفلها مالم تكن مستخدمة بداخل برنامج.
- البيانات الأعضاء بفصائل المؤسسة MFC مثل **m_pMainWnd**.
- الثوابت والماكرو وأسماء الرسائل مثل **WM_PAINT** و **RGB**.
- المصطلحات الجديدة عند ظهورها لأول مرة فى فقرات الكتاب.

أما الفصائل والدوال والبيانات المبتكرة فسوف تكتب بالبنط العادى.

وتجب ملاحظة أننا لم نقدم فى الكتاب نص البرامج النوافذية الكاملة واكتفينا بوضعها على القرص المصاحب ، ولذلك لتقليل حجم الكتاب وتوجيه الهدف نحو تقديم أكبر جرعة من مهارات البرمجة فى مجلد واحد.

ملاحظة:

تذكر:

١. للاطلاع على الصيغة التفصيلية لإحدى الدوال أو الفصائل ، ضع مؤشر الفأر فوقها ثم اضغط زر اللوحة F1.

٢. لعرض الخريطة الكاملة لشجرة فصائل المؤسسة MFC ابحث فى شاشة النجدة (باستخدام البطاقة Search) عن "Hierarchy Chart".