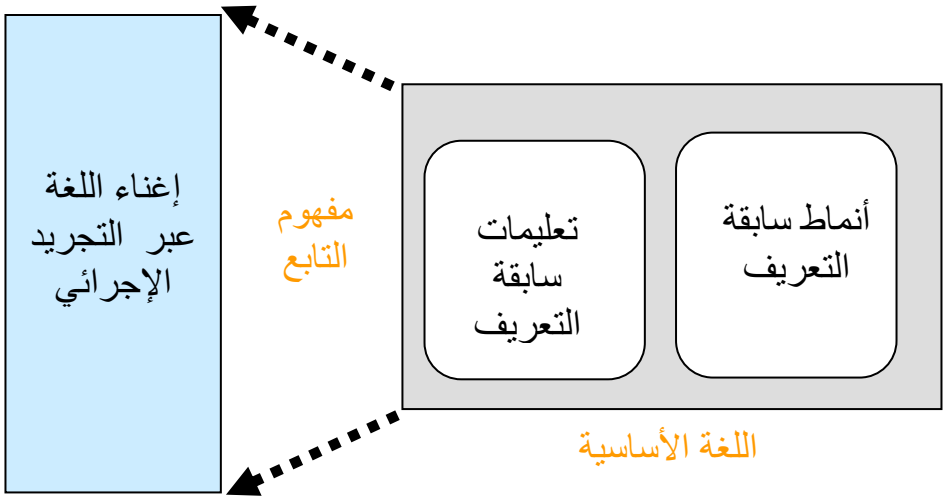
الفصل الرابع
الصفوف والأغراض

مقدمة

تقدم لغة **C++** للمبرمج وسائل تعريف أنماط جديدة، مع إمكان دمج هذه الأنماط على وجهٍ أفضل في مجموعة الأنماط المعرفة سابقاً، يرافق ذلك كله قيام المترجم بجميع عمليات التحكم والمراقبة.

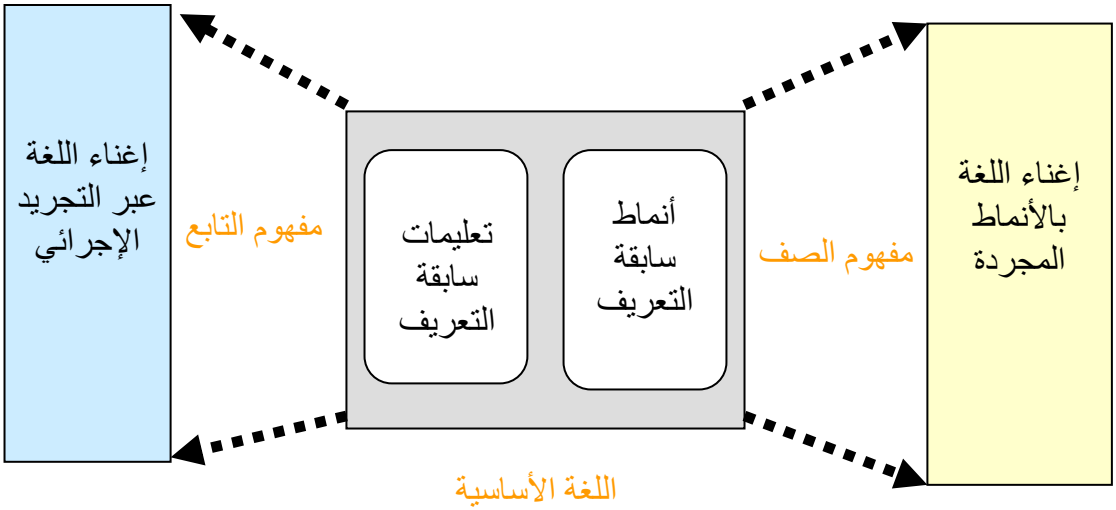
إن بناء أنماط جديدة قابلة للتكامل التام مع ما هو موجود في اللغة، هو خطوةٌ نحو بناء برمجيات أفضل جودةً، ونحو قابلية إعادة استعمال الترميز الموجود، باستغلال أنماط المعطيات التي يبنيها المبرمج أثناء عمله.

وقد مر معك أنه يمكنك في **C++** بناء عمليات جديدة إضافة إلى العمليات السابقة التعريف في اللغة باستعمال مفهوم التابع. تسمى هذه الآلية **التجريد الإجرائي procedural abstraction** وهي آلية تسمح بها غالبية لغات البرمجة الإجرائية.



غير أن التجريد الإجرائي لا يشمل إلا جانباً واحداً من جوانب السيطرة على التطوير البرمجي.

في الحقيقة، من المفيد غالباً أن نتمكن من زيادة قوة تعبير اللغة ببناء أنماط جديدة، وذلك اعتماداً على ما يسمى بنى المعطيات المجردة **abstract data types**.



وكما سنرى لاحقاً، فإن المفهوم الذي يسمح لنا بتنفيذ **أنماط المعطيات المجردة** في C++ هو مفهوم **الصف class**.

ملاحظة:

يعود السبب في تسمية هذه الأنماط بالأنماط المجردة إلى أن مستعملها لا يعرف بالضرورة تمثيلها الداخلي. وهذه مِيْزة هامة تفيد أثناء كتابة البرامج التي تستعمل هذه الأنماط. فعندما يتعامل المبرمج مع الأنماط السابقة التعريف في لغة البرمجة، فإنه يستعمل وضوحاً أنماطاً مجردة. ذلك أنه عندما يعطي المبرمج نمطاً ما لمتحول، يصبح للمتحول بنية داخلية معرفة جيداً، وتكون العمليات المسموح بها على هذه البنية مشروحة في دليل اللغة (int, char,...) دون الإشارة عموماً إلى تمثيلها الداخلي.

وهنالك عدة أسباب تدعم العمل مع أنماط المعطيات منها:

- يعكس البرنامج العمل الذهني للمبرمج على وجهٍ أفضل، عندما نسمح لهذا الأخير بالعمل في مستوى إدراكي أعلى.
- يعزز تكرار استعمال الأنماط الوقايةَ من الأخطاء، إذ يعلن المبرمج نواياه بصراحة،

بتحديد عدد القيم والعمليات.

- يسهل استعمال أنماط المعطيات صيانة البرامج.

يتيح استعمال الأنماط المجردة للمعطيات التفكير بعمق أكبر بوظائف النمط بدلاً من بنية المعطيات الواجب تنفيذها لتحقيق هذا النمط المجرد.

وقبل الدخول في التفاصيل سنعرف كل من الغرض والصف.

الغرض

الغرض هو المفهوم الأساسي في البرمجة الغرضية التوجه. فالأغراض هي الكيانات الأساسية عند تنفيذ البرنامج. وكما رأينا في الفصل الأول، يحتل الغرض مكاناً في الذاكرة الرئيسية وتُحدد المعطياتُ المضمنَة فيه حالته **state**. ولكن الغرض يتضمن إضافة إلى حالته جميع التوابع التي تحدد سلوكه **behavior**، وهذا ما نسميه عادة الكبسلة **encapsulation.**

تسمح الكبسلةُ بجمع المعطيات والتوابع التي تنمذج كياناً ما بعضها إلى بعض، وبذلك يمكن رؤيتها ككل. تُقلص الكبسلةُ إذن تشتت المعطيات والتوابع، وتمكن المبرمج من فهم المسألة فهماً أفضل، وهذا مما يؤدي إلى تقليص تعقيدها.

الغرض = معطيات + عمليات

ومع أن جميع المعلومات تُخزن ضمن الغرض، فإن ذلك لا يعني أن بالإمكان الوصول إليها جميعها. إذ تقسم معلومات غرض ما إلى قسمين:

قسم عام Public، مرئي ويمكن الوصول إليه من خارج الغرض.

قسم خاص Private، غير مرئي ولا يمكن الوصول إليه من خارج الغرض. يمكن، بالطبع، الوصول إلى معلومات هذا القسم واستخدامها من قبل التوابع الأعضاء (**members**) في الغرض.

الصف Class

يعرفُ الصفُ مجموعةً ممكنةً من الأغراض. ينْتُج كلُّ غرضٍ، عند تنفيذ البرنامج، من استنساخ **instantiation** صف موجود. أما الصفُ فهو تمثيلٌ ساكن، جامد (في البرنامج) لمجموعة ممكنة من الأغراض المستَنسخَة.

لتوضيح الفرق بين الغرض والصف، يستَعملُ عادةً مجاز المصنع (الصف) الذي ينتج منتجات (الأغراض). وبإمكان المبرمج تحديد الطريقة التي يجب اتباعها لإنتاج الغرض (البواني في **C++**).

لا نرى في البرنامج إلا الصفوف، في حين لا يوجد عند التنفيذ إلا الأغراض. يمكن مقارنة هذا السلوك بذلك الموافق للنمط، فتعريف صف يكافئ تعريفَ نمطٍ جديد.

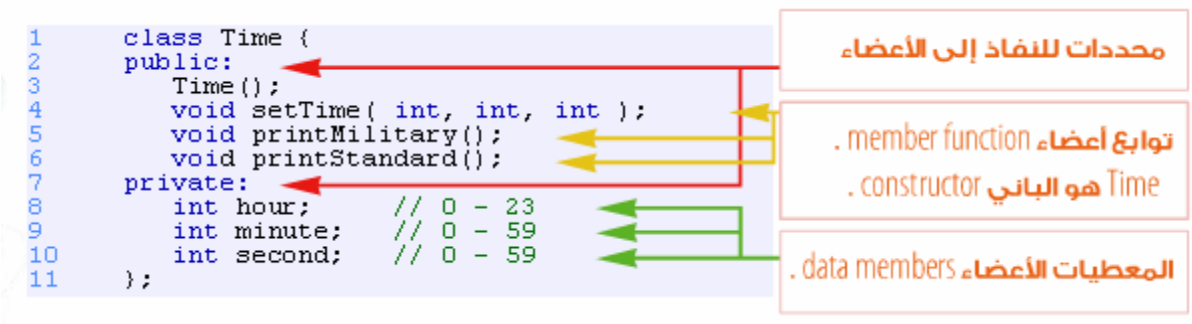
الصف هو نمط يتضمن وصفاً موحداً:

- للواصفات وهي مجموعة من المعطيات التي ترتبط فيما بينها وتكون غالباً من أنماط مختلفة (وتسمى "المعطيات الأعضاء")
- للسلوك وهو مجموعة من الطرائق (وتسمى "التوابع الأعضاء" التي تسمح بالتعامل مع المعطيات الأعضاء).

يعرف الصف باستخدام الكلمة المفتاحية **class** ، ويقع جسم الصف بين العلامتين } و {، وينتهي تعريف الصف بفاصلة منقوطة.

مثال: تنجيز بنية المعطيات المجردة Time في صف

يبين المثال التالي تعريفاً للصف **Time** :



يتوفر عموماً في الصف أربعة أنواع مختلفة من التوابع الأعضاء العامة:

البواني Constructors: تفيد في بناء أغراض من الصف وإعطائها قيماً ابتدائية على وجهٍ صحيح

الهادم Destructor: يسمح الهادم بهدم الغرض في نهاية حياته

النوافذ Accessors: تسمح بالنفاذ إلى المعطيات الأعضاء الخاصة المحتواة في الغرض

المعدلات Modificators: تسمح بتغيير حالة المعطيات الأعضاء الداخلية في الغرض

داخل تعريف كل صف، تكون ضمناً جميع الأعضاء (المعطيات والتوابع) خاصة **private**، ما لم يصرح خلاف ذلك. في حال رغبتنا في جعل المعطيات و/أو التوابع قابلة للنفاذ من الخارج، يجب أن نضعها في قسم مسبوق بالكلمة المحجوزة **public** التي تتبع بالرمز " : ". وهكذا يمكن النفاذ إلى جميع الحقول والتوابع المصرح عنها في القسم العام من قبل جميع مستخدمي الصف.

هناك كلمة أخرى محجوزة وهي **private** للدلالة على أن المعطيات والتوابع الأعضاء التي تظهر في هذا القسم لا يمكن النفاذ إليها إلا من خلال التوابع الأعضاء في الصف نفسه. من ثَم لا يحق للمستخدم النهائي للصف أن ينفذ إلى المعطيات المعرفة في القسم الخاص.

بوجهٍ عام، إذا لم ترد كلمة **private** أو **public**، فهذا يعني أن الحقول ستعامل كما لو كانت معرفة في القسم الخاص.

في تعريف الصف الواحد، يمكن أن ترد الكلمات **private** و **public** عدة مرات وبأي ترتيب. ويمتد كل قسم من الكلمة المحجوزة حتى الكلمة المحجوزة التالية أو حتى نهاية الصف.
من وجهة نظر منهجية يفضل أثناء تعريف الصف أن نُظهر أولاً التوابع والمعطيات الأعضاء العامة، ثم تُسرد المعطيات والتوابع الأعضاء الخاصة.
يجري التصريح عن **المعطيات الأعضاء** بالطريقة ذاتها التي يصرح بها عن المتحولات، عدا أن المعطيات الأعضاء **لا يمكن إعطاؤها** قيماً ابتدائية.
يمكن أن يحوي الصف **تصريحات و/أو تعريفات لتوابع**. ويجري التصريح عن هذه التوابع وتعريفها وفقاً للقواعد المتّبعة في الفصل السابق.
تستطيع التوابع الأعضاء النفاذ إلى جميع المعطيات في الصف سواء أكانت خاصة أم لا. ولا يمكن النفاذ إلى الأعضاء الخاصة **private** إلا من خلال التوابع الأعضاء العامة **public**.

تعريف التوابع الاعضاء

يغلب علينا أن نعرف التوابع الأعضاء خارج الصف أي بعد القوس الكبيرة ;{ التي تنهي تعريف الصف. وإذا عرف تابع عضو **خارج الصف**، يجب أن يسْبق معرف التابع باسم الصف الذي يصرح عن التابع. ويستعمل المؤثر " :: " لهذه الغاية. غير أنه يمكن أن يعرف التابع العضو **داخل الصف**، عندها لا نحتاج لمعامل تمييز المدى :: ولا لاسم الصف قبل اسم التابع.
إن تعريف التابع خارج الصف الذي تنتمي إليه لا يعني أي تعديل في كونه **public** أو **private**

الشكل العام لتعريف التوابع الأعضاء التي تنتمي إلى صف خارج الصف:

```
returnType ClassName::MemberFunctionName(){
    …
    }
```

معامل تمييز المدى scope resolution operator

يسمى المعامل (::) معامل تمييز المدى الثنائي **binary scope resolution operator**، وهو يحدد الصف الذي يمتلك التابع العضو (ذلك أنه من الممكن للتوابع الأعضاء في الصفوف المختلفة أن يكون لها نفس الاسم).

استخدام الأغراض في البرنامج

بعد تعريف الصف، يصبح اسم الصف اسم نمط جديد يمكن أن يستخدم كنمط في تعريف الأغراض، والمصفوفات، على النحو التالي:

```
Time sunset,              // object of type Time
arrayOfTimes[5];          // array of Time objects
```

معامل النفاذ إلى أعضاء الصف

بالطبع **لا يحق** لنا **في البرنامج** النفاذ إلى الأعضاء الخاصة أو استدعاء التوابع الخاصة. أما التوابع العامة، فيحق لنا استدعاءها ولكن باستخدام **معامل النقطة (.)**. يميز هذا المعامل طريقة استدعاء التوابع الأعضاء في صف عن طريقة استدعاء التوابع التي لاتنتمي إلى أي صف (مثل pow(x,n)، sin(x)، ...).

أما **التوابع الأعضاء في الصف**، فهي **تستطيع** النفاذ إلى جميع المعطيات في الصف سواءً أكانت خاصة أم لا، دون استخدام النقطة.

تطبيق

فيما يلي التصريح الكامل عن الصف **Time** وتوابعه:

```cpp
// Time class.
#include <iostream>

using std::cout;
using std::endl;

// Time abstract data type (ADT) definition
class Time {
public:
  Time();                          // constructor
  void setTime( int, int, int );   // set hour, minute,
  void printMilitary();            // print military time
  void printStandard();            // print standard time format
private:
  int hour;     // 0 - 23
  int minute;   // 0 - 59
  int second;   // 0 - 59
};

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
 {
   hour = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
 }

// Print Time in military format
void Time::printMilitary()
 {
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
 }

// Print Time in standard format
void Time::printStandard()
 {
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12)
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
 }

```

لاحظ وجود :: قبل أسماء التوابع

```cpp
// Driver to test simple class Time
int main()
 {
   Time t;  // instantiate object t of class Time

   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();

   t.setTime( 99, 99, 99 ); //attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:"
        << "\nMilitary time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
}
```

لاحظ استدعاء التوابع باستخدام المعامل (.)

خرج البرنامج:

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

ملاحظة1: \n في الرماز السابق هي نفس **endl**

ملاحظة2: التعليمة التالية:

```cpp
hour = ( h >= 0 && h < 24 ) ? h : 0;
```

هي طريقة ثانية لكتابة:

```
if ( h >= 0 && h < 24 ) hour=h; else hour=0;
```

ملاحظة3: تستخدم بعض المترجمات سطراً واحداً بدلاً من تكرار هذه الأسطر

```
using std::cout;
using std::endl;
```

وذلك بكتابة :

```
using namespace std;
```

فصل الواجهات عن التنجيز

عادة، يصرح عن **واجهة الصف** في ملف، ويعرف **جسم التوابع الأعضاء** تعريفاً منفصلاً في ملف آخر.

إيضاح: واجهة الصف تتضمن اسم الصف، والتعريف بالمعطيات الأعضاء والتوابع الأعضاء دون كتابة الرماز التفصيلي للتوابع.

تفيد عملية الفصل فيما يلي:

- إن فصل الواجهات عن التنجيز في الصف يسهل عمليات تعديل الصفوف ومن ثم البرامج.
- عند التصريح عن واجهة الصف **class interface** نخبر المترجم **compiler** عن واصفات الصف والعمليات المتاحة عليه.
- يصف تنجيز الصف **class implementation** كيفية الترميز الداخلي لكل عملية من عمليات الصف.
- تعرف واجهة الصف زبائن الصف بالخدمات التي يقدمها هذا الصف.
- يخفي تنجيز الصف التفاصيل عن كيفية إنجاز الصف لمهامه (التنجيز).

من الناحية العملية، يمكن تقسيم البرنامج في C++ إلى :

- **ملفات الترويسات Header files:** وتتضمن تعريف الصفوف وترويسات التوابع **prototypes function**
- **ملفات الرماز المصدري Source-code files:** وتتضمن تعريف التوابع الأعضاء

مخطط شبه الرماز Pseudocode لإنشاء ملفي الواجهة والتنجيز لكل صف

1- ننشئ ملف ترويسة للصف

2- ننشئ ملف رماز مصدري على النحو التالي:

1. نحمل ملف الترويسة لنحصل على تعريف الصف
2. نعرف التوابع الأعضاء في الصف

تطبيق على الصف Time

```
1     // time1.h
2     // Declaration of the Time class.
3     // Member functions are defined in time1.cpp
4
5     // prevent multiple inclusions of header file
6     #ifndef TIME1_H
7     #define TIME1_H
8
9     // Time abstract data type definition
10    class Time {
11    public:
12       Time();                          // constructor
13       void setTime( int, int, int ); // set hour, minute
14       void printMilitary();            // print military time
15       void printStandard();            // print standard time
16    private:
17       int hour;     // 0 – 23
18       int minute;   // 0 – 59
19       int second;   // 0 – 59
20    };
21
22    #endif
```

نستبدل (.) بالعلامة (_) في اسم الملف

```
1     // time1.cpp
2     // Member function definitions for Time class
3     #include <iostream>
4
5     using std::cout;
6
7     #include "time1.h"
8
9     // Time constructor initializes each data member to zero.
10    // Ensures all Time objects start in a consistent state
11    Time::Time() { hour = minute = second = 0; }
12
13    // Set a new Time value using military time. Perform validity
14    // checks on the data values. Set invalid values to zero.
15    void Time::setTime( int h, int m, int s )
16    {
17       hour   = ( h >= 0 && h < 24 ) ? h : 0;
18       minute = ( m >= 0 && m < 60 ) ? m : 0;
19       second = ( s >= 0 && s < 60 ) ? s : 0;
20    }
21
22    // Print Time in military format
23    void Time::printMilitary()
24    {
25       cout << ( hour < 10 ? "0" : "" ) << hour << ":"
26            << ( minute < 10 ? "0" : "" ) << minute;
27    }
28
29    // Print time in standard format
30    void Time::printStandard()
31    {
32       cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
33            << ":" << ( minute < 10 ? "0" : "" ) << minute
34            << ":" << ( second < 10 ? "0" : "" ) << second
35            << ( hour < 12 ? " AM" : " PM" );
36    }
```

يستخدم الملف المصدري #include لتحميل ملف الترويسة

يتضمن الملف المصدري تعريفات التوابع

```
1     //prog using the class Time
2     // Demonstrate errors resulting from attempts
3     // to access private class members.
4     #include <iostream>
5
6     using std::cout;
7
8     #include "time1.h"
9
10    int main()
11    {
12       Time t;
13
14       // Error: 'Time::hour' is not accessible
15       t.hour = 7;
16
17       // Error: 'Time::minute' is not accessible
18       cout << "minute = " << t.minute;
19
20       return 0;
21    }
```

محاولة تعديل hour وهو عضو من أعضاء المعطيات الخاصة private في المتحول t.

محاولة النفاذ إلى minute وهو عضو من أعضاء المعطيات الخاصة private في المتحول t.

خرج البرنامج:

```
Compiling...
Fig06_06.cpp
D:\Fig06_06.cpp(15) : error C2248: 'hour' : cannot
access private
member declared in class 'Time'
D:\Fig6_06\time1.h(18) : see declaration of 'hour'
D:\Fig06_06.cpp(18) : error C2248: 'minute' : cannot
access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

البواني والهوادم

1- البواني

ما الذي يحدث عندما نصرح في البرنامج الرئيسي عن متحول (غرض) من النمط (الصف) **Time**؟

في الحقيقة، يستدعي المترجم آلياً تابعاً خاصاً من التوابع الأعضاء يسمى الباني **constructor**، فيقوم هذا التابع ببناء الغرض وإعطائه قيمة ابتدائية صحيحة.

يمكن تمييز باني الغرض عن غيره من التوابع بكونه يحمل **اسم الصف** نفسه. **لا يقبل الباني أي نمط عودة** (ولا حتى من النوع **void**)، ولا يسمح بالعمل مع عنوان الباني.

إذاً نعرف الباني كما يلي:

الباني constructor : تابع عضو خاص يقوم بإنشاء غرض من الصف وإعطاء قيمة ابتدائية للمعطيات الأعضاء في هذا الغرض

- يسمى بنفس اسم الصف
- لا يمكن أن يعيد الباني قيمة **return value**

في مثال الصف **Time**، الباني هو التابع **()Time** الذي ليس له موسطات. يقوم الباني **Time** بوضع القيمة الابتدائية **0** في المعطيات الأعضاء مما يضمن أن يكون الغرض في حالة سليمة عند إنشائه.

يمكن أن يتوفر عدة بوانٍ في الصف، وبالطبع، يختلف نموذج هذه البواني رغم كونها جميعاً **تشترك بالاسم نفسه** (راجع فقرة التحميل الزائد للتوابع في الفصل الثالث) ، كما يسمح بتعريف بوانٍ تقبل قيماً افتراضية لمحدداتها.

مثال

لديك فيما يلي واجهة الصف **Point** حيث جرى تعريف ثلاثة بواني أحدها افتراضي والآخران أحدهما يأخذ الإحداثيين **x** و **y** والآخر يأخذ الإحداثيين القطبيين للنقطة:

```
lass Point{
    public:              //  public methods
        Point () { x = 0; y = 0;}  //  a default constructor
        Point (int x0, int y0);    //  a constructor
        Point(double alpha, double r);  //  a constructor
        void move (int dx, int dy);
        void rotate (double alpha);
        int distance (Point p);
    private:
        double x, y;    // private data members
};
```

عند التصريح عن المتحولات يمكننا أن نكتب:

```
int main() {
             Point p2;        // call default constructor
             Point p1 (20,10),
                   p3 (3.14 / 4, 2.5);
```

1- البواني، الباني الافتراضي default constructor

نطلق اسم الباني الافتراضي **default constructor** في الصف على الباني الذي لا يتضمن أية محددات، أو الباني الذي تُعطى قيم افتراضية لجميع محدداته.

إذا لم يحدد المبرمج في تعريفه للصف أي بانٍ، يولّد المترجم بانياً افتراضياً. تكمن فائدة هذه العملية في كون الباني الافتراضي يقوم بانتظام باستدعاء جميع بواني المعطيات الأعضاء إن وجدت. ويعتبر هذا الأمر مفيداً عندما تكون المعطيات الأعضاء في الصف من نمط صفوف أخرى.

عندما ترغب باستدعاء الباني الافتراضي، تكتفي بالتصريح عن الغرض كما يلي:

```
Point p2;
```

إيضاح: راجع الباني الافتراضي للصف **Point** .

1- البواني، البواني ذوات المحددات constructors with arguments

يمكن أن يرغب المبرمج عند استعمال غرض من صف ما أن ينشئ الغرض بقيم ابتدائية يحددها هو. توجد بوانٍ أخرى غير الباني الافتراضي يمكنها تمرير هذه القيم كموسطات لباني الصف.

فمثلاً عند التصريح عن غرض كما يلي:

```
Point p1 (20,10),
      p3 (pi / 4, 2.5);
```

فإن هذا يعني استدعاء بان غير الباني الافتراضي.

تقويم:

إذا وجد عدة بوان تأخذ موسطات، كيف يميز المترجم أيها هو الباني المقصود؟ حدد الإجابة بما يخص تعريف النقطتين **p1** و **p3**.

1- البواني، استخدام المحددات الافتراضية Default argument في البواني

تعامل التوابع الأعضاء في الصفوف -ومن ضمنها البواني- كما يعامل أي تابع في **C++**. وعليه، يمكننا أن نعطي لبعض موسطات الباني قيماً افتراضية. وهنا يجب الانتباه للنقاط التالية:

- يعطى المحدد **argument** قيمة في ترويسة التابع الباني (في تعريف الصف).
- قد يكون عدد القيم الممررة للباني أقل من عدد المتحولات الأعضاء، عندها تعطى هذه القيم للمتحولات الموافقة من اليسار إلى اليمين، وتعطى المتحولات التي لم يبق لها قيم والواقعة على اليمين قيماً افتراضية .

تطبيق

```
1     // time2.h
2     // Declaration of the Time class.
3     // Member functions are defined in time2.cpp
4
5     // preprocessor directives that
6     // prevent multiple inclusions of header file
7     #ifndef TIME2_H
8     #define TIME2_H
9
10    // Time abstract data type definition
11    class Time {
12    public:
13       Time( int = 0, int = 0, int = 0 );  // default
14       void setTime( int, int, int ); // set hour, minute,
15       void printMilitary();          // print military time
16       void printStandard();          // print standard time
17    private:
18       int hour;     // 0 - 23
19       int minute;   // 0 - 59
20       int second;   // 0 - 59
21    };
22
23    #endif
```

لاحظ هنا أن القيم الافتراضية للمتحولات الأعضاء الثلاثة قد وضعت في ترويسة الباني. لاحاجة بنا لأسماء المتحولات الأعضاء. تطبق القيم الافتراضية بترتيب التصريح عن المتحولات الأعضاء في الصف.

```
24
25    // Demonstrating a default constructor
26    // function for class Time.
27    #include <iostream>
28
29    using std::cout;
30    using std::endl;
31
32    #include "time2.h"
33
34    int main()
35    {
36       Time t1,           // all arguments defaulted
37            t2(2),        // minute and second defaulted
38            t3(21, 34),   // second defaulted
39            t4(12, 25, 42), // all values specified
40            t5(27, 74, 99); // all bad values specified
41
42       cout << "Constructed with:\n"
43            << "all arguments defaulted:\n   ";
44       t1.printMilitary();
45       cout << "\n   ";
46       t1.printStandard();
47
48       cout << "\nhour specified; minute and second defaulted:"
49            << "\n   ";
50       t2.printMilitary();
51       cout << "\n   ";
52       t2.printStandard();
53
54       cout << "\nhour and minute specified; second defaulted:"
55            << "\n   ";
56       t3.printMilitary();
57       cout << "\n   ";
58       t3.printStandard();
59
60       cout << "\nhour, minute, and second specified:"
61            << "\n   ";
62       t4.printMilitary();
63       cout << "\n   ";
64       t4.printStandard();
65
66       cout << "\nall invalid values specified:"
67            << "\n   ";
68       t5.printMilitary();
69       cout << "\n   ";
70       t5.printStandard();
71       cout << endl;
72
73       return 0;
74    }
```

لاحظ كيفية إعطاء قيم ابتدائية للأغراض:

Constructor ObjectName (value1,value2…) ;

عند وجود نقص في عدد القيم، يعتبر المترجم كأن القيم الافتراضية موجودة في أقصى اليمين لتكمل العدد.

خرج البرنامج:

```
 OUTPUT
Constructed with:
all arguments defaulted:
   00:00
   12:00:00 AM
hour specified; minute and second defaulted:
   02:00
   2:00:00 AM
hour and minute specified; second defaulted:
   21:34
   9:34:00 PM
hour, minute, and second specified:
   12:25
   12:25:42 PM
all invalid values specified:
   00:00
   12:00:00 AM
```

عندما تخصص قيمة للساعة hour،
توضع القيم الافتراضية لـ minute و
second وهي 0.

1- البواني، تنجيز الباني

عند تنجيز الباني تسمح لغة C++ للمبرمج باستخدام قائمة قيم ابتدائية initilization list لإعطاء القيم الابتدائية للمعطيات الأعضاء. ويجري ذلك بكتابة الباني (الترويسة والجسم) وفق الشكل التالي:

```
ClassName::ClassName(T1 arg1…,Tm argm,…,Tn argn):data1(arg1),…, datan(argn)
{……
……}
```

حيث data1,...,datan هي أسماء الحقول في المعطيات الأعضاء وهي من الأنماط T1,...Tm,...Tn على الترتيب.

في حال استخدام قائمة القيم الابتدائية يبدأ العمل بتنفيذ العمليات الواردة في قائمة القيم الابتدائية initilization list ، قبل التعليمات الواردة في جسم التابع.

مثال: باني الصف Rectangle:

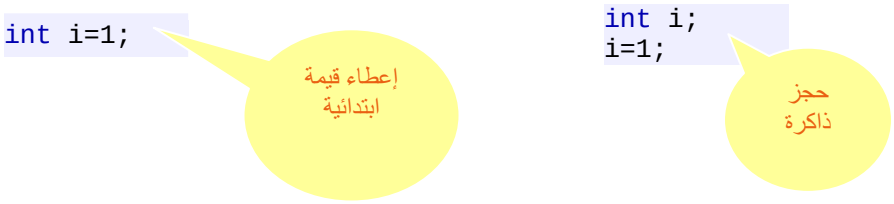
```
Rectangle:: Rectangle(double l, double w): length(l), width(w)
{ }
```

ولكن ما الهدف من إعطاء قيم ابتدائية للباني؟ وما الفرق بين إعطاء قيم ابتدائية للباني واستخدام التعليمات في جسم الباني لإعطاء القيم المرغوب بها للمعطيات الأعضاء؟

قبل الإجابة عليك أن تتذكر الحالة البسيطة التي تصرح فيها عن متحول **i** من نمط بسيط وليكن **int**.

ما الفرق بين أن نكتب: **int i=1;** أو نكتب: **int i; i=1;** ؟

بمعنى آخر ما الفرق بين حجز ذاكرة لمتحول **allocation** وإعطاء قيمة ابتدائية **initialization** ؟



إعطاء قيمة ابتدائية **initialization** : إنشاء متحول وإعطاؤه قيمة خاصة لحظة الإنشاء .

حجز ذاكرة لمتحول **allocation**: تخصيص ذاكرة للمتحول وإعطائه قيمة لاحقاً.

```
int num=0, round, position=2;
.
.
round=7;
```

إن الفرق بين استخدام قائمة القيم الابتدائية في ترويسة الباني أو التعليمات الواردة في جسم التابع يشبه تماماً الفرق بين إعطاء قيمة ابتدائية للمتحول لحظة الإنشاء و إسناد قيمة له لاحقاً.

مثال

```
class Rectangle{

public:
//default constructor
 Rectangle(double = 0.0, double = 0.0);

//compute rectangle measurements
double perimeter();
double area();

//data access function
double getLength();
double getWidth();

//data update function
void setSides(double l, double w);

private:
   double length,width;
};
Rectangle:: Rectangle(double l, double w): length(l), width(w)
 { }
```

يمكن أن يكون تنجيز هذا الباني الأخير على النحو التالي:

```
Rectangle:: Rectangle(double l, double w)
{  length=l;
 width=w };
```

تقويم:

هل هذا الباني هو بان افتراضي؟

نعم لأن موسطاته تأخذ قيماً افتراضية في تعريف الصف، كما في السطر التالي:

```
Rectangle(double = 0.0, double = 0.0);
```

في بعض الحالات كما في مثال المستطيل لا يوجد فرق بين الطريقتين، لكن هنالك حالات يجب فيها استخدام قائمة القيم الابتدائية **initilization list** عند تنجيز البواني كما هي الحال في الوراثة، والثوابت، والمرجع.

2- الهوادم

هادم الغرض هو تابع يحمل اسم الصف ذاته، مسبوقاً بالرمز "~". وكما في البواني، لا يقبل الهادم نمط عودة. إن الهادم لا يقبل محددات، ومن ثَم لا يمكن زيادة تحميله.

إن دور الهادم هو أن يقوم بجميع أعمال الصيانة اللازمة بعد أن ينتهي "وجود" الغرض في ذاكرة الحاسوب. وهذا العمل يهدف تحديداً إلى استرداد الذاكرة **المحجوزة ديناميكياً** للغرض الذي سوف يختفي.

إيضاح: سنتكلم لاحقاً عن الحجز الديناميكي للذاكرة في فصل :

"المؤشرات **pointers** والإدارة الديناميكية للذاكرة **dynamic memory management**"

عند نهاية تنفيذ كتلة، يستدعي نظام البرمجة C++ هادم الغرض آلياً، وذلك لكل متحول من نمط صف معرف في هذه الكتلة، كما يستدعي النظام الهادم للتخلص من جميع الأغراض المؤقتة التي أنشأها دون علم المبرمج. ويسمح باستدعاء هادم الغرض صراحةً (في حالات نادرة) وذلك على نقيض البواني.

أمثلة عن صفوف

الصف Person:

```
class Person{
   char name[20];
   int yearOfBirth;

public:
   void displayDetails()
{
           cout << name << " born in   "
              <<yearOfBirth << endl;
}
     //…..
};
```

إن عدم وجود الكلمة **public** أو **private** يعني ضمنياً **private**

الصف Creature:

لاحظ وجود طريقتين لتعريف توابع الصف:

1- تعرف التوابع كاملة داخل الصف ويستخدم هذا الأسلوب إذا كانت تعريف التوابع قصيراً.

2- تعرف التوابع خارج تعريف الصف ويستخدم المعامل :: لهذا الغرض

الطريقة الأولى:

```
class Creature
   {
 private:
   int yearOfBirth;
 public:
   Creature()
    {
      yearOfBirth = 1970;
      cout << “Hello.”;
    }
Creature(int year){
      yearOfBirth = year;
      }
Creature(Creature & otherCreature){
      yearOfBirth=
          otherCreature.getYearOfBirth();
}

void setYearOfBirth(int year)
{
            yearOfBirth = year;
            }
   int getYearOfBirth()
{
            return yearOfBirth;
            }
};
```

باني افتراضي.

باني النسخ لانه يستخدم للحصول على غرض ثان من نفس النمط يحتوي نفس المعطيات.

الطريقة الثانية:

```
class Creature {
private:
   int yearOfBirth;
public:
   Creature();
   Creature(int year);
   Creature(Creature & otherCreature);
   void setYearOfBirth(int year);
   int getYearOfBirth();
};
Creature:: Creature()
   {
      yearOfBirth = 1970;
      cout << “Hello.”;
}
 Creature::  Creature(int year){
      yearOfBirth = year;
      }
  Creature:: Creature(Creature& otherCreature){
      yearOfBirth= otherCreature.getYearOfBirth();
      }

void Creature::setYearOfBirth(int year){
            yearOfBirth = year;
            }
int Creature::getYearOfBirth(){
            return yearOfBirth;
            }
```

باني افتراضي.

باني النسخ لانه يستخدم للحصول على غرض ثان من نفس النمط يحتوي نفس المعطيات.

باني افتراضي.

يسمى هذا الباني باني النسخ لأنه يستخدم للحصول على غرض ثان من نفس النمط يحتوي نفس المعطيات. فمثلاً لو بنينا الغرض MyDog كما يلي: Creature myDog(1995); يمكننا باستخدام باني النسخ الحصول على غرض له نفس قيم المعطيات (أي نفس القيمة للعضو yearOfBirth) كما يلي:
Creature myCat(myDog);

الأغرض الثابتة

سبق لك أن تعاملت مع المتحولات الثابتة في **C++** كما في التعريف:

```
const double pi=3.14;
```

على غرار هذا التعريف يمكنك بناء أغراض ثابتة باستخدام الكلمة المفتاحية **const**. إن استخدام **const** يحدد أن الغرض غير قابل للتعديل، بحيث أن أية محاولة لتعديل الغرض تؤدي إلى خطأ نحوي.

مثال

```
const Time noon( 12, 0, 0);
```

تصريح عن الغرض الثابت **noon** من الصف **Time** وإعطاؤه القيمة 12 ظهراً.

التوابع الأعضاء الثابتة

إن وجود أغراض ثابتة **const** يتطلب تعريف توابع أعضاء ثابتة **const** ، إذ أنه لايمكن لتابع عضو أن يتعامل مع غرض ثابت ما لم يكن هذا التابع ثابتاً.

- لاتستطيع التوابع الثابتة تعديل الغرض
- تضاف كلمة **const** في ترويسة التابع وفي تنجيزه

في الترويسة:

```
ReturnType FunctionName(param1,param2…) const;
```

في التنجيز:

```
ReturnType FunctionName(param1,param2…) const {….};
```

مثال

```
int A::getValue( ) const
              {return privateDataMember};
```

تطبيق على الصف Time

```
1     // time5.h
2     // Declaration of the class Time.
3     // Member functions defined in time5.cpp
4     #ifndef TIME5_H
5     #define TIME5_H
6
7     class Time {
8     public:
9        Time( int = 0, int = 0, int = 0 );  // default
10
11       // set functions
12       void setTime( int, int, int );  // set time
13       void setHour( int );     // set hour
14       void setMinute( int );   // set minute
15       void setSecond( int );   // set second
16
17       // get functions (normally declared const)
18       int getHour() const;     // return hour
19       int getMinute() const;   // return minute
20       int getSecond() const;   // return second
21
22       // print functions (normally declared const)
23       void printMilitary() const;  // print military time
24       void printStandard();        // print standard time
25    private:
26       int hour;              // 0 - 23
27       int minute;            // 0 - 59
28       int second;            // 0 - 59
29    };
30
31    #endif
```

```cpp
// time5.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time5.h"

// Constructor function to initialize private data.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
void Time::setTime( int h, int m, int s )
{
   setHour( h );
   setMinute( m );
   setSecond( s );
}

// Set the hour value
void Time::setHour( int h )
   { hour = ( h >= 0 && h < 24 ) ? h : 0; }

// Set the minute value
void Time::setMinute( int m )
   { minute = ( m >= 0 && m < 60 ) ? m : 0; }

// Set the second value
void Time::setSecond( int s )
   { second = ( s >= 0 && s < 60 ) ? s : 0; }

// Get the hour value
int Time::getHour() const { return hour; }

// Get the minute value
int Time::getMinute() const { return minute; }

// Get the second value
int Time::getSecond() const { return second; }

```

إن الباني ليس تابعاً ثابتاً لكنه يمكن أن يستدعى للحصول على أغراض ثابتة.

لاحظ استعمال الكلمة المفتاحية const في تعريف التابع وترويسته

لايمكن للتوابع غير الثابتة functions non-const

استخدام أغراض ثابتة وإن كانت لاتقوم بتعديلها مثل printStandard.

```
74    // Display military format time: HH:MM
75    void Time::printMilitary() const
76    {
77       cout << ( hour < 10 ? "0" : "" ) << hour << ":"
78            << ( minute < 10 ? "0" : "" ) << minute;
79    }
80
81    // Display standard format time: HH:MM:SS AM (or PM)
82    void Time::printStandard()  // should be const
83    {
84       cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
85            << ( minute < 10 ? "0" : "" ) << minute << ":"
86            << ( second < 10 ? "0" : "" ) << second
87            << ( hour < 12 ? " AM" : " PM" );
88    }
89
90    // Attempting to access a const object with
91    // non-const member functions.
92    #include "time5.h"
93
94    int main()
95    {
96       Time wakeUp( 6, 45, 0 );       // non-constant object
97       const Time noon( 12, 0, 0 );   // constant object
98
99                                 // MEMBER FUNCTION   OBJECT
100      wakeUp.setHour( 18 );     // non-const         non-const
101
102      noon.setHour( 12 );       // non-const         const
103
104      wakeUp.getHour();         // const             non-const
105
106      noon.getMinute();         // const             const
107      noon.printMilitary();     // const             const
108      noon.printStandard();     // non-const         const
109      return 0;
110   }
```

أخطاء سيولدها المترجم.

خرج البرنامج:

```
Compiling...
Fig07_01.cpp
d:fig07_01.cpp(14) : error C2662: 'setHour' : cannot
convert 'this' pointer from 'const class Time' to
'class Time &'
Conversion loses qualifiers
d:\fig07_01.cpp(20) : error C2662: 'printStandard' :
cannot convert 'this' pointer from 'const class Time'
to 'class Time &'
Conversion loses qualifiers
Time5.cpp
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

الصفوف المركبة

رأيت حتى الآن الصفوف التي تكون أنماط معطياتها الأعضاء أنماطاً بسيطة. ولما كنا نستخدم الأغراض للتوسع في الأنماط، فيمكننا بناء صفوف تكون أنماط معطياتها الأعضاء صفوفاً أيضاً. في هذه الحالة:

- يمتلك الصف أعضاء هي أغراض من صفوف أخرى.
- يسمى الصف الذي ينتمي إليه الغرض العضو صفاً مزوداً **supplier**.
- يسمى الصف الذي يحتوي الأغراض الأعضاء صفاً زبوناً **client** أو صفاً مضيفاً **host**.

بناء الأغراض المركبة

ينجز الباني في الصفوف المركبة باستخدام قائمة القيم الابتدائية حصراً. ولكن عند استدعاء الباني للحصول على غرض مركب، كيف تبنى الأغراض المركبة؟ ما الذي يبدأ بناؤه الغرض المركب أم أعضاؤه الأغراض التي تحتاج بدورها إلى بان؟ في الواقع ، تبنى الأغراض الأعضاء قبل الغرض المضيف، وذلك بترتيب التصريح عنها في الصف لا بترتيب قائمة القيم الابتدائية للأعضاء في الباني. أما عند الهدم، فتهدم الأغراض المضيفة، ثم تهدم الأغراض الأعضاء:

- تستدعى البواني من الداخل للخارج.

- تستدعى الهوادم من الخارج للداخل.

مثال

الصف **Employee** يتضمن حقول من النمط **Date**

```
class Employee    // maintain employee information
{
      private:
        // String (ID) and Date are supplier classes
        String name;                // employee ID (Supplier)
        Date birthDate;             // date of birth (Supplier)
         Date hireDate;             // date of hire (Supplier)
      public:
            . . .
};
```

```
Employee::Employee( char *fname ,
                    int bmonth, int bday, int byear,
                    int hmonth, int hday, int hyear)
                  : birthDate( bmonth, bday, byear),
                    hireDate( hmonth, hday, hyear)
```

مثال

```
1     // date1.h
2     // Declaration of the Date class.
3     // Member functions defined in date1.cpp
4     #ifndef DATE1_H
5     #define DATE1_H
6
7     class Date {
8     public:
9        Date( int = 1, int = 1, int = 1900 ); // default constructor
10       void print() const;  // print date in month/day/year format
11       ~Date();  // provided to confirm destruction order
12    private:
13       int month;  // 1-12
14       int day;    // 1-31 based on month
15       int year;   // any year
16
17       // utility function to test proper day for month and year
18       int checkDay( int );
19    };
20
21    #endif
```

```
22    // date1.cpp
23    // Member function definitions for Date class.
24    #include <iostream>
25
26    using std::cout;
27    using std::endl;
28
29    #include "date1.h"
30
31    // Constructor: Confirm proper value for month;
32    // call utility function checkDay to confirm proper
33    // value for day.
34    Date::Date( int mn, int dy, int yr )
35    {
36       if ( mn > 0 && mn <= 12 )       // validate the month
37          month = mn;
38       else {
39          month = 1;
40          cout << "Month " << mn << " invalid. Set to month 1.\n";
41       }
42
43       year = yr;                      // should validate yr
44       day = checkDay( dy );           // validate the day
45
46       cout << "Date object constructor for date ";
47       print();         // interesting: a print with no
48       cout << endl;
49    }
50
51    // Print Date object in form  month/day/year
52    void Date::print() const
53       { cout << month << '/' << day << '/' << year; }
54
55    // Destructor: provided to confirm destruction order
56    Date::~Date()
57    {
58       cout << "Date object destructor for date ";
59       print();
60       cout << endl;
61    }
62
63    // Utility function to confirm proper day value
64    // based on month and year.
65    // Is the year 2000 a leap year?
66    int Date::checkDay( int testDay )
67    {
68       static const int daysPerMonth[ 13 ] =
69          {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
71       if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
72          return testDay;
73
74       if ( month == 2 &&      // February: Check for leap
75            testDay == 29 &&
76            ( year % 400 == 0 ||
77             ( year % 4 == 0 && year % 100 != 0 ) ) )
78          return testDay;
```

```
79
80       cout << "Day " << testDay << " invalid. Set to day 1.\n";
81
82       return 1;  // leave object in consistent state if bad
83    }
84    // emply1.h
85    // Declaration of the Employee class.
86    // Member functions defined in emply1.cpp
87    #ifndef EMPLY1_H
88    #define EMPLY1_H
89
90    #include "date1.h"
91
92    class Employee {
93    public:
94       Employee( char *, char *, int, int, int, int, int, int);
95       void print() const;
96       ~Employee();  // provided to confirm destruction order
97    private:
98       char firstName[ 25 ];
99       char lastName[ 25 ];
100      const Date birthDate;
101      const Date hireDate;
102   };
103
104   #endif
105   // emply1.cpp
106   // Member function definitions for Employee class.
107   #include <iostream>
108
109   using std::cout;
110   using std::endl;
111
112   #include <cstring>
113   #include "emply1.h"
114   #include "date1.h"
115
116   Employee::Employee( char *fname, char *lname,
117                       int bmonth, int bday, int byear,
118                       int hmonth, int hday, int hyear )
119      : birthDate( bmonth, bday, byear ),
120        hireDate( hmonth, hday, hyear )
121   {
122      // copy fname into firstName and be sure that it fits
123      int length = strlen( fname );
124      length = ( length < 25 ? length : 24 );
125      strncpy( firstName, fname, length );
126      firstName[ length ] = '\0';
127
128      // copy lname into lastName and be sure that it fits
129      length = strlen( lname );
130      length = ( length < 25 ? length : 24 );
131      strncpy( lastName, lname, length );
132      lastName[ length ] = '\0';
133
134      cout << "Employee object constructor: "
135           << firstName << ' ' << lastName << endl;
```

```
136   }
137
138   void Employee::print() const
139   {
140      cout << lastName << ", " << firstName << "\nHired: ";
141      hireDate.print();
142      cout << "  Birth date: ";
143      birthDate.print();
144      cout << endl;
145   }
146
147   // Destructor: provided to confirm destruction order
148   Employee::~Employee()
149   {
150      cout << "Employee object destructor: "
151           << lastName << ", " << firstName << endl;
152   }
153
154   // Demonstrating composition: an object with member objects.
155   #include <iostream>
156
157   using std::cout;
158   using std::endl;
159
160   #include "emply1.h"
161
162   int main()
163   {
164      Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
165
166      cout << '\n';
167      e.print();
168
169      cout << "\nTest Date constructor with invalid values:\n";
170      Date d( 14, 35, 1994 );  // invalid Date values
171      cout << endl;
172      return 0;
173   }
```

التابع print هو تابع ثابت const وسيطبع كلما جرى بناء غرض Date أو هدمه. يستطيع طباعة غرض ثابت لأنه تابع ثابت.

نحمل فقط emply.h لأن هذا الملف يحمل بدوره date.h .

خرج البرنامج:

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Jones

Jones, Bob
Hired: 3/12/1988  Birth date: 7/24/1949

Test Date constructor with invalid values:
Month 14 invalid. Set to month 1.
Day 35 invalid. Set to day 1.
Date object constructor for date 1/1/1994

Date object destructor for date 1/1/1994
Employee object destructor: Jones, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

تدريب:

ورد في المثال السابق استخدام بعض التوابع الخاصة بالتعامل مع مصفوفات من النمط **char** مثل:**strlen** لتحديد طول المصفوفة و **strncpy** لنسخ مصفوفة على مصفوفة أخرى. أعد كتابة المثال والصفوف باستخدام النمط **string**.

التوابع الصديقة friend function والصفوف الصديقة friend class

مر معك أن التوابع الأعضاء هي فقط التي يحق لها النفاذ إلى المعطيات الأعضاء. وأن كبسلتها معاً ضمن الصف هو الذي يؤمن خاصية إخفاء المعلومات هذه.

غير أنه في بعض الحالات نجد حاجة لخرق هذه القاعدة وبشروط خاصة جداً. نستخدم لهذا الغرض التوابع والصفوف الصديقة.

يمكن أن تنفذ التوابع الصديقة والصفوف الصديقة إلى الأعضاء الخاصة **private** و المحمية **protected** في صف آخر. ولا تكون التوابع الصديقة توابع أعضاء في الصف، كما أنها تعرف خارج مدى الصف.

خصائص الصداقة

- الصداقة تمنح ولا تؤخذ
- الصداقة ليست علاقة متناظرة (إذا كان B صديق A فليس بالضرورة أن يكون A صديق B)
- ليست علاقة متعدية (إذا كان A صديق B، B صديق C فليس بالضرورة أن يكون A صديق C)

التصريح عن صديق friend، التوابع الصديقة

- تضاف الكلمة **friend** قبل ترويسة التابع في الصف الذي يمنح الصداقة.

أي أننا نكتب في الصف الذي يمنح الصداقة لتابع ما تصريحاً عن هذه الصداقة على النحو التالي:

```
friend int myFunction(int x);
```

- تضاف الكلمة **friend** قبل كلمة **class** في الصف الذي يمنح الصداقة.
- إذا كان الصف **ClassOne** يمنح الصداقة لـ **ClassTwo** يكون لدينا مايلي:

```
friend class ClassTwo;
```

في تعريف الصف **ClassOne**

مثال

```
1
2     // Friends can access private members of
3     #include <iostream>
4
5     using std::cout;
6     using std::endl;
7
8     // Modified Count class
9     class Count {
10       friend void setX( Count &, int ); // friend declaration
11    public:
12       Count() { x = 0; }                // constructor
13       void print() const { cout << x << endl; }  // output
14    private:
15       int x;  // data member
16    };
17
18    // Can modify private data of Count because
19    // setX is declared as a friend function of Count
20    void setX( Count &c, int val )
21    {
22       c.x = val;  // legal: setX is a friend of Count
23    }
24
25    int main()
26    {
27       Count counter;
28
29       cout << "counter.x after instantiation: ";
30       counter.print();
31       cout << "counter.x after call to setX friend function:
";
32       setX( counter, 8 );  // set x with a friend
33       counter.print();
34       return 0;
35    }
```

setX هو friend للصف Count (يمكنه النفاذ للأعضاء الخاصة في Count)

setX يعرف بشكل عادي وهو ليس عضواً في Count)

يسمح بتعديل متحول private.

خرج البرنامج:

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

مثال

```
1
2     // Non-friend/non-member functions cannot access
3     // private data of a class.
4     #include <iostream>
5
6     using std::cout;
7     using std::endl;
8
9     // Modified Count class
10    class Count {
11    public:
12       Count() { x = 0; }                    // constructor
13       void print() const { cout << x << endl; }  // output
14    private:
15       int x;  // data member
16    };
17
18    // Function tries to modify private data of Count,
19    // but cannot because it is not a friend of Count.
20    void cannotSetX( Count &c, int val )
21    {
22       c.x = val;  // ERROR: 'Count::x' is not accessible
23    }
24
25    int main()
26    {
27       Count counter;
28
29       cannotSetX( counter, 3 ); // cannotSetX is not a friend
30       return 0;
```

cannotSetX ليس تابعاً صديقاً. لا يمكنه النفاذ للأعضاء الخاصة في Count

cannotSetX يحاول تعديل متحول خاص...

(private:
int x;)

خطأ متوقع للمترجم

cannot access private data

```
Compiling...
Fig07_06.cpp
D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\Fig07_06.c
pp(22) :
   error C2248: 'x' : cannot access private member
declared in
   class 'Count'
        D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\
        Fig07_06.cpp(15) : see declaration of 'x'
Error executing cl.exe.

test.exe - 1 error(s), 0 warning(s)
```

زيادة تحميل المؤثرات

قد يكون استدعاء التوابع مزعجاً في بعض أنواع الصفوف لاسيما الرياضية. فلو فرضنا وجود تابع لجمع عددين وتابع لإسناد قيمة إلى متحول، فلا بد أنك تلاحظ الفرق في وضوح العملية المطلوبة بين السطرين التاليين:

```
a = 2 + 3;
assign(a, add(2,3));
```

وفي **C++** مع تعريف أنماط جديدة، نرغب لو أننا نستطيع
استخدام الرموز المعروفة في اللغة لتحقيق بعض العمليات بمعناها المألوف على الأغراض التي تنتمي للصفوف الجديدة، كجمع عددين عقديين (بعد التصريح عن الصف الذي يعرف الأعداد العقدية) و جمع مصفوفتين من الأعداد الصحيحة (بعد التصريح عن الصف الذي يعرف مصفوفات الأعداد الصحيحة).

سنتناول المؤثرات الأصيلة في **C++** إذاً (**C++'s built-in operators**) للتعامل مع هذا النوع من الصفوف.

تتوفر في **C++** إمكانات تسمح للمبرمج بزيادة تحميل بعض المؤثرات المعرفة سابقاً. وفيما يلي قائمة المؤثرات التي يمكن زيادة تحميلها في **C++**.

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	<-	[ ]	( )	new	Delete
new []	Delete[]						

يمكن زيادة تحميل المؤثرات "+" و "-" و "*" و "&" بشكلها الأحادي أو الثنائي. أما المؤثرات التي لا يمكن زيادة تحميلها فهي:

::	.*	.	?:	sizeof

ولئن كانت زيادة تحميل المؤثرات ممكنة، فإنه لا يمكن تغيير قواعد الأولوية الخاصة بها. كذلك الأمر فيما يتعلق بالخاصة التجميعية للمؤثرات، وبعدد المعاملات الخاصة بها إذ لا يمكن تعديلها. ثُم إنه لا يمكن بناء مؤثرات جديدة.

كيفية زيادة تحميل المؤثرات وشروطها

- اكتب تعريف التابع بشكل عادي
- يتألف اسم التابع من الكلمة المفتاحية **operator** يليها رمز العملية التي يجري زيادة تحميلها فمثلاً لزيادة تحميل مؤثر الجمع (+) نعرف التابع **operator+**
- لايمكن اختراع مؤثرات جديدة بل تُستخدم المؤثرات المتوفرة فقط
- إن محاولة زيادة تحميل المؤثرات للأنماط الأصيلة في اللغة يؤدي إلى خطأ نحوي **syntax error** (لايمكنك أن تعيد تعريف عملية جمع عددين صحيحين).
- لايمكن تغيير عدد الأطراف التي يؤثر فيها المؤثر
- يمكن أن تكون توابع المؤثرات توابع أعضاء أو توابع غير أعضاء في الصف

توابع المؤثرات أعضاء في الصف أم توابع صديقة؟

- توابع المؤثرات كتوابع أعضاء
  - يجب أن يكون العنصر الأيسر غرضاً (أو مرجعاً على غرض) من الصف.
  - إذا كان العنصر الأيسر من نمط آخر، فيجب أن لا يكون تابع المؤثر تابعاً عضواً
  - في حال لم يكن تابع المؤثر عضواً، فيجب أن يكون صديقاً **friend** وذلك في حال النفاذ المباشر إلى معطيات خاصة أو محمية في هذا الصف.

- تسمح توابع المؤثرات التي يجري زيادة تحميلها والتي لم تعتبر توابع أعضاء بجعل المؤثرات تبديلية.

```
HugeInteger bigInteger;
int integer;
bigInteger = integer + bigInteger; //or
bigInteger = biginteger + integer;
```

زيادة تحميل المؤثرين الخاصين بالإقحام والاستخراج في الفئوات

يجب أن يمتلك التابعان << و>> عند زيادة تحميلهما طرفاً أيسر من نمط **istream &** و **ostream &** على الترتيب .

- يجب أن لايكون هذين التابعين عضوين في الصف (لأن الطرف الأيسر ليس غرضاً من الصف).
- يجب أن يكون هذان التابعان صديقين للصف إذا كانا يتضمنان نفاذاً إلى المعطيات الخاصة.

تطبيق

```
1
2     // Overloading the stream-insertion and
3     // stream-extraction operators.
4     #include <iostream>
5
6     using std::cout;
7     using std::cin;
8     using std::endl;
9     using std::ostream;
10    using std::istream;
11
12    #include <iomanip>
13
14    using std::setw;
15
16    class PhoneNumber {
17       friend ostream &operator<<( ostream&, const PhoneNumber &);
18       friend istream &operator>>( istream&, PhoneNumber & );
19
20    private:
21       char areaCode[ 4 ];  // 3-digit area code and null
22       char exchange[ 4 ];  // 3-digit exchange and null
23       char line[ 5 ];      // 4-digit line and null
24    };
25
26    // Overloaded stream-insertion operator (cannot be
27    // a member function if we would like to invoke it with
28    // cout << somePhoneNumber;).
29    ostream &operator<<( ostream &output, const PhoneNumber &num)
30    {
31       output << "(" << num.areaCode << ") "
32              << num.exchange << "-" << num.line;
33       return output;     // enables cout << a << b << c;
34    }
35
36    istream &operator>>( istream &input, PhoneNumber &num )
37    {
```

```
38       input.ignore();                      // skip (
39       input >> setw( 4 ) >> num.areaCode; // input area code
40       input.ignore( 2 );                   // skip ) and space
41       input >> setw( 4 ) >> num.exchange; // input exchange
42       input.ignore();                      // skip dash (-)
43       input >> setw( 5 ) >> num.line;     // input line
44       return input;      // enables cin >> a >> b >> c;
45    }
46
47    int main()
48    {
49       PhoneNumber phone; // create object phone
50
51       cout << "Enter phone number in the form (123) 456-7890:\n”;
52
53       // cin >> phone invokes operator>> function by
54       // issuing the call operator>>( cin, phone ).
55       cin >> phone;
56
57       // cout << phone invokes operator<< function by
58       // issuing the call operator<<( cout, phone ).
59       cout << "The phone number entered was: " << phone <<endl;
60       return 0;
61    }
```

خرج البرنامج:

```
Enter phone number in the form (123) 456-7890:
(800) 555-1212
The phone number entered was: (800) 555-1212
```

تدريب:

1. يطلب بناء صف يمثل النمط عدد كسري **Rational** لإجراء حسابات على الأعداد الكسرية. اكتب تعريف الصف وقم بزيادة مؤثري الجمع والضرب و المؤثرين << و >>. اكتب برنامجاً لاختبار الصف.
2. يطلب بناء صف يمثل النمط عدد عقدي **Complex** لإجراء حسابات على الأعداد العقدية. اكتب تعريف الصف وقم بزيادة مؤثري الجمع والضرب و المؤثرين << و >>. اكتب برنامجاً لاختبار الصف.
3. يطلب بناء صف يمثل النمط مستطيل **Rectangle** الذي مر ذكره في المثال الوارد في الفقرة

تنجيز الباني بحيث يمكن القيام بالعمليات الواردة في المثال: حساب المحيط، حساب المساحة، الحصول على الطول الحصول على العرض، تعديل البعدين. كما يطلب تزويد الصف بتوابع << و >> . يطلب كتابة التعريف والتنجيز في ملفين منفصلين.