類別：深入探討 (上) 9

My object all sublime
I shall achieve in time.
—W. S. Gilbert
Is it a world to hide virtues in?
—William Shakespeare
Don't be "consistent," but be simply true.
—Oliver Wendell Holmes, Jr.

學習目標

在本章中，你將學到：

- 如何使用前置處理包裝，避免多次含括同一標頭檔。
- 了解類別使用域(class scope)，以及如何透過物件名稱、參照、或指標存取類別成員。
- 運用預設引數定義建構子(constructor)。
- 了解解構子 (destructor) 在物件摧毀之前進行的資源回收工作。
- 了解建構子和解構子被呼叫的時機及順序。
- 由 `public` 成員函式傳回指向 `private` 資料的參照時，產生的邏輯錯誤。
- 如何透過預設情況下的逐成員賦值 (memberwise assignment)，把物件的資料成員設給另一個物件。

本章綱要

9.1 簡介

本書進行至此，我們已經在前面的章節帶領讀者了解許多 C++物件導向程式設計的基本名詞和概念。我們談到了程式設計的一般方法：即選定各類別的屬性和行爲，再讓物件與 C++標準函式庫的類別合作，以完成指定的工作。

本章將對類別進行更進一步的討論。我們將在本章和第 10 章使用 Time 這個類別爲例子，說明和類別建構有關的主題。首先，我們將用類別 Time 複習前面章節介紹過的一些功能，並介紹 C++中一個重要的概念-如何運用前置處理包裝 (preprocessor wrapper)，防止標頭檔在同一個原始檔中被含括多次。類別的定義只能出現一次，故此技巧可避免重複定義的錯誤。

接下來，我們會討論類別使用域 (class scope) 與類別成員之間的關係。我們將示範類別的客戶程式——即使用類別的程式；可用來存取類別 public 成員的三種方式，即透過物件名稱、透過物件的參照或指向物件的指標。透過物件和物件參照時，用的是點號 (.) 成員選擇運算子；透過指標時，用的是箭號 (->) 成員選擇運算子。

本章接下來會討論可以讀取或顯示物件成員資料的存取函式 (access functions)。存取函式常見的用途之一，是測試條件的眞僞，因此又稱爲判斷函式 (predicate functions)。我們還會介紹工具函式 (utility functions，又稱助手函式 helper functions)；這種函式是類別的 private 成員函式，用來幫助類別的 public 函式完成工作，使用類別的外部程式無法直接呼叫它們。

在類別 Time 的第二個例子中，我們示範如何將引數傳入建構子 (constructor)，以及如何在建構子中使用預設引數，用各式引數爲物件進行初始化。接下來，我們要討論一種稱爲解構子 (destrcutor) 的特殊成員函式，用來在物件被摧毀之前進行資源回收工作。我們還會說明建構子與解構子呼叫的順序，因爲物件被使用時，必須已正確初始化且還沒被摧毀，程式才能正確運作。

類別 Time 的最後一個例子，說明當成員函式傳回 private 資料的參照時，可能會帶來的危險。這麼做會破壞物件的封裝 (encapsulation)，讓外部程式得以直接存取物件的資料。這個例子還會說明，相同類別的物件可以藉逐成員賦值 (memberwise assignment) 彼此設值，讓等號右邊的物件成員逐一設給左邊物件的成員。最後，我們以有關軟體再利用性的討論做爲本章的結束。

9.2　案例研究：類別 Time

我們藉第一個例子 (圖 9.1-9.3) 建立類別 Time，以及用來測試該類別的程式。到目前爲止，本書已經帶領讀者建立了許多類別。在這一節裡，我們要複習第 3 章討論過的許多概念，並介紹 C++軟體工程中一項重要的觀念，也就是在標頭檔中使用「前置處理包裝」(preprocessor wrapper)，避免在原始檔中重複含括同一個標頭檔。類別的定義只能出現一次，故此技巧可避免重複定義的錯誤。

```
1  // Fig. 9.1: Time.h
2  // Declaration of class Time.
3  // Member functions are defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time class definition
10 class Time
11 {
12 public:
13    Time(); // constructor
14    void setTime( int, int, int ); // set hour, minute and second
15    void printUniversal(); // print time in universal-time format
16    void printStandard(); // print time in standard-time format
17 private:
18    int hour; // 0 - 23 (24-hour clock format)
19    int minute; // 0 - 59
20    int second; // 0 - 59
21 }; // end class Time
22
23 #endif
```

圖 9.1　Time 類別的定義

類別 Time 的定義

類別 Time 的定義在圖 9.1 中，包括成員函式 Time、setTime、printUniversal 和 printStandard 的原型 (第 13-16 行)，以及 hour、minute 和 second 等整數 priviate 成員 (第 18-20 行)。只有類別的四個成員函式才能存取它們。第 12 章討論繼承 (inheritance) 和它在物件導向程式設計中所扮演的角色時，我們會介紹第三種成員存取修飾字 protected。

良好的程式設計習慣 9.1

爲使程式碼更明確易讀，通常在類別的定義中，每個成員存取修飾字只會出現一次，並請將 public 成員放在最前面，這樣比較容易找到。

軟體工程的觀點 9.1

類別的函式與資料成員的權限應爲 private，除非該元件的確需要 public 權限。這是最小權限原則的另一個例子。

在圖 9.1 中，類別定義包含在**前置處理包裝** (**preprocessor wrapper**，第 6、7 和 23 行) 之中，如下所示：

```
// prevent multiple inclusions of header file
#ifndef TIME_H
#define TIME_H
   ...
#endif
```

建構較大型的程式時，通常會將其它定義和宣告也放到標頭檔中。如果程式已經定義過 TIME_H，則以上前置處理指令不會重複含括**#ifndef** (意思爲「如果尚未定義」) 和 **#endif** 之間的程式碼。如果某個檔案先前未含括過此標頭檔，則**#define** 指令會定義 TIME_H，並含括標頭檔中的敘述。如果程式先前已經加入標頭檔，因爲 TIME_H 已於上次含括時定義，所以不會再次含括。大型的程式通常有好幾個標頭檔，甚至標頭檔本身還含括其他的標頭檔，因此很容易造成重複含括的狀況。[請注意：使用前置處理指令時，我們常讓符號常數的名稱和標頭檔案名稱很像，但使用大寫字母，並以 "_" 符號取代 "." 符號。]

測試和除錯的小技巧 9.1

利用#ifndef、#define 和#endif 等前置處理指令，撰寫前置處理包裝，可避免程式重複含括標頭檔。

良好的程式設計習慣 9.2

請在標頭檔的前置處理指令#ifndef 和#define 中，使用大寫的標頭檔名稱，並將 "." 置換成 "_"。

類別 Time 的成員函式

在圖 9.2 中，建構子 Time (程式第 10-13 行) 會將所有資料成員的初始值設爲 0 (相當於通用格式時間的 12 AM)，確保物件一開始就處於已定義的狀態。建立 Time 物件時，程式會自動呼叫建構子，而所有對資料成員的修改均需透過 setTime 函式來執行，故 Time 物件的資料成員不可能爲無效值。值得一提的是，程式設計師可以爲類別定義不同的多載建構子 (overloaded constructor)。

請注意，類別的資料成員宣告於於類別主體之內，但不能在類別的主體內設定初始值。我們強烈建議讀者在類別的建構子中設定它們的初始值，因爲 C++不會自動設定基本型別資料成員的初始值。我們也可以透過屬於 Time 類別，以 set 開頭的函式賦值給這些資料成員。[請注意：第 10 章會談到，只有類別的 static const 整數型別或 enum 型別資料成員，才能在類別的主體中設定初始值。]

```
1  // Fig. 9.2: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  #include <iomanip>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  // Time constructor initializes each data member to zero.
9  // Ensures all Time objects start in a consistent state.
10 Time::Time()
11 {
12    hour = minute = second = 0;
13 } // end Time constructor
14
15 // set new Time value using universal time; ensure that
16 // the data remains consistent by setting invalid values to zero
17 void Time::setTime( int h, int m, int s )
18 {
19    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
20    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
21    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
22 } // end function setTime
23
24 // print Time in universal-time format (HH:MM:SS)
```

圖 9.2　Time 成員函式的定義

```
25  void Time::printUniversal()
26  {
27     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
28        << setw( 2 ) << minute << ":" << setw( 2 ) << second;
29  } // end function printUniversal
30
31  // print Time in standard-time format (HH:MM:SS AM or PM)
32  void Time::printStandard()
33  {
34     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
35        << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
36        << second << ( hour < 12 ? " AM" : " PM" );
37  } // end function printStandard
```

圖 9.2 Time 成員函式的定義 (續)

常見的程式設計錯誤 9.1

嘗試在類別的定義中，明確設定非 static 資料成員的初始值，會造成語法錯誤。

函式 setTime (第 17-22 行) 是一個 public 函式，接收三個 int 參數，用它們來設定時間。該函式用條件運算式測試每一個引數，判斷它們的值是否符合指定的範圍。例如，參數 hour 的值 (第 19 行) 必須大於等於 0 且小於 24，因爲在通用時間格式中，時的範圍必須爲 0 到 23 間的整數 (例如：1 PM 以 13 時表示，11 PM 以 23 時表示，0 表示午夜，12 表示中午)。同樣的，minute 和 second 的值 (第 20 及 21 行) 也必須大於等於 0 且小於 60。範圍不合的值會設定爲 0，因此即使傳給 setTime 不正確的參數，Time 物件仍會包含合法 (consistent) 的資料。在這個例子中，數值 0 對 hour、minute 和 second 而言是合法值 (consistent value)。

傳入 setTime 的值，只要在要用來設值的成員的合法範圍內，就稱爲是正確值 (correct value)。因此，對 hour 而言，0-23 間的值均正確。正確值必爲合法值，但合法值不一定是正確值。例如，如果接收的引數超過範圍，setTime 會把 hour 設成 0，但除非現在時間剛好是午夜，否則 hour 的值並不正確。

圖 9.2 第 25-29 行的 printUniversal 函式不接收任何引數，以通用時間格式印出時間，此格式爲三個以冒號隔開的數字，分別代表時、分和秒。舉例來說，假設時是 1:30:07 PM，函式 printUniversal 會印出 13:30:07。我們在第 27 行使用參數化串流操作子 **setfill** 設定**填充字元 (fill character)**，當印出的數字位數小於輸出欄位時，用這個字元把欄位填滿。在預設情況下，填充字元會出現在數字的左邊。以這個例子來說，如果 minute 的值爲 2，會顯示成 02，因爲填充字元設定成零 ('0')。如果數字的位數夠多，就不會顯示填充字元。一但使用 setfill，其後所有輸出欄位都會使用該函式所設

定的填充字元；這稱爲粘著性設定 (sticky setting)。另一個操作子 `setw` 就只會對下一個欄位作用；這稱爲非粘著性設定 (non-sticky setting)。

測試和除錯的小技巧 9.2

各種粘著性設定 (如填充字元或浮點數精準度) 使用完畢後，應把它恢復爲原來的設定，否則將會讓後面的程式產生不正確的輸出格式。我們會在第 15 章「串流輸入與輸出」中討論如何重置填充字元和浮點數精準度的設定。

第 32-37 行的函式 `printStandard` 不接收任何引數，將日期按標準時間格式印出，包括 `hour` (時)、`minute` (分) 和 `second` (秒) 的值，中間以冒號隔開，最後加上 AM 或 PM 表示上午或下午 (如 13:27:06 PM)。和函式 `printUniversal` 一樣，函式 `printStandard` 透過 `setfill ('0')` 用兩位數字印出 `minute` 和 `second` 的值，必要時在數值前面補零。第 34 行用條件運算子 (`?:`) 決定如何顯示 `hour` 的值－若其值爲 0 或 12 (無論是 AM 或 PM)，都顯示 12；否則以 1 至 11 顯示之。第 36 行的條件運算式決定要顯示 AM 或 PM。

在類別定義之外定義成員函式：類別使用域

即使我們在類別的定義之內宣告成員函式，將其定義置於類別的主體之外，並透過二元使用域解析運算子結合二者，成員函式仍屬於**類別使用域 (class scope)**；意思是說，只有此類別的其它成員能夠直接使用它的名稱，否則，程式必須透過此類別的物件、參照此類別的物件或指向此類別物件的指標，或是二元使用域解析運算子，才能夠取用它的名稱。我們稍後會更深入討論類別使用域的議題。

成員函式若定義於類別定義中，C++編譯器會嘗試將該成員函式變成行內函式。請記住，編譯器有權決定是否要讓該函式成爲行內函式。

增進效能的小技巧 9.1

請將小型的成員函式定義放在類別定義之中，若編譯器許可，就可讓此函式成爲行內函式，提高程式的效能。

軟體工程的觀點 9.2

將小型的成員函式定義於類別定義的內部，對軟體工程的品質沒有幫助，因爲類別的使用者會看到函式的實作細節；且函式內容一旦變更，就需重新編譯程式，所以這並不是很好的軟體工程習慣。

軟體工程的觀點 9.3

在類別的標頭檔中，最好只定義最簡單和最穩定 (不太會變更實作) 的成員函式。

成員函式與全域函式

printUniversal 和 printStandard 這兩個成員函式沒有任何引數，這是因爲成員函式已經隱含知道所要印出的內容，即被呼叫的 Time 物件的資料成員。這麼做可讓成員函式呼叫，相較於程序化程式設計的傳統函式呼叫，顯得簡潔許多。

軟體工程的觀點 9.4

物件導向程式設計可以減少參數數目，因此可以簡化函式呼叫。物件導向程式設計的優點，來自於透過物件封裝資料成員和成員函式，且讓成員函式擁有存取資料成員的權限。

軟體工程的觀點 9.5

成員函式的程式碼和非物件導向程式設計的函式相較，通常較爲精簡，因爲儲存在資料成員的資料，已透過建構子或存放新資料的成員函式驗證過了。因爲資料已經存放在物件中，使用該資料的成員函式通常不需要使用引數；就算需要，也會比非物件導向程式設計的函式少。因此，函式呼叫，以及成員函式的原型和定義就比較精簡。這對程式發展的許多面向都很有幫助。

測試和除錯的小技巧 9.3

呼叫類別的成員函式時，通常不需使用引數，就算需要，通常也會比非物件導向程式語言中的傳統函式呼叫少，故傳遞錯誤引數、錯誤引數型別和錯誤引數個數的可能性大大降低。

使用類別 Time

完成類別 Time 的定義之後，就可把它用做物件、陣列和參照定義中的型別，如下所示：

```
Time sunset; // object of type Time
Time arrayOfTimes[ 5 ]; // array of 5 Time objects
Time &dinnerTime = sunset; // reference to a Time object
Time *timePtr = &dinnerTime; // pointer to a Time object
```

圖 9.3 使用類別 Time，其中第 10 行產生類別 Time 的一個物件，名爲 t。物件產生時，會呼叫建構子 Time，將每個 private 資料成員的初始值設爲 0。接下來，第 14 和 16 行分別將時間以通用和標準格式印出，確認資料成員已正確設定初始值。程式第 18 行透過呼叫成員函式 setTime 設定新的時間，且第 24 和 22 行再次以兩種格式列印

時間。第 26 行嘗試使用 `setTime` 把資料成員設爲無效的值，而函式 `setTime` 會發現這種情況，將無效值設爲 0，確保物件是在合法的狀態。最後，程式第 31 和 33 行再次以兩種格式列印時間。

```
1  // Fig. 9.3: fig09_03.cpp
2  // Program to test class Time.
3  // NOTE: This file must be compiled with Time.cpp.
4  #include <iostream>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  int main()
9  {
10    Time t; // instantiate object t of class Time
11
12    // output Time object t's initial values
13    cout << "The initial universal time is ";
14    t.printUniversal(); // 00:00:00
15    cout << "\nThe initial standard time is ";
16    t.printStandard(); // 12:00:00 AM
17
18    t.setTime( 13, 27, 6 ); // change time
19
20    // output Time object t's new values
21    cout << "\n\nUniversal time after setTime is ";
22    t.printUniversal(); // 13:27:06
23    cout << "\nStandard time after setTime is ";
24    t.printStandard(); // 1:27:06 PM
25
26    t.setTime( 99, 99, 99 ); // attempt invalid settings
27
28    // output t's values after specifying invalid values
29    cout << "\n\nAfter attempting invalid settings:"
30       << "\nUniversal time: ";
31    t.printUniversal(); // 00:00:00
32    cout << "\nStandard time: ";
33    t.printStandard(); // 12:00:00 AM
34    cout << endl;
35 } // end main
```

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

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

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

圖 9.3　　類別 `Time` 的測試程式

組合與繼承

我們不需完全從頭建立新的類別。類別可以包含其它類別的物件作爲其成員，或**衍生 (derived)** 自其它類別，讓其它類別提供新類別的屬性和行爲。這稱爲軟體再利用 (software reuse)，可大大提高程式設計者的生產力，並讓程式碼更容易維護。讓類別包含其它類別的物件作爲資料成員，稱爲**組合** (**composition**，又稱**聚合 aggregation**)，我們將在第 10 章討論這個主題。類別衍生自其它類別，稱爲**繼承 (inheritance)**，我們將在第 12 章討論。

物件大小

剛接觸物件導向程式設計的新手常會認爲，物件一定很大，因爲它包含資料成員和成員函式。邏輯上來說，這沒有錯：程式設計者很自然地會把物件想成是包含資料和函式的集合體 (特別是，我們的解說也許更增強了這個信念)，但這個觀念其實是錯的。

增進效能的小技巧 9.2

物件如果包含函式的話，的確是蠻大的；但它只包含資料，所以其實小很多。讀者只要對類別名稱，或屬於該類別的物件使用 `sizeof` 運算子就能發現，所傳回的值只是該類別的資料成員大小。編譯器只會爲整個類別建立一套成員函式，供該類別所有物件使用。每個物件包含的類別資料可能不同，故物件必須擁有屬於自己的資料副本。函式的程式碼是不可修改的，因此可讓類別的所有物件共用。

9.3 類別使用域及存取類別的成員

類別的資料成員 (於類別定義中宣告的變數) 和成員函式 (於類別定義中宣告的函式) 都屬於類別使用域 (class scope)；非成員函式屬於全域命名空間使用域 (global namespace scope)。

在類別使用域中，類別的所有成員可透過成員函式存取，且可透過名稱參照。在類別使用域之外，可透過物件的**代表 (handle)** 參照類別的 `public` 成員，如物件名稱、以及指向物件的參照或指標。物件、參照或指標的型別提供介面 (即成員函式)，供外部程式使用。[在第 10 章中會談到，每當在物件內部參照資料成員或成員函式時，編譯器會在程式碼中自動插入隱含的 (implicit) 代表。

我們可對類別的成員函式進行多載，但多載函式也必須爲該類別的成員函式。要多載某成員函式，只需在類別定義中列出每個多載函式的原型，並提供它們個別的定義。

在成員函式中宣告的變數具有區域使用域，只能在該函式中使用。如果某成員函式定義了一個變數，其名稱與另一個類別使用域的變數名稱相同，則在此函式使用域中，類別使用域的變數會被隱藏。此時，可以在變數前面放置類別名稱，以及使用域解析運算子 (::)，存取被隱藏的變數。被隱藏的全域變數可以透過單元使用域解析運算子存取 (請參閱第 6 章)。

將點號 (.) 成員選擇運算子 (member selction operator) 置於物件名稱或指向物件的參照之後，就可存取該物件的成員。**箭號成員選擇運算子 (->，arrow member selection operator)** 則可配合指向物件的指標使用，存取物件的成員。

圖 9.4 透過一個簡單的類別 Count (第 7-24 行) 示範如何透過成員選擇運算子存取類別的成員。該類別包含一個型別爲 int 的 private 資料成員 x (第 23 行)，一個 public 成員函式 setX (第 11-14 行) 以及 public 成員函式 print (第 17-20 行)。爲簡單起見，我們把這個類別和使用它的 main 函式放在同一個檔案之中。程式第 28-30 行建立三個和 Count 型別有關的變數，即 counter (Count 物件)、counterRef (指向 Count 物件的參照) 以及 counterPtr (指向 Count 物件的指標)。變數 counterRef 參照 counter，而變數 counterPtr 爲指向 counter 的指標。在第 33-34 行與第 37-38 行中，程式分別用點號 (.) 與箭號 (->) 成員選擇運算子，透過物件名稱 (counter) 和物件參照 (counterRef，即 counter 的別名) 呼叫成員函式 setX 和 print。同樣的，程式在第 41-42 行透過指標(countPtr) 和箭號 (->) 成員選擇運算子，呼叫成員函式 setX 和 print。

```
1  // Fig. 9.4: fig09_04.cpp
2  // Demonstrating the class member access operators . and ->
3  #include <iostream>
4  using namespace std;
5
6  // class Count definition
7  class Count
8  {
9  public: // public data is dangerous
10    // sets the value of private data member x
11    void setX( int value )
12    {
13       x = value;
14    } // end function setX
15
16    // prints the value of private data member x
17    void print()
```

圖 9.4　透過不同的物件代表 (物件名稱、物件參照和物件指標)，存取物件的資料成員和成員函式

```
18      {
19         cout << x << endl;
20      } // end function print
21
22   private:
23      int x;
24   }; // end class Count
25
26   int main()
27   {
28      Count counter; // create counter object
29      Count *counterPtr = &counter; // create pointer to counter
30      Count &counterRef = counter; // create reference to counter
31
32      cout << "Set x to 1 and print using the object's name: ";
33      counter.setX( 1 ); // set data member x to 1
34      counter.print(); // call member function print
35
36      cout << "Set x to 2 and print using a reference to an object: ";
37      counterRef.setX( 2 ); // set data member x to 2
38      counterRef.print(); // call member function print
39
40      cout << "Set x to 3 and print using a pointer to an object: ";
41      counterPtr->setX( 3 ); // set data member x to 3
42      counterPtr->print(); // call member function print
43   } // end main
```

```
Set x to 1 and print using the object's name: 1
Set x to 2 and print using a reference to an object: 2
Set x to 3 and print using a pointer to an object: 3
```

圖 9.4　透過不同的物件代表 (物件名稱、物件參照和物件指標)，存取物件的資料成員和成員函式 (續)

9.4　區分介面與實作

在第 3 章中，一開始我們把類別與成員函式的定義放在同一個檔案裡面；後來，我們示範如何把程式分成兩個檔案：分別是類別定義的標頭檔 (即類別的介面 interface)，以及成員函式定義的原始檔 (即類別的實作 implementation)。這麼做能讓程式更容易修改，因爲對使用該類別的外部程式而言，只要提供的界面不變，修改界面的實作對外部程式沒有任何影響。

軟體工程的觀點 9.6

類別的使用者使用類別時，不需用到其原始碼，但必須連結類別的目的碼 (即編譯過的類別)。這能鼓勵獨立軟體供應商 (ISVs) 銷售或授權使用它們發展的類別函式庫。ISV 只需在產品中提供標頭檔和目的檔模組，讓機密性的資訊不致外流；若提供原始碼就會有此風險。C++社群也因更多的軟體業者願意推出類別函式庫而獲益。

事實上，事情並非如此美好；因爲標頭檔還是會包含一部分的實作程式碼和其它提示。例如，行內成員函式必須置於標頭檔中，讓編譯器編譯外部程式時，直接在需要處插入行內成員函式的定義。標頭檔中的類別定義會列出類別的 `private` 成員；使用者雖無法存取它們，但仍能藉此得知成員的相關訊息。在第 10 章中，我們會介紹如何使用代理類別 (proxy class) 隱藏類別的 `private` 資料，避免類別的使用者獲取該資訊。

軟體工程的觀點 9.7

類別介面的重要資訊應該放在標頭檔中；只在類別內部使用，或使用者不需要的資訊，則應放在不公開的原始碼檔中。這是最小權限原則的另一個例子。

9.5　存取函式與工具函式

存取函式 (access functions) 可以用來讀取或顯示類別的資料。存取函式另一項常用的功能是用來測試條件式的眞僞，因此又稱爲**判斷函式 (predicate function)**。舉例來說，許多容器類別 (container class) 都有一個叫做 `isEmpty` 的成員函式；所謂容器類別，是指包含許多物件的類別，例如 `vector`。程式嘗試由容器物件讀取項目之前，會先以 `isEmpty` 測試此容器是否爲空的。同樣的，判斷函式 `isFull` 可用來測試容器類別物件是否沒有額外的空間存放新物件。以本章前面建構的 `Time` 類別來說，我們可加入 `isAM` 和 `isPM` 作爲判斷函式。

圖 9.5-9.7 的程式示範工具函式 (utility function，又稱爲**助手函式 helper function**)。工具函式不屬於類別的 `public` 介面；它們是支援類別其他 `public` 成員函式的 `private` 成員函式。工具函式不能讓使用類別的外部程式呼叫，但可由夥伴類別 (friend class) 使用；我們將在第 10 章討論這個問題。

類別 `SalesPerson` (圖 9.5) 宣告一個 12 個月份銷售圖的陣列 (第 17 行)，類別建構子、以及用來操作陣列的成員函式的原型。

```
1  // Fig. 9.5: SalesPerson.h
2  // SalesPerson class definition.
3  // Member functions defined in SalesPerson.cpp.
4  #ifndef SALESP_H
5  #define SALESP_H
6
7  class SalesPerson
8  {
9  public:
```

圖 9.5　`SalesPerson` 類別定義

```
10      static const int monthsPerYear = 12; // months in one year
11      SalesPerson(); // constructor
12      void getSalesFromUser(); // input sales from keyboard
13      void setSales( int, double ); // set sales for a specific month
14      void printAnnualSales(); // summarize and print sales
15   private:
16      double totalAnnualSales(); // prototype for utility function
17      double sales[ monthsPerYear ]; // 12 monthly sales figures
18   }; // end class SalesPerson
19
20   #endif
```

圖 9.5 SalesPerson 類別定義 (續)

在圖 9.6 中，SalesPerson 建構子 (第 9-13 行) 將 sales 陣列的初始值設爲零。程式在第 30-37 行透過 public 成員函式 setSales 設定陣列 sales 中一個月的銷售量。程式第 40-45 行用 public 成員函式 printAnnualSales 印出過去 12 個月的總銷售額。private 工具函式 totalAnnualSales (程式第 48-56 行) 則會加總最近 12 個月的每月銷售額，再用函式 printAnnualSales 印出。成員函式 printAnnualSales 會把銷售成績以適當金額表示法印出。

```
1   // Fig. 9.6: SalesPerson.cpp
2   // SalesPerson class member-function definitions.
3   #include <iostream>
4   #include <iomanip>
5   #include "SalesPerson.h" // include SalesPerson class definition
6   using namespace std;
7
8   // initialize elements of array sales to 0.0
9   SalesPerson::SalesPerson()
10  {
11     for ( int i = 0; i < monthsPerYear; i++ )
12        sales[ i ] = 0.0;
13  } // end SalesPerson constructor
14
15  // get 12 sales figures from the user at the keyboard
16  void SalesPerson::getSalesFromUser()
17  {
18     double salesFigure;
19
20     for ( int i = 1; i <= monthsPerYear; i++ )
21     {
22        cout << "Enter sales amount for month " << i << ": ";
23        cin >> salesFigure;
24        setSales( i, salesFigure );
25     } // end for
26  } // end function getSalesFromUser
```

圖 9.6 SalesPerson 的成員函式定義

```
27
28  // set one of the 12 monthly sales figures; function subtracts
29  // one from month value for proper subscript in sales array
30  void SalesPerson::setSales( int month, double amount )
31  {
32     // test for valid month and amount values
33     if ( month >= 1 && month <= monthsPerYear && amount > 0 )
34        sales[ month - 1 ] = amount; // adjust for subscripts 0-11
35     else // invalid month or amount value
36        cout << "Invalid month or sales figure" << endl;
37  } // end function setSales
38
39  // print total annual sales (with the help of utility function)
40  void SalesPerson::printAnnualSales()
41  {
42     cout << setprecision( 2 ) << fixed
43        << "\nThe total annual sales are: $"
44        << totalAnnualSales() << endl; // call utility function
45  } // end function printAnnualSales
46
47  // private utility function to total annual sales
48  double SalesPerson::totalAnnualSales()
49  {
50     double total = 0.0; // initialize total
51
52     for ( int i = 0; i < monthsPerYear; i++ ) // summarize sales results
53        total += sales[ i ]; // add month i sales to total
54
55     return total;
56  } // end function totalAnnualSales
```

圖 9.6　　SalesPerson 的成員函式定義 (續)

請注意，在圖 9.7 應用程式的函式 main 只進行一連串的成員函式呼叫，沒有使用任何的控制結構； 這是因爲 SalesPerson 類別的成員函式，已經完全封裝 sales 陣列的邏輯操作。

軟體工程的觀點 9.8

物件導向程式設計的特徵之一，就是定義類別後，建立和操作類別物件只需要呼叫一連串成員函式，很少需要使用控制結構。相對地，在類別的成員函式實作中，就常會使用控制結構。

```
1  // Fig. 9.7: fig09_07.cpp
2  // Utility function demonstration.
3  // Compile this program with SalesPerson.cpp
4
5  // include SalesPerson class definition from SalesPerson.h
6  #include "SalesPerson.h"
7
8  int main()
9  {
10    SalesPerson s; // create SalesPerson object s
11
12    s.getSalesFromUser(); // note simple sequential code; there are
13    s.printAnnualSales(); // no control statements in main
14 } // end main
```

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

圖 9.7　　示範使用工具函式

9.6　類別 Time 案例研究：建構子與預設引數

圖 9.8-9.10 的程式對類別 Time 進行強化，示範如何隱含 (implicit) 地將引數傳給建構子。圖 9.2 的建構子將 hour、minute 和 second 的初始值設爲 0 (即通用時間格式的午夜 12 點)。與一般函式相同，建構子可以擁有預設引數。圖 9.8 的第 13 行宣告 Time 建構子，其中包含預設引數，將每個引數的預設值設定爲 0。在圖 9.9 中，程式第 10-13 行定義 Time 建構子的新版本，接收 hr、min 和 sec 等參數，用來初始化 private 資料成員 hour、minute 和 second。類別 Time 爲每個資料成員提供名稱以 set 和 get 開頭的函式。建構子 Time 呼叫 setTime，而該函式呼叫 setHour、setMinute 和 setSecond 檢查和設定資料成員的值。建構子的預設引數能確保即使呼叫時未提供引數，建構子仍能爲類別物件的資料成員設定初始值，保持 Time 物件狀態的合法性。每個引數都有預設值的建構子又稱預設建構子 (default constrtutor)；這種建構子呼叫時不需任何引數。類別最多只能有一個預設建構子。

```
1  // Fig. 9.8: Time.h
2  // Time class containing a constructor with default arguments.
3  // Member functions defined in Time.cpp.
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time abstract data type definition
10 class Time
11 {
12 public:
13    Time( int = 0, int = 0, int = 0 ); // default constructor
14
15    // set functions
16    void setTime( int, int, int ); // set hour, minute, second
17    void setHour( int ); // set hour (after validation)
18    void setMinute( int ); // set minute (after validation)
19    void setSecond( int ); // set second (after validation)
20
21    // get functions
22    int getHour(); // return hour
23    int getMinute(); // return minute
24    int getSecond(); // return second
25
26    void printUniversal(); // output time in universal-time format
27    void printStandard(); // output time in standard-time format
28 private:
29    int hour; // 0 - 23 (24-hour clock format)
30    int minute; // 0 - 59
31    int second; // 0 - 59
32 }; // end class Time
33
34 #endif
```

圖 9.8　　具預設引數建構子的類別 `Time`

圖 9.9 中，建構子在第 12 行呼叫成員函式 `setTime`，以傳入建構子的值 (或預設值) 做爲其引數。函式 `setTime` 呼叫 `setHour` 確保 `hour` 的值在 0-23 之間，呼叫 `setMinute` 和 `setSecond` 確保 `minute` 與 `second` 的值在 0-59 之間。若超出範圍，則設爲 0，確保資料成員處於合法的狀態。在第 16 章「例外處理」中，我們會介紹遇到這種情況時，除了將資料成員的值設爲預設值外，還可以擲出例外通知使用者傳入的值超出範圍。

```
1  // Fig. 9.9: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  #include <iomanip>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  // Time constructor initializes each data member to zero;
9  // ensures that Time objects start in a consistent state
10 Time::Time( int hr, int min, int sec )
11 {
12    setTime( hr, min, sec ); // validate and set time
13 } // end Time constructor
14
15 // set new Time value using universal time; ensure that
16 // the data remains consistent by setting invalid values to zero
17 void Time::setTime( int h, int m, int s )
18 {
19    setHour( h ); // set private field hour
20    setMinute( m ); // set private field minute
21    setSecond( s ); // set private field second
22 } // end function setTime
23
24 // set hour value
25 void Time::setHour( int h )
26 {
27    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
28 } // end function setHour
29
30 // set minute value
31 void Time::setMinute( int m )
32 {
33    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
34 } // end function setMinute
35
36 // set second value
37 void Time::setSecond( int s )
38 {
39    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
40 } // end function setSecond
41
42 // return hour value
43 int Time::getHour()
44 {
45    return hour;
46 } // end function getHour
47
48 // return minute value
49 int Time::getMinute()
50 {
51    return minute;
52 } // end function getMinute
53
```

圖 9.9　類別 `Time` 的成員函式，其中包括接收引數的建構子

```
54  // return second value
55  int Time::getSecond()
56  {
57     return second;
58  } // end function getSecond
59
60  // print Time in universal-time format (HH:MM:SS)
61  void Time::printUniversal()
62  {
63     cout << setfill( '0' ) << setw( 2 ) << getHour() << ":"
64        << setw( 2 ) << getMinute() << ":" << setw( 2 ) << getSecond();
65  } // end function printUniversal
66
67  // print Time in standard-time format (HH:MM:SS AM or PM)
68  void Time::printStandard()
69  {
70     cout << ( ( getHour() == 0 || getHour() == 12 ) ? 12 : getHour() % 12 )
71        << ":" << setfill( '0' ) << setw( 2 ) << getMinute()
72        << ":" << setw( 2 ) << getSecond() << ( hour < 12 ? " AM" : " PM" );
73  } // end function printStandard
```

圖 9.9　類別 Time 的成員函式，其中包括接收引數的建構子 (續)

其實，我們可以把 Time 建構子的程式碼寫成直接包含成員函式 setTime 的程式碼，甚至直接包括 setHour、setMinute 和 setSecond 的敘述。直接在建構子中呼叫 setHour、setMinute 和 setSecond 的效率較高，因爲不用透過 setTime。同樣的，若直接把第 27、33 和 39 行的程式碼複製到建構子中，就可完全省去呼叫 setTime、setHour、setMinute 和 setSecond 的額外負擔。但是當我們將第 27、33 和 39 行的程式複製到建構子或成員函式中時，會讓程式碼變得不容易維護。若修改 setHour、setMinute 和 setSecond 的實作，每一個複製第 27、33 和 39 行的成員函式程式碼也都需要修改。讓 Time 建構子呼叫 setTime 再讓 setTime 呼叫 setHour、setMinute 和 setSecond，能把檢查 hour、minute 或 second 程式碼的任何變更，限制在對應的 set 函式 (以 set 爲名稱開頭的函式) 內。實作內容常需變更時，這麼做可以降低程式設計出錯的可能。此外，只需將 Time 建構子宣告成行內函式，或將建構子定義在類別定義中 (即定義爲行內函式)，便可提高 Time 建構子的執行效率。

軟體工程的觀點 9.9

若類別的成員函式已經提供建構子 (或其它成員函式) 所需的全部或部分的功能，則可讓建構子 (或其它的成員函式) 直接呼叫該成員函式。這麼做能讓程式碼更容易維護，並降低因修改程式碼實作而發生錯誤的可能性。總之，請避免重複的程式碼。

軟體工程的觀點 9.10

具預設引數的函式，其預設值若有變更，使用該函式的程式碼均需重新編譯，以確保程式能夠正確執行。

圖 9.10 的 `main` 函式會初始化 5 個 `Time` 物件；其中第一個建構子使用的三個引數都是預設值 (第 9 行)，另一個傳入一個引數 (第 10 行)，一個傳入兩個指定的引數 (第 11 行)，一個傳入三個引數 (第 12 行)，而最後一個傳入三個無效的引數 (第 13 行)。然後，程式以通用和標準時間格式，顯示每個物件的內容。

```
1  // Fig. 9.10: fig09_10.cpp
2  // Demonstrating a default constructor for class Time.
3  #include <iostream>
4  #include "Time.h" // include definition of class Time from Time.h
5  using namespace std;
6
7  int main()
8  {
9     Time t1; // all arguments defaulted
10    Time t2( 2 ); // hour specified; minute and second defaulted
11    Time t3( 21, 34 ); // hour and minute specified; second defaulted
12    Time t4( 12, 25, 42 ); // hour, minute and second specified
13    Time t5( 27, 74, 99 ); // all bad values specified
14
15    cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
16    t1.printUniversal(); // 00:00:00
17    cout << "\n  ";
18    t1.printStandard(); // 12:00:00 AM
19
20    cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
21    t2.printUniversal(); // 02:00:00
22    cout << "\n  ";
23    t2.printStandard(); // 2:00:00 AM
24
25    cout << "\n\nt3: hour and minute specified; second defaulted\n  ";
26    t3.printUniversal(); // 21:34:00
27    cout << "\n  ";
28    t3.printStandard(); // 9:34:00 PM
29
30    cout << "\n\nt4: hour, minute and second specified\n  ";
31    t4.printUniversal(); // 12:25:42
32    cout << "\n  ";
33    t4.printStandard(); // 12:25:42 PM
34
```

圖 9.10　使用預設引數的建構子

```
35      cout << "\n\nt5: all invalid values specified\n  ";
36      t5.printUniversal(); // 00:00:00
37      cout << "\n  ";
38      t5.printStandard(); // 12:00:00 AM
39      cout << endl;
40   } // end main
```

```
Constructed with:

t1: all arguments defaulted
  00:00:00
  12:00:00 AM

t2: hour specified; minute and second defaulted
  02:00:00
  2:00:00 AM

t3: hour and minute specified; second defaulted
  21:34:00
  9:34:00 PM

t4: hour, minute and second specified
  12:25:42
  12:25:42 PM

t5: all invalid values specified
  00:00:00
  12:00:00 AM
```

圖 9.10　使用預設引數的建構子 (續)

類別 Time 的 Set、Get 函式與建構子注意事項

類別 Time 在整個本體中多次呼叫名稱以 set 和 get 開頭的函式。舉例來說，圖 9.9 第 17-22 行函式 setTime 呼叫 setHour、setMinute 和 setSecond 等函式，而函式 printUniversal 和 printStandard 分別在第 63–64 行和第 70–72 行呼叫 getHour、getMinute 和 getSecond 等函式。其實，這些函式都能直接存取類別的 private 資料，無需呼叫 set 及 get 函式。然而，請考慮改變內部時間表示法的情況：原來我們用三個 int 表示時、分、秒 (共需 12 個位元組)，假設現在改用一個 int，表示自午夜以來經過的秒數 (只需四個位元組) 。在這個情況下，只有直接存取 private 資料的函式需要修改；即存取 hour、minute 和 second 的 set 和 get 函式，但 setTime、printUniversal 和 printStandard 等函式並不需改變，因爲它們未直接存取資料。類別的程式內容改變時，這種程式設計方法可降低錯誤發生的可能。

同樣的，我們也可以在 Time 建構子中直接加入函式 setTime 中的敘述。這麼做能稍微提高執行效率，因爲可消除呼叫建構子和 setTime 的額外負擔。不過，若把敘

述複製到建構子或函式中，稍後若想改變內部的資料表示法，就會比較困難。如果 Time 建構子直接呼叫 setTime，以後 setTime 的程式碼若有任何變更，只需修改程式碼一次即可。

常見的程式設計錯誤 9.2

建構子可以呼叫類別的其它成員函式 (如 set 或 get 函式)，但建構子原是用來設定物件的初始值，此時資料成員可能尚未處於合法的狀態。在資料成員正確設定初始值之前使用其值，會造成邏輯錯誤。

9.7 解構子

解構子 (destructor) 是一種特別的成員函式，類別解構子的名稱，是類別名稱前面加**上波浪符號 (~，tilde character)**。如此命名有直覺上的意義：在後面的章節中，讀者將會瞭解波浪符號是位元運算的補數運算子 (complement)，代表解構子和建構子是相對的。有些文獻喜歡用 "dtor" 作爲解構子的簡稱；我們不建議這樣用。

物件清除時，程式會自動呼叫其類別的解構子。舉例來說，程式離開該物件的使用域時，自動物件會將自己清除。解構子本身不會釋放物件占用的記憶體；它負責執行**資源回收 (termination housekeeping)**，結束之後，系統才會收回物件占用的記憶體，供其它物件使用。

解構子不接收任何參數，也沒有傳回值。事實上，解構子不能有傳回型態，連 void 也不行。類別只能擁有一個解構子；解構子也不允許多載。解構子一定是 public 的。

常見的程式設計錯誤 9.3

嘗試將引數傳給解構子，指定解構子的傳回值型別 (包括 void 型別)，讓解構子傳回數值，或多載解構子，都會造成語法錯誤。

讀者可能會發現，前面的範例都沒有定義解構子，但其實每個類別都一定有解構子。程式設計師若未定義解構子，編譯器會自動產生空的解構子。[請注意：稍後我們會討論到，這種自動建立的解構子，對由組合 (第 10 章) 和繼承 (第 12 章) 而成的物件執行重要的運算。] 在第 11 章中，我們會爲類別建立適當的解構子，這些類別的物件會動態配置記憶體 (陣列和字串) 或使用其它的系統資源 (如磁碟的檔案，在第 17 章中我們會學到)。我們在第 10 章將討論如何動態配置和回收記憶體空間。

軟體工程的觀點 9.11

在本書後續的章節中，我們將帶領讀者瞭解，建構子和解構子在 C++與物件導向程式設計中，佔有非常重要的地位，不只是這裡討論的那麼簡單。

9.8　建構子與解構子的呼叫時機

建構子和解構子會由編譯器自動呼叫。它們的呼叫順序，與程式的執行進入和離開物件被建立的使用域的順序有關。一般來說，解構子的呼叫順序，與建構子的呼叫順序相反，但如圖 9.11-9.13 所見，物件的儲存類別會影響解構子的呼叫順序。

在任何函式 (包括 `main`) 開始執行之前，程式首先呼叫全域使用域物件的建構子。在多個檔案之中，全域物件建構子的執行順序無法確定。相對應的解構子會在 `main` 結束時被呼叫。函式 **`exit`** 會強迫程式立刻終止執行，不會呼叫自動物件的解構子。程式輸出發生錯誤，或程式無法開啓要處理的檔案時，我們常會呼叫它終止程式執行。函式 **`abort`** 的功能和 `exit` 很像，不同處在於呼叫 `abort` 時，程式會立刻終止，不會呼叫任何物件的解構子。函式 `abort` 通常用來表示程式不正常結束執行。若想更深入瞭解 `exit` 和 `abort` 函式，請參閱附錄 F。

程式執行到定義自動物件處時，會呼叫其建構子；程式執行權離開物件的使用域時 (離開定義該物件的區塊)，會呼叫對應的解構子。對自動物件來說，程式會在進入和離開其使用域時，呼叫其建構子和解構子各一次。但我們若透過呼叫 `exit` 或 `abort` 終止程式執行，就不會呼叫自動物件的解構子。

當程式第一次執行 `static` 區域物件的定義處時，會呼叫其建構子，而於 `main` 結束或程式呼叫 `exit` 時，呼叫對應的解構子。全域和 `static` 物件的清除順序，恰與建立的順序相反。程式若呼叫 `abort` 函式終止程式，就不會呼叫 `static` 物件的解構子。

圖 9.11-9.13 的程式示範，對類別 `CreateAndDestroy` (圖 9.11 和 9.12) 具各種不同使用域及不同儲存類別的變數而言，建構子和解構子的呼叫順序。每個 `CreateAndDestroy` 類別的物件都包含一個整數 (`objectID`) 和一個字串 (`message`)，作爲識別之用 (圖 9.11 第 16-17 行) 。這個簡單的類別是爲教學目的而設計的。圖 9.12 中解構子的第 21 行，會判斷被清除物件的 `objectID` 值是否爲 1 或 6，若是，則輸出換行字元，讓輸出資料清晰可讀。

```
1  // Fig. 9.11: CreateAndDestroy.h
2  // CreateAndDestroy class definition.
3  // Member functions defined in CreateAndDestroy.cpp.
4  #include <string>
5  using namespace std;
6
7  #ifndef CREATE_H
8  #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13    CreateAndDestroy( int, string ); // constructor
14    ~CreateAndDestroy(); // destructor
15 private:
16    int objectID; // ID number for object
17    string message; // message describing object
18 }; // end class CreateAndDestroy
19
20 #endif
```

圖 9.11 CreateAndDestroy 類別定義

```
1  // Fig. 9.12: CreateAndDestroy.cpp
2  // CreateAndDestroy class member-function definitions.
3  #include <iostream>
4  #include "CreateAndDestroy.h"// include CreateAndDestroy class definition
5  using namespace std;
6
7  // constructor
8  CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
9  {
10    objectID = ID; // set object's ID number
11    message = messageString; // set object's descriptive message
12
13    cout << "Object " << objectID << "   constructor runs   "
14       << message << endl;
15 } // end CreateAndDestroy constructor
16
17 // destructor
18 CreateAndDestroy::~CreateAndDestroy()
19 {
20    // output newline for certain objects; helps readability
21    cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
22
23    cout << "Object " << objectID << "   destructor runs    "
24       << message << endl;
25 } // end ~CreateAndDestroy destructor
```

圖 9.12 CreateAndDestroy 的成員函式定義

圖 9.13 在全域使用域定義物件 first (第 10 行)；程式會在執行 main 中任何敘述之前，呼叫其建構子，且在程式結束，其它所有物件都清除之後，才呼叫其解構子。

函式 main (第 12-23 行) 宣告了三個物件。物件 second (第 15 行) 和 fourth (第 21 行) 都是區域自動物件，而物件 third (第 16 行) 則是 static 區域物件。這些物件的建構子都是在程式進入這些物件的宣告處時，才進行呼叫。物件 fourth 和 second 的解構子會在 main 函式結束時呼叫 (即呼叫建構子的相反順序)。物件 third 是 static 物件，它的生命週期會一直延續到程式終止。程式會在呼叫全域物件 first 的解構子之前，但在清除所有其它物件之後，呼叫物件 third 的解構子。

第 26-33 行的函式 create 宣告三個物件；區域自動物件 fifth (第 29 行) 和 seventh (第 31 行) 以及 static 區域物件 sixth (第 30 行)。seventh 和 fifth 的解構子會在 create 函式結束時，按建構子的相反順序被呼叫。sixth 是 static 物件，所以會一直存在，直到程式結束爲止。因此，程式會在呼叫 third 和 first 的解構子之前，清除完畢其他物件之後，呼叫 sixth 的解構子。

```
1  // Fig. 9.13: fig09_13.cpp
2  // Demonstrating the order in which constructors and
3  // destructors are called.
4  #include <iostream>
5  #include "CreateAndDestroy.h" // include CreateAndDestroy class definition
6  using namespace std;
7
8  void create( void ); // prototype
9
10 CreateAndDestroy first( 1, "(global before main)" ); // global object
11
12 int main()
13 {
14    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
15    CreateAndDestroy second( 2, "(local automatic in main)" );
16    static CreateAndDestroy third( 3, "(local static in main)" );
17
18    create(); // call function to create objects
19
20    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
21    CreateAndDestroy fourth( 4, "(local automatic in main)" );
22    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
23 } // end main
24
25 // function to create objects
26 void create( void )
27 {
28    cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
```

圖 9.13　建構子和解構子的呼叫順序

```
29      CreateAndDestroy fifth( 5, "(local automatic in create)" );
30      static CreateAndDestroy sixth( 6, "(local static in create)" );
31      CreateAndDestroy seventh( 7, "(local automatic in create)" );
32      cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
33   } // end function create
```

```
Object 1   constructor runs   (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2   constructor runs   (local automatic in main)
Object 3   constructor runs   (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5   constructor runs   (local automatic in create)
Object 6   constructor runs   (local static in create)
Object 7   constructor runs   (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7   destructor runs    (local automatic in create)
Object 5   destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4   constructor runs   (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4   destructor runs    (local automatic in main)
Object 2   destructor runs    (local automatic in main)

Object 6   destructor runs    (local static in create)
Object 3   destructor runs    (local static in main)

Object 1   destructor runs    (global before main)
```

圖 9.13　建構子和解構子的呼叫順序 (續)

9.9 類別 Time 案例研究：小心陷阱；傳回 private 資料成員的參照

指向物件的參照，相當於其名稱的別名，因此可用在賦值敘述的左邊，接受設定的數值。在此情況下，參照會作爲 lvalue，接受要設定的數值。使用這項功能的方法之一，是透過類別的 public 成員函式，傳回指向該類別 private 資料成員的參照。若傳回 const 參照，就無法用作可修改的 lvalue。

圖 9.14-9.16 的程式藉簡化過的 Time 類別 (圖 9.14 和 9.15) 示範，使用成員函式 badSetHour (宣告於圖 9.14 第 15 行，定義於圖 9.15 第 27-31 行) 傳回指向 private 資料成員的參照。這個傳回動作，會讓函式 badSetHour 產生 private 資料成員 hour 的別名。這麼做會暴露 private 資料成員，甚至讓它成爲賦值敘述中的 lvalue，讓外部程式碼任意修改類別的 private 資料。函式若傳回指向 private 資料成員的指標，也會有同樣的問題。

```
1  // Fig. 9.14: Time.h
2  // Time class declaration.
3  // Member functions defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  class Time
10 {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int ); // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 }; // end class Time
21
22 #endif
```

圖 9.14　Time 類別的宣告

```
1  // Fig. 9.15: Time.cpp
2  // Time class member-function definitions.
3  #include "Time.h" // include definition of class Time
4
5  // constructor function to initialize private data; calls member function
6  // setTime to set variables; default values are 0 (see class definition)
7  Time::Time( int hr, int min, int sec )
8  {
9     setTime( hr, min, sec );
10 } // end Time constructor
11
12 // set values of hour, minute and second
13 void Time::setTime( int h, int m, int s )
14 {
15    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
16    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
17    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
18 } // end function setTime
19
20 // return hour value
21 int Time::getHour()
22 {
23    return hour;
24 } // end function getHour
25
26 // POOR PRACTICE: Returning a reference to a private data member.
```

圖 9.15　Time 成員函式的定義

```
27  int &Time::badSetHour( int hh )
28  {
29     hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
30     return hour; // DANGEROUS reference return
31  } // end function badSetHour
```

圖 9.15 Time 成員函式的定義 (續)

圖 9.16 的程式宣告型別爲 Time 的物件 t (第 10 行)和參照 hourRef (第 13 行) ，並將 t.badSetHour (20) 傳回的參照，指定給 hourRef。程式第 15 行顯示別名 hourRef 的值。這顯示 hourRef 破壞了類別的封裝性 (encapsulation)，因爲 main 中的敘述應不得存取類別的 private 資料。接下來，第 16 行透過這個別名，把 hour 的值設成不合法的 30，而第 17 行顯示 getHour 傳回的這個值，顯示透過 hourRef 的確更改了物件 t 的 private 成員的值。最後第 23 行使用 badSetHour 函式本身做爲 lvalue，將回傳的參照設爲 74 (另一個不合法的值)。第 26 行再次顯示 getHour 函式傳回的值，確認將值指定給第 21 行函式呼叫的傳回值結果，的確修改了 Time 物件 t 的 private 資料。

```
1  // Fig. 9.16: fig09_16.cpp
2  // Demonstrating a public member function that
3  // returns a reference to a private data member.
4  #include <iostream>
5  #include "Time.h" // include definition of class Time
6  using namespace std;
7
8  int main()
9  {
10    Time t; // create Time object
11
12    // initialize hourRef with the reference returned by badSetHour
13    int &hourRef = t.badSetHour( 20 ); // 20 is a valid hour
14
15    cout << "Valid hour before modification: " << hourRef;
16    hourRef = 30; // use hourRef to set invalid value in Time object t
17    cout << "\nInvalid hour after modification: " << t.getHour();
18
19    // Dangerous: Function call that returns
20    // a reference can be used as an lvalue!
21    t.badSetHour( 12 ) = 74; // assign another invalid value to hour
22
23    cout << "\n\n*********************************************\n"
24       << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
25       << "t.badSetHour( 12 ) as an lvalue, invalid hour: "
26       << t.getHour()
```

圖 9.16 傳回指向 private 資料成員的參照

```
27         << "\n************************************************" << endl;
28   } // end main
```

```
Valid hour before modification: 20
Invalid hour after modification: 30

************************************************
POOR PROGRAMMING PRACTICE!!!!!!!!
t.badSetHour( 12 ) as an lvalue, invalid hour: 74
************************************************
```

圖 9.16　傳回指向 private 資料成員的參照 (續)

測試和除錯的小技巧 9.4

傳回指向 private 物件的參照或指標，會破壞類別的封裝，讓使用類別的程式碼依賴物件的內部資料表示方式。這是很危險的，請盡量避免。

9.10　預設逐成員賦值

指定運算子 (=) 可將某個物件指定給另一個相同型別的物件。預設情況下，該運算子會執行**逐成員賦值 (memberwise assignment)**：將指定運算子右邊物件的每個成員，逐一指定給運算子左邊物件的每個成員。圖 9.17-9.18 定義類別 Date 做爲例子。圖 9.19 的第 18 行透過**預設的逐成員賦值 (default memberwise assignment)**，把 Date 物件 date1 的資料成員設給型別同爲 Date 的物件 date2 的對應資料成員。意思是說，date1 的成員 month 會設給 date2 的成員 month；date1 的成員 day 設給 date2 的成員 day；date1 的成員 year 會設給 date2 的成員 year。[請注意：類別的資料成員若爲指向動態配置記憶體的指標，逐成員賦值可能會造成嚴重錯誤；我們將在第 11 章討論這些問題和解決方法)。] 建構子 Date 未執行任何錯誤檢查；我們把它留到習題來解決。

```
1   // Fig. 9.17: Date.h
2   // Date class declaration.  Member functions are defined in Date.cpp.
3
4   // prevent multiple inclusions of header file
5   #ifndef DATE_H
6   #define DATE_H
7
8   // class Date definition
9   class Date
10  {
11  public:
12     Date( int = 1, int = 1, int = 2000 ); // default constructor
```

圖 9.17　Date 類別的宣告

```
13    void print();
14 private:
15    int month;
16    int day;
17    int year;
18 }; // end class Date
19
20 #endif
```

圖 9.17 Date 類別的宣告

```
1  // Fig. 9.18: Date.cpp
2  // Date class member-function definitions.
3  #include <iostream>
4  #include "Date.h" // include definition of class Date from Date.h
5  using namespace std;
6
7  // Date constructor (should do range checking)
8  Date::Date( int m, int d, int y )
9  {
10    month = m;
11    day = d;
12    year = y;
13 } // end constructor Date
14
15 // print Date in the format mm/dd/yyyy
16 void Date::print()
17 {
18    cout << month << '/' << day << '/' << year;
19 } // end function print
```

圖 9.18 Date 成員函式定義

```
1  // Fig. 9.19: fig09_19.cpp
2  // Demonstrating that class objects can be assigned
3  // to each other using default memberwise assignment.
4  #include <iostream>
5  #include "Date.h" // include definition of class Date from Date.h
6  using namespace std;
7
8  int main()
9  {
10    Date date1( 7, 4, 2004 );
11    Date date2; // date2 defaults to 1/1/2000
12
13    cout << "date1 = ";
14    date1.print();
15    cout << "\ndate2 = ";
16    date2.print();
17
```

圖 9.19 預設逐成員賦值

```
18      date2 = date1; // default memberwise assignment
19
20      cout << "\n\nAfter default memberwise assignment, date2 = ";
21      date2.print();
22      cout << endl;
23   } // end main
```

```
date1 = 7/4/2004
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004
```

圖 9.19　預設逐成員賦值 (續)

物件可以作爲引數傳入函式，也可以從函式傳回。在預設情況下，物件的傳入和傳回動作以傳值呼叫進行，即程式會傳入和傳回物件的副本。因此，C++會建立新的物件，透過**複製建構子 (copy constructor)** 將原物件的值設給新的物件。編譯器會爲所有類別提供預設的複製建構子，用來把原物件的每個成員，複製到新物件的對應成員。和逐成員賦值一樣，類別資料成員若包含指向動態配置記憶體的指標，複製建構子可能會造成嚴重的問題。我們將在第 11 章談到如何自行定義複製建構子，以便適當處理指向動態配置記憶體的指標成員。

增進效能的小技巧 9.3

從安全性的角度來看，以傳值呼叫的方式傳遞物件是很好的，因爲被呼叫的函式無法存取原來的物件；不過，傳遞大型物件的副本，會影響程式的執行效率。另一種方式是，傳遞指向該物件的指標或參照。傳參考呼叫的效率較佳，但安全性比較差，因爲被呼叫的函式能存取原始的物件。傳 `const` 參照綜合以上二者的優點，作法是用 `const` 參照或指向 `const` 資料的指標作爲引數傳入函式。

9.11　總結

本章深入討論了「類別」這個主題，使用豐富的範例類別 `Time` 來介紹類別的幾個新功能。你已經認識到，成員函式通常比全域函式要短很多，因爲成員函式可以直接存取物件的資料成員，所以比起程序式程式設計語言中的一般函式，成員函式所接收的引數比較少。你也學到了怎麼使用箭號運算子，透過物件類別型別的指標，來存取物件的成員。

你學到了成員函式具有類別使用域，成員函式的名稱只有同類別中的其它成員才能使用，除非透過此類別的物件、參照此類別的物件、指向此類別物件的指標，或使用二元使用域解析運算子。我們也討論了存取函式 (通常用來讀取資料成員的值，或測試條件的眞僞)，以及工具函式 (`private` 成員函式，用來協助類別的 `public` 成員函式的操作)。

你學到了，建構子可以指定預設引數，以各種方式呼叫。你也學到了，呼叫時不需任何引數的建構子稱爲預設建構子，類別最多只能有一個預設建構子。我們討論了解構子，它的用途是在類別物件被清除之前，執行資源回收的工作。我們也示範了物件的建構子和解構子被呼叫的順序。

我們說明了，當成員函式回傳一個指向 `private` 資料成員的參照時，可能會破壞類別的封裝。我們也展示了同一個型別的物件可以透過預設的逐成員賦值指派給另一個物件。最後，我們討論了使用類別函式庫的優點，它可以增進軟體開發的速度以及品質。

第 10 章會介紹類別的其它功能。我們會介紹要如何使用 `const` 來表示成員函式不能被類別的物件修改。你會學到如何組合其它類別來產生新類別，也就是在類別中包含其它類別的物件，做爲成員。我們會展示類別如何讓夥伴 (friend) 函式來存取它的非 `public` 成員。我們也會展示類別的非 `static` 成員函式，使用特殊的指標「`this`」來存取物件的成員。

摘要

9.2 案例研究：類別 `Time`

- 前置處理指令`#ifndef` (意思是「若未定義」，if not defined) 和`#endif` 可用來避免重複含括同一標頭檔。若這兩個指令中間的程式碼未曾含括過，`#define` 會定義一個名稱，避免程式碼以後被再度含括，並將程式碼含括進原始碼檔案中。
- 資料成員不能在類別主體中宣告時設定初始值 (唯一的例外是類別的 `static const` 整數資料成員或 `enum` 型別)。在類別的建構子中設定資料成員的初始值，因爲系統不會自動設定基本型別資料成員的初始值。
- 串流操作子 `setfill` 可用來設定填充字元，當要輸出的整數位數小於欄位寬度時，用來填滿欄位。
- 預設情況下，填充字元會出現在數字之前。
- 串流操作子 `setfill` 是一種粘著性設定 (sticky setting)，意思是一旦設定，就會對之後所有的輸出欄位發生作用。
- 即使我們在類別的定義之內宣告成員函式，將其定義置於類別的主體之外，並透過二元使用域解析運算子結合二者，成員函式仍屬於類別使用域 (class scope)。
- 成員函式若定義於類別定義中，C++編譯器會嘗試將該成員函式變成行內函式。
- 類別可以包含其它類別的物件作爲其成員，或衍生自其它類別，讓其它類別提供新類別的屬性和行爲。

9.3　類別使用域及存取類別的成員

- 類別的資料成員與成員函式屬於類別使用域。
- 非成員函式屬於全域命名空間使用域 (global namespace scope)。
- 在類別使用域中，類別的所有成員可透過成員函式存取，且可透過名稱參照。
- 在類別使用域之外，可透過物件的代表 (handle) 參照類別成員，如物件名稱、以及指向物件的參照或指標。
- 我們可對類別的成員函式進行多載，但多載函式也必須爲該類別的成員函式。
- 要多載成員函式，只需在類別定義內列出每個多載函式的原型，再爲它們提供定義即可。
- 在成員函式中宣告的變數具有區域使用域，只能在該函式中使用。
- 如果某成員函式定義了一個變數，其名稱與另一個類別使用域的變數名稱相同，則在此函式使用域中，類別使用域的變數會被隱藏。
- 將點號 (`.`) 成員選擇運算子 (member selction operator) 置於物件名稱或指向物件的參照之後，就可存取該物件的 `public` 成員。
- 箭號成員選擇運算子 (`->`) 則可配合指向物件的指標使用，存取物件的 `public` 成員。

9.4　區分介面與實作

- 因爲標頭檔還是會包含一部分的類別實作程式碼和其它提示。例如，行內成員函式必須置於標頭檔中，讓編譯器編譯外部程式時，直接在需要處插入行內成員函式的定義。
- 標頭檔中的類別定義會列出類別的 `private` 成員；使用者雖無法存取它們，但仍能藉此得知成員的相關訊息。

9.5　存取函式與工具函式

- 工具函式是用來幫助類別其它 `public` 成員函式完成工作的 `private` 成員函式，工具函式不能讓使用類別的外部程式呼叫。

9.6　類別 `Time` 案例研究：建構子與預設引數

- 與一般函式相同，建構子可以擁有預設引數。

9.7　解構子

- 物件消滅時，系統會自動呼叫其類別的解構子。
- 類別解構子的名稱，是類別名稱前面加上波浪符號 (~)。

- 解構子不會釋放物件的儲存空間 - 它會在系統收回物件所占記憶體之前，執行資源回收，讓新物件得以利用清出來的記憶體。
- 解構子不接收任何參數，也沒有傳回值。類別至多可擁有一個解構子。
- 若程式設計者未指定類別解構子，編譯器會自動提供，所以每個類別都有解構子。

9.8 建構子與解構子的呼叫時機

- 建構子和解構子的呼叫順序與建立物件時，程式執行權進出每個物件範圍的順序有關。
- 一般來說，解構子的呼叫順序，與建構子的呼叫順序相反，但物件的儲存類別會影響解構子的呼叫順序。

9.9 類別 `Time` 案例研究：小心陷阱；傳回 `private` 資料成員的參照

- 指向物件的參照，相當於其名稱的別名，因此可用在賦值敘述的左邊，接受設定的數值。在此情況下，參照會作爲 lvalue，接受要設定的數值。
- 若函式傳回 `const` 參照，則該參照就不能用爲可修改的 lvalue。

9.10 預設逐成員賦值

- 指定運算子 (=) 可將某個物件指定給另一個相同型別的物件。預設情況下，該運算子會執行逐成員賦值 (memberwise copy)。
- 物件可以用傳值呼叫的方式傳入函式，也可以從函式傳回。C++會建立新的物件，透過複製建構子 (copy constructor) 將原物件的值設給新的物件。
- 編譯器會爲所有類別提供預設的複製建構子，用來把原物件的每個成員，複製到新物件的對應成員。

術語

函式 `abort` (abort function)
存取函式 (access function)
聚合 (aggregation)
箭號成員選取運算子 (`->`) (arrow member selection operator (`->`))
類別使用域 (class scope)
組合 (composition)
複製建構子 (copy constructor)
預設的逐成員賦值 (default memberwise assignment)
前置處理指令`#define` (#define preprocessor directive)
由類別衍生出另一個類別 (derive one class from another)
衍生 (derived）
解構子 (destructor)

前置處理指令`#endif` (#endif preprocessor directive)
函式 `exit` (exit function)
填充字元 (fill character)
物件代表 (handle on an object)
助手函式 (helper function)
前置處理指令`#ifndef` (#ifndef preprocess directive)
繼承 (inheritance)
逐成員賦值 (memberwise assignment)
判斷函式 (predicate function)
前置處理包裝 (preprocessor wrapper)
參數化串流操作子 `setfill` (setfill parameterized stream manipulator)
資源回收 (termination housekeeping)
解構子名稱中的波浪字元 (~) [tilde character (~) in a destructor name]

自我測驗

9.1 請填寫以下空格：

a) 類別成員可透過_____運算子配合類別的物件名稱，或用_____運算子，配合指向此類別物件的指標存取之。
b) 類別成員若爲_____，則只能由類別的成員函式或夥伴函式存取。
c) 類別的成員若爲_____，則程式可以在此類別物件使用域內的任何地方存取之。
d) _____可用來把類別物件設給另一個相同類別的物件。

9.2 找出下列敘述中的錯誤，並說明如何更正：

a) 假設在類別 Time 中有下列原型宣告：

```
void ~Time( int );
```

b) 以下是類別 Time 定義的一部分：

```
class Time
{
public:
   // function prototypes

private:
   int hour = 0;
   int minute = 0;
   int second = 0;
}; // end class Time
```

c) 假設類別 Employee 有以下的原型宣告：

```
int Employee( string, string );
```

自我測驗解答

9.1 a) 點號運算子 (.)，箭號運算子 (->)。b) `private`。c) `public`。d) 預設的逐成員賦值 (由指定運算子執行)。

9.2 a) 錯誤：解構子不能傳回任何值 (甚至不能有傳回值型別) 或接收任何引數。

更正：移除宣告中的傳回值型別 void 和參數 int。

b) 錯誤：在類別定義中，不能明確設定成員的初始值。
更正：移除類別定義中的設定成員初始值的敘述，改於建構子中進行。

c) 錯誤：建構子不可以有傳回值。
更正：從宣告中移除傳回值型別 int。

習題

9.3 使用域解析運算子的功用爲何？

9.4 (改良 Time 類別) 請撰寫建構子 `Time`，使用 `time` 和 `localtime` 函式 (於 C++標準函式庫標頭檔<ctime>中宣告) 取得目前時間，用來設定 `Time` 物件的初始值。

9.5 (類別 Complex) 請撰寫名爲 `Complex` 的類別，用來表示數學中的複數 (complex number)，並寫一個程式測試之。複數的格式爲

```
realPart + imaginaryPart * i
```

其中 `i` 爲

$\sqrt{-1}$

請使用 `double` 變數表示此類別的 `private` 資料。提供一個建構子，於此類別的物件被宣告時，設定其初始值。此建構子應具預設引數，供未提供參數時使用。此外，請提供 `public` 成員函式執行以下工作：

a) 兩個複數相加：實部和實部相加，虛部和虛部相加。

b) 兩個複數相減：先將左運算元的實部減去右運算元的實部，再將左運算元的虛部減去右運算元的虛部。

c) 用形如 `(a,b)` 的格式，列印 `Complex` 的值；其中 `a` 表示實部，`b` 表示虛部。

9.6 (類別 Rational) 請撰寫名爲 `Rational` 的類別，用來執行非整數算術，並寫一個程式測試之。

請使用整數變數表示類別的 `private` 資料，即分母和分子。提供一個建構子，於此類別的物件被宣告時，設定其初始值。建構子應具預設值，供未提供參數時使用，並應以最簡分數的形式儲存分數。例如，以下分數

$\frac{2}{4}$

會在物件內將分子儲存爲 1，而將分母儲存成 2。請提供執行以下工作的 `public` 成員函式：

a) 將兩個 `Rational` 數字相加，結果存成最簡分數。

b) 將兩個 `Rational` 數字相減，結果存成最簡分數。

c) 將兩個 `Rational` 數字相乘，結果存成最簡分數。

d) 將兩個 `Rational` 數字相除，結果存成最簡分數。

e) 以形如 a/b 的格式印出 Rational 的值，其中 a 爲分子，b 爲分母。

f) 以浮點數格式印出 Rational 的值。

9.7 (改良 Time 類別) 修改圖 9.8-9.9 的類別 Time，加入成員函式 tick，其功能是把 Time 物件儲存的時間增加一秒。Time 物件的狀態應永遠保持合法。撰寫一個測試程式，利用迴圈測試 tick 成員函式，每執行一次迴圈操作，以標準時間格式列印其內容，確認 tick 成員函式運作正確。請測試以下的情況：

a) 進位到下一分鐘。

b) 進位到下一小時。

c) 進位到下一日 (即 11:59:59PM 進位到 12:00:00AM) 。

9.8 (改良 Date 類別) 修改圖 9.17-9.18 的 Date 類別，檢查用來設定資料成員 month、day 和 year 的初始值。另外，請提供成員函式 nextDay，用來遞增日數。Date 物件的狀態應該永遠維持合法。撰寫一個測試程式，利用迴圈測試 nextDay 函式，每跑一次迴圈，就印出時間日期，以確認 nextDay 成員函式運作正確。請測試以下的情況：

a) 進位到下個月。

b) 進位到明年。

9.9 (結合類別 Time 與 Date) 請結合習題 9.7 和 9.8 的類別 Time 和 Date，設計名爲 DateAndTime 的類別。(第 12 章會討論繼承，到時就可快速完成這項任務，不需修改現有的類別定義)。請修改 tick 函式，當時間遞增到明天時，呼叫 nextDay 函式。並請修改函式 printStandard 和 printUniversal，輸出日期和時間，並撰寫一個程式測試新類別 DateAndTime，特別是時間遞增到明天的狀況。

9.10 (藉類別 Time 的 set 函式的傳回值指出錯誤) 修改圖 9.8-9.9 中類別 Time 的 set 函式，當傳入的值無效時，會傳回一個特別的值，指出發生了錯誤。請撰寫程式測試新版的 Time 類別； 當 set 函式傳回錯誤值時，顯示錯誤訊息。

9.11 (類別 Rectangle) 請設計類別 Rectangle，此類別包含 length 和 width 兩個屬性，預設值均爲 1。請提供成員函式，計算周長和面積。此外，請提供處理 length 和 width 屬性的 set 和 get 函式。set 函式應檢查 length 及 width 是否爲浮點數，且其值介於 0.0 和 20.0 之間。

9.12 (改良類別 Rectangle) 請改良前一個習題設計的類別 Rectangle，使其儲存矩形四個頂點的直角座標。建構子呼叫一個接收四組座標的 set 函式，並驗證每個座標都位於第一象限內，且沒有任何一個 x 或 y 座標大於 20.0。此外，set 函式亦須確認所提供的 4 組座標值的確形成一矩形。成員函式須計算 length (長)、width (寬)、perimeter (周長) 和 area (面積)。較長的邊爲長 (length)。最後，請提供判斷函式 square 判斷該矩形是否爲正方形。

9.13 (改良類別 `Rectangle`) 請修改習題 9.12 的 `Rectangle` 類別，加入 `draw` 函式，於 25x25 的四方形區域 (第一象限) 中繪出矩形，並加入 `setFillcharacter` 函式，用來指定填入矩形內部的字元。另外，請加入 `setPerimeterCharacter` 函式，指定描繪矩形邊框的字元。讀者若有興趣，還可加入函式縮放矩形的大小，或在第一象限指定的區域內，對它進行移動和旋轉。

9.14 (類別 `HugeInteger`) 請建立使用具 40 個元素陣列的 `HugeInteger` 類別，用來表示最高可達 40 位數的整數。請提供成員函式 `input`、`output`、`add` 和 `substract`。爲比較 `HugeInteger` 物件，請提供 `isEqualTo`、`isNotEqualTo`、`isGreaterThan`、`isLessThan`、`isGreaterThanOrEqualTo` 和 `isLessThanOrEqualTo` 等函式。它們都是判斷函式，如果兩個大整數之間的關係成立，則傳回 `true`，否則傳回 `false`。此外，請提供另一個判斷函式 `isZero`。若有興趣，讀者還可加入 `multiply`、`divide` 和 `modulus` 等成員函式。

9.15 (類別 `TicTacToe`) 請建立類別 `TicTacToe` 及一個完整的程式，實作井字遊戲 (tic-tac-toe)。類別應包含一個 `private` 的 3x3 二維整數陣列。建構子將每個方格的初始值設爲 0。此遊戲可以有兩位玩家；第一位玩家指定的方格置入 1，第二位玩家置入 2。每步只能放到空白的方格中。每進行一步，必須判斷是否有人贏了或是和局。讀者如果有興趣，可讓人和電腦比賽。此外，你可以讓遊戲者決定誰先下誰後下。若還意猶未盡，可以再寫一個程式，用 4x4x4 的陣列玩三度空間的井字遊戲。[請注意：此專案極具挑戰性，完成它可能要花好幾個星期。]