

Example

27

類別與物件： 定義與宣告類別

園遊會裡有許多的攤位，每個攤位要繳交營收的 10% 給主辦單位作為租金。寫一程式，輸入攤位的數量，計算每個攤位的淨利與主辦單位所收取的總租金。假設每個攤位的營業額都介於 1-10 萬。攤位以類別定義，租金使用靜態資料成員，攤位的淨利等於營收減去租金。

一、學習目標

C++ 語言可視為是 C 語言的加強版或擴增版，除了保有 C 語言的特性、功能、標準函式庫之外，最大的特色為提供了物件導向程式設計（Object oriented programming：OOP）。

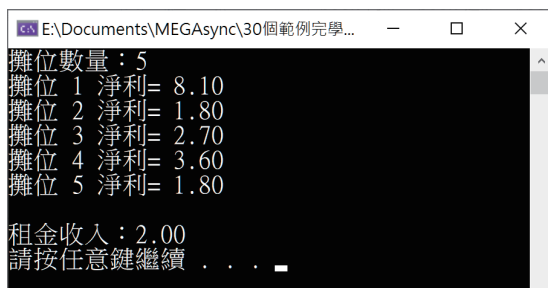
簡單地說，物件導向程式設計與傳統的程式設計最大的差別，在於傳統的程式設計對於資料和程式碼仍然是各自獨立的 2 個部分，只是透過程式結構化的方式讓系統更好維護與開發。

物件導向程式設計則是將資料與程式碼包裹在一起，以類別的方式呈現，撰寫程式是以如何操作類別裡的資料的角度去編寫程式碼；因此，使得程式撰寫得以提高效率，也使得軟體的開發更趨向簡易。就如同拼積木一樣，需要什麼樣形狀、功能的積木，就去取得積木即可；開發程式也是如此，需要什麼樣的功能，就去取得提供此種功能的類別就可以了。

因此，學習物件導向程式設計，有很大的一部分在學習了解什麼是類別、如何定義類別、撰寫類別、使用類別。物件導向程式設計不只是技術，還包含了觀念；因此，不單只是幾個章節或是範例就可以完全學習。所以，本書只討論撰寫程式時所需要的類別技術與觀念。市面上有許多探討物件導向程式設計的專業的書籍，可供讀者參考。

二、執行結果

如下圖所示：園遊會有 5 個攤位，每個攤位的營收以亂數模擬 1-10 萬之間的整數，扣除管理費之後即為所得淨利；並計算所有攤位繳交的租金總金額。



```
E:\Documents\MEGAsync\30個範例完學...
攤位數量：5
攤位 1 淨利= 8.10
攤位 2 淨利= 1.80
攤位 3 淨利= 2.70
攤位 4 淨利= 3.60
攤位 5 淨利= 1.80

租金收入：2.00
請按任意鍵繼續 . . . _
```

27-1 認識類別

一個類別（Class）把資料與操作這些資料相關的函式包裹在一起，資料包含了常數與變數；對於類別裡的資料特別稱之為資料成員（Data member）或稱為屬性（Attribute）。要存取類別裡的資料需要透過類別裡所提供的自訂函式，因此這些函式特別稱之為成員函式、成員函數（Member function）或是稱為方法（Method）。

以製作一個杯子為例，杯子有杯子高度、杯口直徑、杯子的容量、有沒有把手、杯子顏色、杯子形狀這些屬性，這些屬性如同一個類別的資料成員。製作這個杯子的方法有：製作杯體、製作杯口、製作杯子的把手、杯子上色、檢查杯子的容量是否恰當等；這些製作的方法如同類別裡的成員函式。

若要存取類別裡的資料成員，則必須透過類別裡所提供的成員函式。例如：製作杯子把手的方法裡，規範了使用什麼方式製作把手、如何製作把手、把手的形狀、把手的高度與寬度、把手的粗細、黏在杯體的哪個部位等；所以要製作杯子的把手就必須使用這些規範，而不能自行想做杯子的把手就隨意做杯子的把手。

由於物件導向程式設計把要處理的事情或是功能，以類別的方式把相關的資料、變數與函式包裹在一起，因此也產生了資料封裝、繼承與多型的特色；這也是軟體工程裡的一個重要議題。

定義與宣告類別

類別通常是給整個專案或是整支程式所使用，因此是以全域的方式定義類別。定義類別以關鍵字 `class` 開頭，接著是類別的名稱，在左右大括弧裡面是類別的內容，語法如下所示：

```
class 類別名稱
{
    public/private/protected:
    資料型別 變數名稱 ; ← 資料成員
    ⋮
    [inline] [修飾字] 回傳值型別 函式名稱 ([ 參數 1, 參數 2, ⋮ ])
    {
        程式碼敘述 ;
        [return 回傳值 ;]
    }
    ⋮
};
```

成員函式

在左右大括弧裡面是由存取控制關鍵字 `public`、`private` 或 `protected` 所宣告的區段。`public` 區段內的成員可以供整支程式存取，`private` 區段內的成員只可以由類別裡的成員存取；27-2 節會有更詳細的介紹。在區塊內可以宣告資料成員與成員函式；例如：定義一個類別 `myClass1`，如下所示：

```

1 class myClass1
2 {
3     public :
4         char name[10]; // 姓名
5         int age;       // 年齡
6 };

```

`myClass1` 類別裡有一個 `public` 區段，其中宣告了 2 個資料成員 `name` 與 `age`，分別為字元型別的一維陣列與整數型別。再看另一個例子：定義類別 `myClass2`，如下所示：

```

1 class myClass2
2 {
3     private:
4         char schNo[6]; // 學號
5
6     public :
7         string name;   // 姓名
8         int age;       // 年齡
9
10        void setName(string str) // 設定姓名
11        {
12            name = str;
13        }
14 };

```

程式碼第 3-4 行宣告 `private` 區段，此區段裡面只有 1 個字元陣列 `schNo` 的資料成員。第 6-13 行為 `public` 區段，此區段裏面包含了 2 個資料成員 `name` 與 `age`，並且有 1 個成員函式 `setName()`，此函式帶有一個 `string` 型別的參數 `str`，用於設定資料成員 `name`。

定義好類別之後，便可以使用此類別宣告變數，如下所示：

```
myClass1 mycls;
```

使用類別所宣告的變數，通常稱為物件（Object）或是實體（Instance）。類別把相關的資料與函式包裹在一起，這樣的特性便稱為封裝（Encapsulation）。

🔗 練習 1：設計三角形的類別

定義一個三角形的類別，包含 3 個資料成員：底邊、高度與面積。底邊與高度放置於 `public` 區段，面積放置於 `private` 區段。使用此三角形類別宣告物件，並顯示此物件佔用的空間容量。

■ 解說

假設此三角形的類別名稱爲 `Triangle`，有 3 個 `double` 資料型別的資料成員 `base`、`height` 與 `size`，分別代表三角形的底邊、高度與面積；則 `Triangle` 類別如下所示：

```

1 class Triangle
2 {
3     private:
4         double size; // 三角形面積
5
6     public:
7         double base;   // 三角形底邊
8         double height; // 三角形高度
9 };

```

資料成員 `size` 位於 `private` 區段，而資料成員 `base` 與 `height` 則位於 `public` 區段。宣告此類別的物件，如下所示：

```
Triangle tri;
```

要取得物件 `tri` 的大小，則可以使用 `sizeof()` 函式：

```
sizeof(tri);
```

■ 執行結果

```
Triangle 類別的大小 (bytes) : 24
```

■ 程式碼列表

```

1 #include <iostream>
2 using namespace std;
3
4 class Triangle
5 {
6     private:
7         double size;
8

```

```

9      public:
10         double base;
11         double height;
12     };
13
14     int main()
15     {
16         Triangle tri;
17
18         cout << "Triangle 類別的大小 (bytes) : " << sizeof(tri) << endl;
19         system("pause");
20     }

```

程式講解

1. 程式碼第 1-2 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼 4-12 行定義三角形類別 `Triangle`。類別裡有 `public` 與 `private` 此 2 個區段，並有 3 個 `double` 資料型別的資料成員。`private` 區段裡有代表三角形面積的資料成員 `size`，而代表三角形底邊長與高度的資料成員 `base` 與 `height` 則位於 `public` 區段。
3. 程式碼第 16 行宣告 `Triangle` 類別的物件 `tri`。
4. 程式碼第 18 行使用 `sizeof()` 函式取得並顯示物件 `tri` 的大小。

27-2 資料成員

類別裡的資料成員由常數與變數組成，在類別裡宣告變數與一般變數宣告無異，也能設定初始值。存取類別內的資料成員使用範圍存取運算子 `.`，如下語法所示：

類別物件名稱．資料成員

例如：練習 1 中的三角形類別 `Triangle`，則要顯示物件 `tri` 裡的資料成員 `base`：

```
cout << tri.base << endl;
```

存取控制關鍵字 `public`、`private` 與 `protected`

資料成員需宣告在 `public`、`private` 或 `protected` 區段內；此 3 個關鍵字稱為存取控制，由此 3 個關鍵字所宣告的區段其在整支程式的有效範圍不同。

public 區段

放置於 `public`（公開）區段內的資料成員可以供整支程式存取；也就是可以隨意讀取與改變宣告在 `public` 區段內的資料成員的值。此區段可以被繼承，請參考範例 29。

例如，定義一個記錄學生姓名與年齡的類別 `myClass`，如下所示。程式碼第 1-6 行定義類別 `myClass`，並宣告 `public` 區段。區段裡面宣告了 2 個資料成員 `name` 與 `age`，分別表示姓名與年齡；並且 `age` 的初始值等於 0。

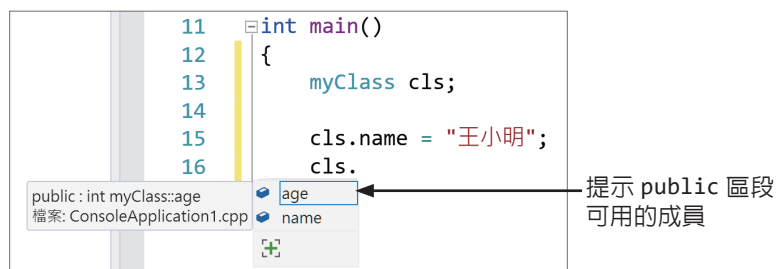
```

1  class myClass
2  {
3      public:
4          string name; // 姓名
5          int age = 0;  // 年齡
6  };
7
8  int main()
9  {
10     myClass cls;
11
12     cls.name = "王小明";
13     cls.age = 20;
14
15     cout << "姓名：" << cls.name << endl;
16     cout << "年齡：" << cls.age << endl;
17 }

```

程式碼第 8-17 行為主函式 `main()`，第 10 行宣告 `myClass` 類別的物件 `cls`，第 12 行設定物件 `cls` 的資料成員 `name` 等於 "王小明"，第 13 行設定資料成員 `age` 等於 20。第 15-16 行分別顯示物件的 2 個資料成員 `cls.name` 與 `cls.age`。

在使用 Visual Studio C++ 編寫程式時，當輸入程式敘述 `cls.` 的時候，便會出現提示 `public` 區段內可供存取的成員，如下圖所示。這樣的提醒機制提供在撰寫程式的時候，用於檢查類別成員是否宣告在正確的區段內。



private 區段

放置於 **private**（私有）區段內的資料成員只能在類別裡使用；也就是整支程式都不能讀取與修改宣告在 **private** 區段內的資料成員。因此，若類別中有不可隨意被讀取與修改的資料成員，以及只供類別內部使用的自訂函式，都可以宣告在此區段內。此區段不可被繼承，請參考範例 29。

例如，定義一個記錄學生姓名與年齡的類別 **myClass1**，如下所示。程式碼第 1-6 行定義類別 **myClass1**，並宣告 **private** 區段。區段裡面宣告了 2 個資料成員 **name** 與 **age**，分別表示姓名與年齡；並且 **age** 的初始值等於 0。

```

1  class myClass1
2  {
3      private:
4          string name;
5          int age = 0;
6  };
7
8  int main()
9  {
10     myClass cls;
11
12     cls.name = "王小明";
13     cls.age = 20;
14 }
```

← 錯誤，因為此 2 個資料成員宣告在 **private** 區段。

程式碼第 12-13 行會發生錯誤，因為資料成員 **name** 與 **age** 宣告在 **private** 區段內，所以無法供類別本身以外的程式敘述使用。因此，要特別撰寫成員函式來存取這些宣告在 **private** 區段內的資料成員。

protected 區段

放置於 **protected**（保護）區段內的資料成員除了無法被類別以外的程式所存取之外，也能被繼承的類別使用，請參考範例 29。因此，當類別裡的資料成員可能會被繼承，並且被繼承之後也只允許在類別裡使用，便要宣告在 **protected** 區段之內。

🔗 練習 2：定義三角形類別

定義一個三角形的類別，包含 3 個資料成員：底邊、高度與面積。底邊與高度放置於 `public` 區段，面積放置於 `private` 區段。使用此三角形類別宣告物件，並輸入三角形的底邊與高度。

解說

練習 2 與練習 1 所定義的三角形類別相同，只多了要輸入三角形的底邊與高度。因此，讀取輸入的底邊與高度的程式敘述應如下之形式：

```
1 Triangle tri;
2     :
3 cin >> tri.base;    // 讀取資料並儲存於物件 tri 的資料成員 base
4 cin >> tri.height;  // 讀取資料並儲存於物件 tri 的資料成員 height
```

程式碼第 1 行宣告 `Triangle` 類別的物件 `tri`。第 3-4 行讀取輸入的三角形底邊與高度，並分別儲存於物件 `tri` 的資料成員 `base` 與 `height`。

執行結果

```
輸入三角形的底邊：2.6
輸入三角形的高度：4.2
三角形的底邊為 2.6，高度為 4.2
```

程式碼列表

```
1 #include <iostream>
2 using namespace std;
3
4 class Triangle
5 {
6     private:
7         double size;
8
9     public:
10        double base;
11        double height;
12 };
13
14 int main()
15 {
16     Triangle tri;
```



```

17
18     cout << " 輸入三角形的底邊：";
19     cin >> tri.base;
20
21     cout << " 輸入三角形的高度：";
22     cin >> tri.height;
23
24     cout << " 三角形的底邊為 " << tri.base;
25     cout << "，高度為 " << tri.height << endl;
26
27     system("pause");
28 }

```

程式講解

1. 程式碼第 1-2 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼 4-12 行定義三角形類別 `Triangle`。類別裡有 `public` 與 `private` 此 2 個區段，並有 3 個 `double` 資料型別的資料成員；`private` 區段裡有代表三角形面積的資料成員 `size`，而代表三角形底邊長與高度的 `base` 與 `height` 則位於 `public` 區段。
3. 程式碼第 16 行宣告 `Triangle` 類別的物件 `tri`。
4. 程式碼第 18-19 行顯示輸入三角形底邊的提示訊息，將輸入的三角形底邊儲存於物件 `tri` 的資料成員 `base`。
5. 程式碼第 21-22 行顯示輸入三角形高度的提示訊息，將輸入的三角形高度儲存於物件 `tri` 的資料成員 `height`。
6. 程式碼第 24-25 行顯示三角形物件 `tri` 的底邊 `tri.base` 與高度 `tri.height`。

27-3 成員函式

類別裡的成員函式即是定義在類別裡的自訂函式；因此，宣告與撰寫成員函式的方式與自訂函式相同；只是這些成員函式的用途是作為存取類別裡的資料成員，以及與類別相關的操作與運算。

宣告於 `public` 區段的成員函式，可被類別外的程式敘述呼叫，也可以被其他的類別繼承。宣告於 `private` 區段的成員函式，不可被類別外的程式敘述呼叫，也無法被其他類別繼承。宣告於 `protected` 區段的成員函式，不可被類別外的程式敘述呼叫，但可被其他的類別繼承，並且會自動轉變成 `private` 區段的性質。

成員函式的位置

成員函式可以直接撰寫於類別之內；或是在類別內只宣告成員函式的原型，然後在類別之外再撰寫成員函式的程式本體。

宣告成員函式在類別之內

例如，練習 2 的三角形類別 `Triangle`，現在增加 1 個用於計算三角形面積的成員函式 `compute()`，如下所示：

```

1  class Triangle
2  {
3      private:
4          double size; // 三角形面積
5
6      public:
7          double base;    // 三角形底邊
8          double height;  // 三角形高度
9
10         double compute() // 計算並回傳三角形面積
11         {
12             size = base * height / 2;
13             return size;
14         }
15 };

```

程式碼第 10-14 行宣告計算三角形面積的成員函式 `compute()`，第 12 行計算三角形面積，並將計算結果儲存於資料成員 `size`，第 13 行回傳三角形面積 `size`。因為成員函式 `compute()` 宣告於 `public` 區段，因此可由類別以外的程式敘述呼叫。

在主函式 `main()` 中，程式碼第 3 行宣告 `Triangle` 類別的物件 `tri`，第 4 行宣告 `double` 型別的變數 `size`，用於儲存三角形的面積。當第 6、7 行讀取並設定了三角形的底邊與高之後，呼叫 `compute()` 成員函式取得三角形的面積，並儲存於資料變數 `size`；如下程式碼第 8 行所示。

```

1  int main()
2  {
3      Triangle tri;
4      double size;
5
6      cin >> tri.base;
7      cin >> tri.height;
8      size = tri.compute(); ← 呼叫 compute() 成員函式
9  }                               計算三角形面積。

```

宣告成員函式員在類別之外

若將所有成員函式的程式本體都寫在類別之內，則會使得類別內的程式敘述過長，造成不容易閱讀與維護；因此，有時只會將成員函式的原型宣告在類別之內，而成員函式的程式本體則寫於類別之外。例如：

```

1  class Triangle
2  {
3      private:
4          double size; // 三角形面積
5
6      public:
7          double base;    // 三角形底邊
8          double height;  // 三角形高度
9
10         double compute(); ← 成員函式的原型宣告
11     };
12     :
13
14     double Triangle:: compute()
15     {
16         size = base * height / 2; ← 成員函式的程式本體
17         return size;              寫在類別之外。
18     }

```

程式碼第 1-11 行為類別 `Triangle` 的定義，第 10 行只是成員函式 `compute()` 的函式原型宣告而已；而成員函式 `compute()` 的程式本體則在類別定義之外，如第 14-18 行。

因為成員函式的程式本體寫於類別之外，因此第 14 行在函式名稱 `compute` 之前需要使用範圍運算子 `::` 標明函式 `compute()` 是屬於 `Triangle` 類別。因此，在自訂函式 `compute()` 的前面才需要加上 `"Triangle::"`。

範圍運算子 `::` 標明函式 `compute()`
是屬於 `Triangle` 類別。

```

14 double Triangle:: compute()

```

↑
函式回傳值型別

friend 函式

類別 `private` 區段內的成員無法被類別之外的程式敘述所使用，因此存取這些 `private` 區段內的成員便顯得麻煩。若將類別之外的自訂函式，在類別中以 `friend` 修飾字修飾並宣告在類別中，便可以存取類別 `private` 區段中的成員。例如：自訂函式 `getSize()` 宣告在類別 `Triangle` 的 `public` 區段內，並以修飾字 `friend` 修飾；如下程式碼第 10 行所示：

```

1  class Triangle
2  {
3      private:
4          double size;
5
6      public:
7          double base;
8          double height;
9
10     friend double getSize(Triangle);
11 }
```

在類別之外，宣告自訂函式 `getSize()` 的程式本體，如程式碼第 13-16 行。`getSize()` 自訂函式接收一個 `Triangle` 類別的參數 `t`，第 15 行回傳 `t` 的私有資料成員 `size`。

原本 `Triangle` 的資料成員 `size` 因為宣告在 `private` 區段，是無法被類別以外的程式敘述所使用。因為自訂函式 `getSize()` 在類別中宣告並以 `friend` 修飾字修飾，因此在自訂函式 `getSize()` 中便可以使用 `size` 資料成員。

```

13 double getSize(Triangle t)
14 {
15     return t.size;
16 }
17
18 int main()
19 {
20     Triangle tri;
21     double size;
22
23     cin >> tri.base;
24     cin >> tri.height;
25     size = tri.compute();
26     cout << " 三角形面積 =" << getSize(tri) << endl;
27 }
```

程式碼第 20 行宣告 `Triangle` 類別的物件 `tri`，第 23-25 行分別取得輸入的三角形底邊與高度，並計算三角形面積。第 26 行呼叫自訂函式 `getSize()` 並傳入引數 `tri`，便能取得三角形的面積。

`friend` 函式的宣告與程式本體也能直接撰寫於類別之內。`friend` 函式可以很方便地存取類別裡 `private` 區段內的成員；然而，也會造成類別封裝與資料安全上的問題；請參考本範例的分析與討論。

練習 3：投票選班長

班上有 50 位學生，推出王小明、真美麗與李小強此 3 位班長候選人。寫一程式，定義投票選班長的類別，並利用亂數模擬全班投票的結果。

■ 解說

定義候選人的類別 `Elect`，其架構如下之形式。類別中有 `private` 與 `public` 區段，在 `private` 區段內有 1 個資料成員：得票數。在 `public` 區段內有 3 個成員：1 個 `string` 型別的資料成員為候選人的姓名；並有 2 個成員函式，分別用於投票與取得得票數。

```

1  class Elect
2  {
3      private:
4          int 得票數 = 0;
5
6      public:
7          string 姓名;
8
9          void 投票 ()
10         {
11             將得票數加 1;
12         }
13
14         void 取得得票數 ()
15         {
16             回傳得票數;
17         }
18 };

```

因為有 3 位候選人，所以可以宣告 `Elect` 類別的一維陣列物件 `candidate`，如下所示。第 1 位候選人為 `candidate[0]`，第 2 位候選人為 `candidate[1]`，第 3 位候選人為 `candidate[2]`。

```
Elect candidate[3];
```

執行結果

王小明：得票 = 18
真美麗：得票 = 19
李小強：得票 = 13

程式碼列表

```
1 #include <iostream>
2 #include <time.h>
3 using namespace std;
4
5 class Elect
6 {
7     private :
8         int vote_num = 0; // 得票數
9
10    public:
11        string name; // 姓名
12        void vote()
13        {
14            vote_num++;
15        }
16
17        int getVoteNum()
18        {
19            return vote_num;
20        }
21 };
22
23 int main()
24 {
25     Elect candidate[3];
26     int no;
27
28     srand((unsigned)time(NULL));
29
30     candidate[0].name = " 王小明 ";
31     candidate[1].name = " 真美麗 ";
32     candidate[2].name = " 李小強 ";
33
34     for (int i = 0; i < 50; i++)
```

```

35     {
36         no = rand() % 3;
37         candidate[no].vote();
38     }
39
40     for (int i = 0; i < 3; i++)
41         cout << candidate[i].name << "：得票 = " <<
42         candidate[i].getVoteNum() << endl;
43
44     system("pause");
45 }

```

程式講解

1. 程式碼第 1-3 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 5-21 行定義候選人的類別 `Elect`，在 `private` 區段裡有 1 個型別為整數的資料成員 `vote_num`，其初始值等於 0，作為候選人的得票數。`public` 區段內有 3 個成員，1 個資料型別為 `string` 的資料成員 `name`，作為候選人的姓名。另外有 2 個成員函式 `vote()` 與 `getVoteNum()`，分別用於累加得票數以及回傳得票數。
第 12-15 行為成員函式 `vote()`，第 14 行將 `vote_num` 加 1，表示增加了 1 票。第 17-20 行為成員函式 `getVoteNum()`，第 19 行回傳 `vote_num` 給呼叫者。
3. 程式碼第 25 行宣告 `Elect` 類別的一維陣列物件 `candidate[3]`，用於表示 3 位候選人的得票情形。第 1-3 位候選人分別為 `candidate[0]`-`candidate[2]`。第 28 行使用 `srand()` 函式初始化亂數。第 30-32 行設定 3 位候選人的姓名。
4. 程式碼第 34-38 行為一個 `for` 重複敘述，用於模擬班上 50 位學生投票的情形。第 36 行使用 `rand()%3` 隨機取得數字 0-2 表示候選人的號碼，並儲存於變數 `no`，表示投票給第 `no` 位候選人。因此第 37 行呼叫第 `no` 候選人 `candidate[no]` 的成員函式 `vote()` 增加 1 票。
5. 程式碼第 40-42 行使用 `for` 重複敘述，呼叫每一位候選人 `candidate[i]` 的成員函式 `getVoteNum()` 取得並顯示得票數。

27-4 類別的指標物件與參考物件

類別所宣告的物件如同變數一般，可以宣告為指標型別或是參考型別的物件，其作用也相同；只是在操作上略有差異。

指標物件

宣告類別的指標物件，與宣告一般指標變數相同。以 27-3 節的 `Triangle` 類別為例，宣告 `Triangle` 類別的指標物件，如下所示。程式碼第 1 行宣告 `Triangle` 類別的物件 `tri`，以及指標物件 `ptrTri`。第 4 行將物件 `tri` 的位址設定給指標物件 `ptrTri`；因此，指標物件 `ptrTri` 指向物件 `tri`。

指標物件存取自己的成員時必須改用 `"->"` 運算子，如第 6-8 行所示。第 6-7 行設定指標物件 `ptrTri` 的資料成員 `base` 與 `height`，第 8 行呼叫成員函式 `compute()` 計算三角形的面積，並將回傳的三角形面積儲存於變數 `size1`；因此，`size1` 等於 100。

```

1 Triangle tri, *ptrTri;
2 double size1,size2;
3
4 ptrTri = &tri;
5
6 ptrTri->base = 20;
7 ptrTri->height = 10;
8 size1 = ptrTri->compute();
9
10 (*ptrTri).base = 4;
11 (*ptrTri).height = 6;
12 size2 = (*ptrTri).compute();

```

也可以使用取值運算子 `"*"` 的方式存取指標物件的成員，如上述程式碼第 10-12 行；先取得指標物件所指位置的物件 (`*ptrTri`)，然後再使用 `"."` 運算子存取類別成員。第 10-12 行設定指標物件 `ptrTri` 的資料成員 `base` 與 `height`，第 12 行呼叫成員函式 `compute()` 計算三角形的面積，並將回傳的三角形面積儲存於變數 `size2`；因此，`size2` 等於 12。

指標物件如同一般的指標變數，也能使用動態記憶體配置的方式，取得實際的類別實體，但是當不再使用此指標物件的時候，要使用 `delete` 釋放所佔用的空間（或稱為銷毀物件）；如下所示。程式碼第 1 行宣告 `Triangle` 類別的指標物件 `ptrTri`，第 3 行使用 `new` 配置空間給指標物件 `ptrTri`，並初始化 `ptrTri`。第 5-6 設定 `ptrTri` 的資料成員 `base` 與 `height`，第 7 行呼叫 `ptrTri` 的成員函式 `compute()` 取得並顯示三角形的面積。第 9 行使用 `delete` 釋放 `ptrTri` 所佔用的記憶體空間。

```

1 Triangle *ptrTri;
2
3 ptrTri = new Triangle();
4
5 ptrTri->base = 20;
6 ptrTri->height = 10;
7 cout << ptrTri->compute();
8
9 delete ptrTri;

```

參考物件

宣告類別的參考物件，也與宣告一般參考型別的變數相同。以 27-3 節的 `Triangle` 類別為例，宣告 `Triangle` 類別的參考物件，如下所示。程式碼第 1 行宣告 `Triangle` 類別的物件 `tri`，第 2 行宣告 `Triangle` 類別的參考型別的物件 `&refTri`，並且初始值指向物件 `tri`。參考型別的物件不能只有宣告物件，必須在宣告時一併設定初始值：即指向哪個物件。

```

1 Triangle tri;
2 Triangle &refTri = tri;
3 double size;
4
5 refTri.base = 20;
6 refTri.height = 10;
7 size = refTri.compute();

```

宣告參考物件同時，也必須設定初始值。

參考型別的物件存取其成員時，使用的方式與一般的物件相同，都使用 `"."` 運算子；如程式碼第 5-7 行所示。程式碼第 5-7 行使用參考型別的物件 `refTri` 設定三角形的底邊、高度，並使用成員函式 `compute()` 計算三角形的面積。

傳遞物件

將物件作為引數傳遞給自訂函式，或是將物件作為接收的參數；其預設的方式為傳值呼叫。當然也可以使用傳址呼叫與參考呼叫，就與一般自訂函式傳遞參數的方式相同。須留意的一點，當自訂函式以傳址呼叫的方式接收指標型別的物件參數，則在自訂函式內的指標物件參數就必須使用 `"->"` 運算子存取物件參數的成員；或是使用取值運算子 `"*"` 的方式存取物件參數的成員。

this 指標

當類別裡的成員函式所接收的參數名稱，與類別裡的資料成員名稱相同時，便無法區分得出來；要避免這樣的情形發生，除了不要使用相同的變數名稱之外，也可以使用 `this` 關鍵字來加以區隔。

this 指的是自己、本身的意思，是一個指向自己的指標；因此，在類別裡使用 **this** 關鍵字，**this** 便是指向類別本身。若使用類別宣告物件變數，**this** 便是指向物件本身。例如，定義 1 個簡單的類別 **myClass**，如下所示。

myClass 類別裡有 1 個位於 **private** 區段內的整數資料成員 **number**。在 **public** 區段內則有 2 個成員函式 **setNumber()** 與 **getNumber()**，分別用於設定資料成員 **number** 的值與取得其值。

```

1  class myClass
2  {
3      private:
4          int number;
5
6      public:
7          void setNumber(int number)
8          {
9              this->number = number;
10         }
11
12         int getNumber()
13         {
14             return number;
15         }
16     };

```

程式碼第 7-10 行為成員函式 **setNumber()** 的程式本體，接收 1 個整數型別的參數 **number**。因為參數的名稱與類別的資料成員的名稱都為 **number**，若要將參數 **number** 設定給資料成員 **number**，如下所示；根本無法區分哪個 **number** 是資料成員或是參數。

```

void setNumber(int number)
{
    number = number;
}

```

因此，在資料成員的名稱之前加上關鍵字 **this**，如程式碼第 9 行所示，表示 **this->number** 是資料成員，而 **number** 是參數。

```

    this->number = number;
    ↑           ↑
    資料成員    參數

```

🔄 練習 4：指標物件與參考物件 - 計算成績總分與平均

設計學生成績之類別，其資料成員為 `private` 區段內的總分與平均分數，`public` 區段內的姓名、國英數 3 科成績。有 2 個成員函式：計算總分與平均分數的函式，以及使用參考呼叫方式回傳總分與平均分數的函式。寫一程式，宣告學生成績類別的物件之後，將此物件以傳址呼叫的方式，傳遞給另一個自訂函式去設定學生姓名與成績，並計算與顯示總分與平均分數。

■ 解說

記錄學生成績之類別 `stuScore` 的架構，如下所示。成員函式 `compute()` 用於計算總分與平均分數，`getScore()` 則用於回傳總分與平均分數。因為需使用參考呼叫的方式傳遞參數，因此函式 `getScore()` 會有 2 個參考呼叫形式的參數 `&total` 與 `&avg`，分別表示總分與平均分數。

```
class stuScore
{
    private:
        int total;        // 總分
        double avg;       // 平均

    public:
        string name;      // 姓名
        int chi, math, eng; // 國英數成績；

        void compute(){...}
        void getScore(int &total, double &avg){...}
}
```

因為成員函式 `getScore()` 所使用的參數名稱與資料成員的名稱相同，在 `getScore()` 函式中要以 `this->total` 與 `this->avg` 表示資料成員，才不至於與參數混淆，如下所示：

```
void getScore(int &total, double &avg)
{
    total = this->total;
    avg = this->avg;
}
```

■ 執行結果

```
姓名 = 王小明
國文 = 90, 數學 = 82, 英文 = 93
總分 = 265, 平均 = 88.3333
```

程式碼列表

```
1  #include <iostream>
2  #include <fstream>
3  using namespace::std;
4
5  class stuScore
6  {
7      private:
8          int total = -1;
9          double avg = -1;
10
11     public:
12         string name;
13         int chi = -1, math = -1, eng = -1;
14
15         void compute()
16         {
17             total = chi + math + eng;
18             avg = (double)total / 3.0;
19         }
20
21         void getScore(int& total, double& avg)
22         {
23             total = this->total;
24             avg = this->avg;
25         }
26 };
27
28 void getData(stuScore* stu)
29 {
30     stu->name = "王小明 ";
31     stu->chi = 90;
32     stu->math = 82;
33     stu->eng = 93;
34
35     stu->compute();
36 }
37
38 int main()
39 {
40     stuScore stut;
41     int total;
```

```

42     double avg;
43
44     getData(&stut);
45     stut.getScore(total, avg);
46
47     cout << " 姓名 = " << stut.name << endl;
48     cout << " 國文 = " << stut.chi << ", ";
49     cout << " 數學 = " << stut.math << ", ";
50     cout << " 英文 = " << stut.eng << endl;
51
52     cout << " 總分 = " << total << ", ";
53     cout << " 平均 = " << avg << endl;
54
55     system("pause");
56 }

```

程式講解

1. 程式碼第 1-3 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 5-26 行定義學生成績的類別 `stuScore`。在 `private` 區段內有 2 個資料成員 `total` 與 `avg`，分別表示總分與平均分數。在 `public` 區段有 4 個資料成員 `name`、`chi`、`math` 與 `eng`，分別代表姓名、國英數 3 科成績。在 `public` 區段內還有 2 個成員函式：函式 `compute()` 用於計算總分與平均分數，函式 `getScore()` 回傳總分與平均分數。

第 15-19 行為成員函式 `compute()` 的程式本體，用於計算總分與平均分數。第 17 行計算總分，將國英數 3 科成績相加後儲存於變數 `total`。第 18 行計算平均分數。因為總分 `total` 為整數型別，但計算平均分數有可能會出現小數；因此，先將總分 `total` 強制轉型為 `double` 型別後，再除以 3 計算平均分數，並儲存於變數 `avg`。

第 21-25 行為成員函式 `getScore()` 的程式本體，用於回傳總分與平均分數。此函式接收 2 個參考呼叫的參數 `&total` 與 `&avg`。第 23-24 將資料成員的 `this->total` 與 `this->avg` 設定給參數的 `total` 與 `avg`；此 2 個參數的名稱與資料成員的總分 `total` 和平均分數 `avg` 的名稱相同；因此，資料成員的名稱前面必須加上 `this` 關鍵字才能區分資料成員與參數。

3. 程式碼第 28-36 行為自訂函式 `getData()` 的程式本體，此函式用於設定學生資料，並接收一個 `stuScore` 類別的指標物件 `stu`，因此是以傳址呼叫的方式傳遞參數。第 30-33 行設定學生資料給參數 `stu`；因為 `stu` 是指標形式的物件參數；所以 `stu` 物件的成員都必須使用 `"->"` 運算子。第 35 行呼叫 `stu` 的成員函式 `compute()` 計算總分與平均分數。

4. 程式碼第 40-42 行宣告變數，`stuScore` 類別的物件 `stut` 用於表示學生的資料，變數 `total` 與 `avg` 分別表示總分與平均分數。第 44 行呼叫自訂函式 `getData()`，並以傳址呼叫的方式將物件 `stut` 的位址作為引數。第 45 行呼叫物件 `stut` 的成員函式 `getScore()`，並以參考呼叫的方式傳入變數 `total` 與 `avg`，藉以取得計算之後的總分與平均分數。第 47-53 行顯示學生的資料。

27-5 靜態成員

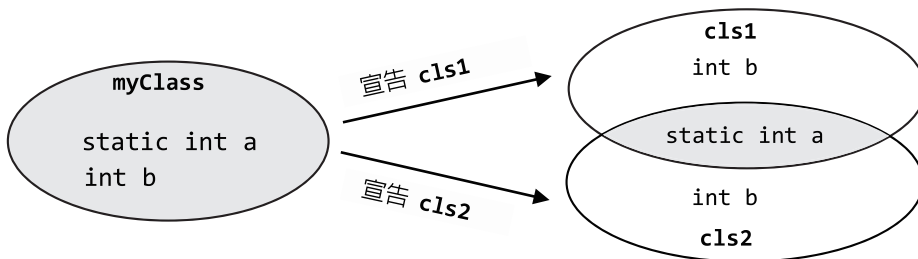
由類別宣告的多個物件，雖然是由同一個類別所衍生出來的實體，但各有自己所擁有的獨立記憶體空間，因此也有各自的資料成員與成員函式；所以每個物件彼此並無關連。

但若以靜態修飾字修飾的類別成員稱為靜態成員，則不管類別宣告了多少的物件，這些物件並不會有各自的靜態成員，而是全部的物件都共用這一個靜態的成員；這個靜態成員可以是資料成員或是成員函式。

以靜態資料成員為例，例如：類別 `myClass` 的定義如下程式碼第 1-6 行所示；`public` 區段內有 2 個 `int` 型別的資料成員 `a` 與 `b`，並且以 `static` 修飾字修飾資料成員 `a`。

```
1 class myClass
2 {
3     public:
4         static int a;
5         int b;
6 };
7     :
8 myClass cls1, cls2;
```

程式碼第 8 行以類別 `myClass` 宣告 2 個物件 `cls1` 與 `cls2`，則此 2 個物件的記憶體關係如下所示。物件 `cls1` 與 `cls2` 有各自的記憶體空間，也有各自的資料成員 `b`；但卻共用相同的資料成員 `a`。



換句話說，在物件 `cls1` 設定了資料成員 `a` 的值之後，物件 `cls2` 的資料成員 `a` 的值也會更改。若改變物件 `cls2` 的資料成員 `a`，則物件 `cls1` 的資料成員 `a` 也會被改變；因為 2 個物件各自認為自己的資料成員 `a`，其實是同一個資料成員 `a`。

靜態資料成員

靜態資料成員可以宣告在類別中的 `public`、`private` 或是 `protected` 區段內，並且仍然具有在其區段內之特性。唯需要注意的是靜態資料成員無法在宣告時設定初始值；並且在沒有設定初始值之前，也無法存取此靜態資料成員。換句話說，在未替靜態資料成員設定初始值之前，是無法使用此靜態資料成員。

如下所示，程式碼第 1-11 行為類別 `myClass` 的定義，在 `public` 區段內有 1 個靜態的 `int` 型別的資料成員 `a`，如第 4 行所示；但第 4 行會發生錯誤，因為在類別之內靜態資料成員不能設定初始值。原因如下所說明：

若使用此類別宣告了物件 `clsA`，並將資料成員 `a` 設定等於 `10` 之後，再宣告另一個物件 `clsB`，此時資料成員 `a` 又被初始化一次，因為靜態資料成員 `a` 是 `clsA` 與 `clsB` 所共用的，此時 `clsA` 之前針對資料成員 `a` 設定為 `10` 就無效了；這便是為什麼不允許靜態的資料成員在類別內設定初始值的原因。

```

1  class myClass
2  {
3      public:
4          static int a=0;
5          int b;
6
7          void add()
8          {
9              a++;
10         }
11 };
12
13 int main()
14 {
15     myClass cls;
16
17     cls.a += 10;
18     cls.add();
19 }
```

錯誤，靜態資料成員不可以在類別內設定初始值。

錯誤，靜態資料成員在沒有設定初始值之前無法存取。

因此，在當靜態的資料成員 **a** 沒有被設定初始值之前，針對資料成員 **a** 的存取自然是錯誤的，所以上述程式碼第 17-18 行會發生錯誤。第 18 行發生錯誤的原因是成員函式 **add()** 的程式碼中有針對資料成員 **a** 做處理，如上述第 9 行所示。

既然無法在類別內設定靜態資料成員的初始值，所以只能在類別之外設定靜態資料成員的初始值；如下所示：程式碼第 13 行針對 **myClass** 類別的靜態資料成員 **a** 設定初始值。因為資料成員 **a** 的資料型別為 **int** 型別，因此在類別名稱 **myClass** 之前加上了資料型別 **int**。

```

1  class myClass
2  {
3      public:
4          static int a;
5          int b;
6
7          void add()
8          {
9              a++;
10         }
11     };
12
13     int myClass::a = 0; ← 對靜態資料成員 a 設定初始值。
14
15     int main()
16     {
17         myClass cls;
18
19         cls.a = 10;
20         cls.add();
21     }

```

練習 5：儲存旅遊經費

Joanna 和 Leo 兩人一起儲存 50000 元的旅遊經費。使用類別定義與儲存旅遊經費相關的資料與函式；並寫一程式持續輸入兩人的存款，直到兩人的共同存款超過旅遊經費為止。

■ 解說

在定義類別之前，要先規劃類別的內容有哪些資料成員或是成員函式。因此，要定義共同存款的類別 **trip**，應該會有這些資料成員：預定達成的存款目標 **amount**、目前的共同存款金額 **money**、個人累計的存款金額 **individual**。並且，因為是 2 個人一起存款，所以將資料成員 **money** 設定為靜態型別；所以無論是誰存錢都會增加到這個資料成員。**amount** 的值不可以被改變，所以加上 **const** 修飾字修飾。

並且將資料成員 `amount` 與 `money` 放置於 `private` 區段，要存取資料成員 `money` 則需透過成員函式。將資料成員 `individual` 放置於 `public` 區段，以方便取得其值。

而成員函式則提供：存款 `deposit()`、檢查是否達到存款目標 `check()` 這 2 個函式。成員函式 `deposit()` 接收一個整數型別的參數，用於表示要存錢的金額。並且設計 `check()` 函式若是 2 人的共同存款已經達到預定的金額目標，則回傳 0；否則回傳差額。

因此，2 人同存款的類別 `trip` 的架構，如下所示。

```
class trip
{
    private:
        const int amount = 50000;    // 預定達成的存款目標
        static int money;            // 目前的共同存款金額

    public:
        int individual = 0;          // 個人累計的存款金額

        void deposit(int money)      // 存款
        {
        }

        int check()                  // 檢查是否達到存款目標
        {
        }
}
```

執行結果

```
差額：50000
誰要存錢 (0:Joanna 1:Leo)：0
存多少錢：20000

差額：30000
誰要存錢 (0:Joanna 1:Leo)：1
存多少錢：20000

差額：10000
誰要存錢 (0:Joanna 1:Leo)：0
存多少錢：10000

已經達到金額
Joanna 儲存了：30000
Leo 儲存了：20000
```

程式碼列表

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4
5  class trip
6  {
7      private:
8          const int amount=50000;
9          static int money;
10
11      public:
12          int individual=0;
13
14          void deposit(int money)
15          {
16              this->money += money;
17              individual += money;
18          }
19
20          int check()
21          {
22              if (money >= amount)
23                  return 0;
24              else
25                  return amount - money;
26          }
27 };
28
29 int trip::money=0;
30
31 int main()
32 {
33     trip Joanna, Leo;
34     int sel;
35     int money, diff;
36
37     diff = Leo.check();
38
39     while (diff!=0)
40     {
41         cout << " 差額：" << diff << endl;
```

```

42
43     cout << " 誰要存錢 (0:Joanna 1:Leo) : ";
44     cin >> sel;
45     if (sel == 0 || sel == 1)
46     {
47         cout << " 存多少錢 : ";
48         cin >> money;
49         cout << endl;
50     }
51
52     switch (sel)
53     {
54         case 0:
55             Joanna.deposit(money);
56             break;
57
58         case 1:
59             Leo.deposit(money);
60             break;
61
62         default:
63             cout << " 輸入錯誤，重新輸入 ";
64             break;
65     }
66
67     diff = Leo.check();
68 }
69
70 cout << " 已經達到金額 " << endl;
71 cout << "Joanna 儲存了 : " << Joanna.individual << endl;
72 cout << "Leo 儲存了 : " << Leo.individual << endl;
73
74 system("pause");
75 }

```

程式講解

1. 程式碼第 1-3 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 5-27 行定義儲存旅遊經費的類別 `trip`。在 `private` 區段內有 2 個資料成員：`amount` 與 `money`；分別代表要達成的存款目標，以及目前儲存的金額。`amount` 初始值等於 50000，並以 `const` 修飾字修飾，所以 `amount` 的值不可以被更改。`money` 是 Joanna 和 Leo 共同儲存的旅遊費用，所以使用 `static` 修飾字修飾。

在 `public` 區段裡有 1 個資料成員 `individual`，用於表示個人累積的存錢金額。程式碼第 14-18 行為成員函式 `deposit()` 的程式本體，接收一個整數型別的參數 `money`；此函式用於累加旅遊經費。第 16 行將參數 `money` 累加到資料成員 `this->money`，第 17 行也將參數 `money` 累加到個人的存錢金額 `individual`。

第 20-26 行為成員函式 `check()` 的程式本體，此函式用於判斷 2 人所存的錢是否已經達到預定的旅遊經費。第 22-25 行判斷若資料成員 `money` 大於或等於資料成員 `amount`，表示 2 人所存的錢已經達到或超過預定的旅遊經費，則回傳 0；否則回傳還差了多少錢才能達到預定的旅遊經費。

3. 程式碼第 29 行設定類別 `trip` 的靜態資料成員 `money` 的初始值。
4. 程式碼第 33 行使用 `trip` 類別宣告 2 個物件 `Joanna` 與 `Leo`，表示 2 人的存款金額。第 35 行整數變數 `money` 和 `diff` 用於表示要存款的金額以及目前的共同存款與旅遊經費的差額。第 37 行先使用物件 `Leo` 呼叫成員函式 `check()` 取得目前的差額，並儲存於變數 `diff`。
5. 程式碼第 39-68 行為 `while` 重複敘述，當差額 `diff` 不等於 0 時則反覆執行。第 41 行先顯示目前的差額，第 43-44 行讀取是誰要存錢的選擇，並儲存到變數 `sel`；第 45-50 行當輸入的 `sel` 值正確時，再次輸入要儲存多少錢，並儲存到變數 `money`。
6. 程式碼第 52-65 行為 `switch...case` 選擇敘述，根據變數 `sel` 的值執行相對應的功能。第 54-56 行為 `Joanna` 要存錢，將輸入的金額 `money` 當成引數，呼叫 `Joanna` 物件的成員函式 `deposit()`。第 58-60 行為 `Leo` 要存錢，將輸入的金額 `money` 當成引數，呼叫 `Leo` 物件的成員函式 `deposit()`。

因為 `Trip` 類別中的資料成員 `money` 為靜態型別，因此無論是 `Joanna` 或是 `Leo` 存錢，都會增加到這個靜態資料成員 `money`。第 67 行使用 `Leo` 物件的成員函式 `check()` 取得差額，並儲存到變數 `diff`。

7. 當 2 人的存款達到旅遊經費時，便會離開第 39-68 行的 `while` 重複敘述，執行第 70-72 行顯示存款已經達到旅遊經費的訊息，並各自顯示自己累計的存款金額 `Joanna.individual` 與 `Leo.individual`。

靜態成員函式

成員函式也能宣告為靜態型別，如同靜態的資料成員一般，是屬於此類別所宣告的所有物件共用；因此，靜態的成員函式不需要藉由宣告類別物件之後才能被呼叫，在使用上更顯得方便。但須注意：靜態成員函式內只能使用靜態型別的資料成員，無法使用類別內一般的資料成員；並且，在靜態成員函式內無法使用 `this` 指標。

例如：設計計算圓形面積的類別 `circleSize`；如下所示。程式碼第 1-11 行定義類別 `circleSize`。在 `private` 區段內宣告靜態資料成員 `PI`，為圓周率的值，其初始值定義在第 13 行。在 `public` 區段內宣告靜態型別的成員函式 `getSize()`，用於計算圓形的面積。此函式接收一個 `double` 型別的參數 `radius`，為圓形的半徑。

```

1  class circleSize
2  {
3      private:
4          static double PI;
5
6      public:
7          static double getSize(double radius)
8          {
9              return radius * radius* PI;
10         }
11 };
12
13 double circleSize::PI = 3.14159;
14
15 int main()
16 {
17     circleSize cir;
18
19     cout << cir.getSize(12.2) << endl;
20
21     cout << circleSize::getSize(2.31) << endl;
22 }
```

直接使用類別的靜態成員函式。

程式碼第 17 行使用類別 `circleSize` 宣告物件 `cir`，第 19 行呼叫物件 `cir` 的成員函式 `getSize()` 計算半徑等於 12.2 的圓形面積；這樣的方式不僅需要花費時間與記憶體空間建立類別的物件 `cir`，然後再透過物件 `cir` 呼叫其函式 `getSize()` 計算圓形的面積。

因為類別 `circleSize` 的成員函式 `getSize()` 為靜態型別；因此使用第 21 行的方式會更直接、方便：直接執行 `circleSize::getSize()` 計算半徑等於 2.31 的圓形面積。

🔗 練習 6：設計計算面積之靜態成員函式

設計一個類別，提供計算三角形與矩形面積之靜態成員函式。

■ 解說

假設此類別為 `computeSize`，提供計算三角形與矩形面積的成員函式分別為 `triSize()` 與 `rectSize()`；則類別 `computeSize` 之結構如下所示。在類別中只有一個 `public` 區段，在此區段內有 2 個靜態的成員函式：`triSize()` 與 `rectSize()`。

```
class computeSize
{
    public:
        static double triSize(...) // 計算三角形面積
        {
        }

        static double rectSize(...) // 計算矩形面積
        {
        }
}
```

■ 執行結果

三角形之底邊 = 10，高 = 3，面積 = 15
 矩形之寬 = 3.4，高 = 6，面積 = 20.4

■ 程式碼列表

```
1 #include <iostream>
2 using namespace std;
3
4 class computeSize
5 {
6     public:
7         static double triSize(double base, double height)
8         {
9             return base * height /2.0;
10        }
11
12        static double rectSize(double width, double height)
13        {
```

```

14         return width * height;
15     }
16 };
17
18 int main()
19 {
20     double base = 10, height1 = 3;
21     double width = 3.4, height2 = 6;
22
23     cout << " 三角形之底邊 = " << base << ", 高 = " << height1;
24     cout << ", 面積 = " << computeSize::triSize(base, height1) << endl;
25
26     cout << " 矩形之寬 = " << width << ", 高 = " << height2;
27     cout << ", 面積 = " << computeSize::rectSize(width, height2) << endl;
28
29     system("pause");
30 }

```

程式講解

1. 程式碼第 1-2 行引入所需要的標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 4-16 行定義 `computeSize` 類別。類別裡只有 `public` 區段，此區段內有 2 個靜態型別的成員函式：`triSize()` 與 `rectSize()`，前者用於計算三角形的面積，後者用於計算矩形的面積。
3. 程式碼第 7-10 行為靜態成員函式 `triSize()` 的程式本體，接收 2 個 `double` 型別的參數 `base` 與 `height`，分別表示三角形的底邊與高。第 9 行計算並回傳三角形的面積。第 12-15 行為靜態成員函式 `rectSize()` 的程式本體，接收 2 個 `double` 型別的參數 `width` 與 `height`，分別表示矩形的寬與高。第 14 行計算並回傳矩形的面積。
4. 程式碼第 20 行宣告 `double` 型別的變數 `base` 與 `height1`，分別表示三角形的底邊與高。第 21 行宣告 `double` 型別的變數 `width` 與 `height2`，分別表示矩形的底邊與高。

第 23 行顯示三角形的底邊 `base` 與高 `height1`。第 24 行呼叫類別的成員函式 `computeSize::triSize()`，並傳入變數 `base` 與 `height1` 作為引數，計算並取得三角形的面積。第 26 行顯示矩形的寬 `width` 與高 `height2`。第 27 行呼叫類別的成員函式 `computeSize::rectSize()`，並傳入變數 `width` 與 `height2` 作為引數，計算並取得矩形的面積。

三、範例程式解說

1. 建立專案，程式碼第 1-4 行引入所需要的標頭檔與宣告使用 `std` 命名空間。

```

1 #include <iostream>
2 #include <math.h>
3 #include <time.h>
4 using namespace std;

```

2. 程式碼第 6-33 行定義攤位類別 `Booth`，類別中包含 `private` 與 `public` 區段。在 `private` 區段內有 2 個浮點數型別的資料成員：`rent` 與 `netProfit`。`rent` 為靜態型別，用於表示佔營收 10% 的租金。因為 `rent` 宣告為靜態型別，所有的攤位會共用此資料成員；所以當每個攤位將租金累加至資料成員 `rent`，就等同所有攤位要繳交給主辦單位的租金總和。`netProfit` 為營收扣除租金後的淨利。

在 `public` 區段內有 1 個資料成員與 3 個成員函式。程式碼第 13 行為浮點數型別的資料成員 `revenue`，代表營收。程式碼第 15-22 行的成員函式 `compute()` 用於計算租金與淨利，第 19 行計算營收的 10% 作為租金，並儲存於變數 `r`。第 20 行將租金 `r` 累加到靜態資料成員 `rent`。第 21 行計算淨利 `netProfit`。

```

6 class Booth
7 {
8     private :
9         static float rent; // 租金
10        float netProfit = 0; // 淨利
11
12    public:
13        float revenue = 0; // 營收
14
15        void compute() // 計算租金與淨利
16        {
17            float r;
18
19            r=revenue * 0.1;
20            rent += r;
21            netProfit = revenue - r;
22        }

```

第 24-27 行的成員函式 `getNetprofit()` 用於回傳淨利；第 26 行回傳資料成員 `netProfit`。第 29-32 行的成員函式 `getRent()` 用於回傳租金；第 31 行回傳靜態資料成員 `rent`。第 35 行在類別 `Booth` 的定義之外，設定其靜態資料成員 `rent` 的初始值等於 0。

```

24         float getNetprofit() // 回傳淨利
25     {
26         return netProfit;
27     }
28
29     static float getRent() // 回傳租金
30     {
31         return rent;
32     }
33 };
34
35 float Booth::rent = 0;

```

3. 開始於 `main()` 主函式中撰寫程式，程式碼第 39-40 行宣告變數。因為尚未決定有多少的攤位，因此第 39 行只宣告了類別的指標物件 `booths`，並將初始值設定為 `NULL`，表示尚未初始化與配置記憶體空間。第 40 行宣告整數變數 `number`，表示攤位的數量。第 42 行使用函式 `srand()` 初始化亂數產生器。

```

39 Booth *booths=NULL;
40 int number; // 攤位數量
41
42 srand((unsigned)time(NULL));

```

4. 程式碼第 44-45 行顯示輸入攤位數量的提示訊息，以及將所輸入的攤位數量儲存於變數 `number`。第 46-53 為 `try...catch` 例外處理敘述。第 47 行使用 `new` 配置所需要的記憶體空間給指標物件 `booths`；若配置發生錯誤則執行第 50-53 行，顯示錯誤訊息並結束程式。

```

44 cout << " 攤位數量：";
45 cin >> number;
46 try {
47     booths = new Booth[number];
48 }
49 catch (...)
50 {
51     cout << " 攤位配置錯誤 " << endl;
52     exit(0);
53 }

```

5. 程式碼第 55 行設定小數顯示位數為 2 位數。第 56-62 行為 `for` 重複敘述，第 58 行使用 `rand()` 函式模擬第 `i` 個攤位介於 1-10 萬之間的營收，第 59 行呼叫第 `i` 個攤位的 `compute()` 成員函式計算攤位的租金與淨利。第 60-61 行呼叫第 `i` 個攤位的 `getNetprofit()` 成員函式取得與顯示淨利。

```

55  cout.precision(2);
56  for(int i=0;i<number;i++)
57  {
58      booths[i].revenue = rand() % 10 + 1;
59      booths[i].compute();
60      cout << "攤位 " << i + 1 << " 淨利 = " << \
61      fixed << booths[i].getNetprofit() << endl;
62  }
63
64  cout << "\n 租金收入：" << fixed << Booth::getRent() << endl;
65
66  delete[] booths;
67
68  system("pause");

```

第 64 行直接使用類別 `Booth` 的靜態成員函式 `getRent()` 取得並顯示主辦單位的租金總和。第 66 行使用 `delete` 釋放指標物件 `booths` 所佔用的記憶體空間。

重點整理

1. C++ 引入了類別的相關機制，使得撰寫程式的方式轉為物件導向程式設計；軟體的開發變得更方便與模組化。
2. 類別裡包含資料與函式，通稱為類別的成員。資料成員為宣告在類別內的常數與變數，成員函式則是用於操作這些資料成員，或與操作類別相關的自訂函式。
3. 類別裡的資料與函式都放置於 `private`、`public` 或 `protected` 其一區段內。`private` 區段內的成員只供類別內使用。`public` 區段內的成員可供類別之外使用。
4. 類別將資料與操作這些資料相關的函式包裹在一起，並且使用存取控制權限界定資料與函式是否可被存取；這樣的方式便是物件導向程式設計的第一個特色：資料封裝（Data encapsulation）。因為在不同的存取權限的區段內，資料與函式的存取權限不同，也產生了資料隱藏（Data hiding）的優點。
5. 類別內的靜態成員，無論類別產生多少個物件，靜態成員只有一份，並且這些物件都共用此靜態成員。

分析與討論

1. 物件導向程式設計雖然提供了許多傳統程式設計所沒有的優點，然而效率比較慢（C++ 經過這麼多年的改良，也有部分的人認為目前的執行效率也幾乎和 C 差不多了）、開發時間較長，也是普遍認為的缺點。

再加上如果只是撰寫簡單的程式、開發嵌入式系統，似乎只需要使用 C 語言就已經很足夠了。如同廚師一樣，會有許多的刀具，看要烹飪何種菜餚選擇適合的刀具。因此，使用 C 或是 C++ 語言、使用傳統程式設計還是要使用物件導向程式設計，都是要看當時的情形而定；混合 C 與 C++ 使用也是一個不錯的考量；本書中的範例便是採用如此的方式。

2. 在 27-3 節中所討論的 **friend** 函式，可以用於存取類別 **private** 區段中的成員，使得存取類別內的成員變得容易。然而，在類別內將成員分別宣告在不同的區段，就是為了資料隱藏、封裝；使存取類別內的資料趨於更安全，以防止類別之外的程式，沒有經過一定的步驟隨意取得類別中不安全的資料。例如：在尚未輸入購買物品的數量與計算金額之前，就要取得消費總金額；或輸入的物品數量不正常，卻要計算購買金額等。

再看一個更危險的例子，以 27-3 節所使用的計算三角形面積的類別 **Triangle** 為例，如下所示。程式碼第 1-16 行為類別 **Triangle** 的定義，其中第 10-13 行為計算三角形面積的函式 **compute()**，第 15 行是以 **friend** 修飾字修飾的自訂函式 **getSize()** 的原型宣告。

```

1  class Triangle
2  {
3      private:
4          double size;
5
6      public:
7          double base;
8          double height;
9
10         void compute()
11         {
12             size = base * height / 2;
13         }
14
15         friend double getSize(Triangle);
16 };

```

程式碼第 18-22 行為自訂函式 `getSize()` 的程式本體，並接收一個 `Triangle` 類別的參數 `t`；特別注意第 20 行將所傳入的物件 `t` 的私有成員 `size` 設定為 `-1`。

```

18 double getSize(Triangle t)
19 {
20     t.size = -1;
21     return t.size;
22 }

```

程式碼第 24-38 行為 `main()` 主函式，第 26 行宣告 `Triangle` 類別的物件 `tri`，第 28-32 行取得三角形的底邊與高度，第 34 行呼叫物件 `tri` 的成員函式 `compute()` 計算三角形的面積。第 35 行呼叫自訂函式 `getSize()`，並傳入物件 `tri` 作為引數，取得三角形的面積；但輸出的結果是 `-1`。

```

24 int main()
25 {
26     Triangle tri;
27
28     cout << " 輸入三角形的底邊：";
29     cin >> tri.base;
30
31     cout << " 輸入三角形的高度：";
32     cin >> tri.height;
33
34     tri.compute();
35     cout << " 三角形面積 =" << getSize(tri) << endl;
36
37     system("pause");
38 }

```

這是因為在自訂函式 `getSize()` 中，無意或是失誤地將 `Triangle` 類別的私有成員 `size` 設定為 `-1`，所以無論在成員函式 `compute()` 如何地計算，最後都會被程式碼第 20 行所覆蓋；這也是使用 `friend` 函式危險的地方。

要取得類別裡的私有成員，可以有更安全的方法，例如：提供取得私有資料成員的成員函式，如下程式碼第 15-18 行所示。當宣告 `Triangle` 類別的物件之後，再呼叫 `compute()` 計算面積，就可以呼叫 `obtainSize()` 取得計算後的面積。

```

1 class Triangle
2 {
3     private:
4         double size;
5
6     public:
7         double base;
8         double height;
9
10        void compute()
11        {
12            size = base * height / 2;
13        }
14
15        double obtainSize()
16        {
17            return size;
18        };
19 }

```

程式碼列表

```

1 #include <iostream>
2 #include <math.h>
3 #include <time.h>
4 using namespace std;
5
6 class Booth
7 {
8     private :
9         static float rent; // 租金
10        float netProfit = 0; // 淨利
11
12    public:
13        float revenue = 0; // 營收
14
15        void compute() // 計算租金與淨利
16        {
17            float r;
18
19            r=revenue * 0.1;
20            rent += r;

```

```

21         netProfit = revenue - r;
22     }
23
24     float getNetprofit() // 回傳淨利
25     {
26         return netProfit;
27     }
28
29     static float getRent() // 回傳租金
30     {
31         return rent;
32     }
33 };
34
35 float Booth::rent = 0;
36
37 int main()
38 {
39     Booth *booths=NULL;
40     int number; // 攤位數量
41
42     srand((unsigned)time(NULL));
43
44     cout << " 攤位數量 : ";
45     cin >> number;
46     try {
47         booths = new Booth[number];
48     }
49     catch (...)
50     {
51         cout << " 攤位配置錯誤 " << endl;
52         exit(0);
53     }
54
55     cout.precision(2);
56     for(int i=0;i<number;i++)
57     {
58         booths[i].revenue = rand() % 10 + 1;
59         booths[i].compute();
60         cout << " 攤位 " << i + 1 << " 淨利 = " << \
61             fixed << booths[i].getNetprofit() << endl;
62     }

```

```

63
64     cout << "\n 租金收入：" << fixed << Booth::getRent() << endl;
65
66     delete[] boothses;
67
68     system("pause");
69 }

```

本章習題

1. 設計一個代表營業員資料的類別。營業員資料有以下屬性：員工編號、姓名、電話與四季的業績（萬元計算）。四季的業績使用陣列表示，員工編號、姓名與電話放置於 **private** 區段，其餘的屬性放於 **public** 區段。使用此類別宣告物件，並顯示此物件佔用的空間容量。
2. 接續第 1 題。設計 2 個成員函式，一個函式用於增加營業員的資料，另一個函式則用於顯示營業員的資料。
3. 接續第 2 題，增加計算業績總和與平均的成員函式，以及取得業績總和與平均的成員函式。取得業績總和使用傳址呼叫的方式，取得業績平均使用參考呼叫的方式。
4. 寫一擲骰子的程式，骰子使用類別定義。輸入要擲多少顆的骰子，並動態配置所需數量的骰子物件。寫 2 個自訂函式，一個自訂函式接收骰子物件，並模擬擲骰子；另一個自訂函式顯示所有骰子的點數。
5. 寫一個類別，提供公斤與台斤互相轉換的靜態成員函式。
6. 類別 **myClass** 裡只有 **public** 區段。區段內有 2 個資料成員與 1 個成員函式。資料成員為字串型別的 **ID**，以及靜態型別的整數資料 **count**；成員函式為 **add()**，用於設定 **ID** 與累計 **count**。

寫一程式連續產生 10 個 **myClass** 類別的物件，並設定物件的 **ID** 為："ID: 1"、"ID: 2"… "ID: 10"。每產生一個物件，便將 **count** 累加 1，作為所產生的物件數量。最後顯示所有產生的物件的 **ID** 以及所產生的物件數量。

