

Example

30

樣板

有一抽獎活動，一共有 5 個獎項：小電鍋、行動電源、藍芽耳機、讀卡機與藍芽滑鼠。抽獎時沒有出席抽獎者、參加抽獎但沒有中獎者都能獲得鼓勵獎品：USB 隨身碟。請用類別樣板設計獎品的類別，再設計抽獎的類別來繼承獎品類別。寫一程式，輸入抽獎者姓名、是否有出席抽獎，並顯示抽獎所得之獎品。

一、學習目標

樣板（Template，或稱為範本）在不同的程式語言中被廣泛地使用；使用樣板所寫的程式，也稱為樣板程式或泛型程式。樣板可以加強程式語言傳遞參數時的彈性，使得參數傳遞時不用指定特定的資料型別。樣板由一個或多個的型別參數所組成；型別參數表示尚未確定的資料型別，直到編譯程式的時後才確定其資料型別。

因此，樣板免去了最麻煩的事情：寫不完的多載函式。樣板若使用於一般的函式，則稱為函式樣板；若使用於類別，就稱為類別樣板。

例如：自訂函式 `add()` 用於接收 2 個數值，並回傳此 2 個數值相加的結果。因此，可以呼叫自訂函式 `add()`，並傳入整數 2 與 3 執行 2 個整數相加；如下所示：

```
1 int add(int a, int b){...}
2     :
3 add(2, 3);
```

當要執行 2 個 `float` 型別的數值相加時，因為自訂函式 `add()` 所接收的是 2 個整數型別的參數；因此，便以多載的方式再宣告另 1 個自訂函式 `add()`，接收 2 個 `float` 型別的參數；如下程式碼第 2 行所示：

```
1 int add(int a, int b){...}
2 float add(float a, float b){...}
3     :
4 add(2, 3);
5 add(1.2, 5.6);
```

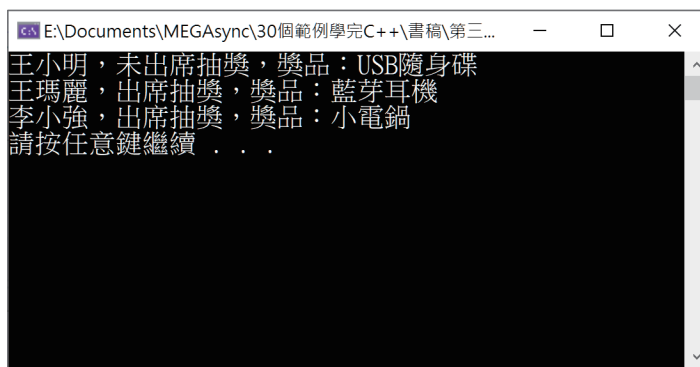
如果要執行相加的 2 個數，分別是 `double` 型別與整數型別的數值；或者又是別的资料型別的數值，或者是 3 個數值相加；如此一來，即使是定義再多的多載函式 `add()` 都難以應付。

因此，樣板有點像是函式的參數的资料型別定義；先建好一個通用的定義之後，然後讓自訂函式都依照這個定義去接收、使用這些參數。所以，當呼叫此自訂函式時，傳遞給自訂函式的引數，便能依照這個定義自動轉換為正確的资料型別。

通常使用樣板的時機為：撰寫自訂函式、定義類別。因此，也將樣板的用途大致上分為函式樣板（或函數樣板，Function template）與類別樣板（Class template）。

二、執行結果

如下圖所示，若未親自參加抽獎活動，只會得到鼓勵獎品獎：USB 隨身碟。親自參加抽獎活動者，則隨機抽出獎品。



```

E:\Documents\MEGAsync\30個範例學完C++\書稿\第三...
王小明，未出席抽獎，獎品：USB隨身碟
王瑪麗，出席抽獎，獎品：藍芽耳機
李小強，出席抽獎，獎品：小電鍋
請按任意鍵繼續...
  
```

30-1 定義與使用函式樣板

樣板若用來定義函式的參數型態，則稱為函式樣板（函數樣板）；而使用樣板所撰寫的程式，便稱為樣板程式或泛型程式（後者是比較常被使用的稱呼）。

函式樣板

函式樣板由關鍵字 `Template` 開始，之後是型別參數列 "`< 型別參數... >`"，在型別參數列裡以 "`class 型別參數`"（或 "`typename 型別參數`"）為一個型別參數單元，用來定義不同的型別參數，型別參數可以視為尚未被決定的資料型別。此處的關鍵字 "`class`" 並不是指類別的意思；如下第 1 行所示。在函式樣板之後，就是使用此函式樣板的自訂函式的定義，如第 2-5 行所示。

```

1 Template <class 型別參數 1, class 型別參數 2,...>
2 [ 修飾字 ] 回傳值型別 函式名稱 ([ 參數 1, 參數 2,...])
3 {
4     :
5 }

```

若擔心樣板單元的關鍵字 `class` 與類別的 `class` 混淆，也能使用 `typename` 這個關鍵字，如下所示：

```

1 Template <typename 型別參數 1, typename 型別參數 2,...>
2 [ 修飾字 ] 回傳值型別 函式名稱 ([ 參數 1, 參數 2,...])
3 {
4     :
5 }

```

型別參數的名稱與變數的命名方式相同，只是通常會使用：`Type`、`T1`、`T2` 等簡單的命名方式。例如：以下都是合法的型別參數名稱：

```

<typename Type>
<typename T1>
<typename a1, typename AB>

```

使用樣板的自訂函式的撰寫方式與一般的自訂函式無異，只是多了型別參數可以做為資料型別。例如：使用 `<typename T>` 樣板的自訂函式 `func()`，如下所示。

```

1 template <typename T>
2 static T func(T a, T b)
3 {
4     T c;
5
6     c = a + b;
7     return c;
8 }

```

程式碼第 1 行定義了函式樣板 `<typename T>`，`T` 即為型別參數。第 2-8 行定義靜態的自訂函式 `func()`，函式回傳值的型別為 `T`，並接收 2 個資料型別為 `T` 的參數 `a` 與 `b`。第 4 行使用資料型別 `T` 宣告了變數 `c`，第 6 行將參數 `a` 與 `b` 相加的結果儲存於變數 `c`，第 7 行回傳變數 `c`。當呼叫自訂函式 `func()` 並傳入 2 個整數 1 與 2，如下所示：

```

int c;
c = func(1, 2);

```

傳入自訂函式 `func()` 的 2 個引數為 `int` 型別；對應到程式碼第 2 行的 `func()` 自訂函式定義：接收 2 個 `T` 型別的參數，因此 `T` 會被置換為 `int` 型別，所以自訂函式 `func()` 被轉換為：

```

2 static int func(int a, int b)
3 {
4     int c;    資料型別 int 取代型別參數 T1
5
6     c = a + b;
7     return c;
8 }

```

因此，若傳進 `func()` 自訂函式的 2 個引數為 `double` 型別，則函式樣板中的型別參數 `T` 便會被 `double` 型別取代，所以自訂函式 `func()` 則被轉換為：

```

2 static double func(double a, double b)
3 {
4     double c;  資料型別 double 取代型別參數 T1
5
6     c = a + b;
7     return c;
8 }

```

由此可知，函式樣板中的型別參數 `T`，會依據所傳入的參數的資料型別而改變；所以免除了要根據不同的參數的資料型別，而定義各種多載函式的麻煩。

定義函式樣板與使用樣板的自訂函式，可以予以合併，例如上述的程式碼第 1-2 行可以合併為下列程式碼的第 1 行：將函式樣板與自訂函式宣告寫在同一行。

```

1 template <typename T> static T func(T a, T b)
2 {
3     ↑      ↑
4     : 函式樣板      自訂函式宣告
5 }

```

多個型別參數

上述所定義的函式樣板 `<typename T>` 只有 1 個型別參數 `T`；因此，透過此函式樣板所定義的自訂函式 `func()` 便只能接受 2 個資料型別皆為 `T` 的參數。若傳入自訂函式 `func()` 的 2 個參數的資料型別不同，例如：整數 4 與浮點數 1.2，則會發生如下的錯誤：

`func(4, 1.2);`

```
template<class T> void func(T a, T b)
```

函式樣板 "func" 沒有任何執行個體符合引數清單
引數類型為: (int, double)

Visual Studio C++ 的編譯視窗出現的錯誤是：沒有接收引數類型 (`int`, `double`) 的自訂函式 `func()`。因為上述所定義的自訂函式 `func()` 所接收的是 2 個相同型別 `T` 的參數，但在呼叫自訂函式 `func()` 時，卻傳入了 1 個整數與 1 個浮點數；因此，參數型態不符合預設的資料型別而發生了錯誤。

因此，只要將函式樣板修改為可接受 2 種資料型別的樣板，即可以修正這種錯誤；如下所示：

```

      2 個型別參數 T1 與 T2
      ↓      ↓
1  template<typename T1, typename T2> static double func(T1 a, T2 b)
2  {
3      double c; ← 函式回傳值型別與變數 c 改為 double 型別。
4
5      c = a + b;
6      return c;
7  }
```

程式碼第 1 行的函式樣板有 2 個型別參數單元：`<typename T1>` 與 `<typename T2>`，而自訂函式 `func()` 的 2 個參數的資料型分別為 `T1` 與 `T2`；因此，可以接受 2 個不同（或相同：`T1` 等於 `T2`）的參數。例如：

```

func(4, 1.2);
func(3.4, 6);
func(6, 12);
func(3.2, 1.6);
```

以第 1 個為例：呼叫自訂函式 `func()` 所傳入的引數為整數 4 與浮點數 1.2；因此，自訂函式 `func()` 透過函式樣板轉換為如下的形式：

```

2  static double func(int a, double b)
3  {
4      double c;
5      c = a + b;
6      return c;
7  }
```

↑
型別參數 T1 與 T2 分別轉換為 int 與 double 型別。

由於無法事先知道所傳入的參數的資料型別，因此 2 個參數數相加後的結果有可能是整數或是小數。所以為了安全起見，也將函式回傳值型別與變數 `c` 宣告為 `double` 型別；藉以避免 2 數相加所產生的小數部分被無條件捨去，或是產生溢位的錯誤。

以此類推，可以自行定義多個型別參數，以符合功能設計上的需求。

非型別參數

樣板對於函式的參數的資料型別提供了很好的彈性，可以依照呼叫函式時，所傳入的引數的資料型別才決定參數的資料型別。但是，有時候卻希望固定某些參數的資料型別，例如：表示「物品數量」的參數應該是整數型別，表示「體重」的參數應該是浮點數型別。此時便可以使用非型別參數（Non-Type）的方式設計函式樣板（參考範例檔案 01-1）；例如：

```

1  template<int age, typename T1, typename T2>
2  void BMI(const char* name, T1 weight, T2 height)
3  {
4      double bmi;
5
6      bmi = (double)weight / powf(height, 2);
7      cout << " 姓名：" << name << " " << age << " 歲, BMI= " << bmi << endl;
8  }

```

第 1 行宣告樣板，其中第 1 個為 `int` 型別的非型別參數，並且指明了參數名稱 `age`；因為在樣板裡直接使用明確的資料型別，所以稱為非型別參數。這是計算 BMI 的自訂函式 `BMI()`，接收 3 個參數，分別為 `name` 表示姓名、`weight` 為體重（公斤）與 `height` 表示身高（公尺）；身高與體重可能是整數或是浮點數，所以就使用型別參數 `T1` 與 `T2`。因此，呼叫此自訂函式計算 BMI 時，可以是如下的型式：

```

15 BMI<20, int, double>(" 王小明 ", 60, 1.7);
16 BMI<19>(" 真美麗 ", 47.6, 1.62);

```

第 15 行呼叫 `BMI()` 自訂函式，並指定了樣板所需要的 3 個型別參數 `<20, int, double>`。因為第 1 行的樣板中第 1 個參數為非型別參數，因此直接指定其值等於 `20`；另外 2 個型別參數則使用資料的型別 `int` 與 `double`。此種寫法也可以簡化如第 16 行的方式，只列出非型別參數。上述 2 行程式敘述的輸出結果為：

```

姓名：王小明 20 歲, BMI= 20.7612
姓名：真美麗 19 歲, BMI= 18.5185

```

練習 1：計算平均成績

定義函式樣板，並使用此樣板設計自訂函式 `average()`，用於計算學生的平均成績。輸入多筆成績資料，並利用自訂函式 `average()` 計算平均成績。

■ 解說

一個學生應該有多項成績，才能計算平均成績；因此記錄學生成績的變數應使用陣列。自訂函式接收 2 個參數：成績陣列與陣列長度。因為無法事先知道成績陣列的資料型別，只知道是數值陣列，因此可以使用樣板的型別參數。所以，自訂函式 `average()` 以及函式樣板，應如下之形式：

```

1  Template <typename T>
2  double average(T score[], const int num)
3  {
4      :
5  }
```

自訂函式 `average()` 的第 1 個參數 `score[]` 用於表示學生的成績陣列，並且資料型別為 `T`。第 2 個參數 `num` 表示成績陣列的長度，為了避免此參數被修改，造成計算平均成績時的錯誤，所以使用 `const` 修飾字。

■ 執行結果

```

約翰的平均成績：86
瑪莉的平均成績：88.25
```

■ 程式碼列表

```

1  #include <iostream>
2  using namespace std;
3
4  template<typename T>double average(T sco[],const int num)
5  {
6      double avg=0;
7
8      for (int i = 0; i < num; i++)
9          avg += sco[i];
10
11     avg /= num;
12     return avg;
13 }
14
```

```

15 int main()
16 {
17     double John[] = { 80,86,92 };
18     double Mary[] = { 91,89,88,85 };
19
20     cout << " 約翰的平均成績：" <<
21         average(John, size(John)) << endl;
22
23     cout << " 瑪莉的平均成績：" <<
24         average(Mary, size(Mary)) << endl;
25
26     system("pause");
27 }

```

程式講解

1. 程式碼第 1-2 行引入 `iostream` 標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 4-13 行定義函式樣板與自訂函式 `average()`。第 6 行宣告 `double` 型別的變數 `avg`，用於儲存平均成績。第 8-9 行先使用 `for` 重複敘述累加成績陣列裡的每一項成績，並儲存在變數 `avg`，第 11 行計算平均成績，第 12 行回傳平均成績。
3. 開始撰寫主函式 `main()`。程式碼第 17-18 行宣告 2 個整數陣列 `John` 與 `Mary`，長度分別等於 3 與 4；用於表示約翰和瑪莉 2 人的成績。第 20-21 行顯示約翰的平均成績；其中呼叫自訂函式 `average()`，並傳入 2 個引數：成績陣列 `John`，以及成績陣列的長度 `size(John)`。

第 23-24 行顯示瑪麗的平均成績，作法與顯示約翰的平均成績相同。

30-2 函式樣板的多載

函式樣板的功能避免了過多的多載函式；然而，諸如函式回傳值型別不同、參數數量不同等，還是需要用到多載函式；因此，函式樣板也提供了多載的方式來因應。

多載型式的函式樣板

以下為函式樣板多載的例子：

```

1 template <typename T> void func(T a){...}
2 template <typename T> void func(T a, T b){...}
3 template <typename T1, class T2> void func(T1 a, T2 b){...}
4 template <typename T> void *func(T a, int b){...}
5 template <typename T> void func(T *a, int b){...}

```


以上 5 個自訂函式 `func()`，除了函式名稱、函式回傳值相同之外，參數的個數或是參數的資料型別並不完全相同，因此被視為是多載的函式：第 1 個 `func()` 接收 1 個型別參數 `T` 的參數，第 2 個 `func()` 接收 2 個型別參數 `T` 的參數。

第 3 個 `func()` 接收 2 個參數，並且此 2 個參數的型別參數不同。第 4 個 `func()` 雖然和第 2 個很像，但因為其第 2 個參數直接指定了 `int` 型別，所以也被視為是多載的函式。第 5 個 `func()` 的第 1 個參數多了指標，也被視為是多載的函式。

以下的自訂函式 `func()`，並不是多載的函式樣板。雖然第 1 個與第 2 個函式 `func()` 的型別參數的名稱不同，但函式樣板中的型別變數的名稱，在呼叫函式時會被置換為參數的真實資料型別，例如 `int` 型別；如此一來，第 1 個與第 2 個函式 `func()` 都符合被呼叫的條件，因此產生了模稜兩可的錯誤。

```
1 template <typename T> void func(T a){...}
2 template <typename S> void func(S a){...}
3 template <typename T> void *func(T a){...}
```

第 3 個自訂函式 `func()` 的簽名與另外 2 個函式 `func()` 完全相同，只是回傳值型別多了指標，因此不是多載函式。

樣板特製化

函式樣板中的型別參數，若所接收的是字元陣列或字元指標型別的字串時，容易發生模稜兩可的結果與錯誤。先看一個使用 `string` 型別宣告字串的正確範例（參考範例檔案 02-1）；如下所示。

程式碼第 1-4 行定義了函式樣板 `<typename T>` 的自訂函式 `maxStr()`，接受 2 個 `T` 型別的參數 `str1` 與 `str2`；此自訂函式用於比較 2 個字串參數誰比較大，並顯示比較大的字串參數。

```
1 template <typename T> void maxStr(T str1, T str2)
2 {
3     cout << " 比較大者：" << ((str1 > str2) ? str1 : str2) << endl;
4 }
5
6 int main()
7 {
8     string s1= "abcde";
9     string s2= "fghij";
10
11     maxStr(s1, s2);
12 }
```

程式碼第 8-9 行宣告 2 個 `string` 型別的字串變數 `s1` 與 `s2`，其內容等於字串 "abcde" 與 "fghij"。第 11 行呼叫自訂函式 `maxStr()`，並傳入 2 個字串變數作為引數。因此，此範例的執行結果為：

比較大者：fghij

現在將程式碼第 8-9 行的 2 個變數 `s1` 與 `s2` 改為字元陣列型別的字串（參考範例檔案 02-2）；如下所示：


```
8 char s1[] = "abcde";
9 char s2[] = "fghij";
```

執行結果為：

比較大者：abcde

明顯地，這是錯誤的執行結果。這是因為當字元陣列型別的字串傳遞到自訂函式 `maxStr()` 之後，型別參數 `T` 會將所接收到的字串自動換為字元指標 `char *`；因此，自訂函式 `maxStr()` 視同：

```
1 void maxStr(char * str1, char * str2)
2 {
3     cout << " 比較大者：" << ((str1 > str2) ? str1 : str2) << endl;
4 }
```



比較 2 個參數的記憶體位址誰比较大

程式碼第 3 行的 `(str1 > str2)` 變成了比較 `str1` 的記憶體位址是否大於 `str2` 的記憶體位址，在此例中參數 `str1` 與 `str2` 在電腦中的記憶體位址分別為 0097FE20 與 0097FE10，因此比較之後的結果是參數 `str1` 大於 `str2`，所以執行結果為字串 "abcde" 比较大。若換台電腦，或換個時間執行此程式，也許會有不同的結果。

例如，筆者又換了一台電腦執行此程式，結果這次參數 `str1` 與 `str2` 在電腦中的記憶體位址分別為 000000F6467EF854 與 000000F6467EF874；因此，這次的執行結果是參數 `str2` 大於 `str1`，所以執行結果又變成了：

比較大者：fghij

有不同的方式可以解決這樣模稜兩可的問題，在此提供 2 種常被使用的方法。

► 指定型別參數

第 1 種方法，在呼叫 `maxStr()` 自訂函式時，加上明確的型別參數 `<string>`；如下所示（參考範例檔案 02-3），將程式碼第 11 行改爲：

```
11 maxStr<string>(s1, s2);
```

► 樣板特製化

第 2 種方法，當呼叫使用函式樣板的自訂函式，在型別參數轉換爲特定的資料型別，若會發生此種模稜兩可的結果時，可以替這種資料型別特別撰寫多載的自訂函式，稱爲樣板特製化（Template specialization）。當呼叫此自訂函式時，就不會去執行這個有函式樣板的自訂函式，而是去執行特別撰寫的多載的自訂函式（參考範例檔案 02-4）。例如，將此範例修改爲：

```
1 template <typename T> void maxStr(T str1, T str2)
2 {
3     cout << " 比較大者：" << ((str1 > str2) ? str1 : str2) << endl;
4 }
5
6 void maxStr(char* str1, char* str2)
7 {
8     cout << " 比較大者：" << ((*str1 > *str2) ? str1 : str2) << endl;
9 }
10
11 int main()
12 {
13     char s1[] = "abcde";
14     char s2[] = "fghij";
15
16     maxStr(s1, s2);
17 }
```

程式碼第 6-9 行提供了另一個多載的自訂函式 `maxStr()`，接收 2 個字元指標型別的字串參數 `str1` 與 `str2`。第 8 行所比較的是字串參數 `str1` 與 `str2` 的內容：`*str1` 與 `*str2`，而不再是比較 `str1` 與 `str2` 的記憶體位址。

因此，當第 16 行呼叫自訂函式 `maxStr()`，便會執行第 6-9 行的自訂函式 `maxStr()`，而不是第 1-4 行的自訂函式 `maxStr()`。

🔄 練習 2：購物商場周年慶

某一購物商場周年慶好禮相送，購物滿 1500-3000 元則贈送精美收納盒。購物超過 3000 元，並在 6000（含）元以內，則可以隨機抽 1000、1500 或 2000 的購物禮卷。購物金額超過 6000 元，除了可以抽購物禮卷之外，還隨機贈送超值禮盒。使用多載的函式樣板，寫一程式輸入顧客姓名與購物金額，顯示所獲得之禮品。

■ 解說

依照題意，分為 4 種購物金額發放禮品：低於 1500 元不發放任何禮品。介於 1500-3000 元發放精美收納盒，3001-6000 元發放禮卷，大於 6000 元以上發放禮卷與禮盒；因此，發放禮品的原則可以使用巢狀的 `if...else` 來處理。

隨機贈送禮卷或禮盒，可以使用亂數來處理。事先設定好 3 種禮卷的額度，再產生亂數來隨機挑選禮卷的金額；隨機贈送禮盒的處理方式也相同。

至於發放禮品的自訂函式 `select()`，可以使用多載的方式來處理：分別處理 3 種不同購物金額的禮品，如下所示：

```

1 void select(string name) ← 處理 1500-3000 元
2 {
3     :
4 }
5                                     處理 3001-6000 元
6                                     ↓
6 template <typename T> void select(string name, T sel)
7 {
8     :
9 }
10                                     處理 6001 元以上
11                                     ↓
11 template <typename T1, typename T2> void select(string name,
12                                             T1 sel, T2 item)
13 {
14     :
15 }

```

程式碼第 1-4 行、第 6-9 行與第 11-15 行分別為 3 個多載的自訂函式 `select()`，分別處理 3 種不同購物金額的禮品。3 個自訂函式的第 1 個參數都是字串型別，表示顧客的姓名。第 2、3 個多載自訂函式使用了函式樣板。樣板 `<typename T>` 與 `<typename T1>` 用於表示不同金額的禮卷種類，樣板 `<typename T2>` 則表示不同的禮盒。

執行結果

消費金額為 1500-3000 元，只贈送精美收納盒；如下所示。

輸入姓名與今日消費金額：王小明 2000
顧客名稱：王小明，獲得精美收納盒

消費金額為 3001-6000 元，可抽購物禮卷；如下所示。

輸入姓名與今日消費金額：真美麗 4000
顧客名稱：真美麗，獲得禮卷 1000 元

消費金額高於 6000 元，除了可抽購物禮卷，還隨機贈送禮盒；如下所示。

輸入姓名與今日消費金額：李小強 7000
顧客名稱：李小強，獲得禮卷 2000 元，以及骨瓷杯組

程式碼列表

```

1  #include <iostream>
2  #include <time.h>
3  using namespace std;
4
5  void select(string name)
6  {
7      cout<<" 顧客名稱："<<name<<"，獲得精美收納盒 " << endl;
8  }
9
10 template <typename T> void select(string name, T sel)
11 {
12     cout << " 顧客名稱：" << name;
13     cout << "，獲得禮卷 " << sel * 500 << " 元 " << endl;
14 }
15
16 template <typename T1, typename T2> void select(string name, T1 sel, T2 item)
17 {
18     cout << " 顧客名稱：" << name;
19     cout << "，獲得禮卷 " << sel * 500 << " 元，";
20     switch (item)
21     {
22         case 0:
23             cout << " 以及手工皂禮盒 " << endl;

```

```

24         break;
25     case 1:
26         cout << " 以及高級紅酒禮盒 " << endl;
27         break;
28     case 2:
29         cout << " 以及骨瓷杯組 " << endl;
30         break;
31     }
32 }
33
34 int main()
35 {
36     int total;
37     string name;
38
39     srand((unsigned)time(NULL));
40
41     cout << " 輸入姓名與今日消費金額：";
42     cin >> name >> total;
43
44     if (total >= 1500)
45     {
46         if (total <= 3000)
47             select(name);
48         else
49         {
50             if (total <= 6000)
51                 select(name, rand() % 3 + 2);
52             else
53                 select(name, rand() % 3 + 2, rand() % 3);
54         }
55     }
56
57     system("pause");
58 }

```

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 5-8 行為第 1 個多載的自訂函式 `select()`，接收一個字串變數 `name`；用以表示顧客的姓名。第 7 行顯示顧客獲得精美收納盒。

3. 程式碼第 10-14 行為使用函式樣板 `<typename T>` 的第 2 個多載的自訂函式 `select()`，接收 2 個參數。第 1 個為字串型別的參數 `name`，表示顧客的姓名。第 2 個為型別參數 `T` 的 `sel` 參數，表示隨機取得的購物禮卷。第 12 行顯示顧客姓名，第 13 行計算並顯示所獲得的購物禮卷金額。
4. 程式碼第 16-32 行為使用函式樣板 `<typename T1, typename T2>` 的第 3 個多載的自訂函式 `select()`；並接收 3 個參數。第 1 個為字串型別的參數 `name`，表示顧客的姓名。第 2 個為型別參數 `T1` 的 `sel` 參數，表示隨機取得的購物禮卷。第 3 個為型別參數 `T2` 的 `item` 參數，表示隨機取得的禮盒。

第 18 行顯示顧客姓名，第 19 行計算並顯示所獲得的購物禮卷金額。第 20-31 行為 `switch...case` 選擇敘述，根據 `item` 的值分別顯示不同的禮盒。

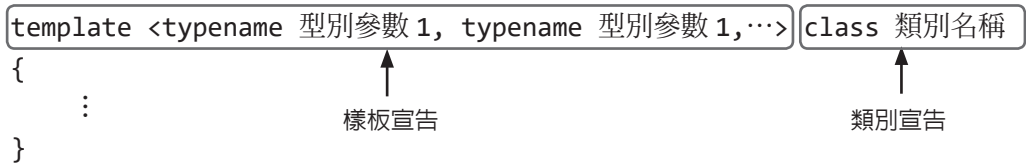
5. 開始撰寫主函式 `main()`。程式碼第 36-37 行宣告 2 個變數 `name` 與 `total`，分別代表顧客姓名與購物金額。第 41-42 行分別顯示輸入的提示訊息，以及將輸入的顧客姓名與購物金額，儲存於變數 `name` 與 `total`。
6. 程式碼第 44 行判斷若購物新額大於 1500 元，才可以獲得禮品或禮卷。第 46 行判斷若購物金額在 1500-3000 元，則第 47 行呼叫自訂函式 `select()`，並只傳遞顧客姓名 `name` 作為引數；因此會執行第 1 個多載的 `select()` 自訂函式。否則，進入程式碼第 48-54 行，表示購物金額超過 3000 元。
7. 程式碼第 50 行判斷購物金額若超過 3000 元，並小於含 6000（含）元，則第 51 行呼叫自訂函式 `select()`，並傳遞顧客姓名 `name` 與亂數 2-4 作為引數；因此會執行第 2 個多載的 `select()` 自訂函式。否則，第 53 行呼叫第 3 個多載的自訂函式 `select()`，表示購物金額超過 6000 元，並傳遞 3 個參數：顧客姓名、亂數 2-4 與亂數 0-2；此 2 個亂數分別表示隨機抽取的禮卷與禮盒。

30-3 類別樣板

使用於類別的樣板，稱為類別樣板；使用上與函式樣板有些許不同處。例如：函式樣板在函式被呼叫時，會自動由編譯器決定型別參數的真正資料型別；但是類別樣板必須在撰寫程式時，明確指定型別參數的資料型別。因此，在設計類別與使用類別樣板時，需要特別的留意。

定義類別樣板

定義類別樣板的語法，只是在類別定義之前加上了樣板的宣告，其餘的與定義類別並無差別；如下所示。



例如，定義一個類別樣板 `<typename T>`，使用此樣板定義類別 `Rectangle`，用於計算矩形的面積；如下所示（參考範例檔案 03-1）。程式碼第 1 行在定義類別之前，加上了 `template<typename T>` 的類別樣板。第 3-5 行 `protected` 區段內，宣告了 3 個型別參數 `T` 的資料成員 `size`、`length` 與 `width`，分別表示矩形的面積、長度與寬度。

在 `public` 區段內有 3 個成員函式。第 8-12 行為建構元，接收 2 個型別參數 `T` 的參數 `len` 與 `wid`，表示矩形的長度與寬度。第 14-17 行為成員函式 `computeSize()`，用於計算矩形的面積。第 19 行為成員函式 `getSize()` 的函式原型宣告，用於回傳矩形的面積，函式回傳值型別為型別參數 `T`。

```

1  template<typename T> class Rectangle
2  {
3      protected:
4          T size = 0; // 面積
5          T length = 0, width = 0; // 長度與寬度
6
7      public:
8          Rectangle(T len, T wid)
9          {
10             length = len;
11             width = wid;
12         }
13
14         void computeSize() // 計算矩形的面積
15         {
16             size = length * width;
17         }
18
19         T getSize(); // 取得矩形面積
20     };
21
22     template<typename T> T Rectangle<T>::getSize()
23     {
24         return size;
25     }

```

因為類別 `Rectangle` 的成員函式 `getSize()` 只有函式的原型宣告，因此需要在類別之外定義此成員函式；程式碼第 22-25 為此成員函式的本體定義。因為類別 `Rectangle` 使用了類別樣板；因此，寫在類別之外的成員函式也需要加上樣板 `template<typename T>`，並且要在類別的名稱之後也加上型別參數 `T`；如第 22 行所示；第 24 行回傳矩形的面積。

在主函式 `main()` 中，程式碼第 29 行宣告類別 `Rectangle` 類別的物件 `rect1`，並且需要指定參數的資料型別 `double`；此資料型別會取代類別樣板中的型別參數 `T`。這個指定的資料型別 `double` 代表了傳遞給 `Rectangle` 類別建構元的 2 個引數 12.4 與 5.6 的資料型別為 `double`。

第 30 行宣告了另一個 `Rectangle` 類別的物件 `rect2`，但參數型別為 `int`，因此類別樣版中的型別參數 `T` 會被取代為 `int` 型別；因此，所傳入的 2 個引數 10 與 20 也才會是 `int` 型別。

```

27 int main()
28 {
29     Rectangle <double> rect1(12.4, 5.6);
30     Rectangle <int> rect2(10, 20);
31
32     rect1.computeSize();
33     cout << "矩形面積 = " << rect1.getSize() << endl;
34
35     rect2.computeSize();
36     cout << "矩形面積 = " << rect2.getSize() << endl;
37 }

```

第 32、35 行分別呼叫物件 `rect1` 與 `rect2` 各自的成員函式 `computeSize()`，計算矩形的面積。相同的方式，第 33、36 行則分別呼叫各自的成員函式 `getSize()` 取得並顯示矩形的面積。此範例的執行結果為：

```

矩形面積 = 69.44
矩形面積 = 200

```

多個與非型別參數的類別樣板

類別樣板如同函式樣板一樣，也能定義多個型別參數，或是使用非型別參數。但是定義類別與使用上有些許不同（參考範例檔案 03-2）；修改類別 `Rectangle` 如下所示：

程式碼第 1 行的類別樣板有 3 個型別參數 `T1`、`T2` 與 `int` 型別。第 3 個非型別參數明確指定為 `int` 資料型別，並且有參數名稱 `num` 與預設值 1；如果在類別樣板中指定了非型別參數，也必須提供參數的名稱。此類別修改為可以計算多個相同大小的矩形面積；因此，類別樣板

的非型別參數的參數 `num` 用於表示有幾個矩形。因此，第 4 行的資料成員 `size` 表示所有矩形的面積總和。

程式碼第 8 行 `T1` 型別的資料成員 `name`，表示形狀的名稱。第 10-16 行為類別 `Rectangle` 的建構元，接收 3 個參數 `str`、`len` 與 `wid`，分別代表形狀的名稱、長度與寬度。第 15 行將類別樣本的參數 `num` 直接設定給資料成員 `number`，表示矩形的數量。第 18-21 行為成員函式 `computeSize()`，計算 `number` 個矩形的面積總和。

```

1  template< typename T1, typename T2, int num = 1> class Rectangle
2  {
3      private:
4          T2 size = 0; // 所有矩形的面積總和
5          T2 length = 0, width = 0; // 長與寬
6
7      public:
8          T1 name; // 形狀的名稱
9          int number = 0; // 矩形的數量
10         Rectangle(T1 str, T2 len, T2 wid)
11         {
12             name = str;
13             length = len;
14             width = wid;
15             number = num;
16         }
17
18         void computeSize() // 計算 number 個矩形的面積
19         {
20             size = length * width * number;
21         }
22
23         T2 getSize(); // 取得矩形面積總和。函式原型宣告。
24     };
25
26     template<typename T1, typename T2, int num>
27     T2 Rectangle< T1, T2, num>::getSize()
28     {
29         return size;
30     }
```

非型別參數 num 的初始值 1

使用參數的名稱

程式碼第 23 行為成員函式 `getSize()` 的函式原型宣告，第 26-30 行則為此成員函式的外部定義。除了第 26 行將完整的類別樣板重新描述一次之外，第 27 行在類別名稱之後，需要再

次標明所有用到的型別參數 `<T1, T2, num>`；其中非型別參數需與第 1 行的類別的非型別參數的名稱 `num` 相同。

主函式 `main()` 中，程式碼第 34 行宣告 `Rectangle` 類別的物件 `rect`，函式樣板中的第 3 個參數並不是資料型別的名稱，而是數值 2；此數值會直接傳遞給非型別參數的參數 `num`。第 36 行呼叫物件 `rect` 的成員函式 `computeSize()` 計算 2 個矩形的面積。

```

32 int main()
33 {
34     Rectangle <string, double, 2> rect(" 矩形 ", 12.4, 5.6);
35
36     rect.computeSize();
37     cout << rect.name << " 面積 = " << rect.getSize() << endl;
38 }

```

因此，此範例程式的執行結果為：

矩形面積 = 138.88

類別樣板與繼承

使用類別樣板所定義的類別，也如同一般的類別，有著繼承與被繼承的特性；但是因為使用了類別樣板關係，所以繼承在定義與實作上有些不同的地方。假設使用類別樣板的類別 `baseClass` 作為父類別，則有 2 種被繼承的方式：

► 第 1 種方式：明確指定父類別的型別參數

第 1 種表達方式如下所示。程式碼第 1-4 行為使用樣板 `<typename T>` 的父類別 `baseClass`，第 6-9 行為繼承 `baseClass` 類別的子類別 `subClass1`，並且明確指定了父類別 `baseClass` 的樣板中的型別參數 `<T>` 為 `int` 型別。

```

1 template<typename T>class baseClass
2 {
3     :
4 };
5
6 class subClass1 :public baseClass<int>
7 {
8     :
9 };

```

↑
明確指定父類別的型別參數 `<T>` 的型別為 `int` 型別。

► 第 2 種方式：保持父類別的型別參數

如果要父類別維持使用類別樣板的彈性，則子類別 `subClass2` 也需要使用類別樣板；如下第 6 行所示。子類別與父類別所使用的類別樣板可以不一樣，但是子類別所使用的類別樣板中，必須包含父類別的類別樣板中的型別參數。

```

6  template<typename T> class subClass :public baseClass<T>
7  {
8      :
9  };

```

↑
類別 `subClass2` 也使用了類別樣板，並且包含了父類別的型別參數 `T`。

例如以下的範例（參考範例檔案 03-3）：程式碼第 1-15 行定義父類別 `baseClass`，並且使用類別樣板 `<typename T>`，此類別只有 1 個 `public` 區段。第 4 行宣告 `T` 型別的資料成員 `base`。第 6-9 行為建構元，接收一個 `T` 型別的參數 `v`，並將此參數設定給資料成員 `base`。第 11-14 行為成員函式 `show()`，用於顯示成員資料 `base`。

```

1  template <typename T> class baseClass
2  {
3      public:
4          T base;
5
6          baseClass(T v)
7          {
8              base = v;
9          }
10
11         void show()
12         {
13             cout << base << endl;
14         }
15 };

```

程式碼第 16-31 行為子類別 `subClass1` 的定義，繼承 `baseClass` 類別，並且明確指定了父類別 `baseClass` 的樣板中的型別參數 `T` 為 `int` 型別；如第 16 行所示。第 19 行宣告了字串型別的資料成員 `sub`。第 21-24 行為建構元，接收 2 個資料型別分別為 `string` 與 `int` 的參數 `v1` 與 `v2`，並呼叫父類別的建構元；參數 `v2` 用於設定父類別的資料成員 `base`。第 23 行將參數 `v1` 設定給資料成員 `sub`。第 26-30 行為成員函式 `showAll()` 的定義。第 28 行先顯示資料成員 `sub`，第 29 行再呼叫父類別的成員函式 `show()` 顯示其資料成員 `base`。

```

16 class subClass1 : public baseClass<int>
17 {
18 public:
19     string sub;
20
21     subClass1(string v1, int v2) :baseClass(v2)
22     {
23         sub = v1;
24     }
25
26     void showAll()
27     {
28         cout << sub <<" , ";
29         show();
30     }
31 };

```

程式碼第 33-49 行定義子類別 `subClass2`，繼承類別 `baseClass`，並且使用的類別樣板有 2 個型別參數 `T` 與 `T1`；其中要包含父類別所用的型別參數 `T`，繼承的父類別名稱之後也要加上型別參數 `<T>`；如第 34 行所示。

第 37 行宣告了 `T1` 型別的資料成員 `sub`，第 39-42 行為建構元，接收 2 個資料型別分別為 `T1` 與 `T` 的參數 `v1` 與 `v2`，並呼叫父類別的建構元；參數 `v2` 用於設定父類別的資料成員 `base`。第 41 行將參數 `v1` 設定給資料成員 `sub`。第 44-48 行為成員函式 `showAll()` 的定義，第 46 行先顯示資料成員 `sub`，第 47 行再呼叫父類別的成員函式 `show()` 顯示其資料成員 `base`；因為父類別使用了類別樣板，所以呼叫父類別的成員函式時，必須在函式名稱之前加上父類別的名稱與其樣板的型別參數 `"baseClass<T>"`。

```

33 template<typename T, typename T1>
34 class subClass2 : public baseClass<T>
35 {
36     public:
37         T1 sub;
38
39         subClass2(T1 v1, T v2) : baseClass<T>(v2)
40         {
41             sub = v1;
42         }
43
44         void showAll()

```

↑
用到父類別就要加上型
別參數 T。

↓

```

45         {
46             cout << sub << ", ";
47             baseClass<T>::show(); ← 要加上父類別的名稱與樣板的型別參數 T。
48         }
49     };

```

在主函式 `main()` 中，程式碼第 53-54 行分別宣告類別 `subClass1` 與 `subClass2` 的物件 `cls1` 與 `cls2`；此 2 個物件的第 1 個引數為姓名，第 2 個引數為數值。類別 `subClass2` 使用了類別樣板，因此宣告物件時需明確指定樣板中的型別參數的型別：`<int, string>`，此 2 個資料型別對應了子類別 `subClass2` 的型別參數 `T` 與 `T1`。

```

51 int main()
52 {
53     subClass1 cls1("王小明", 12);
54     subClass2<int, string>cls2("真美麗", 15);
55
56     cls1.showAll();
57     cls2.showAll();
58 }

```

程式碼第 56-57 行分別呼叫 2 個物件各自的成員函式 `showAll()`；此範例的執行結果為：

```

王小明 , 12
真美麗 , 15

```

🔄 練習 3：計算長方體體積

使用本章節「定義類別樣板」小節中，所使用的 `Rectangle` 類別作為父類別，設計子類別 `Cuboid` 繼承 `Rectangle` 類別，用於計算長方體的體積。

■ 解說

計算長方體需要將矩形的面積乘上高度；因此，在子類別 `Cuboid` 中需要增加 2 個浮點數型別的資料成員：高度 `height` 與體積 `volumn`。子類別 `Cuboid` 為了接收父類別的類別樣板中的型別參數 `T`，因此也設計為使用類別樣板。

子類別的建構元也需要呼叫父類別的建構元，才能設定矩形的長度與寬度。計算長方體體積的成員函式，可以設計為先呼叫父類別的 `computeSize()` 成員函式計算出矩形的面積，然後再呼叫父類別的 `getSize()` 取得矩形的面積之後，乘上高度 `height`，便能計算長方體的體積。因此，子類別 `Cuboid` 的結構大致如下所示：

```

template<typename T, typename T1>
class Cuboid :public Rectangle<T>
{
    private:
        T volumn, height;  // 體積與高度

    public:
        T1 name = "";      // 名稱

        Cuboid(T1 str, T len, T wid, T hei) :Rectangle<T>(len, wid)
        {
            設定資料成員 ;
        }

        void computeVolumn()  // 計算長方體體積
        {
            呼叫父類別的 computeSize() 計算矩形面積 ;
            計算長方體體積：呼叫父類別的 getSize()，再乘上 height;
        }

        T getVolumn()  // 取得長方體體積
        {
            回傳 volumn;
        }
};

```

執行結果

長方體面積 = 600

程式碼列表

```

1 #include <iostream>
2 using namespace std;
3
4 template <typename T> class Rectangle
5 {
6     protected:
7         T size = 0;  // 面積
8         T length = 0, width = 0; // 長與寬

```

```
9
10     public:
11         Rectangle(T len, T wid)
12         {
13             length = len;
14             width = wid;
15         }
16
17         void computeSize() // 計算矩行的面積
18         {
19             size = length * width;
20         }
21
22         T getSize() // 取得矩形面積
23         {
24             return size;
25         }
26 };
27
28 template<typename T, typename T1> class Cuboid :public Rectangle<T>
29 {
30     private:
31         T volumn, height;
32
33     public:
34         T1 name = "";
35
36         Cuboid(T1 str, T len, T wid, T hei) :Rectangle<T>(len, wid)
37         {
38             name = str;
39             height = hei;
40         }
41
42         void computeVolumn()
43         {
44             Rectangle<T>::computeSize();
45             volumn = Rectangle<T>::getSize() * height;
46         }
47
48         T getVolumn()
49         {
50             return volumn;
```



```

51         }
52     };
53
54     int main()
55     {
56         Cuboid <double,string> cub("長方體",10,20,3);
57
58         cub.computeVolumn();
59         cout << cub.name<<"面積 "<<"= " << cub.getVolumn() << endl;
60
61         system("pause");
62     }

```

程式講解

1. 程式碼第 1-2 行引入 `iostream` 標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 4-26 行為類別 `Rectangle` 的定義，並使用類別樣板 `<typename T>`。第 6-8 行為 `protected` 區段，第 7-8 行宣告 `T` 型別的資料成員 `size`、`length` 與 `width`，分別代表矩形的面積、長度與寬度。第 10-25 行為 `public` 區段，包含了 3 個成員函式。第 11-15 行為建構元，用於設定資料成員 `length` 與 `width`。第 17-20 行為成員函式 `computeSize()`，用於計算矩形的面積。第 22-25 行為成員函式 `getSize()`，用於回傳矩形的面積。
3. 程式碼第 28-52 行為類別 `Cuboid` 的定義，此類別繼承 `Rectangle` 類別，並使用了類別樣板 `<typename T, typename T1>`；使用型別參數 `T` 是因為繼承了 `Rectangle` 類別的關係，所以沿用父類別所使用的類別樣板中的型別參數 `T`。

第 30-31 行為 `private` 區段，宣告了 2 個 `T` 型別的資料成員 `volumn` 與 `height`，分別表示長方體的體積與高度。第 33-51 行為 `public` 區段，包含了 3 個成員函式。第 36-40 行為建構元，並呼叫父類別 `Rectangle` 的建構元。建構元接收 4 個參數：`str`、`len`、`wid` 與 `hei`，分別代表名稱、長方體的長度、寬度與高度；並將長度 `len` 與寬度 `wid` 傳遞給父類別 `Rectangle` 的建構元，用以初始化矩形的長度與寬度。

4. 程式碼第 42-46 行為子類別 `Cuboid` 的成員函式 `computeVolumn()`，用於計算長方體的體積。第 44 行先呼叫父類別的成員函式 `computeSize()` 計算矩形的面積，第 45 行再呼叫父類別的成員函式 `getSize()` 取得矩形的面積，隨即乘上高度 `height` 來計算長方體的體積，並儲存於變數 `volumn`。

第 48-51 行為成員函式 `getVolumn()`，用於取得長方體的體積。第 50 行回傳長方體的體積 `volumn`。

5. 開始撰寫 `main()` 主函式。程式碼第 56 行宣告類別 `Cuboid` 的物件 `cub`，並指定型別參數為 `<double, string>`；因此，第 28 行的類別樣本型別參數 `T` 與 `T1`，分別會被轉換為 `double` 型別與 `string` 型別。並且傳入 4 個引數："長方體"、10、20 與 3；分別表示形狀的名稱、長方體的長度、寬度與高度。第 58 行呼叫物件 `cub` 的成員函式 `computeVolumn()` 計算長方體的體積，第 59 行顯示長方體的訊息：使用物件的屬性 `cub.name` 取得名稱，並呼叫成員函式 `getVolumn()` 取得長方體的體積。

特製化的類別樣板

類別樣板也能使用特製化；當類別接收某些資料型別的參數，需要特別做處理時，便可以使用類別樣板特製化。類別樣板特製化的語法，如下所示。

如第 1 行所示，使用空的用樣板 `template<>` 作為開始，並在類別名稱之後加上需要特製化的資料型別。成員函式的定義若是在類別之外，則需要在類別名稱之後也加上需要特製化的資料型別；如第 7 行所示。

```

1  template <> class 類別名稱 <需要特製化的資料型別>
2  {
3      :
4      void func();
5  }
6
7  void 類別名稱 <需要特製化的資料型別>::func(){...}

```

例如：下列程式碼第 1-8 行為類別 `myClass` 的定義，使用類別樣板 `<typename T>`，並有 1 個成員函式 `func()`。若類別 `myClass` 需要針對 `int` 型別進行特製化；因此，程式碼第 10-19 行為 `int` 型別特製化的類別 `myClass`。

在第 10 行使用了空的樣板 `template<>`，以及在類別名稱 `myClass` 之後加上需要特製化的資料型別 `<int>`。程式碼第 13 行只有成員函式 `func()` 的原型宣告；因此，第 16-19 行為此成員函式的本體定義，並在類別名稱 `myClass` 之後也加上了需要特製化的資料型別 `<int>`

```

1  template<typename T>class myClass
2  {
3      public:
4          void func()
5          {
6              :
7          }
8  };

```

} 類別樣板為 <typename T>
的類別 myClass

```

9
10 template<>class myClass <int>
11 {
12     public:
13         void func();
14 };
15
16 void myClass <int> ::func()
17 {
18     :
19 }

```

針對 int 型別特製化的類別 myClass

🔗 練習 4：打招呼

使用類別樣板並設計 1 個類別 **Hello**，輸入姓名之後（例如：輸入 " 王小明 "），可以顯示：" 您好，王小明 "。若輸入的姓名長度只有 1 個字元，則顯示錯誤訊息。

■ 解說

類別 **Hello** 用於輸入姓名之後，顯示打招呼的訊息，並且需要使用類別樣板。因此，類別 **Hello** 的架構大致如下所示。第 1 行使用類別樣板 `<typename T>` 來定義類別 **Hello**，第 4 行 `T` 型別的資料成員 `name` 用於儲存姓名。

第 7-10 行為建構元，接收一個 `T` 型別的參數 `str`，用來設定資料成員 `name`。第 12-15 行為成員函式 `sayHello()`，用於顯示打招呼的訊息。

```

1 template <typename T> class Hello
2 {
3     private:
4         T name; // 姓名
5
6     public:
7         Hello(T str) // 建構元
8         {
9             設定資料成員 name;
10        }
11
12        void sayHello() // 顯示訊息
13        {
14            顯示打招呼的訊息 ;
15        }
16 }

```

若姓名只有 1 個字元的長度，則要顯示錯誤的訊息。因此，可以使用類別樣板特製化的方式，定義 1 個為 `char` 型別特製化的 `Hello` 類別；當所取得的姓名為 1 個字元時，便會執行此特製化的 `Hello` 類別，如下所示：

第 18 行使用了空的類別樣板 `template<>`，並且在類別名稱之後加上了 `<char>` 型別的特製化。第 21 行的資料成員 `name` 也改成了 `char` 型別。第 24 行建構元所接收的參數 `ch`，其資料型別也改成了 `char`。第 29-32 行的成員函式 `sayHello()`，呼叫了另 1 個成員函式 `showError()`。

第 34 行為成員函式 `showError()` 只有函式原型宣告，此函式的本體定義在類別之外。第 37-40 行為成員函式 `showError()` 的本體定義；因為是在類別 `Hello` 之外所定義的本體，所以需要在類別名稱 `Hello` 之後加上明確的資料型別 `<char>`。

```

18  template <> class Hello <char> ← 針對 char 型別的類別樣
19  {                                版特製化。
20      private:
21          char name; // 姓名
22
23      public:
24          Hello(char ch) // 建構元
25          {
26              設定資料成員 name;
27          }
28
29          void sayHello() // 顯示訊息
30          {
31              呼叫成員函式 showError() 顯示錯誤訊息；
32          }
33
34          void showError(); // 函式原型宣告
35  }
36
37  void Hello<char>::showError()
38  {
39      顯示錯誤訊息；
40  }

```

執行結果

```

您好，王小明
姓名長度需要大於 1 個字元。

```

程式碼列表

```
1  #include <iostream>
2  using namespace std;
3
4  template<typename T> class Hello
5  {
6      private:
7          T name;
8
9      public:
10         Hello(T str)
11         {
12             name = str;
13         }
14
15         void sayHello()
16         {
17             cout << "您好，" << name << "。" << endl;
18         }
19 };
20
21 template<> class Hello<char>
22 {
23     private:
24         char name;
25
26     public:
27         Hello(char ch)
28         {
29             name = ch;
30         }
31
32         void sayHello()
33         {
34             showError();
35         }
36
37         void showError();
38 };
39
40 void Hello<char>::showError()
```

```

41 {
42     cout << " 姓名長度需要大於 1 個字元。" << endl;
43 }
44
45 int main()
46 {
47     Hello <string> cls1(" 王小明");
48     Hello<char>cls2('a');
49
50     cls1.sayHello();
51     cls2.sayHello();
52
53     system("pause");
54 }

```

程式講解

1. 程式碼第 1-2 行引入 `iostream` 標頭檔與宣告使用 `std` 命名空間。
2. 程式碼第 4-19 行為類別 `Hello` 的定義，使用類別樣板 `<typename T>`。第 6-7 行為 `private` 區段，宣告了 `T` 型別的資料成員 `name`，用於表示姓名。第 9-18 行為 `public` 區段，第 10-13 行為建構元，接收 `T` 型別的參數 `str`，用於設定資料成員 `name`。第 15-18 行為成員函式 `sayHello()`，此函式顯示打招呼的訊息：將字串 " 您好，" 與資料成員 `name` 一起輸出顯示。
3. 程式碼第 21-38 行為 `char` 型別的特製化 `Hello` 類別。第 24 行的資料成員 `name` 的資料型別改為 `char`，第 27 行的建構元所接收的參數 `ch`，其資料型別也更改為 `char`。第 32-35 行為成員函式 `sayHello()`，其中第 34 行呼叫了成員函式 `showError()` 來顯示錯誤訊息。第 37 行為成員函式 `showError()` 的函式原型宣告。

第 40-43 行為成員函式 `showError()` 的本體定義，因為寫於類別 `Hello` 之外，所以在函式名稱之前要加上類別名稱以及特製化的資料型別：`Hello<char>`。第 42 行顯示錯誤訊息。

4. 開始撰寫主函式 `main()`。程式碼第 47-48 行宣告 `Hello` 類別的 2 個物件 `cls1` 與 `cls2`，`cls1` 的參數為字串 " 王小明 "，因此會正常顯示：" 您好，王小明。"。`cls2` 的引數為字元 'a'；因此執行的是特製化的 `Hello` 類別，所以會顯示錯誤訊息：" 姓名長度需要大於 1 個字元。"。

三、範例程式解說

1. 建立專案，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。

```

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

```

2. 程式碼第 5-23 行為類別 `Gift` 的定義，此類別用於表示獎品，並使用類別樣板 `<typename T>`。第 7-9 行為 `private` 區段，有 1 個一維字串陣列的資料成員 `items`，並使用 `const` 修飾字修飾（因為獎品的內容不想被更動）；雖然抽獎的獎品只有 5 種，但為了方便起見，把鼓勵獎品也放在資料成員 `items` 的最後一項。

第 10-16 行為 `protected` 區段，有 1 個 `T` 型別的資料成員 `no`，用於表示獎品的號碼。成員函式 `getItem()` 則會回傳獎品的內容：回傳資料成員 `items` 的第 `no` 個元素 `items[no]`；因此，函式回傳值型別為 `string`。

第 18-22 行為 `public` 區段，建構元接收 1 個 `T` 型別的參數 `n`，用於設定資料成員 `no`；如第 21 行所示。

```

5 template<typename T> class Gift
6 {
7     private:
8         const string items[6] = { "小電鍋","行動電源","藍芽耳機",
9                                     "讀卡機","藍芽滑鼠","USB隨身碟" };
10    protected:
11        T no; // 獎品號碼
12
13        string getItem() // 取得獎品名稱
14        {
15            return items[no];
16        }
17
18    public:
19        Gift(T n) // 建構元
20        {
21            no = n;
22        }
23 };

```

3. 程式碼第 25-57 行為類別 `Lottery`，用於表示抽獎。此類別繼承 `Gift` 類別，所以也使用了類別樣版，如第 25 行所示；樣板中使用了 3 個型別參數 `T1`、`T2` 與 `T`，型別 `T` 是爲了父類別 `Gift` 而使用。第 28-29 行為 `private` 區段，宣告了 `T2` 型別的資料成員 `fgPresent`，用於表示抽獎者是否出席抽獎活動（所以 `T2` 型別可以使用 `bool` 型別）。

第 31-56 行為 `public` 區段，第 32 行宣告了 `T1` 型別的資料成員 `name`，表示抽獎者的姓名（所以 `T1` 型別可以使用字串型別）。

```

25 template<typename T1, typename T2, typename T>
26 class Lottery :protected Gift<T>
27 {
28     private:
29         T2 fgPresent; // 是否出席
30
31     public:
32         T1 name; // 中獎者姓名

```

4. 第 34-38 行為建構元，接收 3 個參數，並呼叫父類別的建構元。第 1 個參數 `str` 表示抽獎者的姓名。第 2 個參數 `fg` 表示抽獎者是否出席活動。第 3 個參數 `no` 表示獎品的號碼，預設值等於 5 表示預設獎品爲鼓勵獎品，並把此參數傳遞給父類別的建構元。

第 40-51 行為成員函式 `getPrize()`，用於表示抽獎，並回傳獎品的名稱；所以函式回傳值型別爲字串型別。第 42-46 行判斷資料成員 `fgPresent` 若等於 `true`，則表示出席抽獎活動，所以獎品的號碼 `no` 等於介於 0-9 之間的亂數號碼，並且號碼若大於 4 者表示未抽中獎品，所以將號碼 `no` 設定爲 5。第 48 行設定未出席者的獎品號碼 `no` 等於 5。獎品號碼 `no` 被宣告在父類別 `Gift`；因此，在存取獎品編號 `no` 時，要加上 `Gift<T>`。第 50 行呼叫父類別的成員函式 `getItem()`，回傳獎品的名稱。

第 53-56 行為成員函式 `getPresent()`，根據資料成員 `fgPresent` 的值回傳是否出席抽獎活動的訊息字串。

```

34         Lottery(T1 str, T2 fg, T no=5):Gift<T>(no)
35         {
36             name = str;
37             fgPresent = fg;
38         }
39

```

```

40     string getPrize() // 抽獎
41     {
42         if (fgPresent) // 有出席者，可以抽前 5 種獎品
43         {
44             Gift<T>::no = rand() % 10;
45             Gift<T>::no = (Gift<T>::no > 4) ? 5 : Gift<T>::no;
46         }
47         else
48             Gift<T>::no = 5; // 未出席者只能領取第 6 種獎品
49
50         return Gift<T>::getItem();
51     }
52
53     string getPresent()
54     {
55         return (fgPresent == true) ? "出席抽獎" : "未出席抽獎";
56     }
57 };

```

5. 開始於 `main()` 主函式中撰寫程式。程式碼第 61-63 行分別宣告 `Lottery` 類別的 3 個物件 `Wang`、`Mary` 與 `Li`；樣板的型別參數為 `<string, bool, int>`，分別表示抽獎者姓名、是否出席抽獎活動，以及獎品的號碼。3 個物件的引數都只有 2 個：抽獎者姓名與是否出席抽獎活動，但沒有第 3 個引數：獎品號碼；這是因為在第 34 行 `Lottery` 類別的建構元中，參數 `no` 已經有預設值了。

第 65 行使用函式 `srand()` 初始化亂數產生器，第 67-68 行顯示物件 `Wang` 的抽獎情形：呼叫物件 `Wang` 的成員函式 `getPresent()` 取得是否出席的字串，呼叫物件 `Wang` 的成員函式 `getPrize()` 取得獎品的名稱。相同的做法，第 70-71 行、第 73-74 行分別顯示物件 `Mary` 與 `Li` 的抽獎情形。

```

61 Lottery<string, bool, int> Wang("王小明", false);
62 Lottery<string, bool, int> Mary("王瑪麗", true);
63 Lottery<string, bool, int> Li("李小強", true);
64
65 srand((unsigned)time(NULL));
66
67 cout << Wang.name << ", " << Wang.getPresent() <<
68     "，獎品：" << Wang.getPrize() << endl;
69
70 cout << Mary.name << ", " << Mary.getPresent() <<
71     "，獎品：" << Mary.getPrize() << endl;

```

```

72
73 cout << Li.name << "，" << Li.getPresent() <<
74      "，獎品：" << Li.getPrize() << endl;
75
76 system("pause");

```

重點整理

1. 使用函式樣板或類別樣板，遇到指標、參考、成員指標或函式指標類型都需要樣板特製化處理。
2. 子類別繼承使用別樣板的父類別，在子類別中使用到父類別的成員時，都需要明確指定 " 父類別 < 樣板 >"，例如：

```

父類別 < 型別參數 >:: 成員函式 () { ... };
父類別 < 型別參數 >:: 資料成員 = 0;

```

3. 函式樣板或是類別樣板，都可以使用特製化來處理特定的資料型別。

分析與討論

1. 使用多個型別參數的函式樣板時，每個型別參數都必須被使用，否則會出現錯誤；例如以下的例子：

```

1 template <typename T1, typename T2> void func(T1 a, T1 b)
2 {
3     :
4 }
5
6 int main()
7 {
8     func(4, 5); ← 發生錯誤
9 }

```

↑
沒有使用到型別參數 T2

在程式碼第 1 行定義函式樣板與定義自訂函式時，在參數列沒有使用到型別參數 T2，則在第 8 行撰寫呼叫自訂函式 func() 時便會發生錯誤。

2. 樣板中的非型別參數若為 string 型別，則必須使用指標或是參考呼叫的方式，如下範例所示（參考範例檔案 03-4）。程式碼第 1 行定義函式樣板以及自訂函式 show()，樣板中有 1 個型別參數 T 與 1 個非型別參數 string，並指名參數的名稱為 str，使用參考呼叫的方式接收此參數。第 3 行顯示參數 str 與 name。

```

1  template <typename T, string& str> void show(T name)
2  {
3      cout << str << name << endl;
4  }
5
6  int main()
7  {
8      static string hello = " 你好，";
9
10     show<string, hello>(" 王小明");
11
12     system("pause");
13 }

```

在主函式中，程式碼第 8 行宣告靜態的 `string` 型別的字串變數 `hello`。要設定給樣板中的非型別參數的資料需要是靜態資料，所以字串變數 `hello` 才使用 `static` 修飾字修飾。第 10 行呼叫自訂函式 `show()` 並傳入 2 個資料：變數 `hello` 透過函式樣板中的非型別參數的參數 `str` 所接收，字串 " 王小明 " 則是直接作為引數傳遞給自訂函式 `show()`。此範例的輸出結果為：

```
你好，王小明
```

- 欲查明樣板中的型別參數被轉換為何種資料型態，可以使用 `typeid()` 函式；例如以下範例所示：

```

1  template<typename T1> void func(T1 v)
2  {
3      cout << typeid(v).name();
4  }
5
6  int main()
7  {
8      func(4);
9  }

```

執行結果

```
int
```

程式碼列表

```
1 #include <iostream>
2 #include <time.h>
3 using namespace std;
4
5 template<typename T> class Gift
6 {
7     private:
8         const string items[6] = { " 小電鍋 ", " 行動電源 ", " 藍芽耳機 ",
9                                     " 讀卡機 ", " 藍芽滑鼠 ", "USB 隨身碟 " };
10    protected:
11        T no; // 獎品號碼
12
13        string getItem() // 取得獎品名稱
14        {
15            return items[no];
16        }
17
18    public:
19        Gift(T n) // 建構元
20        {
21            no = n;
22        }
23 };
24
25 template<typename T1, typename T2, typename T>
26 class Lottery :protected Gift<T>
27 {
28     private:
29         T2 fgPresent; // 是否出席
30
31    public:
32        T1 name; // 中獎者姓名
33
34        Lottery(T1 str, T2 fg, T no=5):Gift<T>(no)
35        {
36            name = str;
37            fgPresent = fg;
38        }
39
40        string getPrize() // 抽獎
```

```

41     {
42         if (fgPresent) // 有出席者，可以抽前 5 種獎品
43         {
44             Gift<T>::no = rand() % 10;
45             Gift<T>::no = (Gift<T>::no > 4) ? 5 : Gift<T>::no;
46         }
47         else
48             Gift<T>::no = 5; // 未出席者只能領取第 6 種獎品
49
50         return Gift<T>::getItem();
51     }
52
53     string getPresent()
54     {
55         return (fgPresent == true) ? " 出席抽獎 " : " 未出席抽獎 ";
56     }
57 };
58
59 int main()
60 {
61     Lottery<string, bool, int> Wang(" 王小明 ", false);
62     Lottery<string, bool, int> Mary(" 王瑪麗 ", true);
63     Lottery<string, bool, int> Li(" 李小強 ", true);
64
65     srand((unsigned)time(NULL));
66
67     cout << Wang.name << " , " << Wang.getPresent() <<
68         " , 獎品 : " << Wang.getPrize() << endl;
69
70     cout << Mary.name << " , " << Mary.getPresent() <<
71         " , 獎品 : " << Mary.getPrize() << endl;
72
73     cout << Li.name << " , " << Li.getPresent() <<
74         " , 獎品 : " << Li.getPrize() << endl;
75
76     system("pause");
77 }

```

本章習題

1. 設計一個函式樣板，可用於台斤轉換為公斤。
2. 設計一個函式樣板，可用於計算營業員 1-4 月的平均營業額（萬元為單位）。
3. 班上段考後，平均成績達 80 分的同學，贈送筆記本。平均分數達 90 分則可以抽禮物。寫一多載的函式樣板，用於處理贈送筆記本與禮物。
4. 分析與討論的第 2 點，所使用的範例，請用類別的方式改寫，並使用類別樣板。
5. 使用類別樣板，設計一個類別 `Item`，用於記錄物品的名稱與單價。再設計另一個類別 `Buy`，繼承類別 `Item`，用於記錄購買物品的數量。寫一程式，最多購買 3 樣物品，計算購物之總金額。