第 3 章
變數

本章提要

- 3-1 甚麼是變數？
- 3-2 資料的種類：資料型別
- 3-3 常數
- 3-4 自訂資料型別
- 3-5 綜合演練

3-1 甚麼是變數？

- 當我們在進行心算時，要用大腦記住數字，然後將它們加減乘除，只要數字一多，就會發現我們的腦袋實在不爭氣，可能連要計算的數字都記不住，更不要說想要在腦中計算出答案了。這時候我們就會佩服在電視上表演心算的神童，也會覺得計算機 (電腦) 實在好用，手指頭按一按就能算出答案。
- 其實，電腦也沒有比我們高明多少，只不過它能清楚確實記住要計算的數字，也就是存放到記憶體中。

甚麼是變數？

- 在這個例子中, 數字就是所謂的資料；而計算的動作則稱為電腦在處理資料；計算出來的答案則是處理的結果 (其實也算是資料) 寫程式時, 就是在命令電腦以我們想要的方式處理資料, 而在程式中就是用變數 (Variable) 來表示儲存資料的地方。抽象一點的說法, 就直接說**變數就是儲存資料的地方**。

宣告變數

- 讓我們先來看看以下這個程式：

程式 Ch03-01.cpp 宣告變數並指派內容給變數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06     int i;          // 宣告一個整數變數 i
07     i = 123;        // 將 i 的值設爲 123
08     cout << "變數 i 的內容爲：" << i;
09 }
```

執行結果

```
變數 i 的內容爲：123
```

宣告變數

- 在這個程式中, 第 3 行 的 “int i;” 敘述的功用, 就是**宣告 (Declare)** 一個變數, 它的名字叫 做 i, 而最前面的 int 則是說這個叫做 i 的變數可以用來存放**整數 (Integer) 型態的資料**。當 C++ 編譯器看到這一行時, 就會在程式執行時配置一塊記憶體空間, 以存放資料。在這個例子中, 存放的就是在第 4 行**指定 (assign)** 給 i 的 123 。

宣告變數

- 如果把變數比擬成百貨公司服務櫃檯提供的**保管箱**, 那麼第 6 行程式的意思就等於是向櫃檯人員說**麻煩給我一個可以放整數的保管箱！**。而在程式實際執行這一個敘述時, 就相當於櫃檯的服務人員去找出一個空的保管箱, 並且將保管箱的**號碼牌**給您。如此一來, 您就擁有一個可以存放物品的地方了。

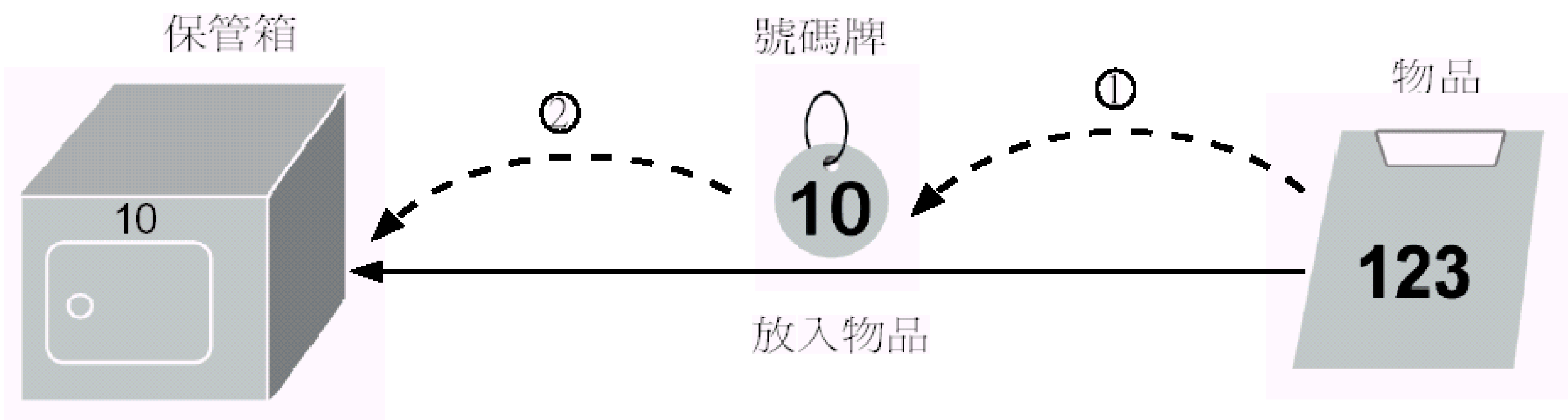
設定變數的內容

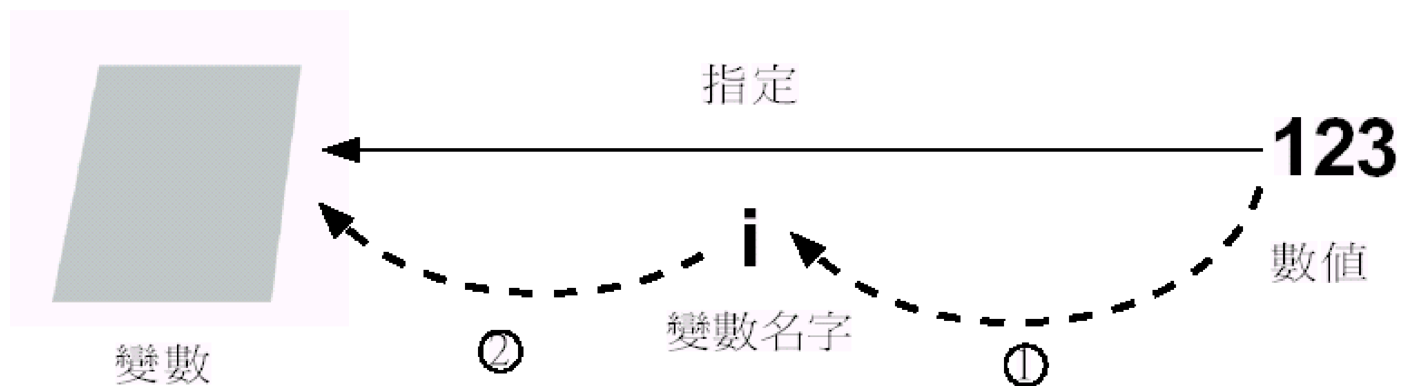
- 如同前面所說, 變數就像是個保管箱, 那麼櫃檯服務人員所給的保管箱號碼牌就相當於是這個保管箱的名字。往後當您需要放置或是取出保管箱中的物品時, 都必須出示這個號碼牌, 讓櫃檯人員依據號碼找出保管箱來幫你取出或是放置物品。

設定變數的內容

- 變數的使用也是一樣, 宣告了變數之後, 往後要存放或是取出資料時, 只要指定變數的名字即可。像是程式中的第 7 行, 就是將 123 這個數值放入名字為 i 的變數中 (就是將 123 這個物品放入 i 這個保管箱中) 在這一行中的 = **稱為指定運算子 (Assignment Operator)**, 它的功用就是將資料放到變數中。

設定變數的內容





設定變數的內容

- 程式中的第 8 行, 則是取出資料的範例。當我們需要取用變數所存的資料時, 只要在需要用到資料的地方寫上變數的名字, 實際執行程式時, 就會將資料由變數中取出, 並且取代變數名字出現在程式中的位置。以第 8 行來說, 先用 cout 輸出 數 i 的內容為：" 的訊息, 後面再用 << 串接變 數 i, 由於 i 所存的值已變成 123, 因此這行程式的功能就相當於：

```
cout  <<  "變數  i  的內容為：" <<  123;
```

↑ 代換為 i 的值

設定變數的內容

- 因此, 最後程式的執行結果就是將 “ 變數 i 的內容為：123” 這段文字顯示出來了。
- 要注意的是, 在數 i 的內容為：“ 中雖然也出現了變數 i 的名字, 但是因為使用了雙引號 ” 括起來, 因此括起來的內容會被視為是單純的一段文字, 而不會將其中 的 i 解譯為變數的名字, 所以就不會以變數 i 的內容取代。

設定變數的內容

- 變數所存的資料 (簡稱**變數值**) 是可以更換的, 所以如果我們中途改變了變數值, 則程式輸出資料時, 會以執行當時變數的內容來取代變數名字出現的位置, 例如：(見下頁)
- 由於在第 9 行重新指定新的值給變 數 i, 所以之後第 10 行程式執行時顯示的就會是 456, 而不是之前的 123 了。

設定變數的內容

程式 Ch03-02.cpp 改變變數值

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int i;
07    i = 123;      // 將 i 的值設爲 123
08    cout << "變數 i 的值爲：" << i << endl;
09    i = 456;      // 將 i 的值改成 456
10    cout << "變數 i 的值變成：" << i;
11 }
```

執行結果

```
變數 i 的值爲：123
變數 i 的值變成：456
```

變數的名稱

- 在前面的範例中, 變數的名字只是很簡單的 i, 就字面來說, 看不出有任何的意義。為了方便閱讀, 最好可以為變數取個具有說明意義的名字。舉例來說, 如果某個變數代表的是學生的年齡, 那麼就可以幫這個變數取像是 age 這樣的名字, 請參考以下實際的範例：

變數的名稱

程式 Ch03-03.cpp 改變變數值

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06     int age;      // 使用能代表變數意義的名稱
07     age = 18;
08     cout << "小妹今年 " << age << "歲";
09 }
```

執行結果

```
小妹今年 18 歲
```

變數的名稱

- 這樣一來，在閱讀程式的時候，就更容易瞭解每個變數的意義與用途，而且如果有變數用在與其意義不符的用途上時，也很容易就能發現，進而修正程式。
- 因此在替變數取名字時，就要稍微花心思替不同用途的變數取個有意義的名字。但要小心的是，在 C++ 中變數的命名方式有幾個原則不能違背，而且有些字是不能用來當成變數的名稱。

變數的命名規則

- C++ 程式中的變數名稱，需符合以下規範：
  - 可使用任何文字 (英文字母、中文字都可以) 或數字，以及"_"符號組成，其他符號都不能使用。以下所列都是合法的變數名稱：

```
aa
a_a
_aa
f15
f_16
變數01      //  不建議使用
```

變數的命名規則

- 在此要提醒讀者一點, C++ 語言標準雖然允許我們在變數名稱中使用中文 (或其他語系的文字), 但一方面某些平台可能未支援此項功能, 二來 C++ 語言本身定義的關鍵字及標準函式庫中定義的函式、變數名稱仍為英文, 寫程式時若要中文、英文切來切去難免不便, 所以建議還是只用英文來替變數命名。

變數的命名規則

- 數字不能當成第 1 個字元。例如以下都是**不合法**的變數名稱：

```
2days
19f
```

- 變數名稱不能重複。真實世界會有人同名同姓，但在程式中則不可有兩個變數使用相同的名稱，否則編譯器會發出重複定義的錯誤訊息。

- 除了以上 2 點，變數名稱也不能和 C++ 語言中的**關鍵字 (keyword)** 重複。

C++ 關鍵字

- 關鍵字 (或稱保留字) 是定義在 C++ 語言標準中, 對 C++ 編譯器有特殊意義的字符。所以在替變數命名時, 必須避開這些關鍵字, 因為 C++ 不會把它當成變數名稱, 而是會遵照該關鍵字所代表的意義及功能來編譯程式。下表就是 C++ 的關鍵字：

C++ 關鍵字

asm	do	f	return	typedef
auto	double	inline	short	typeid
bool	dynamic_cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit_cast	namespace	static_cast	using
char	export	new	struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	while
default	friend	register	true	
delete	goto	reinterpret_cast	try	

變數名稱和關鍵字都屬於字符

- 前一章談過**字符**的觀念，字符是組成 C++ 敘述的要素之一，而變數的名稱就是字符的一種。C++ 的字符可分為以下幾類：
  - 識別字 (identifier)：簡單的說，識別字就是在 C++ 程式中用來代表一樣東西的名稱。例如變數名稱就是識別字的一種，在後面幾章介紹的函式名稱、類別名稱也都是識別字。定義在標準函式庫中的名稱 (例如 cout)，也是識別字。

變數名稱和關鍵字都屬於字符

- 關鍵字 (keyword)：在 C++ 語言中有特別意義的字, 例如我們已用過的 using、int。
- 字面常數 (literal)：在程式中直接寫出來的數字、字串等, 例如在前一個範例程式中出現的 123、456 等。
- 運算子 (operator)：也就 是 C++ 的運算符號, 例如 =>>。
- 標點符號 (punctuator)：例如分號、大括號等。

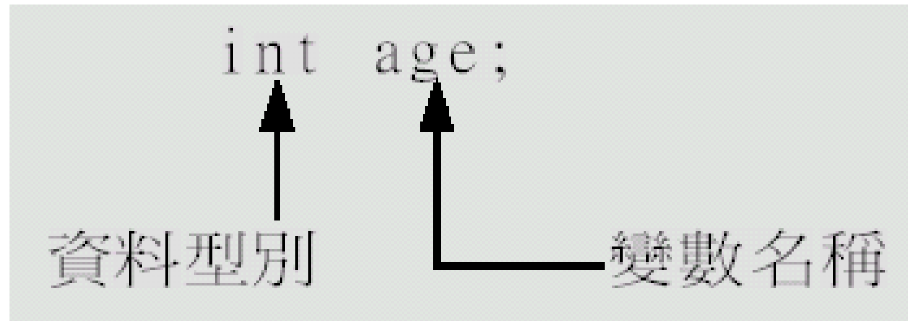
3-2 資料的種類：資料型別

- 使用任何變數之前，都需要先**宣告(declare)**。宣告變數的目的，除了讓編譯器認識這個變數名稱外，另外也是告訴編譯器，這個變數是用來存放什麼類型的資料，這些資料種類，在程式語言中稱為**資料型別(Data Type)**。
- 宣告變數的語法如下：

```
資料型別 變數名稱;
```

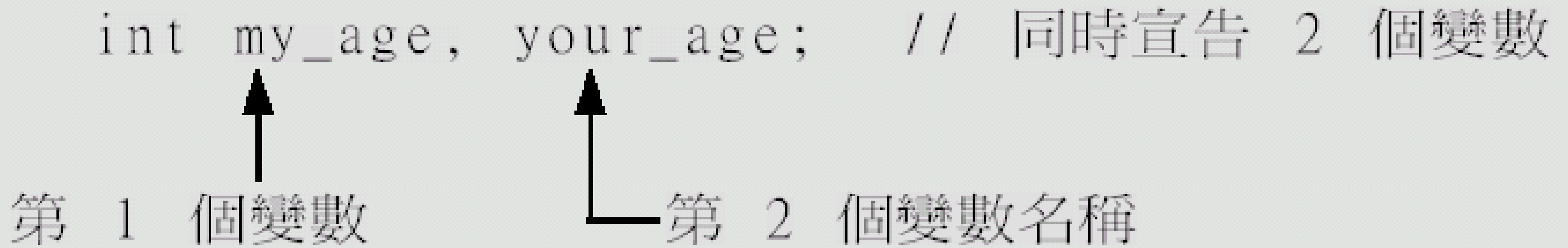
資料的種類：資料型別

- 例如前一個範例程式中的：



- 此外我們可在同一敘述中，同時宣告多個**同型別**的變數，只要用逗號將變數名稱分開即可：

資料的種類：資料型別



- 每一種資料型別所使用記憶體大小不盡相同 (位元組數不同), 所以當我們宣告了變數所屬的型別, 編譯器就會配置適當的空間給該變數來存放資料。

整數

- 前面我們已使用過的 int 是**整數 (integer)** 資料型別, 可用來記錄如 1、2200、-3 之類的正負整數。
- 整數變數所佔的記憶體空間為 32 位元 (4 個位元組), 所以可表示的數字範圍為 -2147483648 (負的 2^{16}) 與 2147483647 (2^{16} -1)。
- 指定給 int 變數的數值若超過 int 允許的範圍, 將會使變數所存的數值變得不正確, 例如：

整數

```
int verybig;
verybig = 2345678900;    // 超過 2147483647
cout << verybig;         // 將在螢幕輸出 "-1949288396"?
```

- 第 2 行指定超過 2147483647 的數值給 verybig 時, 由於其值超過 int 變數可表示的範圍, 所以將導致變數所存的數值不正確。

整數型別的修飾字

- 為了讓我們能以更彈性的方式使用整數變數, C++ 允許我們在宣告整數時, 在 int 前面加上**修飾字**, 來改變整數資料的範圍與使用的記憶體大小, 修飾字有 3 種, 分列如下：
  1. **short**： short int 將整數資料型別變成只有 16 位元 (2 bytes)。所以可表示的數值範圍變成從 -32768～32767。
  2. **long**：在目前的 32 位元個人電腦上, long int 其實和 int 一樣, 都是用 32 位元 (4 bytes) 來記錄整數值。

整數型別的修飾字

但如果在 64 位元的環境下, 則 long 會變成 64 位元, 可表示的數值範圍也變成 -9223372036854775808 (負的 2^{63})～9223372036854775807 (22^{63}-1)。

3.**unsigned**：unsigned 不會改變整數佔用的記憶體大小, 且可與前 2 個修飾字一起使用。其功用是讓 C++ 的整數變成只能表示 0 及正整數。例如 unsigned int 可記錄範圍 0 ~ 4294967295 之間的整數。有些人將 unsigned 類型的整數稱為無號 (無正負號) 的整數。

整數型別的修飾字

- 使用修飾字時, 可將修飾字加在整數型別 int 之前, 但也可省略 int 這個關鍵字, 如以下範例程式所示：(見下頁)
- 第 11 行程式故意將 k 的值設為 long 最大值 (2147483647) 加 1, 結果因超出 long 可表示的範圍, 使 k 的值反而變成負的 -2147483648。至於為何恰好是變成 long 可表示的最小值, 與電腦中記錄數值的方式有關。

整數型別的修飾字

程式 Ch03-04.cpp 整數型別的修飾字

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06     unsigned int i;   // 變數 i 只能記錄正整數或 0
07     short j;   // 省略 "int" 關鍵字
08     long  k;   // 省略 "int" 關鍵字
09     unsigned long  l;   // 省略 "int" 關鍵字
10     i = 1999;
```

整數型別的修飾字

```
11     j = 32767;
12     k = 2147483648;   // 故意設為 long 最大值加 1
13     l = 4294967295;
14     cout << i << ' ' << j << ' ' << k << ' ' << l << endl;
15 }
```

執行結果

```
1999 32767 -2147483648 4294967295
```

整數型別的修飾字

- 讀者可能會被 short、int、long 的大小弄混, 其實平常大部份的程式也只會用到 int 型別, 真要分辨三者, 也只需 把 short 看成是較小的整數型別；long 是較大的整數型別；而 int 則是**最普通**的整數型別, 且其大小是視作業系統、編譯器種類而定。
- 要想知道您所使用的作業系統及編譯器組合, 會產生多大的 short、int、long 變數, 可使用 C++ 的 sizeof() 運算子來查看。

sizeof() 運算子

- sizeof() 運算子的用途是查看變數所佔的記憶體空間大小, 也可查看各資料型別會使用的記憶體空間。而其傳回的大小值是以位元組為單位, 所以當您不確定目前所用的作業系統 / 譯器的 short、int、long 各佔幾個位元組, 就可用如下的程式來檢查：

sizeof() 運算子

程式 Ch03-05.cpp 查看各資料型別所佔的記憶體大小

```
01 #include <iostream>
02 using namespace std;
03
04 int main()
05 {
06    cout << "short: " << sizeof(short) << " 位元組" << endl
07         << "int  : " << sizeof(int)   << " 位元組" << endl
08         << "long : " << sizeof(long)  << " 位元組" << endl;
09 }
```

執行結果

```
short: 2   位元組
int  : 4   位元組
long : 4   位元組
```

sizeof() 運算子

- 此程式分別顯示 short、int、long 三種資料型別的大小 (unsigned 的大小和未加 unsigned 時是一樣的), 在本例中 int 和 long 一樣都是 4 個位元組 (32 位元), 但在不同的環境中可能會有不同的結果。例如我們在 64 位元 Linux 中使用 g++ 編譯此程式, 執行所得的結果是 short 和 int 都是 4 個位元組 (32 位元), 但 long 則是 8 個位元組 (64 位元) 。

字元

- C++ 的字元 (character) 型別有 2 種：
  - **char**：大小為 8 位元, 可記錄一般英文、數字字元, 或是標點符號等。
  - **wchar_t**：大小為 16 位元, 可記錄包括中日韓文字在內的雙位元組 (double bytes) 字集的字母。這是 C++ 為因應國際化而加入的資料型別, 不過一般寫程式時不常使用, 稍後的程式範例也會示範其不適用的理由。

字元

- 每個字元變數都只能記錄 1 個英文字母、數字字元或符號, 在 C++ 要表示一個字元時, 可用單引號 (‘) 括住字元：

'a'	←	小寫字元 a
'M'	←	大寫字元 M
'+'	←	加法符號

- 以下就是個使用字元變數的簡單範例：

字元

程式 Ch03-06.cpp 使用字元變數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   char c1, c2;          // 宣告 2 個字元變數
07   wchar_t c3;
08   c1 = 'a';   // 將字元 'a' 指定給變數
09   c2 = 65;    // 將 ASCII 碼 65 的字元指定給變數
10   c3 = '大';
11   cout << c1 << endl << c2 << endl << c3 << endl;
12 }
```

執行結果

```
a
A
42090
```

字元

- 此程式的內容很簡單, 比較需要解釋是第 9 行, 此敘述將 65 指定給 c2 字元變數, 其作用等於是將字元 'A' 指定給 c2 變數, 因為大寫 A 的 ASCII 碼為 65。什麼 是 ASCII 碼呢?
- 我們知道在電腦中可記錄的資料基本上只有數值, 所以文字其實也是以數值的方式來表示, 為讓所有電腦系統都用一致的方式來表示相同的字元,

字元

例如 65 代表大寫 A、97 代表小寫 A, 這種以特定數值代表某個字元的方法就稱為**編碼**, 目前資訊界一般所用的編碼為美國標準資訊交換碼 (ASCII, American Standard Code for Information Interchange)。

- 所以當我們直接指定數值 65 給變 數 c2 時, 其實和指定 ‘A’ 給它是一樣的意思。因此從 cout 輸出 c2 的值時會看到字元 ‘A’, 而不會看到數值 65。

字元

- 至於程式第 10 行指定 ‘大’ 這個字給 c3 時, 輸出的內容卻變成 42090 這也是文字編碼所造成的, 因為 ‘大’ 這個字 的 Unicode 編碼就是 42090 (若以 16 進位表示則是 A46A)。但 cout 預設在輸出 wchar_t 型別的資料時, 並不會以字元的形式輸出, 而是以數字的形式輸出, 所以我們看到的就是 ‘大’ 的 Unicode 編碼了。這也是前面提到使用 wchar_t 型別不便之處, 因此不建議大家使用這個型別。

利用字元型別處理數字

- 由於 char 本質上存的是數值 (字元的編碼), 所以我們也可用它來存放整數資料, 可代表的數值範圍為 -128～127。當然我們也可在 char 前加上 unsigned 表示變數只能儲存 0 及正整數。然而 cout 預設在輸出時, 都是將 char/unsigned char 變數輸出成對應編碼的文字：

```
unsigned char x;
x = 100;
cout << x;          // 將會顯示字母 'd' (ASCII 碼 100)
```

浮點數

- 當程式需用到含有小數 (例如 3.14), 或是超過整數型別可表示範圍 (例如 10 的 20 次方) 的數字時, 就需改用浮點數 (floating point) 型別的變數。C++ 的浮點數型別有下列 3 種：
  - **float**：浮點數 (floating point) 資料型別, 如 1.1、2.22 大小為 4 個位元組, 可表示範圍為 1.175e-38 與 3.402e38 間的浮點數。

浮點數

- **double**：倍精度浮點數 (double precision floating point) 資料型別, 也是用來存放浮點數型別, 但是小數的位數比 float 多, 而且精確度更高。大小為 8 個位元組, 可表示範圍為 2.225e-308 與 1.7976e308 間的浮點數。
- **long double**：擴充精確度的浮點數資料型別, 和 long int 的情況相似, 在一般 32 位元電腦 上 double 和 long double 都是相同的。

- 以下範例程式就是將圓周率指定給 flaot、double 型別的變數, 並查看其精確度：

浮點數

程式 Ch03-07.cpp 使用浮點數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   float f_pi;
07   double d_pi;
08   // 將圓周率值指定給變數
09   f_pi   = 3.141592653589732f;
10   d_pi   = 3.141592653589732;
11
12   cout.precision(17);
13   cout << "f_pi = "  << f_pi  << endl
14        << "d_pi = "  << d_pi  << endl;
15 }
```

執行結果

```
f_pi = 3.1415927410125732
d_pi = 3.1415926535897931
```

浮點數

1. 第 9、10 行將圓周率值 (表示到小數點後 20 位) 指定給 float 及 double 型別的變數 (其中第 9 行最後一個數字後的 f, 只是表示此數字為浮點數) 。稍後在第 13 及 14 行則輸出 2 變數的值。
2. 第 12 行的程式是設定 cout 在顯示浮點數時, 要顯示的數字位數。讀者可先不管這行程式的語法和意義, 此處只是為示範 float 和 double 的差異才加上此敘述。

浮點數

- 由輸出結果可發 現 float 變數記錄的數值精確度低於 double 很多, 當我們將同樣的數值指定給 f_pid_pi 時, f_pi 從小數點後第 7 位數就開始和我們指定的值不同, 但 d_pi 則在小數點後第 16 位數才不同, 但其精確度已足夠一般情況值用 。因此如果要進行較精確的科學計算時, 應使 用 double 或 long double, 而不要使用 float 。

布林型別

- 布林型別 bool 可表示**真 (true)**、**假 (false)** 兩種狀態, 在做邏輯、比較運算時就會得到布林型別的真假值。在設定 bool 型別的變數 (通常稱為**布林變數**) 值時, 可使用 C++ 關鍵字 truefalse 來表示其值為真或假。
- 其實在 C++ 中, bool 是以整數的方式來記錄, 0 相當於 false, 1 則為 true, 所以在設定布林變數值時, 也可直接用 0、1 來設定。

布林型別

程式 Ch03-08.cpp 示範使用布林變數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    bool test1,test2;
07    test1 = true;
08    test2 = 0;       // 相當於設爲 false
09    cout << "test1 = " << test1 << endl
10         << "test2 = " << test2 << endl;
11
12    // 改用文字的方式輸出布林值
13    cout << boolalpha;
14    cout << "test1 = " << test1 << endl
15         << "test2 = " << test2 << endl;
16 }
```

執行結果

```
test1 = 1
test2 = 0
test1 = true
test2 = false
```

布林型別

1. 第 7、8 行程式分別將不同的值指定給布林變數, 其中第 7 行使用關鍵字 true 來指定；第 8 行則是將變數值指定為 0, 這就相當於指定為 false。
2. 第 9、10 行分別輸出 2 個變數值。由輸出結果可發現, cout 預設是將 true 顯示為 1、false 顯示 為 0。

布林型別

3. 第 13 行的程式是設定 cout 在顯示布林變數時, 改用文字的方式來表示, 而不再使用 0、1 來表示。更改 cout 的顯示方式後, 第 13、14 行再次輸出變數值, 就變成是以 true、false 的文字來顯示了。

設定變數的初值

- 當程式宣告一個變數, 獲得一塊記憶體空間來存放變數值時, 變數最初的值是**不確定的**, 因此我們必須設定變數的初值, 然後讓程式做必要的處理, 才能產生我們所期望的結果。
- 在前面的範例程式中, 都是先宣告好變數, 然後再設定變數的內容, 其實還有較便利的作法, 就是在宣告時就同時設定其初值, 而且可以用運算式來設定初值。

宣告同時設定初值

- 在宣告變數的同時就設定該變數的初值，也稱為**定義 (definition)**。舉例來說，以下的程式就是改寫自前面的範例程式，只是宣告變數時即設定初值：(見下頁)
- 第 6、7 行在宣告變數時，即設定變數的初值，所以稍後輸出時，就會看到所設定的值。利用這種方式，就可以省掉額外撰寫設定變數內容的敘述。

宣告同時設定初值

程式 Ch03-09.cpp 宣告同時設定初值

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int age = 18;
07    double rate = 31.685;
08
09    cout << "小妹今年 " << age << " 歲" << endl;
10    cout << "今天台幣兌美元的匯率是 " << rate << endl;
11 }
```

執行結果

```
小妹今年 18 歲
今天台幣兌美元的匯率是 31.685
```

宣告同時設定初值

- 此外, 如果在同一敘述宣告多個變數時, 也可一一指定不同的初值, 例如：

```
int age = 18, height = 170, weight = 50;
```

- 此敘述就相當於宣告了 age、height、weight 三個整數變數, 同時將它們的初值分別設定為 18、170、50。

用運算式設定變數的初值

- 前面的範例程式中曾用運算式指定給變數, 也就是讓變數值等於運算式的計算結果。同理, 在設定變數初值時, 也可以直接用運算式來設定。因為最基本的四則運算都可以用在 C++ 程式中：(見下頁)
- 第 6 行最後面的 "sum = i + j + k" 意思就是宣告 sum 變數, 同時將 "i + j + k" 的總和設定為其初值。由於前面已先設好 I、j、k 的初值, 所以程式也能自動算出正確的和, 並設定給變數 sum。

用運算式設定變數的初值

程式 Ch03-10.cpp 宣告同時設定初值

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   int i = 10, j = 20, k = 30, sum = i + j + k;
07   cout << "總和等於： " << sum << endl;
08 }
```

執行結果

```
總和等於：60
```

由鍵盤輸入取得變數的值

- 我們也可用**標準輸入** cin, 從鍵盤取得變數值, 例如：(見下頁)
- 第 8 行的敘述會將使用者輸入的數值指定給 age, 例如在上列的執行例中, 我們輸入了 20, 所以 age 所存的數值就是 20。

由鍵盤輸入取得變數的值

程式 Ch03-11.cpp 從鍵盤取得變數值

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int age;
07    cout << "請問今年貴庚：";
08    cin >> age;              // 將使用者輸入的值存到 age
09    cout << "您今年 " << age << " 歲";
10 }
```

執行結果

```
請問今年貴庚：20  ←— 輸入 "20" 後按下 Enter 鍵
您今年 20 歲
```

3-3 常數

- 前面我們介紹了變數的宣告與定義，另外有一種資料和變數相對，稱為**常數 (Constant)**。顧名思義，變數所存放的資料隨時可以改變，因此稱為變數；而常數所儲存的資料則是**恆常不變**，因此稱之為**常數**。
- 在 C++ 中，可將常數的形式歸類為三種：**字面常數 (Literal)**、**唯讀變數**及**巨集常數**。

字面常數

- 所謂的**字面常數**, 就是直接在程式中以文字表達其文字或數值, 在之前的範例程式中其實已經用過許多次了。例如：

```
char c1 ='a';
int i = 10, j = 20, k = 30;
double pi = 3.14159;
```

- 上列的程式片段中, **'a'**、**10**、**20**、**30**、**3.14159** 等就是字面常數, 直接看其文字, 就可以瞭解其所代表的數值。

字面常數

使用字面常數就是這麼簡單，不過有幾點需要注意：

- char 型別的字面常數以字元來表達時，必須以單引號括起來，例如 'a'。
- 如果要表示一串文字，則必須用一對雙引號 (“) 括起來，例如 “ How are You ”、“ 這是一串文字”。

- 此外在 C++ 中還有一些特別的字面常數表示方式，以下就先介紹各種數值常數的表示法。

各種數字表示法

- 前面設定 float 變數的範例中, 我們在 3.1415926535 這個數值後面加了 f, 其作用是告訴編譯器, 3.1415926535 這個字面常數是 float 型別。在 C++ 中, 數值型的字面常數有以下幾個特性：
  - 整數字面常數預設為 int 或 long 型別 (以 Visual C++ 為例, 預設為 int)。我們也可在數字後面加上 u (unsigned)、l (long) 指定其型別, 且這 2 個字可放在一起使用：

各種數字表示法

```
123u      // 無正負號的整數 123
987l      // 長整數 123
555ul     // 無正負號的長整數 555
```

- 任何有小數點的字面常數預設為 double 型別。可加上 f 或 l 指定其為 float 或 long double ：

```
3.14f     // float 常數 3.14
.314l     // long double 常數 0.314
```

- 除了標記字面常數的型別外, 也可用不同的方式來表示數值：

各種數字表示法

- 整數字面常數可使 用 8 進位或 16 進位表示：第 1 個數字是 0 時, 此數值為 8 進位；若以 ‘0x’ 為首, 則此數值為 16 進位表示。
- 浮點數除了用一般的小數點數值表示, 也可用科學記號表示, 也就是在數值後面加上 e (大小寫均可), 再加上該數字為 10 的幾次方。

- 以下就是個簡單的範例：(見下頁)
- 這個範例程式分別用 8 進位及 16 進位的字面常數設定 int 變數的值, 以及用科學數字表示法設定 double 型別變數值。

各種數字表示法

程式 Ch03-12.cpp 各種字面常數表示法應用

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   int i;                  double d;
07   i = 074;                d = 6.02e22;
08   cout << "i= " << i << "      d= " << d << endl;
09   i = 0xACE;              d = 3.14159E-1;
10   cout << "i= " << i << "    d= " << d << endl;
11 }
```

執行結果

```
i= 60        d= 6.02e+022
i= 2766      d= 0.314159
```

特殊字元表示法 - Escape Sequence

- 前面學過字元常數要用單引號 (‘) 括起來表示, 而字串常數則要用雙引號 (“)。但如果我們要將 char 變數值設為單引號, 或是字串內也要包含雙引號, 我們不能用如下的方式表示：

```
char  quote = ''';                    //  不合法
cout << "他對我說："Hello"";         //  不合法
```

特殊字元表示法 - Escape Sequence

- 上述兩種表示方式都是不合法的, 在編譯時會出現錯誤, 因為 C++ 編譯器無法解讀程式。例如 **“ 他對我說：”Hello“”** 會被看成是“ 他對我說：” 及 “” 兩個字串, 而中間的 Hello 則 C++ 編譯器不曉得是什麼, 因此視為語法錯誤。

特殊字元表示法 - Escape Sequence

- 為解決要在字面常數中使用這類特殊符號或字元的問題, 在 C++ 程式中需以 Escape Sequence (跳脫序列) 來表示特殊字元和符號。Escape Sequence 的寫法是以反斜線 (\) 開頭, 後面跟著一個英文小寫字母或符號, 來表示特殊的控制字元及符號：

特殊字元表示法 - Escape Sequence

符號	ASCII 碼	代號符號	用途
\a	0x07	BEL	嗶聲
\b	0x08	BS	退位
\f	0x0C	FF	換頁
\n	0x0A	LF	換行
\r	0x0D	CR	歸位(游標移至行首)
\t	0x09	HT	水平定位(tab)
\v	0x0B	VT	垂直定位
\\	0x5C	\	反斜線
\'	0x2C	'	單引號
\"	0x22	"	雙引號
\?	0x3F	?	問號

特殊字元表示法 - Escape Sequence

- 上表前半的控制字元是屬於**看不到**的字元, 用 cout 輸出這些字元, 只會產生該字元對應的效果, 例如 cout << '\n' 將會使 cout 輸出換行, 和使用 "cout << endl" 的效果一樣。這些 Escape Sequence 也可以放在字串中使用, 請參考以下的應用實例：

特殊字元表示法 - Escape Sequence

程式 Ch03-13.cpp 脫序字元的應用

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    char beep = '\a';
07    cout << "今天天氣很好\n"
08         << "接著你會聽到三聲\"嗶聲\"\n";
09    cout << "嗶!\t" << beep << "嗶!\t" << beep << "嗶!" << beep;
10 }
```

特殊字元表示法 - Escape Sequence

執行結果

```
今天天氣很好
接著你會聽到三聲 " 嗶聲 "
嗶!        嗶!        嗶!
```

← 同時會聽到電腦喇叭發出聲音

1. 第 6 行將字元變數 beep 設為控制字元 '\a', 也就是會使電腦喇叭發出嗶聲的字元。
2. 第 7、8 行分別輸出 2 個字串, 這兩個字串結尾都有加上換行字元 '\n', 所以輸出字串後, 下一個輸出就會換行輸出。

特殊字元表示法 - Escape Sequence

3. 第 9 行交替輸出 “!\t” 字串與 beep 變數, ‘\t’ 是定位字元 (tab), 這就好比我們在打字時先打出 “!” 然後按一下 Tab 鍵, 再打下一組字串一樣, 所以螢幕上輸出的結果就是 3 個 “!”, 且中間有一些空白。至於輸出 beep 並不會在螢幕上看到任何文字輸出, 只會聽到電腦喇叭發出三次嗶聲 (因為我們輸出 beep 三次) 。

用字元編碼指定字元

- Escape Sequence 除了用來指定控制字元與特殊符號外，也可用**指定字元編碼**的方式來表示特定的字元，表示方法如下：
  - 在反斜線後面加上 8 進位數值，就等於是該 ASCII 編碼所代表的字元，例如：

```
'\141'     // 8 進位數字 '141' 等於十進位的 97，
           // ASCII 碼 97 為小寫字元 'a'
'\12'      // ASCII 碼 10 為換行字元 '\n'
```

用字元編碼指定字元

- 若數字的開頭為 x 或 X, 則該數值是 16 進位的 ASCII 碼, 例如：

```
'\x61'      // 同 '\141', 也就是字元 'a'
'\xC'       // 同 '\12' , 也就是 '\n'
```

- 若數字的開頭為 u 或 U, 則該數字是 16 進位的 Unicode 編碼, 例如：

```
'\u706B'   // Unicode 中 '706B' 所指的字元爲 '火'
```

const 唯讀變數

- 如果一個數值具有某種特殊的意義，而且在程式中經常會被用到，那我們就可以用一個較有意義的識別字來代替這個常數，例如：圓周率用 PI 來代表，稅率用 TaxRate 來代表。這樣做有兩個好處：
  - 使常數的意義一目瞭然：假設稅率是 0.05 (5%)，那麼 TaxRate 要比 0.05 容易讓人了解。
  - 使常數的值易於修改：例如當稅率更動時，我們只要改變 TaxRate 的定義值即可，而不必更改程式的其他部份。

const 唯讀變數

- C++ 提供二種方法來定義常數，第一種是用唯讀變數，另一種則是利用前置處理指令。以下先介紹唯讀變數的用法。
- 當我們在定義一個變數時，如果在最前面加上關鍵字 “const”，那麼這個變數就會成為一個**唯讀變數 (constant variable)**，它的值將永遠無法改變。例如：

```
const double PI = 3.14159;
```

const 唯讀變數

- 由於唯讀變數的值不能改變，所以在定義時一定要給它一個初值。在定義好之後，任何企圖改變其值的敘述都將造成編譯時的錯誤。下面是一些錯誤的範例：

```
const char str;              // 錯誤：沒有指定初值

const int maxSize = 10;
maxSize = 20;                // 錯誤：不能更改 const 變數的值
```

- 唯讀變數除了值不能更改以外，它和一般變數的性質完全相同。

const 唯讀變數

程式 Ch03-14.cpp 使用唯讀變數定義圓周率

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06     const double PI = 3.1415926;   // 定義唯讀變數
07     double area;
08
09     // 用唯讀變數計算圓面積
10     area = 5 * 5 * PI;
11     cout << "半徑 5 的圓面積等於 " << area << endl;
12
13     area = 15 * 15 * PI;
14     cout << "半徑 15 的圓面積等於 " << area << endl;
15 }
```

執行結果

```
半徑 5 的圓面積等於 78.5398
半徑 15 的圓面積等於 706.858
```

巨集常數

- 巨集常數是以前置處理指令 #define 來定義，其格式如下：

```
#define    常數名稱    常數值
```

- 例如：

```
#define    PI  3.1415926
#define    TAX_RATE    0.05
#define    KB    1024
```

巨集常數

- 巨集常數通常都是定義在程式的開頭，並且都用大寫文字，以便和其他的變數或函式名稱有所分別。一旦定義好巨集常數之後，我們便可以在程式中使用該常數，在編譯程式前，前置處理器會先把所有的常數名稱代換成常數值，然後才真正開始編譯。以下的範例程式改用 #define 的方式來定義圓周率：

巨集常數

程式 Ch03-15.cpp 使用唯讀變數定義圓周率

```
01 #include<iostream>
02 using namespace std;
03 #define PI 3.1415926      // 定義巨集常數
04
05 int main()
06 {
07    double area;
08
09    // 用巨集常數計算圓面積
10    area = 7 * 7 * PI;
```

巨集常數

```
11     cout << "半徑 7 的圓面積等於 " << area << endl;
12
13     area = 9 * 9 * PI;
14     cout << "半徑 9 的圓面積等於 " << area << endl;
15 }
```

執行結果

```
半徑 7 的圓面積等於 153.938
半徑 9 的圓面積等於 254.469
```

巨集常數

- 當我們編譯此程式時, 前置處理器會將程式中所有的 PI 代換成 3.1415926, 然後才開始編譯。所以第 10、13 行中的 PI, 在編譯前都已變成是**字面常數** 3.1415926。
- 注意, 前置處理器並不懂 C++ 的語法, 它只是單純地執行文字代換的工作, 而不會為我們做型別或是語法的檢查, 這是和使用 const 唯讀變數最大的不同之處。

3-4 自訂資料型別

- 為了模擬真實世界的事物, C++ 允許我們利用內建的資料型別定義出較複雜的**自訂資料型別** (User Defined Data Type), 以表示特定的資料型式。以下就來介紹兩種自訂新型別的方式：
  - **定義新的整數型別**：以 enum 定義列舉型別。
  - **為型別建立別名**：用 typedef 為資料型別定義別名。

列舉型別：enum

- **列舉型別 (Enumeration type)** 可以讓我們定義新的整數型別, 並用列舉的方式來指出所有可能的成員。其定義方式為：

```
enum  新型別名稱  { 列舉出所有成員 } 變數名, 變數名,... ;
```

- 其中, 新型別名稱或變數名都可省略, 而大括弧內所列舉的各成員就稱為**列舉成員** (Enumerator) 。例如, 我們可以定義某種水果茶有三種口味：

列舉型別：enum

```
enum fruit_tea { apple, banana, orange };
fruit_tea taste;              // taste 是 fruit_tea 型別的變數
```

- 如此一來, 我們要設定變數 taste 的值時, 就只能用 3 個列舉成員 apple banana、orange 來設定。而列舉成員其實也依序代表從 0 開始的整數值, 所以 apple、banana、orange 就分別代表 0、1、2。請參考以下的範例：

列舉型別：enum

程式 Ch03-16.cpp 使用列舉型別定義整數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   enum fruit_tea { apple, banana, orange };
07   fruit_tea taste;    // taste 是 fruit_tea 型別的物件
08
09   taste = apple;      // 需用列舉成員來設定其值
10   cout << "taste = " << taste << endl;
11
12   taste = orange;
13   cout << "taste = " << taste << endl;
14 }
```

執行結果

```
taste = 0
taste = 2
```

列舉型別：enum

- 大家或許會覺得這樣不是讓寫程式變得複雜了嗎？直接將 taste 宣告成 int 變數, 然後用字面常數 0、1、2 來設定豈不是比較方便？雖然看起來是如此, 但使用 enum 有下列好處：
  - 讓程式看起來比較有意義。因為我們把單純的數值用有意義的名稱代替, 所以程式的可讀性也提高, 更容易看出設定變數值代表的意義。

列舉型別：enum

- enum 可以把變數的值限定在列舉之名稱集合裡, 如此可減少程式可能的錯誤。例如 taste 值就只能是 0、1、2, 而不能指定為其它的值, 否則編譯時就會出現錯誤。但使用一般 int 變數則無法做到此種限制, 如果不小心設定了不應使用的值, 必須自行檢查程式, 不能靠編譯器幫忙。

列舉型別：enum

- 列舉成員預設的值是從 0 開始, 不過我們也可以自行指定它們所代表的值, 例如：

```
enum fruit_tea { apple=3, banana, orange };
```

- 此時 apple、banana、orange 代表的值就變成 3、4、5。當然我們也可將列舉成員設為不連續的值, 甚至指定相同的值都可以：

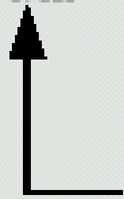
列舉型別：enum

```
enum fruit_tea { apple=5, banana=2, orange };   // 代表值變成 5、2、3

enum boolean { False, True, off=0, on }; // False 和 off 都是 0
                                         // True 和 on 都是 1
```

- 在定義列舉型別時可同時定義所要使用的變數,且型別名稱如果不再使用,也可以將之省略:

```
enum { blue, red, yellow } color;
```



列舉型別：enum

- 最後要提醒讀者：若定義了多個列舉型別，即使不同型別中的列舉成員代表的值是一樣，也不能將它們交錯使用。因為對 C++ 而言，它們代表的是**不同**的資料型別，所以這樣的程式將會造成編譯錯誤：

```
enum fruit_tea { apple, banana, orange } taste;
enum colors { blue, red, yellow } ;

taste = blue;      // 錯誤：blue 不屬於 fruit_tea 型別
```

用 typedef 為型別建立別名

- 用 typedef 關鍵字可以為任何已存在的資料型別建立別名，其目的是為了使用上的方便，以及增加程式的可讀性。其建立方式如下：

```
typedef 型別 新的別名;
```

- 舉例來說，若您想使用 unsigned char 變數，但覺得這個名稱實在太長了，對打字及閱讀都造成一些負擔，可以替它取個較簡短的名字來代替，例如：

```
typedef unsigned char byte;
```

用 typedef 為型別建立別名

- 這行敘述就是將 byte 定義為 unsigned char 型別的別名, 接著我們就能用 byte 來代替 “unsigned char” 這一大串字了。通常在定義了型別的別名後, 我們會希望能在所有的程式都能用到它, 所以會將它放在一個自訂的含括檔中,例如：

程式 Ch03-17.h 用 byte 取代 unsigned char 的含括檔

```
01 typedef unsigned char byte;
```

用 typedef 為型別建立別名

- 以後在撰寫程式時, 只要含括這個自訂的含括檔, 就能改用 byte 來宣告 unsigned char 型別的變數了：(見下頁)
- 前一章提過, 含括自訂的 .h 檔時, 需用雙引號括住檔名, 所以第 2 行的寫法是 #include"Ch03-17.h" 而非 #include<Ch03-17.h>。

用 typedef 為型別建立別名

程式 Ch03-17.cpp 用 byte 宣告變數

```
01 #include<iostream>
02 #include"Ch03-17.h"
03
04 int main()
05 {
06   byte a = 8, b = 88; // a、b 都是 unsigned char 型別的變數
07   std::cout << a * b;
08 }
```

執行結果

```
704
```

用 typedef 為型別建立別名

- 其實在 C++ 標準函式庫中就有很多使用 typedef 的應用實例, 例如代表資料大小或長度的變數, 就經常是宣告為 size_t 型別, 這個資料型別可能是 unsigned int 或其它整數型別 (視各編譯器的實作方式及應用環境各有不同), 但因為在標準函式庫中以 size_t 來代表, 如此一來我們就能以一致的方式來表示一種代表大小或長度的資料。

3-5 綜合演練

- 交換兩變數的值
- 大小寫轉換

交換兩變數的值

- 兩個同型別的變數可以用等號將其值指定給對方, 比如說 a = b, 就是將變數 b 的值指定給 a, 換句話說變數 a 與 b 的值會變成相同。我們可以用這個原理來作兩個變數值的交換：

交換兩變數的值

程式 Ch03-18.cpp 兩變數值互換

```
01 #include <iostream>
02 using namespace std;
03
04 int main()
05 {
06     int a=10, b=20;    // 宣告兩個 int 變數 a, b
07     int temp;          // 宣告暫存整數變數 temp
08
09     cout << "交換前 a = " << a << "\tb = " << b << endl;
10
```

交換兩變數的值

```
11    temp = a;    // 將變數 a 的值指定給暫存變數
12    a = b;       // 把變數 b 的值指定給 a
13    b = temp;    // 將暫存變數所存的 a 值指定給 b
14
15    cout << "交換後 a = " << a << "\tb = " << b << endl;
16 }
```

執行結果

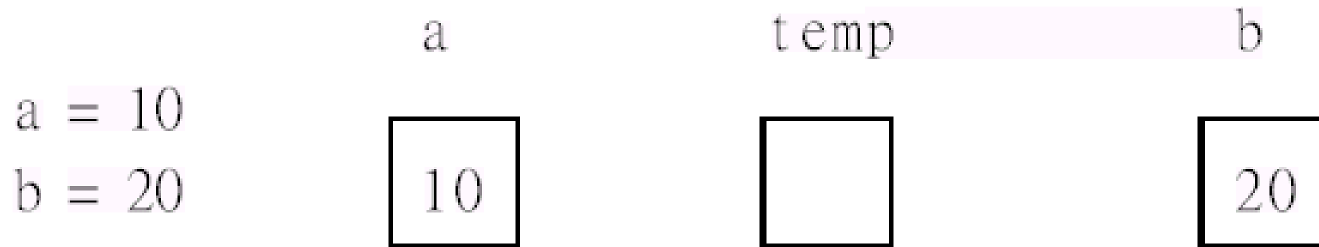
```
交換前 a = 10    b =20
交換後 a = 20    b =10
```

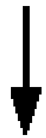
交換兩變數的值

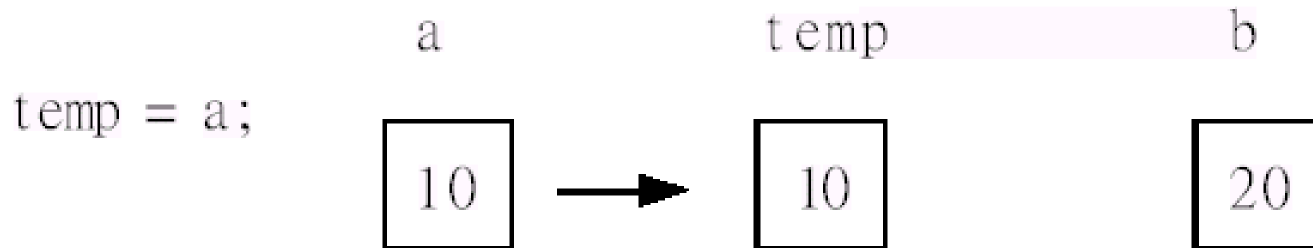
1. 在第 11～13 行中, 先將 a 的變數值指定給 temp 。然後再將 b 的變數值指定給 a。最後, 再將 temp 的變數值指定給 a。
2. 在變數指定過程中, 如果在被指定新變數值的變數中, 已先存有其它變數值。此時, 新變數值會取代舊的變數值。

3 個變數空間中的變數值, 轉換過程如下：

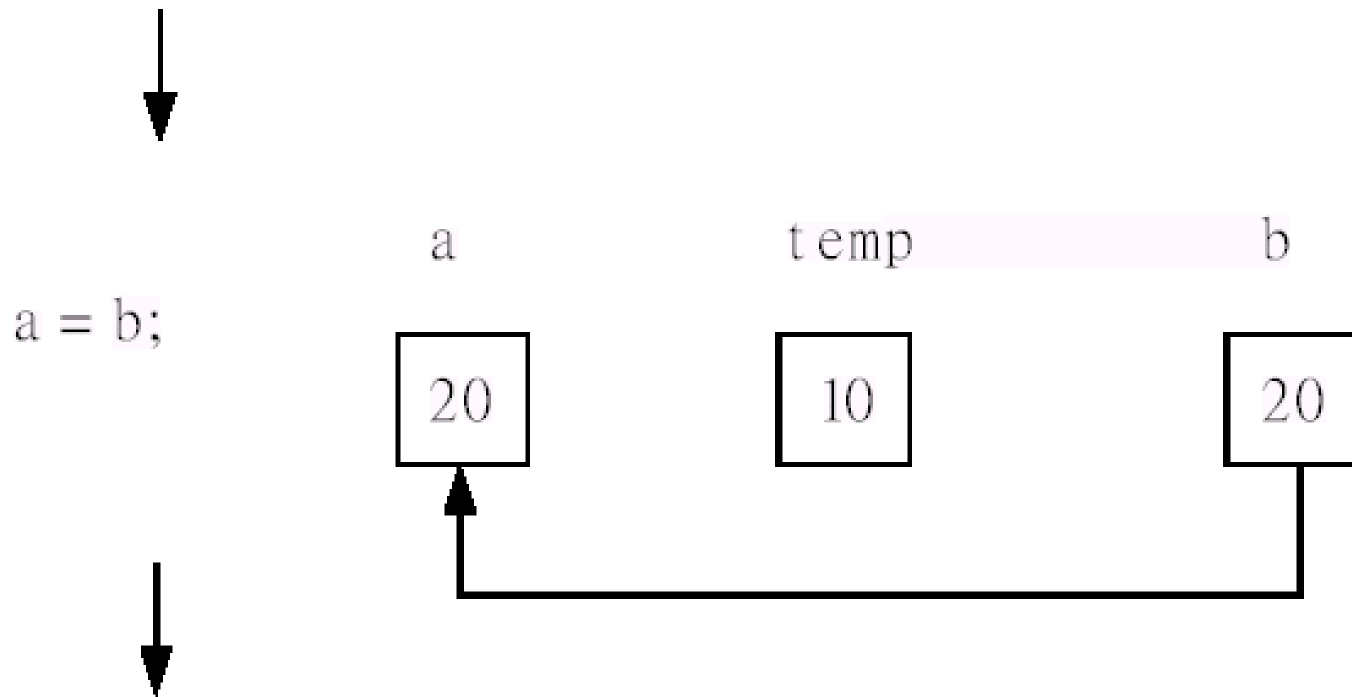
交換兩變數的值







交換兩變數的值



兩數值交換為何一定要有暫存變數呢?

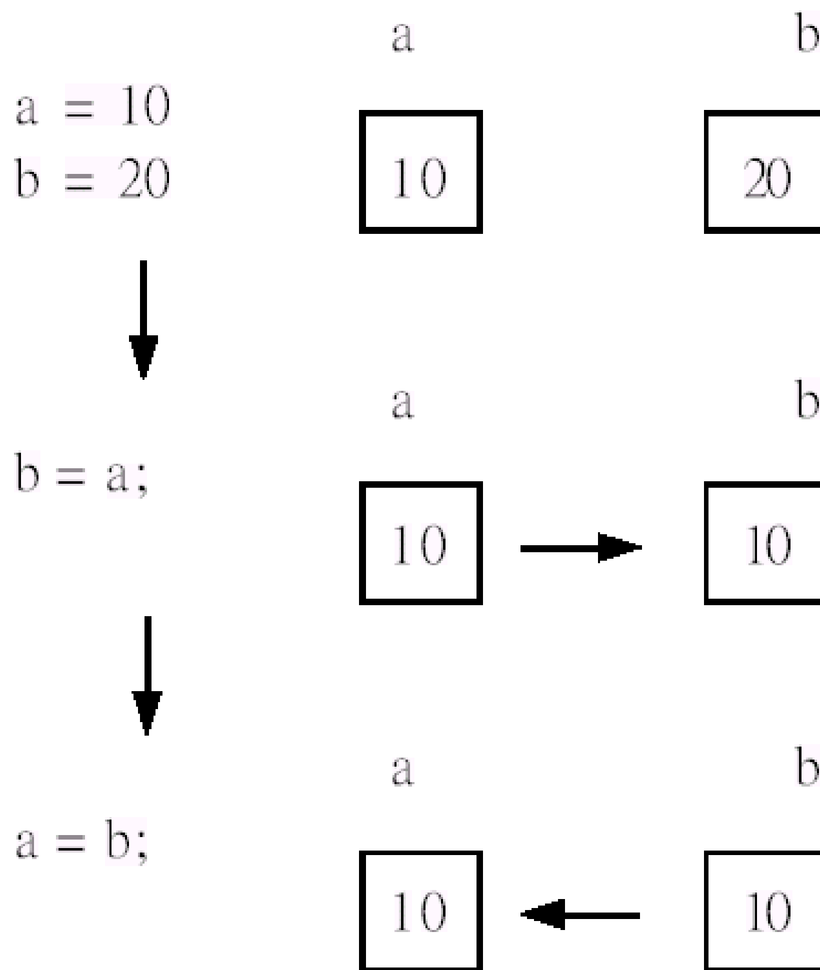
- 兩數值交換時, 如果未使用暫存變數, 會產生甚麼結果：

```
int  a=10,  b=20;


b=a;            //  將變數 a 的值指定給 b
a=b;            //  把變數 b 的值指定給 a
```

兩數值交換為何一定要有暫存變數呢?

a 、 b 最後都變成 10,並沒有如預期般的將兩個數值交換。



大小寫轉換

- 我們已經知道字元是以整數的方式記錄字元 的 ASCII 編碼 (或 Unicode), 其中大寫英文字母的編碼為 65～90, 小寫則 是 97～122, 各字母的大小寫 ASCII 編碼值恰好相差 32。利用這個規律性, 我們就能寫程式作大小寫字母的轉換。(見下頁)
- 第 8 行, 將 upper 變數值加上 32, 也就是前述大小寫字母 ASCII 碼的差值, 所以 lower 就變成小寫的 's' 了。

大小寫轉換

程式 Ch03-19.cpp 大寫字母轉小寫字母

```
01 #include <iostream>
02 using namespace std;
03
04 int main()
05 {
06    char upper = 'S', lower;   // 宣告兩個字元變數
07
08       lower = upper + 32;
09       cout << upper << " 的小寫是 " << lower << endl;
10 }
```

執行結果

```
S 的小寫是 s
```