國立屏東大學 資訊工程學系 C++程式設計入門教材

4. 變數、常數與資料型態

C++語言使用變數（Variable）來保存程式相關的資料內容，例如使用者所輸入的資料或是執行過程中暫時或最終的運算結果。一個變數的內容可於程式執行時視需要加以改變，但其所屬型態不能變更。C++語言也提供常數（Constant）來存放特定的資料內容，但常數的內容一經定義就不可以被改變。我們已經在前一章中，使用變數來進行了幾個簡單的程式開發，例如在BMI計算程式中的weight與height，以及在門號違約金計算程式中的contractDays與subsidy等。本章將進一步為讀者詳細說明變數與常數的意義，以及它們在C++語言中的使用方式，具體的內容包含資料型態、變數與常數的宣告、初始值的設定、以及輸入與輸出等主題。

4.1 變數

變數（Variable）指是在程式執行時，用於儲存特定資料型態（Data Type）的「值（Value）」的記憶體空間。此處的「值」是指資料內容，可以在執行過程中視需要改變其內容，但其所屬的「資料型態」不可改變。在我們正式開始介紹變數前，讓我們先回顧一下，上一章IPO程式設計模型的BMI計算程式與門號違約金計算程式範例，讓我們學到了哪些關於變數的知識與概念：

- 變數其實就是一塊記憶體空間，可以在程式執行期間保存特定「資料型態」的「值（Value）」（也就是「資料內容」）。
- 每個變數都必須有一個用以識別的變數名稱（Variable Name）以便在程式碼中用以存取其值。
- 依據程式設計的需要，變數的值在執行期間可以被改變，但其資料型態不可改變。
- C++語言規定所有變數在初次使用前，都必須進行變數宣告（Variable Declaration）— 包含了其變數名稱以及所屬資料型態的宣告。

本節後續將進一步提供更完整的介紹，包含變數宣告、變數的命名規則與記憶體空間等相關主題。

4.1.1 變數宣告（Variable Declaration）

C++語言規定所有變數在初次使用前，都必須先進行「變數宣告（Variable Declaration）」以便在程式執行時，依據其宣告的內容完成其所需的記憶體空間配置。因此，您必須先學會如何使用C++語言來進行變數的宣告，才能夠在程式中使用變數；能夠在程式中使用變數，也才能夠讓程式完成您想要執行的任務。現在，讓我們來學習如何在C++語言中完成變數的宣告？

變數宣告敘述的語法定義

C++語言使用變數宣告敘述（Variable Declaration Statement）來完成變數宣告，其語法如下：

變數宣告敘述（Variable Declaration Statement）語法定義

資料型態 變數名稱 [=初始值]?[,變數名稱 [=初始值]?]*;

資訊補給站：語法定義符號

在上面的語法定義中，方括號[ ]代表了選擇性（意即可有可無的部份）的語法單元，其後若接續星號 * 則表示該語法單元可使用0次或多次；問號 ? 表示使用0次或1次（也就是可以省略，但至多使用一次）。另外還有此處還未使用到的加號 +，代表的是該語法單元可以使用1次或多次（也就是至少必須使用一次）。後續本書將繼續使用此種表示法做為語法的說明。

在此處的變數宣告敘述語法定義裡，包含了以下三項內容定義：

- 變數名稱（Variable Name）：用以識別的名稱。關於變數的命名規則，將在3-1-2節提供詳細的說明。
- 資料型態（Data Type）：用以定義變數值的範圍，例如被宣告為整數的變數，其值不允許包含小數。關於C++語言所提供的資料型態，除了已經在第2章中介紹過的float與int，其它更完整的介紹請參考本章3-2節。
- 初始值（Initial Value）：用以定義變數值的初始內容。我們將在3-1-3節提供詳細的說明。

注意：變數宣告敘述必須使用分號；結尾

在開始說明如何使用變數宣告敘述前，我們要先提醒讀者所有的「宣告敘述（Statement）」都必須使用分號「;」做為結尾，變數宣告敘述也不例外。請不要忘記在每一行變數宣告敘述後，加上一個分號。

本章後續將依據變數宣告敘述的語法，為讀者詳細說明如何進行變數宣告，但在本小節中將先把選擇性語法單元的部份加以忽略，先行就精簡版的變數宣告敘述加以說明。

精簡變數宣告敘述的語法定義

精簡版的語法定義，略去了原語法中所有可以省略的部份（意即將所有使用方括號包裹的語法單元都已經被省略），是最簡單的變數宣告敘述，只要註明變數的名稱及其資料型態，再加上一個分號「;」就完成了。請參考以下的定義：

精簡版變數宣告敘述語法定義（略去選擇性語法單元）

資料型態 變數名稱;

下列的例子是取自於上一章IPO程式設計模型的BMI計算程式，它們都是符合這種精簡的變數宣告敘述 — 每一行程式敘述各自宣告了一個變數：

```
float weight;  // 體重
float height;  // 身高
```

```
float BMI;   // 身體質量指數
```

這三個敘述皆符合精簡版的變數宣告敘述語法定義，都是以一個「資料型態」開頭，其後再接一個「變數名稱」，最後以分號結尾。

程式設計小技巧：為變數宣告提供註解 另外，我們還在這三個宣告敘述以分號結束後，在後面又加上了以「//」開頭的註解，來補充說明這些變數的意義。關於註解的部份並不屬於變數宣告的語法規範，但卻是筆者建議您可以維持這樣的習慣 — 儘可能在宣告變數時，以註解提供變數的補充說明。如此一來，您所撰寫的程式將比較容易維護。

分隔語法單元的「白色空白（Whitespace）」

還有一點要特別注意的是，在C++語言的語法規則中，我們必須使用一個或多個「空白鍵、Tab鍵」、「Enter鍵」或混合使用它們，來將各個語法單元加以分隔。我們將這種分隔方式稱為「白色空白（Whitespace）」，又將這些用以分隔的鍵稱為「白色空白字元（Whitespace Character）」，例如在上述的例子中，我們使用了一個「空白鍵」將「float」與「weight」加以分隔[1)]。依照這樣的規定，以下的寫法都是正確的宣告：

```
float weight;  // 使用一個空白鍵分隔
float   height;  // 使用一個Tab鍵分隔
float
BMI;   // 使用一個Enter鍵分隔
```

當然，如果您想要使用更多個分隔字元也是可以的，請參考以下的例子：

```
float      weight;  // 使用多個空白鍵分隔
float       height;  // 使用兩個Tab鍵分隔
float



          BMI;   // 混合使用Enter鍵與空白鍵分隔
```

縮排（Indentation）：程式碼排版風格

白色空白不但可以用在語法單元間的分隔，也常常應用在敘述中的第一個語法單元之前。例如以下的例子：

```
     float weight;  // 在float前使用多個空白鍵
```

```
  float height;     // 在float前使用一個Tab鍵
```

相信讀者一定會有一個疑問：「為何要在前面在白色空白呢」？其實這就是所謂的「縮排（Indentation）」又稱為「程式碼排版風格（Coding Style）」透過在每行程式敘述前加入適當的白色空白，將有助於程式碼的閱讀，並可反映出程式碼的結構，是專業程式設計師必備的技能之一[2)]。以下的兩個例子都是在第2章中的BMI計算程式，但使用不同的縮排方法：

全部切齊

```
#include <iostream>
using namespace std;
int main()
{
float weight;
float height;
float BMI;
cout << "請輸入您的體重(公斤): ";
cin >> weight;
cout << "請輸入您的身高(公尺): ";
cin >> height;
BMI = weight / ( height * height
);
cout << "BMI值為" << BMI << endl;
return 0;
}
```

適當地縮排

```
#include <iostream>
using namespace std;
int main()
{
float weight;
float height;
float BMI;

cout << "請輸入您的體重(公斤): ";
cin >> weight;
cout << "請輸入您的身高(公尺): ";
cin >> height;

BMI = weight / ( height * height
);

cout << " BMI值為" << BMI << endl;
return 0;
}
```

其中左側的程式碼完全沒有在每行敘述前加入白色空白，右側的部份則適當地使用白色空白將程式碼對齊，不但在閱讀上比較清爽，同時在main()函式內的程式碼透過內縮對齊也反映了它們是屬於函式內部的程式結構。

其實C++語言只規定敘述要以分號做為結尾，但並沒有規定要寫在獨立的一行，把多個敘述寫在同一行，或是將一個敘述分寫在多行裡都是可以的。雖然在語法上允許我們這麼做，但是筆者並不鼓勵您將多個敘述寫在同一行，或是將一個敘述分散到多行（除非一行寫不下的時候），因為這樣一來會將程式碼的結構破壞殆盡，例如以下的例子：

```
#include <iostream>
using namespace std;
int main(){ float weight; float height; float BMI;
cout << "請輸入您的體重(公斤): "; cin >> weight; cout <<
"請輸入您的身高(公尺): ";cin >> height; BMI = weight
/ ( height * height ); cout << "您的BMI值為"<<
BMI << endl;return 0;
}
```

您還認得出來這是哪個程式嗎？沒錯，它仍然是我們熟悉的那個BMI計算程式，但是看起來實在不怎麼友善！事實上，在我們進行程式碼的編譯時，編譯器會先將程式碼中的白色空白移除，然後才進行編譯的動作。因此上面這個排版很不自然的程式仍然可以正常的執行！只是您真的能接受這種雜亂無章的編排方式嗎？這種編排方式，不但其他人看不懂您所寫的程式碼，可能連您自己都看不太懂！

其實，程式碼縮排方式並沒有硬性的規定，但是儘量將每個敘述寫在獨立的一行，並適當地使用白色空白

來將程式碼對齊，是比較好的程式撰寫習慣。

使用一行敘述宣告多個相同型態的變數

除了前述的精簡版變數宣告敘述外，依據原語法定義，多個相同型態的變數還可以寫在同一個宣告敘述中，請參考以下的語法定義：

精簡版變數宣告敘述語法定義（僅略去部份的選擇性語法單元）

資料型態 變數名稱 [,變數名稱]*;

上述的語法定義僅將部份的選擇性語法單元加以省略（也就是僅省略了原語法中關於初始值的部份）。與精簡版的變數宣告敘述相同，我們仍然使用「資料型態」作為開頭，並在其後再接一個「變數名稱」；但不同的是，您可以在變數名稱的後面，視需要再宣告0個或多個變數名稱。具體來說，在語法單元 [,變數名稱]* 中的 * 表示 [,變數名稱] 可以使用0次或多次。用比較白話的說法，此語法定義也可以解讀為「可以使用單一的變數宣告敘述，同時宣告多個變數，不過必須在連續的兩個變數名稱中，使用逗號 , 加以分隔」。以下的例子同時宣告了三個float型態的變數，分別是weight、height與BMI。

```
float weight, height, BMI;
```

以語法來看，其開頭處的float就是語法定義中的「資料型態」部份，至於weight就是後面所接的「變數名稱」，但我們在其後再重覆使用了兩次 [,變數名稱]* 語法規則，也就是「,height」與「,BMI」最後以分號做為此宣告敘述的結尾。

4.1.2 變數命名規則

變數名稱（VariableName）又被稱為識別字（Identifier）是用以區分在程式中不同的變數。為了識別起見，每一個變數必須擁有獨一無二的變數名稱，其命名規則如下：

1. 只能使用英文大小寫字母、數字與底線（Underscore） _
2. 不能使用數字開頭；
3. 不能與C++語言的關鍵字（keyword）相同。

前兩項規則相當容易理解，至於第三項中所謂的關鍵字（Keyword）是在程式語言中具有（事先賦予的）特定意義的文字串組合。由於每個關鍵字都具有事先定義好的意義與用途，因此我們在宣告變數時，變數名稱不能與任何一個關鍵字相同。

C++的關鍵字

以ANSI/ISO C++（或稱為C++89） 為例，共有以下74個關鍵字：

and	and_eq	asm	auto	bitand	bitor	bool	break	case	catch
char	class	compl	const	const-cast	continue	default	delete	do	double
dynamic_cast	else	enum	explicit	export	extern	false	float	for	friend
goto	if	inline	int	long	mutable	namespace	new	not	not_eq

operator	or	or_eq	private	protected	public	register	reinterpret-cast	return	short
signed	sizeof	static	static_cast	struct	switch	template	this	throw	true
try	typedef	typeid	typename	union	unsigned	using	virtual	void	volatile
wchar_t	while	xor	xor_eq						

Tab. 1: ANSI/ISO C++的關鍵字列表

正確與錯誤的變數命名範例

依據前述變數命名的三項規則，以下的變數名稱使用了不在規則中允許的字元或符號（僅能使用英文大小寫字母、數字與底線），當然都是錯誤的命名：

```
int cert#, someone@taipei;  //使用了不允許的特殊字元
int average-score;          //使用了不允許的連字號(也就是減號)
int miles/per-second;       //使用了不允許的斜線字元與連字號
float ratio!, question?       //使用了不允許的驚嘆號與問號
```

至於數字在使用上也必須注意（不可以用於變數名稱的開頭），以下是錯誤的例子：

```
float 386PC, 486PC;        //使用了數字開頭
int 1stScore, 2ndScore;     //使用了數字開頭
```

除此之外，還要注意不要使用C++的關鍵字做為變數名稱，例如以下的例子都是錯誤的命名：

```
int namespace;        //使用了C++的關鍵字namespace
float auto;            //使用了C++的關鍵字auto
```

看完了上述的錯誤命名的例子之後，讓我們看看一些正確的變數命名範例。以下的變數名稱僅使用了大小寫英文字母、數字與底線所組成，同時也都沒有以數字開頭，或是使用了C++的關鍵字，所以當然都是正確的：

```
int A, B, C;        // 使用大寫英文字母所組成的變數名稱
int x, y, z;         // 使用小寫英文字母所組成的變數名稱
int Wig, xYz;      // 混合使用大小寫英文字母所組成的變數名稱
int H224, i386;     // 混合使用英文字母與數字所組成的變數名稱
float w_1, w_2;    // 混合英文、數字與底線所組成的變數名稱
```

雖然依據變數的命名規則，這些變數名稱都是正確的，但都不是很好的命名選擇，因為這些變數名稱並不具備任何意義，無法從中得到關於變數的用途或意義的提示。雖然變數的命名必須要滿足其命名規則（因為一定要滿足，不然不能通過編譯），但是更重要是為變數賦與「有意義」的名稱；換句話說，變數名稱的做用不單是用以識別不同的變數，更重要的是要能夠傳達該變數在程式中的用途或意義。為了做到這一點，讀者們應該適當地使用大小寫的英文字母、數字與底線（underscore）為變數命名，以增進程式碼的可

讀性。

可讀性（Readability）

所謂的可讀性（Readability）係指程式碼容易被理解的程度，可讀性高的程式碼，閱讀起來就像行雲流水，頃刻之間就可以完全瞭解程式的內容；相反地，可讀性低的程式碼，不但讓人難以理解，有時候甚至是原始的程式創作者自己也無法理解程式的內容。為了增進程式的可讀性，我們建議讀者視變數在程式中的用途，選擇具有意義的英文詞彙為變數命名。為了讓程式碼的可讀性提升，有時我們甚至會使用一個以上的英文單字為變數命名，此時可以適當地調整大小寫或加上底線，例如下面是正確且具有意義的變數名稱：

```
float weight, height, BMI;  // 分別代表體重、身高與BMI值的變數

// 我們可以使用底線連接多個英文單字，以形成更具意義的變數名稱
int student_id;       // 使用底線連接student與id代表學生的學號
float student_score;  // 使用底線連接student與score代表學生的成績

// 我們也可以適當的使用大小寫英文字母，取代使用底線來連接多個單字
int studentID;       // 使用大小寫來區分連接起來的student與ID
float studentScore;  // 使用大小寫來區分連接起來的student與Score
```

除此之外，也可以適當地使用數字的諧音，來精簡表示特定的英文或其它含義。請參考下面的例子：

```
// 使用數字4來代替英文的for
float interest4loan;   // 此變數名稱表示interest for loan（意即貸款的利率
float interest4saving; // 此變數名稱表示interest for saving（意即存款的利率
float discount4group; // 此變數名稱表示discount 4 group（意即團體的折扣
float discount4kids;  // 此變數名稱表示discount 4 kids（意即孩童的折扣

// 使用數字2來代替英文的to
int time2destination;  // 此變數名稱表示time to destination（
       // 意即到達目的地的時間
int amount2pay;     // 此變數名稱表示amount to pay（意即應支付的金額
```

大小寫敏感（Case Sensitive）

除了前述的三項變數命名的規則之外，還必須特別注意的是（C++語言是對大小寫敏感（Case Sensitive））的程式語言，意即大小寫會被視為不同的字元，因此以下的程式碼所宣告的8個變數名稱都是正確的，而且會被C++語言視為是「不相同」的變數：

```
int day, Day, dAy, daY, DAy, DaY, dAY, DAY;
```

儘管您應該不會像這樣宣告一些非常容易混淆的變數名稱，但仍有可能在程式中發生一些錯誤，請參考以

下的例子：

```
float Weight; // 宣告一個float型態的變數Weight

cin >> weight;  // 使用cin取得使用者輸入時，誤將變數名稱寫為w eight
```

在上面的程式碼片段中，出現了「寫錯變數名稱」的錯誤！不小心將Weight寫成了weight與！由於C++語言對於大小寫敏感的關係，在程式碼中的Weight與「weight將會被視為是兩個不同的變數。對於編譯器而言Weight是正確的變數名稱（有事先使用變數宣告敘述宣告），但是用以儲存使用者輸入的Weight變數，因為沒有事先的宣告，所以是不正確的使用（無法通過編譯）。像這樣的錯誤其實很容易發生，例如在程式中不小心將x寫成X或是把v寫成V等。

要避免這樣的問題，除了小心謹慎地使用變數名稱外，最好的方式是維持一定的變數命名習慣。舉例來說，如果您固定全部使用小寫字母為變數命名，那麼在使用變數時您自然也會全部使用小寫字母，那麼將weight誤寫成Weight的機會就會大幅地降地。目前業界已經有一些通用的變數命名方式，我們將其稱之為「命名慣例Naming Convention依據這些慣例來為變數命名除了有助於提升程式的可讀性外，也能夠減少犯錯機會。

命名慣例Naming Convention

目前有一些用於變數命名的規則可參考，例如著名的「駝峰式命名法CamelCase[2]與「匈牙利命名法Hungarian notation[3]等方法。駝峰式命名法是當前主流的變數命名方法，其命名方法是直接使用英文為變數命名，其名稱即為變數的意義或用途，因此具備良好的可讀性。

採用駝峰式命名法命名方法時，直接使用英文為變數命名；若使用到兩個或兩個以上的英文單字時，每個英文單字除首字母外一律以小寫表示，且單字與單字間直接連接（不須空白），但從第二個單字開始，每個單字的首字母必須使用大寫。至於第一個單字的首字母，則依其使用大寫或小寫字母，可將駝峰式命名法再細分為「大寫式駝峰式命名法UpperCamelCase與「小寫式駝峰式命名法lowerCamelCase兩類。例如以下的幾個名稱皆屬於大寫式駝峰式命名法：

```
Number
UserInputNumber
MaxNumber
StudentIdentifier
FulltimeStudent
BestScore
CourseTime
CamelCase
UpperCamelCase
```

至於小寫式駝峰式命名法，請參考下列的例子：

```
amy
userName
```

```
happyStory
setData
getUserInput
lowerCamelCase
```

目前大部份的C++語言程式設計師，都是採用小寫式駝峰式命名法為變數命名（意即使用有意義的英文單字來為變數命名，原則上所有單字皆使用小寫英文字母組成，但從第二個單字開始，每個單字的第一個字母必須使用大寫），本書後續的程式範例，也將繼續使用小寫式駝峰式命名法為變數命名。

標準識別字（Standard Identifier）

本節最後為讀者介紹標準識別字（Standard Identifier），雖然它們完全符合變數命名的各項規定，但仍不建議讀者使用。所謂的標準識別字是一組在特定標頭檔（Header Files）或命名空間（Namespace）中預先定義好的常數、變數或函式名稱，例如我們已經在程式中使用過的cin（cout與endl等，事實上它們是定義在std這個Namespace中的名稱，您可以試著宣告以下的變數：

int cin=5; int cout=10; int endl=20;

相信我，這樣的程式碼是正確的！但如此一來，以後在您的程式碼中的cin（cout與endl（倒底是代表整數值或是其原本定義的輸入、輸出與換行的功能？像這樣的例子就充份說明了什麼是標準識別字 — 雖然不是C++語言的關鍵字，但仍不鼓勵您使用這些名稱來做為您所宣告的變數名稱，因為可能會和其原本的意義不同，對程式碼的意義帶來不必要的錯誤解讀。關於標準識別字還有一些其它的例子，包含NULL（EOF（min（max（open（close（sin（cos（pow與log等，請儘量別使用這些名稱為您的變數命名。

4.1.3 變數初始值宣告

變數就是在程式執行時，用以暫時儲存特定型態的值的一塊記憶體空間；在程式執行的過程中，一個變數所儲存的值可以被改變，這也正是「變數（Variable）」一詞的由來 — vary（變化）加上able（具有…能力）。我們在宣告一個變數時，除了要指定其變數名稱以及型態外，還可以給定一個「初始值（Initial Value）」— 變數一開始所儲存的值。請先回顧以下的變數宣告敘述語法：

變數宣告敘述（Variable Declaration Statement）語法定義

資料型態 變數名稱 [=初始值]?[,變數名稱 [=初始值]?]*;

在上述的語法規定中，每個變數除了宣告其所屬的型態以及其變數名稱外，還可以選擇性地接上「=初始值」；不論是在一個變數宣告敘述中的第一個變數宣告，或是後續其它的變數宣告都可以擁有這個選項 — 給定初始值，請參考以下的例子：

```
float weight=65.5, height=1.72;     // 宣告兩個變數並且都給予初始數值。
int contractDays=100;              // 宣告一個變數並給予初始值。
int productID, price=0, amount=12; // 宣告三個變數並給予其中兩個變數初始值
```

4.1.4 變數記憶體配置

在程式碼中使用變數宣告敘述所宣告的變數，在執行時會依所宣告的資料型態在記憶體中配置適當的位置。截至目前為止，本書已經介紹了兩種資料型態，分別是浮點數float與整數int，而它們都是使用連續的4個位元組（Byte）的記憶體空間。關於資料型態的更多細節，請參考本章3-2節。請考慮以下的變數宣告：

```
float weight, height, BMI;
```

此行變數宣告敘述，在程式執行時，會依據其宣告內容在記憶體中配置三個浮點數型態的計憶體空間，並將其分別命名為weight、height與BMI，後續在程式中使用這些變數時（不論是讀取或改變其值），就是對這些記憶體位置內的值進行操作。figure 1顯示了這三個變數可能的記憶體配置圖：

Fig. 1: 變數記憶體配置圖。

在figure 1中的每個矩形就代表了分配給這些變數的記憶體空間，我們將變數名稱標示在其所分配到的記憶體空間左側，並在其上方標示了該空間的起始位址[3]；由於C++語言為float（浮點數）型態的變數，所配置的是連續的4個位元組的記憶體空間[4]，因此以figure 1左上角的weight變數為例，其所配置的記憶體空間是從0x7ffff69a3232開始（標示於上方）至0x7ffff69a3235的連續4個位元組。由於變數所配置到的記憶體位置，必須等到程式執行時才能得知，如果我們想要存取變數weight的值時，並不是在程式中指定要存取0x7ffff69a3232至0x7ffff69a3235這連續4個位元組的記憶體空間（因為在撰寫程式時，我們還無法得知其記憶體位址），而是以該變數的名稱，也就是weight來進行存取。可是程式執行時，又要如何才能得知weight變數所在的記憶體位址呢？關於此點，其實是透過「符號表（Symbol Table）」來查詢的。

符號表（Symbol Table）

符號表是在程式編譯或執行時所建立的一個表格[5]，是為了管理在程式中用以識別的變數名稱與其所分配到的記憶體空間，其內容包含有變數的名稱、型態以及其所分配到的記憶體空間位址等資訊。以我們前述的weight、height與BMI這3個變數為例，在執行時其符號表可能會有如table 2的內容[6]：

符號(Symbol)	資料型態(Data Type)	記憶體位址(Memory Address)
weight	float	0x7ffff69a3232
height	float	0x7ffff69a3255
BMI	float	0x7ffff69a32f0

Tab. 2: 程式執行時的符號表內容

其中在記憶體位址的部份，符號表僅保存其起始位址，至於其結束的位址可以由同樣保存在符號表中的資料型態來決定。例如weight在符號表中記載了其型態為float，因此其所配置到的空間就是從符號表中所記

載的0x7ffff69a3232開始，一直到0x7ffff69a3235的連續4個位元組（由於float資料型態佔用了連續4個位元組的記憶體空間）。請注意，此處所使用的記憶體位址僅供參考，其真實的位址必須在程式執行時才能得知。

Address-Of運算子

如果讀者對於某個變數在執行時，其所配置到的記憶體位址有興趣，則可以使用「位址（Address-Of）運算子，&符號」來取得變數的記憶體位址（更具體來說，是變數所配置到的起始位址）。只要在變數名稱前加上位址運算子，就可以取得其所配置的記憶體位址。請參考Example 1的addressOf.cpp程式，其中宣告了兩個變數，並使用&（Address-Of運算子）取得它們所配置到的記憶體位址：

```
#include <iostream>
using namespace std;

int main()
{
   float weight, height; // 此處使用了一行敘述來宣告兩個變數

   cout << "變數weight所配置到的記憶體位址為: " << &weight << endl;
   cout << "變數height所配置到的記憶體位址為: " << &height << endl;
   return 0;
}
```

Example 1的addressOf.cpp之執行結果如下（注意，此結果僅供參考，實際輸出的記憶體位址將會有所差異）：

```
變數weight所配置到的記憶體位址為: 0x7fff5104cafc
變數height所配置到的記憶體位址為: 0x7fff5104caf8
```

4.2 常數

除了變數之外，在C++語言的程式碼中，還可以宣告常數（Constant）。常數就如同變數一樣，都擁有名稱、型態與內容值，不過一旦給定初始值後，就不允許變更其數值內容。

4.2.1 常數宣告

C++語言的常數宣告（Constant Declaration）的語法如下：

常數宣告敘述（Constant Declaration Statement）語法定義

const 資料型態 常數名稱 = 數值 [, 常數名稱 = 數值]*;

從上面的語法可以得知，其實constant declaration（常數宣告）的語法與變數宣告非常相似，不過必須在最前面加上const這個關鍵字，並且所有常數的宣告都必須給定數值（Value）[7]。在常數名稱方面，其命名規則與變數的命名規則一致，請自行參考3-1-2的說明。接下來，請參考下面的程式碼片段：

```
const int a=100;
const int b=3,c=5;
...
a=200;  // 改變一個常數的值
...
```

上面的程式碼正確地宣告了三個整數常數a、b與c，但在後續的程式碼中卻又改變了其中一個常數的數值！這樣會導致編譯時的錯誤，您會得到「error: read-only variable is not assignable（錯誤：唯讀的變數不可以指定數值）」的錯誤訊息，因為一個常數一但被宣告後，其值是不允許被改變的。

4.2.2 常數定義

除了使用前述的常數宣告方法外，我們還可以使用「#define」這個前置處理器指令（Preprocessor Directive）[8]來定義常數，其語法如下：

常數定義（Constant Definition）語法

#define 常數名稱 數值

與多個常數可以在一個宣告中同時宣告不同，常數定義一次僅能定義一個常數，且在定義時不需要使用等號（=），也不需要在結尾處的分號（；）。依照上面的語法，我們可以定義一個常數 PI，其值為3.1415926：

```
#define  PI  3.1415926
```

或是定義一個名為size的常數，其值為10：

```
#define  size  10
```

其實常數定義並不是幫我們產生一個常數，而是幫我們以代換的方式，將程式碼中所出現的特定文字串組合改以指定的內容代替。因為在C++語言中，所有以井字號（#）開頭的指令，都是在編譯時由編譯器先啟動一個前置處理器（Preprocessor）來負責加以處理的。包含了我們已經使用過的「#include<標頭檔>」指令，其實是在編譯前，先將指定的標頭檔案內容載入到程式碼中。至於此處所介紹的「#define」也是一個前置處理器指令（Preprocessor Directive），同樣是由編譯器內的前置處理器負責處理，以下面的程式碼為例：

```
#define  PI   3.1415926

int main()
{
   int radius=5;
   float area;
   area = PI * radius * radius;
   …
}
```

在編譯時，編譯器會先啟動前置處理器，依據其中第一行所定義的「#define PI 3.1425926」掃描尋找所有在程式碼中出現PI的地方，並將其改以3.1415926進行代換；完成這個代換後，編譯器才會展開真正的編譯工作。因此，上述的程式碼經過前置處理器處理後，會將以下的程式內容送交給編譯器進行編譯的工作：

```
int main()
{
   int radius=5;
   float area;
   area = 3.1415926 * radius * radius;
   …
}
```

關於前置處理指令更詳細的說明，可參閱本書第X章。

4.3 資料型態

一個資料型態包含了一組特定的資料內容的集合以及一組可以對其進行的操作。例如在數學領域裡，「整數」就是一種資料型態，它是一個包含了在序列{…, -3, -2, -1, 0, 1, 2, 3, …}中的所有正整數、零以及負整數的無窮集合，我們可以對整數的資料內容進行包含加、減、乘、除在內的相關操作。不過要注意的是，因為受限於有限的記憶體空間，在電腦系統裡的資料型態只能是有限的集合。C++語言提供多種資料型態，包含基本內建資料型態（Primitive Built-In Data Type）與使用者自定資料型態（User-Defined Data Type）兩類。本章僅就基本內建資料型態進行說明，使用者自定資料型態請參閱本書第X、Y、X與Z章。

C++語言所提供的基本內建型態，可分為以下幾種：

- 整數型態（Integer）
- 浮點數型態（Floating Point）
- 字元型態（Character）
- 布林型態（Boolean）
- 無值型態（Valueless or Void）

我們可以視需求在程式中宣告這些型態的變數來加以使用，本節後續將針對這些型態逐一加以介紹。

4.3.1 整數型態

顧名思義，整數型態（Integer type）就是用以表示整數的資料型態。在C++語言中的整數型態，是以Integer的前三個字母（int）表示。事實上，我們已經在本書中使用過這個int整數型態，在這個小節中我們將提供更完整的說明。請先回顧我們在第2章所介紹過的Example 5（請參考2-X頁）的compensaton-1.cpp程式，以下的程式碼宣告了一些整數型態的變數：

```
int contractDays;          // 合約總日數
int contractRemainingDays; // 合約剩餘日數
int monthlyFeeDiscount;    // 每月月租費優惠金額
int subsidy;               // 手機補貼款
int compensation;          // 違約金
```

這些使用int整數型態所宣告的變數，在程式執行的過程中會佔用連續的4個位元組（Byte）的記憶體空間，也就是32個位元（Bit） ，雖然int變數的值（Value）在程式執行的過程中可以有所變化，但其值必須符合int整數型態的範圍（Range）。

對於一個資料型態而言，其所謂的範圍係指一個該型態變數之數值的上限與下限，換句話說就是該型態的變數所能表達的最大值與最小值，可由該型態所佔用的記憶體大小加以計算。以4個位元組的int整數為例，其範圍是由在記憶體中連續的32個位元的排列組合所決定的，其中最左邊的位元代表正負號，稱為符號位元（Sign Bit），以0代表非負（Non-Negative）的整數，也就是正整數或0；1則代表負整數（Negative）。此外，讀者必須特別注意的是，不論是正整數或負整數，其值一律以2補數（2’s Complement）表示。依據2補數的運算方式，32位元的int整數其可表達的最大正整數在記憶體中的內容為：

0111 1111 1111 1111 1111 1111 1111 1111

其值為+2,147,483,647，至於最小的整數則為：

1000 0000 0000 0000 0000 0000 0000 0000

其值-2,147,483,648。因此一個int整數型態的變數，其值是介於其最大值2,147,483,647與最小值-2,147,483,648之間。

---

[1] 試想如果不加以分隔，成float weight;，不就變成了，floatweight;，這樣怎麼知道你是在宣告一個型態為，float，的「weight，變數呢？還是在寫一行，floatweight;，敘述呢？當然，，floatweight;，敘述是錯誤的寫法。

[2] 專業的軟體開發團隊也會規範程式碼的縮排方式，以便團隊開發出的程式碼能具有一致性的風格。

[3] 此處使用的是以0x開頭的16進制的數字，同時出現在圖3-1中的記憶體位址僅供參考，其實際的位址於程式執行時才能得知。

[4] 在本書上一章IPO程式設計模型中，我們已經介紹過兩種不同的資料型態，分別是float（浮點數）與int（整數），其中float是使用4個位元組的記憶體空間，至於其它資料型態與其所佔之記憶體空間等細節，將於本章後續第3-3節為你說明。

[5] 依所使用的程式語言及其編譯技術而定，有些時候是在編譯階段產生符號表以將變數名稱轉譯為其對應的記憶體位址，有時則是將符號表嵌入在目的檔或可執行檔中，在執行階段才進行位址的轉譯。

[6] 此處所列示的符號表內容僅供參考，且其內容經過大幅度地簡化，其中所顯示的記憶體位址只是假設的，真正執行時其內容與此表格並不相同。

[7] 由於變數的內容可以在程式執行過程中被改變，所以一開始所給定的數值被稱為「初始值」，以便與後續變動後的不同數值做區隔；但常數在宣告時所設定的數值內容並不允許被改變，因此其數值內容並不存在著「初始」與「後續」的差異，我們將其稱為「數值」。

[8] 前置處理器指令是在程式被編譯前，先執行的指令操作，詳細說明可參考本書第X章。