計算機程式講義

10 小時學會 C 語言

☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆
☆★☆★☆　不要完全相信書上及老師所講的內容，只有自己實作才是真的。　☆★☆★☆
☆★☆★☆　　李棟樑 Sep.22,1997　☆★☆★☆
☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆☆★☆★☆

參考書目：10 小時學會 C 語言[修訂版]　施保旭著　儒林出版社　1997 年 6 月初版 3 版

內容 page

■ 第一章　C 語言簡介與 Turbo C 的使用

◎ C 語言的優點：
○ 效率高：C 語言的編譯器會產生最小的程式碼。
○ 可攜性/移植性高：經過些許的修改，可以在不同的平台使用。
○ 功能強而有彈性。
○ 需要記憶的東西很少，易於寫作。

◎ Turbo C 的安裝：已安裝在學校主機。
Turbo C 的環境設定：Turbo C 安裝的目錄必須設定在 PATH 的系統變數。
如： PATH=C:\TC;C:\DOS;..... 如此 TC 才能正常工作。

◎ Turbo C 的使用

只要設定好 PATH 變數，在 DOS 提示號輸入 TC ，就可以執行 Turbo C 的整合環境了。TC 將編輯(Edit)、編譯(Compile)、連結(Link)、除錯(Debug)、檔案管理(File)、...等等的功能整合在一起，所以我們稱之爲整合環境。
最好先用 CD 的指令，變更工作目錄到你要寫 C 的目錄，再執行 TC，這樣所產生的檔案，就會裡這個目錄裡面，便於備份與管理。

◎ 移動游標
方向鍵 ←↑↓→ 可以用來移動游標。

◎ 刪除文字
將游標移到你要刪除的文字上，再按下 Del 鍵即可。
將游標移到要刪除文字的右邊，再按下 BS 退位鍵也可以。

◎ 載入文字檔(C 語言原始碼檔)

按下功能鍵 F3 或 按 F10 選 File → Load 就會出現一個詢問視窗要求輸入檔名：

```
┌──────── Load File Name ────────┐
│*.C                             │
└────────────────────────────────┘
```

其中的檔名可以使用萬用字元 * 或 ? ，或直接指定你要的檔名。
若是使用萬用字元，TC 會再秀出一個視窗讓你選擇所要的檔案，
你可以用方向鍵移動反白光棒，按 Enter 鍵則是選擇反白的檔案。

◎ 儲存編輯的文字檔

按下功能鍵 F2 或 按 F10 選 File → Save 就會儲存目前編輯檔案。
若你想另外取一個檔名，並加以存檔，就必須 按 F10 選 File → Write to
就會出現一個詢問視窗要求輸入檔名：

```
┌──────── New Name ──────────────┐
│_                               │
└────────────────────────────────┘
```

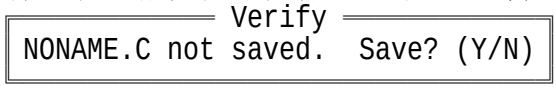
輸入新的檔名，按下 Enter 即可。

◎ 編譯並執行目前的所編輯的程式
Turbo C 是一種編譯語言系統，你所寫的程式，經過 TC 的編譯(pass 1)及連結(pass 2)後，產生可執行檔(.exe)，才能在 PC 上執行。
按下 Ctrl + F9 或 按 F10 選 Run → Run ，TC 會編譯目前所編輯的程式，如果沒有錯誤發生，TC 會立即執行所編輯的程式。
TC 在執行完程式後，會立刻切換回到 TC 的整合環境，如果你還想看剛才程式執行的結果，可以按下 Alt + F5 或 按 F10 選 Run → User screen ，就會切換到執行畫面，再按下任何一鍵，就會回到 TC 的整合環境。

◎ 結束 Turbo C

按下 Alt + X 或 按 F10 選 File → Quit 便可結束 Turbo C。

若你還有已編修尚未儲存的檔案，TC 會問你要不要存。



要存就按 Y ，不想存就按 N 。

■ 第二章 C 程式的結構

◎ C 程式的結構：

hello.c ←— 範例檔名

```
1  #include <stdio.h>
2  main()
3  {
4     printf("Hello!");
5  }
```

←— 範例原始碼

```
Hello!
```

←— 範例執行結果

第一列： `#include <stdio.h>`

是用來定義一些函式的原型(prototype)、資料結構(struct)或是常數(constant)。

C 在使用變數之前，該變數都要先行宣告(declear)才可使用，而使用函式也是一樣，必須先宣告它的原型才可以。宣告函式的原型是爲了讓 C 能在編輯時作資料的型別檢查，以減少錯誤的發生。 內建的函式原型定義都放在 INCLUDE*.H 中，用 #include <stdio.h> 就會將 INCLUDE\stdio.h 這個檔引含。 stdio.h 由檔名可知道是跟標準輸出入(standard I/O)有關，檔內定義了檔案輸出入、螢幕輸出、鍵盤輸入等相關函式的原型、資料結構及常數。 本例中的 printf() 就屬於標準輸出的函式。 引含 .h 檔並不會讓你的執行檔變大或是變慢，而且還能讓編輯器作正確的型別檢查，所以要養成寫 #include 的習慣。 ☆雖然在某些狀況下，不加 #include <> 所產生的執行檔，一樣可以正常的執行。

第二列： `main()`

main() 就是主程式。程式在執行時，就是由這個函式開始執行。 在 C 語言中，內定的型別是 int ，所以原來的 main() 相當於是 int main(int)

★ 在這裡正確的寫法應該是 void main(void), 因爲在這個簡單的程式中，沒有回傳值，也沒有輸入參數。

☆ 回傳值，是指一個函式在執行後傳回的數值。

☆ 輸入參數，是指一個函式可由參數決定執行的結果，這個部分在第八章中有詳細的說明。

第三列： {
第五列： }

在第三列及第五列之間，屬於 main() 函式的程式碼。

{ 表示程式由此開始， } 表示程式到此結束。

第四列： `printf("Hello!");`

是本程式要求系統做動作的指令，稱之爲「敘述」。在 C 語言中，每一敘述都以分號(;)做爲結束。 在這個程式中，利用縮排的方式，使程式的層次分明，增加可讀性。

在 C 語言中，調位字元（ 如：空白(space)、定位(tab)及換列字元 ）在編譯時都會被忽略，所以可適時加入調位字元，使程式好看一點。 要注意的是，別把一個完整的個體拆開，如：main、printf 等，這些字本身是一個完整的個體，不可予以拆開。而在各個個體之間，可以任意加入調位字元。

☆ C 語言中的英文字母是有分大小寫的，printf() 與 PrintF() 不同， 內建的函式大多是小寫的，你自己寫的函式，則可以用大寫來做區隔。

◎ printf 的功用

printf() 的功用是在螢幕上輸出資料。在 TC 中，在編輯區內輸入 printf ，再將游標移到 printf 這個字上，按下 Ctrl + F1 就會看到以下 Help 畫面：

```
──────────────── Help ────────────────

printf: formatted output to stdout

int printf(const char *format, ...);

Prototype in stdio.h

Print formats a variable number of arguments
according to the format, and sends the output
to stdout. Returns the number of bytes output.
In the event of error, it returns EOF.

See also     ecvt    fprintf   putc
             puts    scanf     vprintf
```

← 這是 TC 的 Help 視窗

← printf 是將格式化的資料輸出到 stdout

← printf 的語法

← 要用 printf 應該 #include 的檔

← 使用說明：不同的格式須要不同的參數，這些資料會送到 stdout。傳回值是輸出的 byte 數，若發生錯誤則傳回 EOF

← 可參考相關指令：ecvt,fprintf,putc, puts,scanf,vprintf

在用 TC 的整合環境中，只要將游標移到想進一步了解的指令或內建的函式上，按下 Ctrl + F1 就可以叫出 TC 的 Help 說明視窗，得到該指令或函式的相關說明。

printf 的語法： int printf(const char *format, ...);

其中 const char *format 指的是一個格式化的字串。 const 是常數的意思，在此表示 format 這個字串指標傳入 printf 函式後，它的值不會被改變。

... 指的是不定參數，參數的數目取決於 format 字串的內容，這些參數，通常是一些你要秀出來的變數。 簡單來說： printf("輸出格式(含控制字串)" , 要印出的資料);

在 C 語言中是用雙引號(")來引含字串，也就是在雙引號內的資料，是一個字串。本章只介紹 %d 這個控制字串，其他的控制字串在第四章會講到。 %d 表示以整數 10 進位的方式秀出資料。在輸出格式(含控制字串) 內有幾個%d ，在要印出的資料內就要有幾個整數來對應。

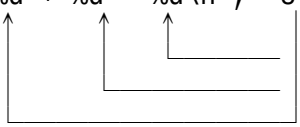
arith.c

```
1  #include <stdio.h>
2  void main(void)
3  {
4     printf("%d + %d = %d\n", 8 , 2 , 8+2 );
5     printf("%d - %d = %d\n", 8 , 2 , 8-2 );
6     printf("%d * %d = %d\n", 8 , 2 , 8*2 );
7     printf("%d / %d = %d\n", 8 , 2 , 8/2 );
8  }
```

```
8 + 2 = 10
8 - 2 = 6
8 * 2 = 16
8 / 2 = 4
```

我們以第四列的敘述作說明：

```
printf("%d + %d = %d\n",  8  ,  2  ,  8+2 );
```



在 輸出格式(含控制字串) 內有 3 個 %d ，所以在 要印出的資料 的部分有 8 , 2 , 及 8+2 三個整數對應，所以輸出來的結果就是

```
8 + 2 = 10
```

在 輸出格式(含控制字串) 的最後有 \n 符號，這是一個控制字元，表示要更換到下一列，其他的控制字元在第四章會提到。
若將本例中的 \n 都刪除，那秀出的結果會像這樣子：

```
8 + 2 = 108 - 2 = 68 * 2 = 168 / 2 = 4
```

◎ C 的四則運算

電腦語言用的四則運算符號幾乎都是相同的：

四則運算符號	意　義	範　例	結　果
+	加法	4 + 2	6
-	減法 (或代表負號)	4 - 2	2
*	乘法	4 * 2	8
/	除法	4 / 2	2
相關運算符號	**意　義**	**範　例**	**結　果**
++	變數值加 1	i++ 或 ++i	i 值加 1
--	變數值減 1	i-- 或 --i	i 值減 1
%	整數除法的餘數	4 % 2	0

在書中提到 C 語言沒有提供次方的功能，指的是在某些電腦語言可以用 ** 表示次方，如： 2 ** 3 ，表示 2 的 3 次方；有的用 ^ 表示，如： 2 ^ 8 ，表示 2 的 8 次方。在 C 語言，沒有運算符號可以表示次方，但是 C 語言有提供次方的函式： pow()， pow(2 , 3) 表示 2 的 3 次方。

一個式子如果有多個運算的話，C 是以先乘除後加減的方法來運算，當然我們也可以用括號 ( ) 來改變這個法則，只要有括號，就優先運算。另外，在 C 語言內中括號 [ ] 及 大括號 { } 是有其他的用途，所以在作數學運算時，只要用括號，就只能用小括號 ( )，小括號可以多層，C 在運算時，是由最內層開始運算。

```
範例：   ( 1 + 2 * ( 3 + 4 ) ) * 5 - 6 * 7 / 2 + 8
      =  ( 1 + 2 * (   7   ) ) * 5 - 6 * 7 / 2 + 8
      =  (       15          ) * 5 - 6 * 7 / 2 + 8
      =            75              -   42  / 2 + 8
      =            75              -     21    + 8
      =   62
```

◎ 註解(Comments)

通常老師會要求初學者在程式的每一列加上註解，這是爲了讓初學者知道自己在寫些什麼程式碼，了解爲什麼要這樣寫，而不只是照著書 Keyin 程式。寫註解有助於自己了解程式的內容，便於日後的修改。但寫註解對於某些程式設計師而言可說是一種噩夢，因爲寫註解所花的時間可能會與寫程式的時間相去不遠，認爲寫註解只是在浪費時間。 對一個相當好的程式設計師而言，也許寫註解真的是浪費時間，因爲好的程式碼本身就已經隱含了註解，這也是寫程式相當高的境界。 對一般的程式設計師而言，寫一些註解還是比較好的作法，特別是某些程式碼是你花了一段時間才想到的「特殊」方法，加上一些註解，說明一下這個「特殊」的方法，以後要修改才能快速進入狀況，否則，你可能會佩服自己當時是如何想到的這個方法，又再花一段時間才知道自己在寫些什麼程式碼。 講了這麼多註解的正反面(正面居多)論調，在 C 語言中要如何寫註解呢？ 只要用 /* 和 */ 將你要的註解內容包起來就可以了。C 在編譯時，會將 /* */ 內的資料略去，就如同調位字元一樣。唯一的例外是：當 /* */ 在一組雙引號 " 內時，它們就屬於這組雙引號所包含的字串。 在 C++ 語言中則可用//當註解.

comments.c or comments.cpp

```
 1  #include <stdio.h>     /* prototype : printf() */
 2  void main(void)  // main program
 3  {
 4  /* 所有的程式碼都變成註解，所以這個程式目前是空的 */
 5  /* printf("%d + %d = %d\n", 8 , 2 , 8+2 );    */
 6  /* printf("%d - %d = %d\n", 8 , 2 , 8-2 );
 7     printf("%d * %d = %d\n", 8 , 2 , 8*2 );
 8     printf("%d / %d = %d\n", 8 , 2 , 8/2 );  //  division
 9  */
10  }    // end of program
```

◎ 巢狀註解(Nested Comments)

nestcom0.c

```
 1  #include <stdio.h>     /* prototype : printf() */
 2  void main(void)
 3  {
 4  /* 這個程式必須把巢狀註解的設定打開，才不會有錯誤 */
 5  /*
 6     printf("%d + %d = %d\n", 8 , 2 , 8+2 );
 7  /* printf("%d - %d = %d\n", 8 , 2 , 8-2 );        */
 8     printf("%d * %d = %d\n", 8 , 2 , 8*2 );
 9     printf("%d / %d = %d\n", 8 , 2 , 8/2 );
10  */
11  }
```

上面的例子，有四組註解 /* */ ，其中第三組及第四組的註解之間有部分重疊。
想要 Compile 沒有錯誤，必須第 5 列的 /* 與 第 10 列的 */ 配，也就是
第 5 列到第 10 列都是註解；另外第 7 列的 /* 與 第 7 列的 */ 配，也就是
第 7 列是註解。這種註解方式，我們稱之爲巢狀註解。
Turbo C 內定是不可使用巢狀註解的，上面的例子會是第 5 列的 /* 與 第 7 列
的 */ 配，結果在第 10 列的 */ 會變成是多餘的，造成 Compile 錯誤。
打開巢狀註解的方法：
按下 F10 → Options → Compiler → Source → Nested comments Off
將 Off 設爲 On 就可以了。

◎ 巢狀註解的使用時機

在前面的例子只是爲了說明巢狀註解，也許你會覺得這樣的用法是自找麻煩，
但是以下的例子，你就會認爲有巢狀註解的功能還是比較好的。
在 nestcom1.c 中，每一列的 printf(); 後面都加上了註解。
若要把這幾列程式變成註解，不使用巢狀註解，就會像 nestcom2.c 一樣，
必須在每一列的 printf(); 前後再加上 /* */，若是使用巢狀註解，
就像 nestcom3.c 一樣，只要在這幾列的前後加 /* */ 就可以了。

nestcom1.c

```
 1  #include <stdio.h>     /* prototype : printf() */
 2  void main(void)
 3  {
 4     /* 這個程式在每一個敘述後都加上了註解 */
 5
 6     printf("%d + %d = %d\n", 8 , 2 , 8+2 );     /* 8 + 2 = 10 */
 7     printf("%d - %d = %d\n", 8 , 2 , 8-2 );     /* 8 - 2 = 6  */
 8     printf("%d * %d = %d\n", 8 , 2 , 8*2 );     /* 8 * 2 = 16 */
 9     printf("%d / %d = %d\n", 8 , 2 , 8/2 );     /* 8 / 2 = 4  */
10
11  }
```

nestcom2.c

```
 1  #include <stdio.h>     /* prototype : printf() */
 2  void main(void)
 3  {
 4  /* 這個程式不用把巢狀註解的設定打開，也不會有錯誤 */
 5
 6  /* printf("%d + %d = %d\n", 8 , 2 , 8+2 ); */  /* 8 + 2 = 10 */
 7  /* printf("%d - %d = %d\n", 8 , 2 , 8-2 ); */  /* 8 - 2 = 6  */
 8  /* printf("%d * %d = %d\n", 8 , 2 , 8*2 ); */  /* 8 * 2 = 16 */
 9  /* printf("%d / %d = %d\n", 8 , 2 , 8/2 ); */  /* 8 / 2 = 4  */
10
11  }
```

nestcom3.c

```
 1  #include <stdio.h>     /* prototype : printf() */
 2  void main(void)
 3  {
 4  /* 這個程式也必須把巢狀註解的設定打開，才不會有錯誤 */
 5  /*
 6     printf("%d + %d = %d\n", 8 , 2 , 8+2 );     /* 8 + 2 = 10 */
 7     printf("%d - %d = %d\n", 8 , 2 , 8-2 );     /* 8 - 2 = 6  */
 8     printf("%d * %d = %d\n", 8 , 2 , 8*2 );     /* 8 * 2 = 16 */
 9     printf("%d / %d = %d\n", 8 , 2 , 8/2 );     /* 8 / 2 = 4  */
10  */
11  }
```

■ 第三章 常數與變數

C 語言的資料可分爲常數(constant)及變數(variable)，常數指的是固定不變的數，例如：0,1,2 等數值，或是用雙引號定義的字串，我們也稱之爲字串常數。
變數指的是數值可以改變的數，例如：一個整數變數，我們可以把它設成 1，然後再改爲 10，或是其他的整數數值。
一個程式若沒有變數，那純粹只是將常數秀出來而已，好比是用文字編輯器編輯一個檔案，再用 type 把它秀出來一樣。有了變數，就可以做不同的變化。

◎ 變數的型態——Char, int, long, float, double etc.。

◎ 變數的命名

如同檔案的命名，變數的名字要取得有意義，在 C 中，名字可以取得很長，但是要用英文的，所以你可以把變數用中翻英來命名。

◎ 變數的命名規則

○ 變數名稱的第一個字元必須是英文字母(A 到 Z 或 a 到 z)或是底線(_)。
○ 第二個字元以後可以使用前述字元，再加上數字 0 到 9 。
○ 變數名稱的大小寫是不同的。
○ 變數名稱的最前面 32 個字元有效。
○ 不可以使用「保留字」當變數的名稱，保留字是給編譯器使用，不可以當成變數名稱。TC 有以下的保留字：

流程：	if	else	for	do	while
	switch	default	case	break	continue
	goto	return			
型別：	char	int	long	float	double
	void	register	signed	unsigned	
	short	near	far	huge	
	typedef	struct	union	enum	
	auto	const	static	volatile	extern
	interrupt		cdecl	pascal	asm
運算：	sizeof				
暫存器：	_AX	_AH	_AL	_cs	_CS
	_BX	_BH	_BL	_ds	_DS
	_CX	_CH	_CL	_es	_ES
	_DX	_DH	_DL	_ss	_SS
	_SI	_DI	_BP	_SP	

◎ 變數的設定

使用變數時，應該先考量這個數可能的數值範圍，用以選定變數的型別，例如：用一個數來存班上的人數，一個合班的大班級可能超過百人，但最大不太可能超過千人，所以選一種變數型別可存 1000 以下的數值，在此可選整數。
若是要用一個數來存你的存款，則整數的上限 32767 可能某些同學一個月的薪資就是它的數倍，所以要選長整數，它的上限是 2147483647 。在數學運算時，想要有小數的就要用浮點數(float)。
在 C 語言中，變數宣告的語法如下：

```
型別  變數名稱 1 [,變數名稱 2 [,...]] ;
```

```
例如： int   NumberOfStudent;                /* 學生人數 */
       long  MoneyInBank, interest;          /* 銀行存款 */
```

```
float RateOfInterest;                  /* 利息利率 */
char  EndOfString;                     /* 字串結束 */
char  OneStudentName[9];               /* 學生姓名 */
```

在宣告變數時，我們可以設定變數的初始值(initial value)，語法如下：

```
型別  變數名稱 1=初始值 1 [,變數名稱 2=初始值 2 [,...]] ;
```

例如：
```
int   NumberOfStudent=60;              /* 學生人數 */
long  MoneyInBank=1000000L;            /* 銀行存款 */
float RateOfInterest=5.0;              /* 利息利率 in % */
char  EndOfString='\0';                /* 字串結束 */
char  OneStudentName[9]="王大明";    /* 學生姓名 */
```

注意：在銀行存款的設定數值 1000000 後加上一個 L ，表示這個常數數值 1000000 是一個長整數。因爲 C 語言內定的型別是整數，爲了防止不可預期的狀況發生，最好是自己把它設定成你想要的型別，不要假設 TC 會幫你做好好的，要假設 TC 很笨不會幫你做，這樣在發展大程式要除錯時，就可以把問題簡化，不必再考慮是不是資料型別錯誤，只要把程式流程或演算法搞定就可以了。
在 TC 中，不加 L，結果還是正確的，但是在其它的環境下可能會不同。多加一個 L 並不會使程式變大或變慢，又能保障正確使用，何樂不爲。

○ 複習一下字元與字串：

char 字元只佔一個 byte，以一組單引號 ' 引含字元資料，其表示法如下：
⊙ 單一字元：　'A'　、'a'　、'0'　。
⊙ 八進位數值：　'\101'、'\141'、'\60' 、'\0'
⊙ 十六進位數值：'\x41'、'\x61'、'\x30'、'\x0'

字串則是由一個以上的字元所組成的，而且以 '\0' 這個字元做爲結尾。
表示法： "123"、"ABC"、"abc"。以上的三個例子都是佔 4 bytes。
用 strlen() 可以取得字串的長度(不含 '\0' 字元)。
如：　int StringLen=strlen("123");
這樣，StringLen 就等於 3 。
☆ 在使用 strlen() 時，必須加入　#include <string.h>

◎ 設定敘述

前面已經說明了變數在宣告時給定初值的方法，接下來是在程式執行的過程中設定變數數值的方法。 即然是變數，表示它的數值可能在程式執行的過程中會改變多次，如果一個變數在整個程式執行中都不會改變，或許你該把它設成常數。
在設定變數時，可以用等號 = 來設定變數新值，語法如下：

```
變數名稱 = 運算式(運算式、函式傳回數值或兩者混合);
```

這個意思是等號左邊「變數名稱」的數值會等於等號右邊「運算式」的運算結果。在 C 中，等號 = 是用來設定變數數值的，所以在等號的左邊必須是變數，不可以是常數。在邏輯上的相等，在 C 中是用兩個等號 == 來表示，有關邏輯的表示，在第五章中會作介紹。以下我們來看一些設定的例子，計算圓的面積：

```
PI = 3.1415926;
r = 4;
area = PI * r * r ;
```

以下是用變數的方式表示的範例：

var.c

```
#include <stdio.h>
void main(void)
{
   int i,j;

   i = 10;
   j = 2;
   printf("%d + %d = %d\n", i , j , i+j );
   printf("%d - %d = %d\n", i , j , i-j );
   printf("%d * %d = %d\n", i , j , i*j );
   printf("%d / %d = %d\n", i , j , i/j );

   i = 20;
   j = 2;
   printf("%d + %d = %d\n", i , j , i+j );
   printf("%d - %d = %d\n", i , j , i-j );
   printf("%d * %d = %d\n", i , j , i*j );
   printf("%d / %d = %d\n", i , j , i/j );
}
```

```
10 + 2 = 12
10 - 2 = 8
10 * 2 = 20
10 / 2 = 5
20 + 2 = 22
20 - 2 = 18
20 * 2 = 40
20 / 2 = 10
```

變數使用的有效範圍:

整體變數(Global Variable): 整體程式內
區域變數(Local Variable):函式內
靜態變數(Static Variable): 單一程式內

■ 第四章　基本輸出入函式

這一章將介紹一些基本輸出入的函式，使用者經由這些函式可以與電腦溝通，讓程式讀取使用者的輸入部分。程式依使用者不同的要求，做不同的事，再將結果輸出給使用者。

◎ 輸出指令：printf()

在第二章中，曾經談過 printf 指令，現在來詳細的探討它。

printf 是一種格式化的輸出指令，換句話說，你可以用它來編排你所要的輸出格式。

printf 的一般型式如下:

```
printf("控制字串" , 運算式 1 , 運算式 2 , ... );
```

控制字串是你打算要秀出的訊息，其中利用 % 與 \ 這兩個字元，來控制數值的輸出格式。

控制字串中每一個 % 符號，表示在後面有一個運算式與它對應，運算式的值會代入這個 % 的位置。在 % 後的字元表示代入數的型別，常用的控制碼如下表：

printf 的控制碼	代表代入的數值型別
%c	字元
%d	十進位之整數
%ld	十進位之長整數
%f	浮點數
%lf	倍精浮點數
%Lf	長倍精浮點數
%s	字串

運算式的型別必須跟控制碼所代表的型別相符，否則會秀出不可預期的資料。

另一個控制符號 \ ，其實只是定義字元而已。在上一章已經介紹了字元的各種表示法，如：'\x41' 表示 A 字元。

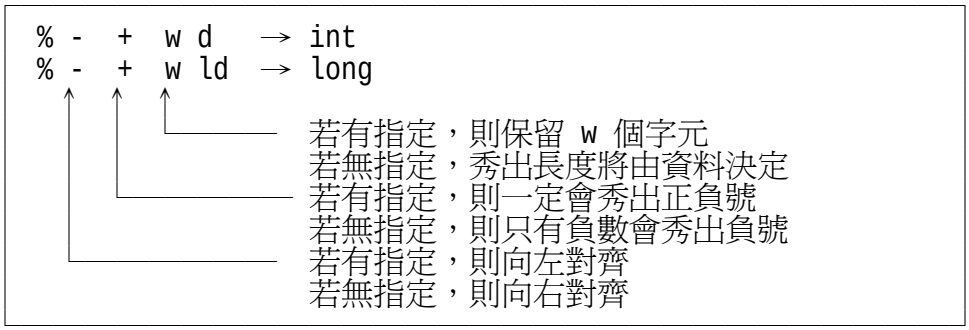
在 C 語言中，將一些特殊的控制字元另外定義，這些控制字元大部份跟游標的控制有關，如下表：

	控制字元	Dec	Hex	功能
	\n	10	0x0A	換列，也就是將游標移到下一列
	\t	9	0x09	將游標移到下一個定位(1+8n)
	\b	8	0x08	退一格，類似按下左鍵
	\a	7	0x07	喇叭叫一聲
	\r	13	0x0D	回到列首
	\f	12	0x0C	跳頁，在列表時控制列表機跳頁
	\\	92	0x5C	印出 \ 字元
	\'	39	0x27	印出 ' 字元
	\"	34	0x22	印出 " 字元
	\xHH		0xHH	印出 0xHH 所表示的字元
*	%%	37	0x25	印出 % 字元

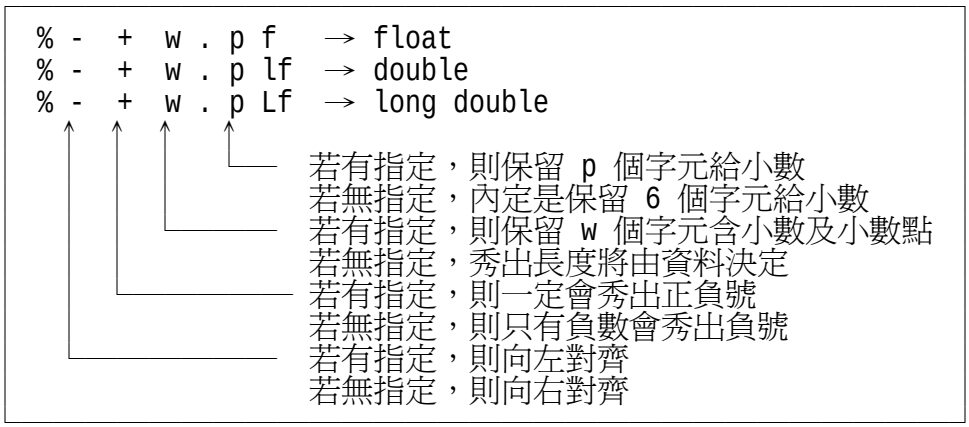
其中，% 字元的定義與在控制字串中的表示法不同，其餘的字元在定義上與在控制

字串中的表示法都相同。
printf 在做輸出時，你也可以指定保留多少位置給對應的運算式放結果。指定的方式是在 % 之後加一個數值，如 %5d：表示保留 5 個字元空間給一個十進位整數；%12ld：表示保留 12 個字元空間給一個十進位長整數。
如果要輸出資料的長度比你指定的保留空間還要大時，printf 就不理會你的設定，把要輸出的資料完整的輸出，所以，你在設定保留空間時，應該注意輸出資料的範圍及長度，保留夠大的空間，確保輸出格式的整齊。
在浮點數方面，你除了可以指定保留的空間外，還可以指定小數點後要取幾位。如 %8.3f ：表示保留 8 個字元空間給一個浮點數，小數部分則是佔 3 個字元空間，由於小數點本身佔一個字元，所以整數部分佔 8 - (3 + 1) = 4 個字元空間。
printf 在輸出資料時，如果你指定保留的空間比要秀的資料長度還要大時，那 printf 先秀一些空白，再秀出資料，使總長度等於你所指定的寬度，這樣等於是讓輸出的資料向右對齊。如果你想要讓資料是向左對齊的話，可以在指定寬度時使用負數，如 %-5d：表示保留 5 個字元空間給一個十進位整數，若資料長度不足 5 ，則在秀出資料後補空白。

○ 整數(int)及長整數(long)



○ 浮點數(float)、倍精浮點數(double)及長倍精浮點數(long double)



說了這麼多，只有自己試試看才知道！以下是個簡單的例子：

print.c

```
1  #include <stdio.h>
2  void main(void)
3  {
4     printf("|%ld|\n",    123456 );
5     printf("|%5ld|\n",   123456 );
6     printf("|%d|\n",     123    );
7     printf("|%5d|\n",    123    );
8     printf("|%-5d|\n",   123    );
```

```
 9     printf("|%f|\n",    12.345 );
10     printf("|%9f|\n",   12.345 );
11     printf("|%9.2f|\n", 12.345 );
12     printf("|%-9.2f|\n",12.345 );
13  }
```

```
|123456|          ← 123456 大於 32767 要長整數才能表示,所以用 %ld
|123456|          ← 所保留的 5 個字元不夠使用，所以 TC 視同你沒設
|123|
|  123|           ← 保留 5 個字元，只使用 3 個字，向右靠齊
|123  |           ← 保留 5 個字元，只使用 3 個字，向左靠齊
|12.345000|       ← 小數沒有指定，所以 TC 使用內定的 6 個小數。
|12.345000|       ← 保留 9 個字元，小數部分仍使用內定值
|    12.35|       ← 保留 9 個字元，小數 2 個字元，向右靠齊
|12.35    |       ← 保留 9 個字元，小數 2 個字元，向左靠齊
```

◎ 輸入指令：scanf()

C 語言使用 scanf 指令來讀取 keyboard 輸入的資料。scanf 的一般型式如下：

```
scanf("控制字串" , &變數 1 , &變數 2 , ... );
```

scanf 與 printf 可以說是相對的，一個用來做輸入，一個用來做輸出。
scanf 的控制字串與 printf 幾乎是一樣。

	scanf 的控制碼	代表輸入的數值型別
	%c	字元
	%d	十進位之整數
	%ld	十進位之長整數
*	%D	十進位之長整數
	%f	浮點數
	%lf	倍精浮點數
	%Lf	長倍精浮點數
	%s	字串

★ 注意：沒有 %F 這種控制碼，課本有誤！
%D 的控制碼只能用在 scanf() ，在 printf() 中無法使用。

在用 scanf 時還有一點要注意，在控制字串後的變數，使用的是指標(pointer)。
什麼是指標？指標就是指向記憶體的一個位址，在那個位址存放著資料。
例如：一個整數變數 i ，它是存在記憶體的某一個位址，那個位址在 C 語言中，是以 &i 來表示，我們通常稱 &i 是 i 的位址，也稱爲 i 的指標。
以下是常用的變數及其指標(因爲它的位址固定不會改變，也稱爲指標常數)：

```
char c;           /* 字元          */    /* c  的指標是 &c  */
int i;            /* 整數          */    /* i  的指標是 &i  */
long l;           /* 長整數        */    /* l  的指標是 &l  */
float f;          /* 浮點數        */    /* f  的指標是 &f  */
double d;         /* 倍精浮點數    */    /* d  的指標是 &d  */
long double ld;   /* 長倍精浮點數  */    /* ld 的指標是 &ld */
```

```
char str[80];     /* 字元陣列(字串) */  /* str[80] 的指標是 str */
int a[100];       /* 整數陣列       */  /* a[100]  的指標是 a   */
long b[100];      /* 長整數陣列     */  /* b[100]  的指標是 b   */
float c[100];     /* 浮點數陣列     */  /* c[100]  的指標是 c   */
```

以下的範例，將第三章的範例 var.c 變數改由 scanf 輸入：

io.c

```
 1  #include <stdio.h>
 2  void main(void)
 3  {
 4     int i,j;
 5
 6     printf("Enter 2 integers:");
 7     scanf("%d %d", &i, &j );        /* 用 i 與 j 的指標 &i,&j */
 8     printf("Now, I find that ...\n");
 9     printf("%d + %d = %d\n", i , j , i+j );
10     printf("%d - %d = %d\n", i , j , i-j );
11     printf("%d * %d = %d\n", i , j , i*j );
12     printf("%d / %d = %d\n", i , j , i/j );
13  }
```

```
Enter 2 integers:20 4            ← 相當於 i = 20 , j = 4
Now, I find that ...
20 + 4 = 24
20 - 4 = 16
20 * 4 = 80
20 / 4 = 5
```

scanf 在讀取多筆資料時，是把調位字元(如：空白、定位(tab)及換列字元)當作資料的分隔的記號，也就是利用 scanf 輸入多筆資料時，必須用調位字元分隔輸入的資料。如本例中，輸入的 i 值與 j 值，可以用空白區隔，也可以用換列或定位來區隔。在結束輸入時則要用換列字元，也就是說要按 Enter 。

進階的程式設計師通常不會用 scanf 作爲資料的輸入，因爲只要輸入的格式有一點點錯誤， scanf 所得到的結果就無法預期。大部分的程式設計師會用 gets() 這個函式讀入字串，gets 是以換列字元作爲結束記號，所以，利用 gets 可以讀入包含空白及定位的字串。讀入的字串可以用 atoi() 轉成整數、用 atol() 轉成長整數、用 atof() 轉成浮點數。
更進階的程式設計師則是自行設計輸入函式，並加入一些設定防止輸入錯誤發生，例如，在輸入整數的函式中，使用者只有按下數字鍵才有反應，若按下非數字鍵，則不會反應。也可以限定使用者輸入的資料長度，防止輸入的數值過大，等等。

以下的範例，將華氏溫度(℉)換算成攝氏溫度(℃)

f2c.c

```
 1  #include <stdio.h>
 2  void main(void)
 3  {
 4     int f,c;
 5
 6     printf("Enter the temperature in F : ");
```

```
 7     scanf("%d", &f );                          /* 用 f 的指標 &f */
 8     c = ( f - 32 ) * 5 / 9 ;                   /* 換算公式 */
 9     printf("%d degrees in F is %d degrees in C.", f, c);
10  }
```

```
Enter the temperature in F : 100
100 degrees in F is 37 degrees in C.
```

華氏溫度(℉)與攝氏溫度(℃)的互換公式如下：

```
C = ( F - 32 ) * 5 / 9
F = C * 9 / 5 + 32
```

以下是課本的範例，將年紀改用"日"來估算：

age.c

```
 1  #include <stdio.h>
 2  void main(void)
 3  {
 4     float years, days;
 5
 6     printf("Enter the age of you : ");
 7     scanf("%f", &years );            /* 用 years 的指標 &years */
 8     days = years * 365.25 ;                        /* 換算公式 */
 9     printf("You are %f days old.", days );
10  }
```

```
Enter the age of you : 28.5
You are 10409.625000 days old.
```

◎ 輸入指令：getche()

使用 scanf() 讀取字元時，必須再按下 Enter 鍵，該字元才會被讀取。
在某些場合中，我們希望每按一個鍵，程式就會讀取，不用再按下 Enter 鍵，
例如：在電腦遊戲中，每按下方向鍵，所控制的人物就依所按的方向移動。
利用 getche() 可以達到這個目的。 getche 的一般型式如下：

```
ch = getche();
```

將 getche() 所讀到的字元傳給 ch 。此外，getche() 會將讀到的字元先秀在螢幕上。

以下的範例，將所按的鍵秀出其 ASCII 碼：

code.c

```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2  #include <conio.h>     /* 宣告 getche() 的原型 */
 3  void main(void)
 4  {
 5     char ch;
 6
```

```
 7     ch = getche();      /* getche() 會傳回你所按下鍵的字元 */
 8     printf(" -- You typed %c.\n", ch );
 9     printf("Character %c has ASCII code %d.\n", ch, ch );
10  }
```

```
A -- You typed A.
Character A has ASCII code 65.
```

這個範例主要是用來查詢一般字元的 ASCII 碼，請勿輸入方向鍵，否則它的結果可能會讓你失望。因爲方向鍵及功能鍵等特殊按鍵會產生兩個碼，必須讀兩次才能得到正確的結果。
第 9 行，用 %c 的控制字元，表示要秀出 ch 所代表的字元；
用 %d 的控制字元，表示要秀出 ch 所代表的數值。

☆★☆★☆★☆★ 練習 ☆★☆★☆★☆★

實作課本習題第 10 題，看看你所得到的結果與課本所列的的結果一不一樣？

☆★☆★☆★☆★ 作業 ☆★☆★☆★☆★

■ Help 參考資料：

Help

```
printf: formatted output to stdout

int printf(const char *format, ...);

Prototype in stdio.h

Print formats a variable number of arguments
according to the format, and sends the output
to stdout. Returns the number of bytes output.
In the event of error, it returns EOF.

See also     ecvt     fprintf    putc
             puts     scanf      vprintf
```

格式化輸出至 stdout

printf 的語法

必須 #include<stdio.h>

不同的格式須要不同的參數，這些資料會送到 stdout。傳回值是輸出的 byte 數，若發生錯誤則傳回 EOF

相關指令

Help

```
              Format Specifiers

  % [flags] [width] [.prec] [F|N|h|l] type

type          Format of Output
 d      signed decimal int
 i      signed decimal int
 o      unsigned octal int
 u      unsigned decimal int
 x      in printf = unsigned hexdecimal int
        lowercase; in scanf = hexadecimal int
 X      in printf = unsigned hexdecimal int
        uppercase; in scanf = hexadecimal long
 f      floating point [-]dddd.ddd
 e      floating point [-]d.ddd e [+/-]ddd
 g      format e or f based on precision
 E      same as e except E for exponent
```

format 的格式

[ ] 表示不一定要用

格式化輸出的型別：
d 帶正負號十進位整數
i 帶正負號十進位整數
o 不帶正負號八進位整數
u 不帶正負號十進位整數
x 不帶正負號 16 進位整數(小寫)
  在 scanf 爲 16 進位整數
X 不帶正負號 16 進位整數(小寫)
  在 scanf 爲 16 進位長整數
f 浮點數 例： 314.159
e 浮點數 例： 3.14159e2
g 由精度決定用 f 或 e 的格式
E 同 e 只是以 E 表示指數符號

```
 G       same as g except E for exponent
 c       single character
 s       print characters till '\0' or [.prec]
 %       the % character
 p       pointer: near - YYYY; far - XXXX:YYYY
 n       stores count of characters written so
         far in the location pointed to by
         input argument

[flag]         What it Specifies

none     right-justify, pad 0 or blank to left
 -       left-justify, pad spaces to right
 +       always begin with + or -
blank    print sign for negative values only
 #       convert using alternate form:
         c,s,d,i,u  no effect
         o          0 prepended to nonzero arg
         x or X     0x or 0X prepended to arg
         e, E, f    always use decimal point
         g or G     same as above but no
                    trailing zeros

[width]        Effect on Output

   n     at least n characters, blank-padded
   0n    at least n characters, 0 left fill
   *     next argument from list is width

[.prec]        Effect on Output

 none    default precision
 .0      d,i,o,u,x    default precision
         e, E, f      no decimal point
 .n      at most n characters
 *       next argument from list is precision

Modifier       How arg is Interpreted

 F          arg is far pointer
 N          arg is near pointer
 h          d,i,o,u,x,X arg is short int
 l          d,i,o,u,x,X arg is long int
 l          e, E, f, g, G  arg is double
            (scanf only)
 L          e,E,f,g,G   arg is long double
```

```
G 同 g 只是以 E 表示指數符號
c 單一字元
s 列印字元直到 '\0' 或指定長度
% 列印 % 這個字元
p 指標：近 YYYY 遠 XXXX:YYYY
n 將目前已經列印出的字元數值
  傳給輸入的參數。在此，
  輸入的參數必須是整數指標。

[旗標]

無 靠右對齊不足在左邊補空白或 0
-  靠左對齊不足在右邊補空白
+     正負號一定會秀出
空白 只有負數才會秀出負號
#     轉換格式
對 c,s,d,i,u 沒有影響
o 如果資料不是 0 就會先秀 0
x,X 在資料前加秀 0x 或 0X
e,E,f 會秀出小數點
g,G   會秀出小數點，但是不補 0


[寬度]

n  指定輸出的寬度，不足補空白
0n 指定輸出的寬度，不足在左補 0
*  由下一個參數決定寬度

[.精度]

無  內定的精度
.0 d,i,o,u,x 是內定的格式
   e,E,f 表示沒有小數
.n 指定 n 個字元長度
.* 由下一個參數決定精度

輸入大小修飾詞

F 遠指標
N 近指標
h 短整數
l 長整數
l 倍精數(只有在 scanf)

L 長倍精數
```

```
Help
scanf: performs formatted input from stdin

int scanf(const char *format, ...);

Prototype in stdio.h

Returns the number of input fields processed
successfully.  It processes input according to
the format and places the results in the
memory locations pointed to by the arguments.

See also     atof    cscanf   fscanf
             getc    printf   sscanf
             vfscanf vscanf   vsscanf
```

由 stdin 讀入格式化的資料

scanf 的語法

必須 #include<stdio.h>

傳回成功讀取的欄位數。
輸入的資料會依照指定格式儲存，並將它們放到參數所指到的記憶體。

相關指令

```
Help
getch: gets character from console, no echoing
getche: gets character from the console and
        echoes to screen

int getch(void);
int getche(void);

Prototype in conio.h

Both functions return the character read.
Characters are available immediately - no
buffering of whole lines.

Special keys such as function keys and arrow
keys are represented by a two character
sequence: a zero character followed by the
scan code for the key pressed.

See also     getpass   cgets     cscanf
             kbhit     ungetch   putch
             getchar   getc
```

由控制台讀入字元，不回應
由控制台讀入字元，並回應到螢幕

getch 的語法
getche 的語法

必須 #include<conio.h>

getch 及 getche 兩者都會回傳所讀到的字元。由於沒有緩衝區，所傳回的字元立即可用。
功能鍵或方向鍵等這些特殊按鍵，會產生連續的兩個字元：第一個字元是 0，第二個是所按下鍵的掃描碼。

相關指令

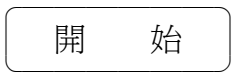
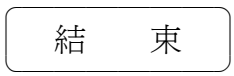
■ 第五章 流程圖與抉擇指令

前一章的 scanf 好不好用呢？在多筆資料輸入時，不小心輸錯一筆資料，可能會導致後面的讀到的資料都是錯的。而錯誤的輸入資料，自然會使輸出的資料也是錯誤的，這就是所謂的「垃圾進、垃圾出」(Garbage In, Garbage Out)。如果程式能分辨輸入的資料是否正確，如果是錯誤的或是超過範圍的，就可以要求使用者再輸入一次。要達到這個目的，我們必須使用「判斷」以及「迴圈」。本章將介紹如何用 C 來做判斷，有關迴圈的部分參考下一章。

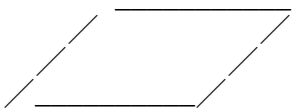
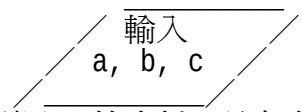
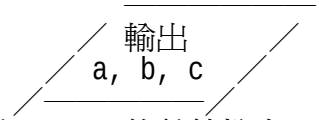
◎ 預習課程：流程圖 (Flow Chart)

在進入本章之前，先介紹流程圖。流程圖不只可以用在程式的流程設計，也可以用在任何事件，例如：計畫執行的流程圖、休閒計畫流程圖等等。 在程式設計上，小程式也許不用流程圖。 但是中、大型程式最好要有流程圖，有了流程圖，可以進一步檢視你的設計，看看是否在邏輯上有問題，看看那個部分可以再加強，那個部分可以簡化，因爲看流程圖比看原始碼要容易除錯。另外流程圖設計得好，程式的部分就顯得相當簡單了，只要把流程圖的符號用對應的程式流程控制敘述取代，程式的架構就完成了。 隔了一段時間，若須要增加或修改程式部分功能，有留下流程圖，就可以快速進入狀況，找到要修改的部分，並新增程式。以下介紹 6 種常用的流程圖符號：

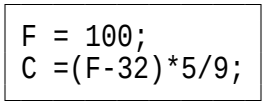
○ 開始／結束符號

符　　號	意　　義	例　　子
	表示流程圖的開始或結束，每個流程圖必須以開始符號開始，以結束符號作結束。	 流程圖的開始  流程圖的結束

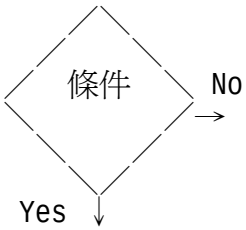
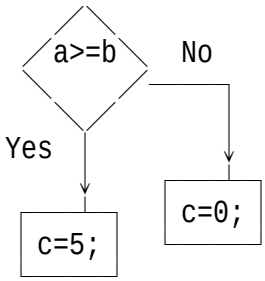
○ 輸入／輸出符號

符　　號	意　　義	例　　子
	表示電腦與外界的連繫，通常程式需從外界輸入資料，程式再將執行後的結果輸出給使用者。 在符號中，輸入資料時要註明是「輸入」，在輸出資料時要註明是「輸出」。	 讀入三筆資料，並存在 a, b, c 三個變數內。  將變數 a,b,c 的數值輸出。

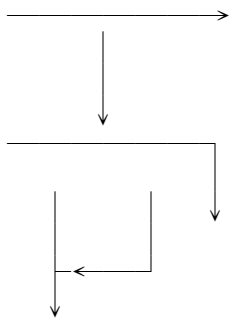
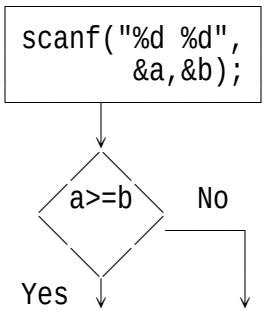
○ 處理符號

符　　號	意　　義	例　　子
	表示各種算術、函數的運算。 在一個處理符號內可表示多個運算。	

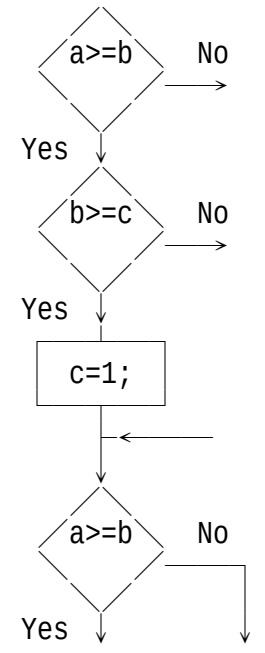
○ 抉擇符號

符　號	意　義	例　子
	表示各種算術、邏輯或關係運算，程式執行到這個地方必須做一決定，此項決定由符號內的條件來判斷。當條件成立時，則程式流程依 Yes 的方向指示繼續執行；當條件不成立時，則程式流程依 No 的方向指示繼續執行。在這個菱形符號的四個角，上面的角通常是被上面的流程符號所指，其它的三個角，你可以任選兩個角做 Yes 及 No 的流程指向，只要標明清楚就可以了。	 當 a 大於或等於 b 時執行 c=5; 的運算，否則執行 c=0; 的運算。

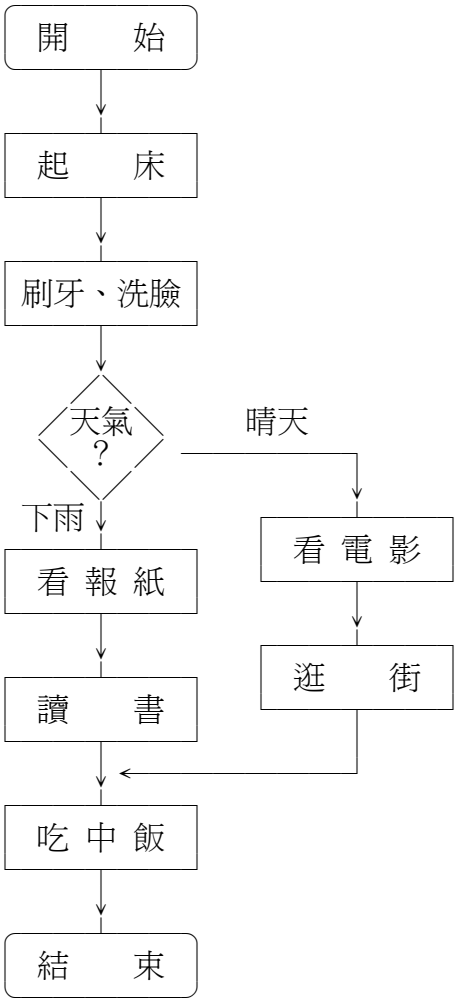
○ 流向線

符　號	意　義	例　子
	箭頭方向表示程式執行的先後次序。	

○ 連接符號

符　號	意　義	例　子
○	繪製流程圖時，常會受到空間的限制，有時流向線需流到比較遠的地方。這種長的流向線往往會使流程圖看起來比較亂，有時流程圖很大，可能須要許多頁來繪製，這時，利用連接符號就可以簡化流向線。 在連接符號內會有一個數字或字母做爲識別。在不同的地方看到連接符號，只要它們的識別數字或字母是相同的，就表示它們是相連接的。	

○ 範例：星期天流程圖



◎ 關係與條件

在數學上任兩個數值一定會符合「三一律」，什麼是「三一律」？假設兩個數：一個是 a ，一個是 b ，那 a 跟 b 的關係一定是下列三種的某一種：
a 大於 b ； a 等於 b 或 a 小於 b 。
在電腦程式中，常常要判斷數值間的關係，例如：一些遊戲軟體會在一開始要求你輸入密碼，輸入的密碼與正確的密碼相等，遊戲才會繼續執行；或是機密性的資料庫管理系統會要求輸入密碼，系統再依輸入的密碼給予相對的權限，如：總經理可以看到全部的資料，一般員工只能看自己的資料。
在 C 語言中提供了下列的關係運算子，讓程式設計師判斷兩個數值的關係：

關係運算子	意　義	例子	狀　況
>	大於	a>b	a 是否大於 b ？
<	小於	a<b	a 是否小於 b ？
>=	大於或等於	a>=b	a 是否大於等於 b ？
<=	小於或等於	a<=b	a 是否小於等於 b ？
==	等於	a==b	a 是否等於 b ？
!=	不等於	a!=b	a 是否不等於 b ？

兩個數的關係符合三一律，所以，當你使用上述任一個關係運算子時，所指定的狀況不是成立，就是不成立。指定的狀況成立，我們稱之爲「真(True)」，狀況不成立，我們稱之爲「假(False)」。
在 C 語言中，假(False)是用 0 來表示。不是假的，就是真的，所以，所有非 0 的值，在 C 語言中都表示真(True)。
★注意：在 C 語言中 = 是設定運算子； == 是關係運算子，兩者不可混用。

a = 3 ;	a 的數值設爲 3
a == 3	a 是否等於 3 ？ a 等於 3 則此關係式爲真，否則爲假。

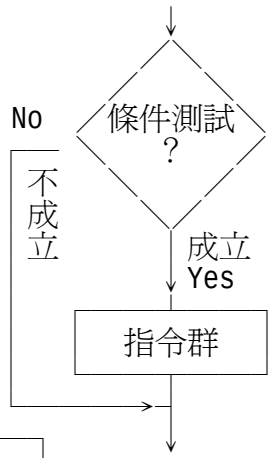
◎ if 敘述

if 敘述可用來作基本的抉擇：

如果某個條件成立，則要做某事。

例：如果天下雨的話，我就要帶傘。

條件　　要做的事



if 敘述的型式如下：

```
if(條件測試) { 指令群 }
```

意義：

如果 (條件測試) 是成立的，則執行 { 指令群 } 。

○ 指令群？
在 C 語言中，所謂的指令群，乃是由數個(包括 0 個,1 個,...)指令括在一對大括號中而形成。例如：

```
{ }                          ;              空指令
{
   i = i + 1;                i = i + 1;     單一指令
}
{
   i = i - 1;
   j = i * 2;
}
```

這三者都可以稱爲指令群。指令群的後面不必以分號做爲結束，因爲右大括號便足以代表結尾在何處了。當指令群內只有一個指令時，我們也可以將這一對大括號省略，此時，該指令群便與一般指令無異。當指令群內沒有任何指令時，如果你要省略這對大括號，就要補上一個分號 (;)，表示是空指令。

以下的範例，將使用者輸入的數值取絕對值後再秀出：

abs.c

```
1 #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
2 void main(void)
3 {
```

```
 4    int i;
 5
 6    printf("Enter an integer:");
 7    scanf("%d",&i);
 8    if( i < 0 ) i = -i;
 9    printf("The absolute value of it is %d.\n", i );
10 }
```

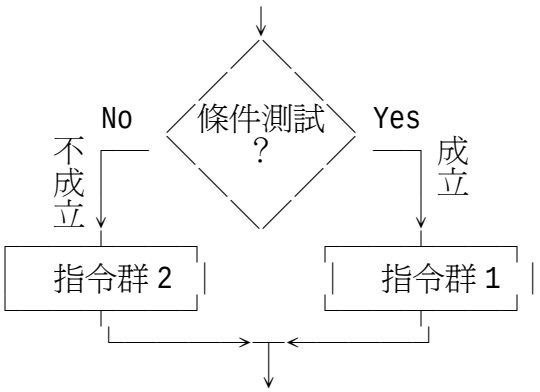
```
Enter an integer:-100
The absolute value of it is 100.
```

絕對值的定義： $|i| = \begin{cases} i & , 若\ i >= 0 \\ -i & , 若\ i < 0 \end{cases}$

所以在程式第 8 行，當 i < 0 成立時，就令 i = -i ，這樣 i 就成爲正數了。

◎ 二重選擇題： if - else 敘述

只有 if 只能設定條件成立要做的事，
再加上 else 則可設定條件不成立時
要做的事。
例如：如果好天氣，就出去逛街，
　　否則，就在家看電視。



if-else 敘述的型式如下：

```
if(條件測試) { 指令群 1 }
else   { 指令群 2 }
```

意義：

```
如果 (條件測試) 是成立的，則執行 { 指令群 1 }；
否則，執行 { 指令群 2 }
```

以下是課本的範例，將使用者輸入的兩個數值找出最大值後再秀出：

max1.c

```
 1 #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
 2 void main(void)
 3 {
 4    int i, j, max;
 5
 6    printf("Enter 2 integer :");
 7    scanf("%d %d", &i, &j);
 8    if( i > j ) max = i;
 9    else        max = j;
10    printf("The maximum of %d and %d is %d.\n", i, j, max );
11 }
```

```
Enter 2 integer : 7 5
The maximum of 7 and 5 is 7.
```

◎ 多重選擇題： if-else if 架構

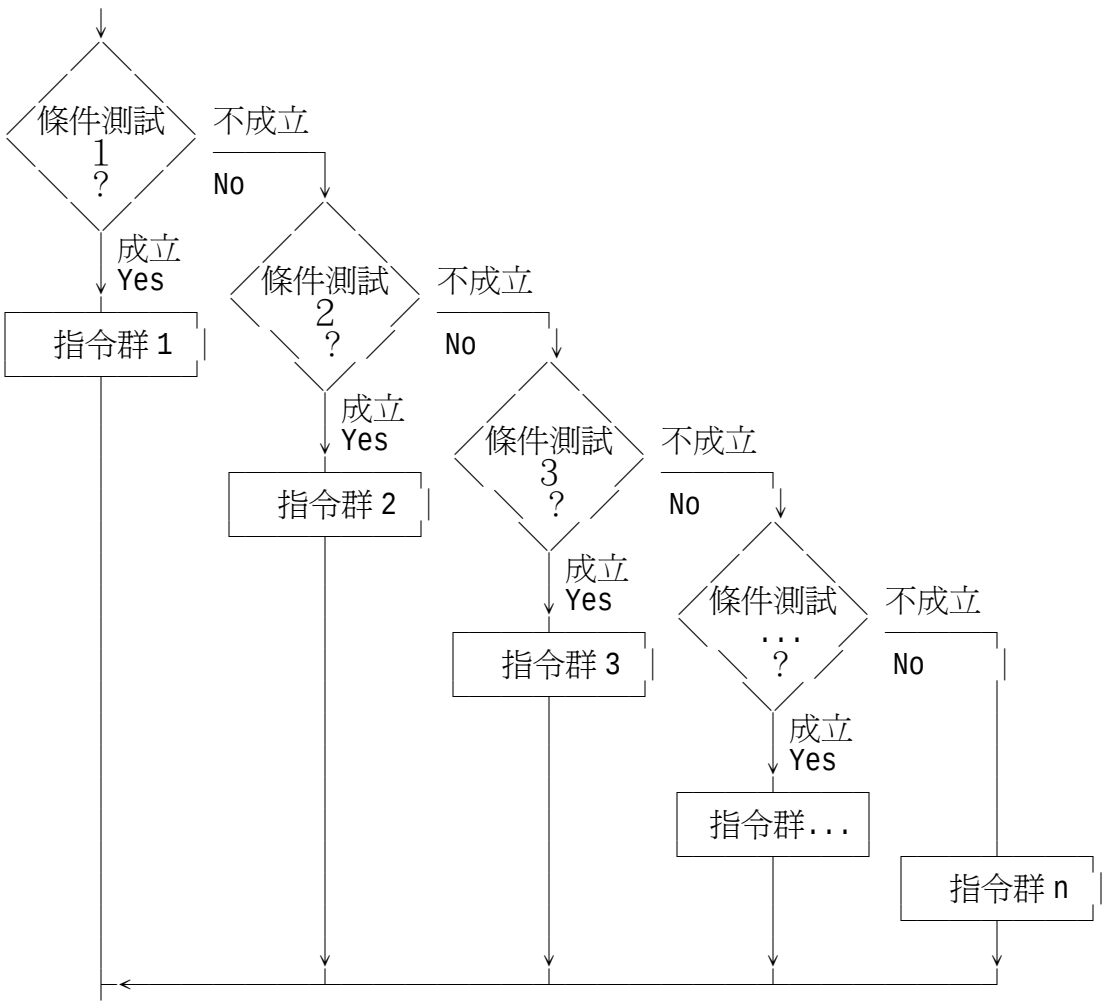
當狀況不只一種時，光是 if-else 就不夠用了。在 C 中，可以在 else 之後再加上 if-else 的架構，如此，就可多一個判斷。同理，可在所加的 else 之後再加上 if-else ，直到所有狀況都包含爲止。
如：如果氣溫小於 20 度，就穿兩件衣服；氣溫小於 15 度，就穿三件衣服；
　　氣溫小於 10 度，就穿四件衣服。

if-else if 架構：

```
if(條件測試 1) { 指令群 1 }
else if(條件測試 2) { 指令群 2 }
else if(條件測試 3) { 指令群 3 }
else if...
else { 指令群 n }
```

意義：

```
如果 (條件測試 1) 是成立的，則執行 { 指令群 1 };
否則，如果 (條件測試 2) 是成立的，則執行 { 指令群 2 };
否則，如果 (條件測試 3) 是成立的，則執行 { 指令群 3 };
否則，如果...
否則，只好執行 { 指令群 n }
```



↓

以下的範例，模擬高速公路交通警察執勤狀況。

police.c

```
 1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
 2  void main(void)
 3  {
 4     int speed;
 5
 6     printf("Enter the speed = ");
 7     scanf("%d", &speed );
 8     if( speed < 60 )
 9        printf("Too slow, speed up!\n");
10     else if( speed < 90 )
11        printf("Good day, boss.\n");
12     else if( speed < 100 )
13        printf("Too fast, slow down!\n");
14     else
15        printf("I will give you a ticket!\n");
16  }
```

```
Enter the speed = 50
Too slow, speed up!
Enter the speed = 80
Good day, boss.
Enter the speed = 95
Too fast, slow down!
Enter the speed = 120
I will give you a ticket!
```

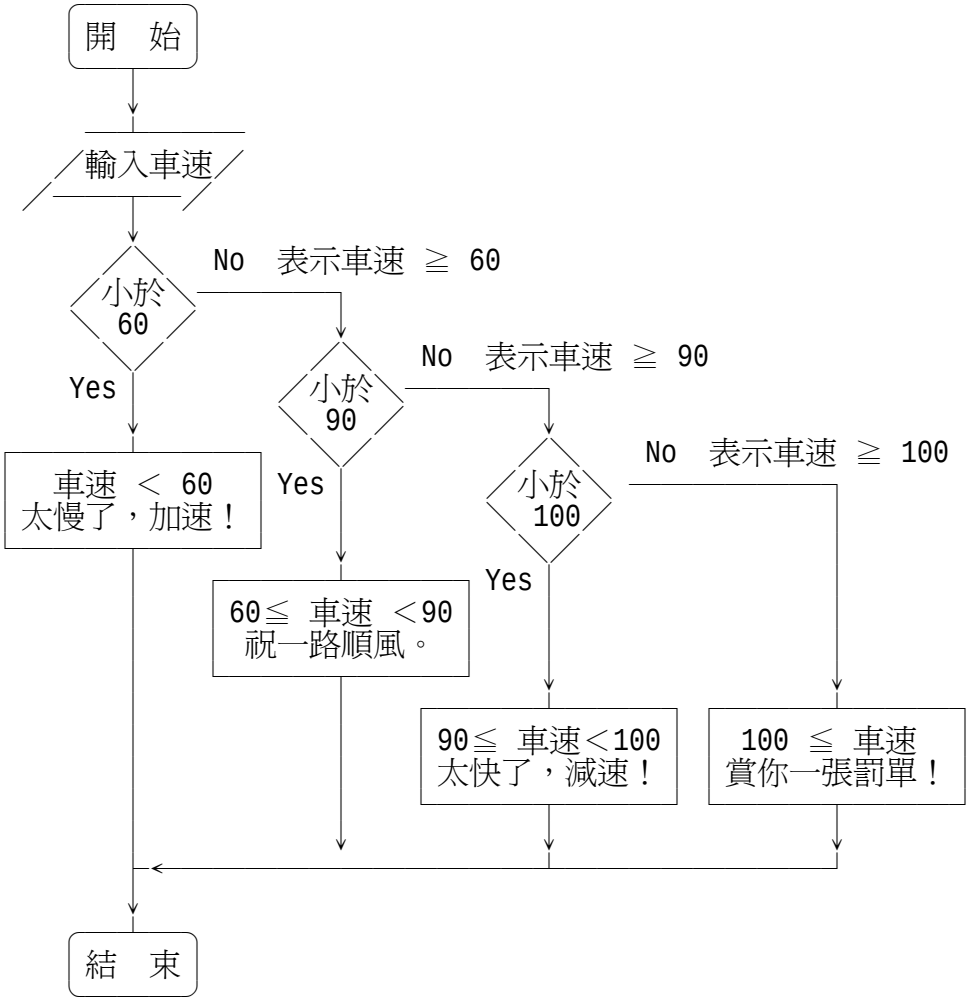
這類的程式，在設計時要注意到條件測試的順序，否則，答案可能會有問題。
如：將第 12 行與第 8 行的判斷式對調，秀出訊息的第 13 行與第 9 行也對調：

```
      if( speed < 100 )
         printf("Too fast, slow down!\n");
      else if( speed < 90 )
         printf("Good day, boss.\n");
      else if( speed < 60 )
         printf("Too slow, speed up!\n");
      else
         printf("I will give you a ticket!\n");
```

這樣的話，程式只會有兩種輸出：
`speed <  100` ，秀出 `Too fast, slow down!`
`speed >= 100` ，秀出 `I will give you a ticket!`

如果你無法確定你寫的程式對不對，那你最好畫畫流程圖來幫助判斷。

此程式的判斷流程如下：



以下是課本的範例：一個簡單的四則運算計算器。

calc.c

```
 1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
 2  void main(void)
 3  {
 4     float num1,num2;
 5     char op;
 6
 7     for(;;)
 8     {
 9        printf("Enter number, operator, number\n");
10        scanf("%f %c %f", &num1, &op, &num2);
11        if( op == '+' )
12           printf("%f + %f = %f\n", num1, num2, num1+num2);
13        else if( op == '-' )
14           printf("%f + %f = %f\n", num1, num2, num1-num2);
15        else if( op == '*' )
16           printf("%f + %f = %f\n", num1, num2, num1*num2);
17        else if( op == '/' )
```

```
18            printf("%f + %f = %f\n", num1, num2, num1/num2);
19        }
20    }
```

```
Enter number, operator, number
4 + 8
4.000000 + 8.000000 = 12.000000
Enter number, operator, number
5 * 7
5.000000 + 7.000000 = 35.000000
^C
```

在這個程式中，會讀取二個數值以及一個運算符號，接著判斷運算符號是否是 + - * / 之中的一個，如果是的話就作相對應的運算，並將結果輸出。
另外，第 7 行使用了下一章才會提到的「迴圈」。這個程式會不斷的執行，你必須按下 Ctrl + Break 或 Ctrl + C 才能中斷它。

☆ 有關「抉擇」☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆

前面所講的抉擇判斷式子都相當簡單，以下介紹三種 C 語言提供的方式，可以讓你的「抉擇」多樣化。

◎ 邏輯運算子

○ && AND

假設有一個運算式 ： exp1 && exp2
則此運算式的結果為真的條件是 exp1 為真 且 exp2 也為真；
否則此運算式的結果為假。

○ || OR

假設有一個運算式 ： exp1 || exp2
則此運算式的結果為真的條件是 exp1 為真 或 exp2 為真；
只有在 exp1 為假 且 exp2 也為假時，此運算式的結果才為假。

○ ! NOT

假設有一個運算式 ： !exp1
則此運算式結果為真的條件是 exp1 為假；
而當 exp1 為真時，此運算式的結果就是假。

◎ 小括號指定敘述

任何用小括號括住的指定敘述是一個運算式，它的值與指定值一樣。
例如：運算式 (sum = 5 + 3) 的值為 8 ，所以運算式
((sum = 5 + 3) <= 10) 結果為真，因為 (8 <= 10) 。

◎ 逗號運算子

你可以利用逗號運算子 (,) 把多個運算式放入括號內。運算式由左至右執行或計算，整個運算式的值以最後一個的值為準。
例如：運算式 (i = 1, j = 2, k = i + j) 的值為 k 也就是 3 ，因為，先設了 i = 1 ，再指定 j = 2 ，最後令 k = i + j ，所以 k 的值為 3 ，因為 k = i + j 是最後一個敘述，所以這個運算式的值就等於 k 也就是 3 。

應用：

yes.c

```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2  #include <conio.h>     /* 宣告 getche() 的原型 */
 3  void main(void)
 4  {
 5     char ch;
 6
 7     printf("Press Y or y to continue xxxxxx...");
 8     if ( ( ch = getche() ) == 'Y' || ch == 'y' )
 9       printf("\nYou press %c.\n", ch),
10       printf("continue xxxxxx...\n");
11     else
12       printf("\nYou press %c.\n", ch),
13       printf("stop xxxxxx!\n");
14  }
```

```
Press Y or y to continue xxxxxx...y
You press y.
continue xxxxxx...
Press Y or y to continue xxxxxx...q
You press q.
stop xxxxxx!
```

在這個程式中，會要求使用者輸入一個字元，如果輸入的字元是 Y 或 y 則秀出 You press y. continue xxxxxx... 類似程式繼續執行某種程序；
如果輸入的不是 Y 也不是 y ，則秀出 You press X. stop xxxxxx! 類似程式停止執行某種程序。

在第 8 行中，

```
if ( ( ch = getche() ) == 'Y'  || ch == 'y' )
```

由鍵盤讀入一個鍵值，並存到 ch
整個小括號的值就等於 ch

所以上式可以簡化爲　if (ch == 'Y' || ch == 'y')
也就是，如果 ch 等於 'Y ' 或 'y' 則此判斷式爲真，否則爲假。

在第 9 行中，　printf("\nYou press %c.\n", ch),
是用逗號 (,) 作爲結尾的，而不是一般以分號 (;) 來結尾，接著的第 10 行，則是以分號 (;) 作結尾，表示第 9 行與第 10 行的指令是一體的。
如果第 9 行要用分號 (;) 作結尾，那就要在第 9 行指令前及第 10 行指令後用大括號 { } 括起來，如：

```
{  printf("\nYou press %c.\n", ch);
   printf("continue xxxxxx...\n");   }
```

同理，在第 12 行與第 13 行也用了相同的方法。

各位也許會發現：第 9 行與第 12 行的程式碼完全相同的！沒錯，你可以想辦法將第 9 行程式碼移到 if 之前，並刪去第 12 行程式碼，這樣程式就可以減小。本例只是用一下逗號運算子，所以才將程式寫成這樣。

在這裡，並不是要各位把程式寫得很難看懂，而是希望各位能看得懂別人所寫的程式，了解一些程式設計師可能的用的「技巧」或「手段」。

以上的程式，我們也可以改寫如下：

yes2.c

```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2  #include <conio.h>     /* 宣告 getche() 的原型 */
 3  void main(void)
 4  {
 5     char ch;
 6
 7     printf("Press Y or y to continue xxxxxx...");
 8     ch = getche();
 9     if ( ch == 'Y' || ch == 'y' )
10     {
11       printf("\nYou press %c.\n", ch);
12       printf("continue xxxxxx...\n");
13     }
14     else
15     {
16       printf("\nYou press %c.\n", ch);
17       printf("stop xxxxxx!\n");
18     }
19  }
```

這樣子寫，雖然程式比較長，但是在 Compile 之後產生的 .exe 檔是一模一樣大的【註】，只是程式碼比較長，卻是比較「好看」。

【註】用 TC 整合環境所 Compile 出來的 .exe 檔中包含了除錯資訊(Debug Information)，這些除錯資訊包含了原始碼的行號資料，因爲 yes.c 與 yes2.c 的行數不同，它們的行號資料就不會相同，所以用 TC 整合環境編譯出來的執行檔大小是不同的。
我們可以用命令列編譯程式 tcc 來編譯 yes.c 及 yes2.c ：

```
DOS_Prompt> tcc yes.c

DOS_Prompt> tcc yes2.c
```

如此，產生的 yes.exe 及 yes2.exe 的檔案大小就是一樣的了。

■ 第六章 迴圈與自動重複

前一章，已經用了 for 迴圈來寫程式，讓程式依需求重複執行某些程式碼。
在 C 中有許多種迴圈，本章就以 for 迴圈爲主，介紹迴圈設計的技巧。

◎ for 敘述

○ for 敘述的格式如下：

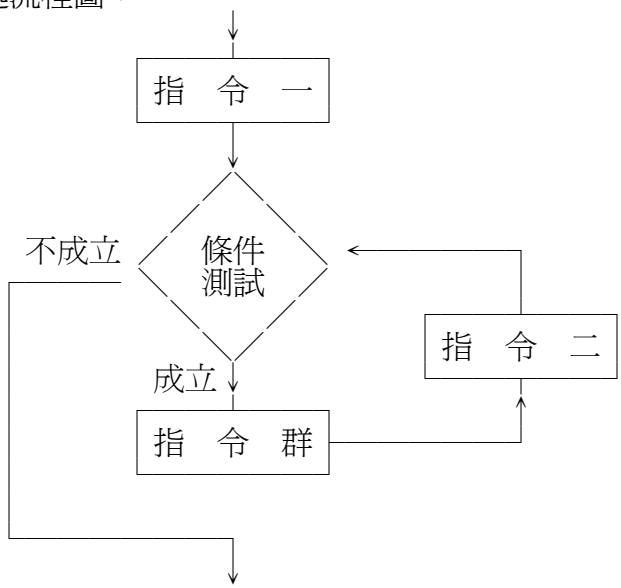
```
for ( 指令一 ; 條件測試 ; 指令二 )
{  指令群  }
```

注意：在 for 後面有對小括號，小括號裡面的東西用分號隔成三個部份，
這二個分隔用的分號絕對不可以省略掉。

○ for 敘述的意義：

1.先執行「指令一」。
2.測試看看「條件測試」是否成立？如果成立，則執行
{ 指令群 }；否則，結束自動重複的動作。
3.在執行完 { 指令群 } 後，執行「指令二」，然後回到 2。

○ for 敘述流程圖：



在這裡，指令一及指令二都是單一指令，如果你要作多項設定，那就要用到
上一章最後所講的「逗號運算子」，可以將多個指令寫在一起。

○ 常見 for 敘述的用法：

```
for ( i = 0  ; i < 100 ; i = i+1 )
   printf("i = %3d\n", i );
```

```
for ( i = 0 , j = 0 ; i < 100 ; i = i+1 , j = j+2 )
   printf("i = %3d j = %3d\n", i, j );
```

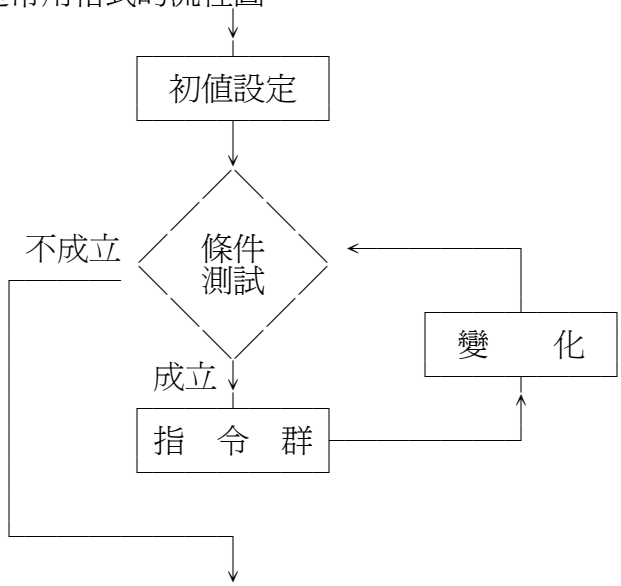
○ for 敘述常用的格式：

```
for ( 初值設定 ; 條件測試 ; 變化 )
{  指令群  }
```

○ for 敘述常用格式的意義：

```
1.首先依「初值設定」的要求，設定你所指定變數的值。
2.測試看看「條件測試」是否成立？如果成立，則執行
  { 指令群 }；否則，結束自動重複的動作。
3.在執行完 { 指令群 } 後，按照「變化」的要求，改變指定
  變數的值，然後回到 2 。
```

○ for 敘述常用格式的流程圖：



○ 省略的 for 敘述：

```
for ( 指令一 ; 條件測試 ; 指令二 )
{  指令群  }
```

在 for 敘述中，指令一、條件測試、指令二以及指令群這四者都可以省略，

△ 省略指令一：以常用格式來說，就是不做初值設定。
　　初值可在 for 之前設定，或者不用做初值設定。

△ 省略條件測試：表示不做條件測試，重複執行 { 指令群 }。
　　我們通常稱這種迴圈爲「無窮迴圈」。

△ 省略指令二：以常用格式來說，就是不做變數數值的改變。
　　變數數值可以在指令群中改變。

△ 省略指令群：表示空迴圈，可用於時間延遲(delay)。
　　若連大括號都要省略，則要補上分號 (;) 表示空指令。

for 敘述括號內的指令是可以省的，但是分隔用的分號 (;) 則不能省。

以下是課本的範例：秀出 ASCII 碼及相對應的字元。

ascii.c

```
1  #include <stdio.h>     /* 宣告 printf() 的原型 */
2  void main(void)
3  {
4     int i;
5
6     for( i = 32 ; i < 256 ; i = i+1 )
7        printf("%3d=%c\t", i, i );
8  }
```

```
 32=     33=!    34="    35=#    36=$    37=%    38=&    39='   ...
 42=*    43=+    44=,    45=-    46=.    47=/    48=0    49=1   ...
 52=4    53=5    54=6    55=7    56=8    57=9    58=:    59=;   ...
...                                                             ...
252=*   253=*   254=*   255=
```

```
for( i = 32 ;  i < 256 ; i = i+1 )
    ========  ========  =========
    初值設定   條件測試     變化
```

在 C 語言中，我們通常會把 `i = i + 1` 這種變數遞增寫成 `i++` 。
同樣的，變數遞減寫成 `i--` 表示 `i = i - 1` 。
如果遞增或遞減的數值不是 1 ，則寫成 `i += 2` 表示 `i = i + 2`
或 `i -= 2` 表示 `i = i - 2` 。
注意，在這裡的 `+=` 或 `-=` 不可以分開。 同樣的，乘除法也有相同的用法，
如 `i *= 2` 表示 `i = i * 2` 、 `i /= 2` 表示 `i = i / 2` 。

以下的範例用來計算 n! 的值。

n!.c

```
1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
2  void main(void)
3  {
4     long fact;
5     int n;
6
7     printf("Enter the value of n to compute n! : ");
8     scanf("%d", &n );
9     printf("%d! = %d", n, n );
10    fact = n;
11    for( n = n-1 ; n >0 ; n-- )
12    {
13       fact *= n;
14       printf("x%d", n);
15    }
16    printf(" = %ld", fact);
17 }
```

```
Enter the value of n to compute n! : 5
n! = 5x4x3x2x1 = 120
```

n 階乘的定義是 `n! = n * (n-1) * (n-2) * ... * 2 * 1`
在上面的程式中以 fact 這個變數來存 n 階乘的數值。此例來說，輸入的數值為 5 ，所以第 9 行的 n 為 5 。第 10 行 `fact = n;` 設定 fact = 5 。
再來進入迴圈，先設定 n 為 n-1 也就是 4 ，因為 n 階乘下一個是要乘 n-1 ，

第 13 行 fact *= n 也就是 fact = fact * n ，因爲每次迴圈 n 的值都會減 1 ，如此，就達到階乘的效果。以下是 fact 及 n 在每次迴圈的數值：

時　刻	fact 值	n 值	印　出　效　果
進入迴圈前	5	5	5! = 5
n 初值設定	5	4	5! = 5
第 1 圈後	5*4	4	5! = 5x4
第 2 圈後	5*4*3	3	5! = 5x4x3
第 3 圈後	5*4*3*2	2	5! = 5x4x3x2
第 4 圈後	5*4*3*2*1	1	5! = 5x4x3x2x1
離開迴圈後	5*4*3*2*1	1	5! = 5x4x3x2x1 = 120

◎ 重複中有重複

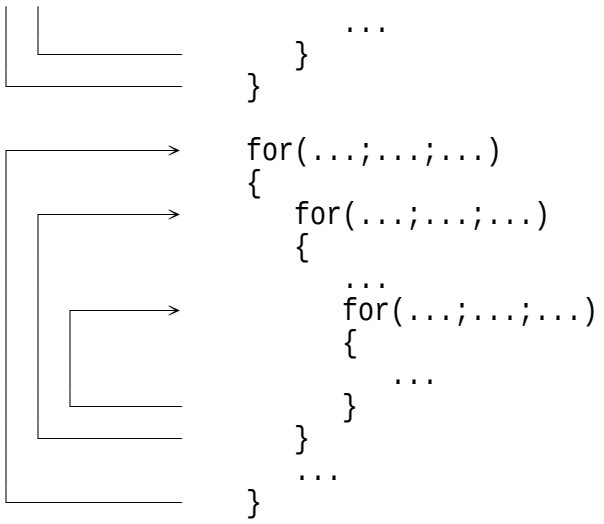
在 for 敘述所要重複執行的指令群中，也可以再含有另一個 for 敘述，於是便形成了重複中有重複的情形。一般我們稱 for 敘述及所重複的指令群爲 for 迴圈，而這種重複中有重複的情形，便稱爲巢狀(nest)迴圈。（課本稱爲套疊式迴圈）
在使用巢狀迴圈時，最好使用縮排的編輯技巧，讓你的程式更好看：

```
敘述;
...
for( ... ; ... ; ... )
{
   敘述;
   ...
   for( ... ; ... ; ... )
   {
      敘述;
      ...
   }
   敘述;
   ...
}
```

同樣的技巧可以用在巢狀 if 敘述，或其他類似的敘述。

巢狀迴圈可以多層，但是，各個迴圈之間絕對不可以交叉。
以下是正確的巢狀迴圈：

```
for(...;...;...)
{
   for(...;...;...)
   {
      ...
   }
   ...
   for(...;...;...)
   {
```



```
            ...
        }
    }

    for(...;...;...)
    {
        for(...;...;...)
        {
            ...
            for(...;...;...)
            {
                ...
            }
        }
        ...
    }
```

以下是不正確的巢狀迴圈：

以下的範例：印出九九乘法表

99.c

```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2  void main(void)
 3  {
 4     int i, j;
 5
 6     for( i=1 ; i<=9 ; i++ )
 7     {
 8        for( j=1 ; j<=9 ; j++ )
 9           printf("%dx%d=%2d ", j, i, j*i );
10        printf("\n");
11     }
12  }
```

```
1x1= 1 2x1= 2 3x1= 3 4x1= 4 5x1= 5 6x1= 6 7x1= 7 8x1= 8 9x1= 9
1x2= 2 2x2= 4 3x2= 6 4x2= 8 5x2=10 6x2=12 7x2=14 8x2=16 9x2=18
1x3= 3 2x3= 6 3x3= 9 4x3=12 5x3=15 6x3=18 7x3=21 8x3=24 9x3=27
1x4= 4 2x4= 8 3x4=12 4x4=16 5x4=20 6x4=24 7x4=28 8x4=32 9x4=36
1x5= 5 2x5=10 3x5=15 4x5=20 5x5=25 6x5=30 7x5=35 8x5=40 9x5=45
1x6= 6 2x6=12 3x6=18 4x6=24 5x6=30 6x6=36 7x6=42 8x6=48 9x6=54
1x7= 7 2x7=14 3x7=21 4x7=28 5x7=35 6x7=42 7x7=49 8x7=56 9x7=63
1x8= 8 2x8=16 3x8=24 4x8=32 5x8=40 6x8=48 7x8=56 8x8=64 9x8=72
1x9= 9 2x9=18 3x9=27 4x9=36 5x9=45 6x9=54 7x9=63 8x9=72 9x9=81
```

九九乘法表是 1 到 9 的兩個數相乘，一個是乘數，一個是被乘數，因此，

我們需要兩個迴圈， i=1　要乘以 1 到 9 、 i=2 也要乘以 1 到 9 、...，所以要用巢狀迴圈的方式來處理這個問題。
第 9 行的 printf("%dx%d=%2d ", j, i, j*i); 是屬於內層迴圈，也就是在某一個 i 值， j 要由 1 變化到 9 。
第 10 行的 printf("\n"); 是屬於外層迴圈，在內層迴圈執行完之後，就跳下一行。

◎ 跳出迴圈： break 敘述

有時，我們可能在重複執行到一半時，檢測到某個情況發生，而希望提早結束重複的動作，便可以利用 break 敘述。 break 敘述的功能相當單純，它用來跳出一層的迴圈。注意！它只能跳出一層迴圈。如果你使用兩層的巢狀迴圈，那麼在內層的 break 只會跳出內層迴圈，在外層的 break 才會跳出外層迴圈。

```
for(...;...;...)            /* 外層迴圈 開始 */
{
   for(...;...;...)         /* 內層迴圈 開始 */
   {
      ...
      if(...) break;        /* 跳出內層迴圈 */
      ...
   }                        /* 內層迴圈 結束 */
   ...
   if(...) break;           /* 跳出外層迴圈 */
   ...
}                           /* 外層迴圈 結束 */
```

○ Ctrl + C　或　Ctrl + Break　　中斷按鍵

當程式進入無窮迴圈時，在某些狀況下，你可以按下 Ctrl + C 或 Ctrl + Break 來中斷程式的執行，但是在某些狀況下，即使你按下 Ctrl + C 或 Ctrl + Break 也沒有用，只有重開機才能中斷程式執行。所以，在程式中最好能正常結束，例如：使用者按下 q 鍵或 x 鍵表示結束程式等等，若你要使用無窮迴圈，最好能確定用 Ctrl + C 或 Ctrl + Break 可以中斷程式。

☆ 其他的迴圈 ☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆

除了 for 以外， C 語言還有 while 迴圈及 do ... while 迴圈。

◎ while 迴圈

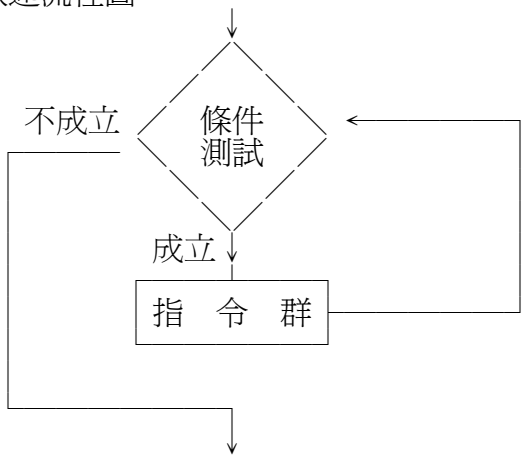
○ while 敘述的格式如下：

```
while ( 條件測試 )
{  指令群  }
```

○ while 敘述的意義：

1.測試看看「條件測試」是否成立？
　如果成立，則執行{ 指令群 }；否則，結束自動重複的動作。
2.執行完{ 指令群 }後，再回到 1 。

○ while 敘述流程圖：



我們可以把 while 迴圈用 for 迴圈來表示：

```
while( i < 10 )
{
   ...
}
```

```
for(  ; i<10 ; )
{
   ...
}
```

◎ do ... while 迴圈

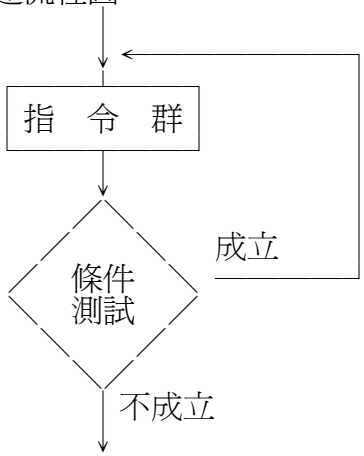
○ do ... while 敘述的格式如下：

```
do
{
   指令群
} while ( 條件測試 );
```

○ do ... while 敘述的意義：

1.先執行{ 指令群 }。
2.測試看看「條件測試」是否成立？
  如果成立，則執行{ 指令群 }；否則，結束自動重複的動作。
3.執行完{ 指令群 }後，再回到 2 。

○ do ... while 敘述流程圖：



while 迴圈與 do ... while 迴圈最大的不同點在於：
while 迴圈要先條件測試，成立才執行指令群，而 do ... while 迴圈則是先執行指令群，才條件測試，成立再一次執行指令群。

■ 第七章 陣列與指標

陣列是由一群相同型態的變數所組成，當你需要一堆相同型態的變數時，用陣列是最適合的。例如，要儲存全班 50 位同學的成績，那就需要 50 個變數，如果要一個一個宣告，那就太沒有效率了。這時就需要用陣列了。指標則是指向變數或陣列位址的運算子，任一變數或陣列的數值均可經由指標的運算獲得。

◎ 陣列的宣告

陣列宣告的格式如下：

```
變數型別  變數名稱[元素個數];
```

以儲存全班 50 位同學的成績爲例，用整數來存成績：

```
int  score[50];
```

C 語言中，陣列的索引值是由 0 開始的，所以這 50 個變數爲：
score[0], score[1], score[2], ..., score[48], score[49] 。

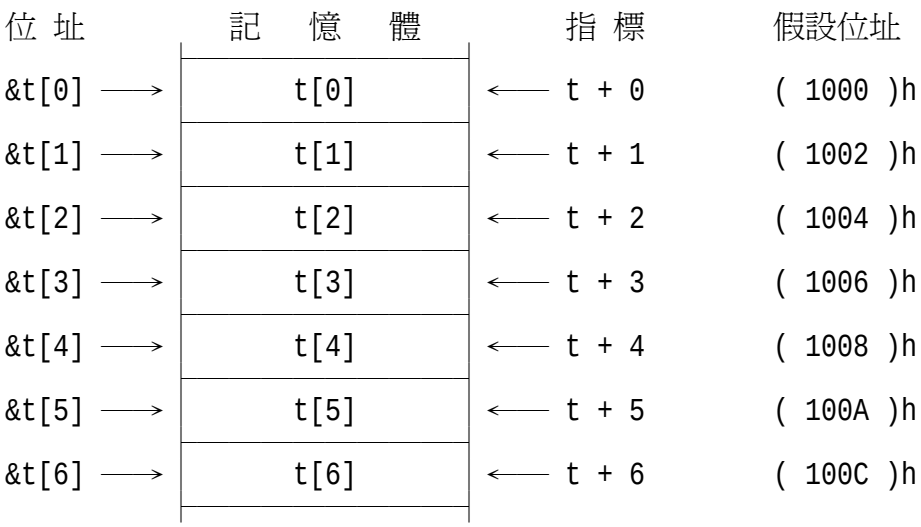
以下的範例：讀取一週的氣溫，再求出其平均值。

temper.c

```
 1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
 2  void main(void)
 3  {
 4     int t[7];
 5     int day, sum;
 6
 7     for( day = 0 ; day < 7 ; day++ )
 8     {
 9        printf("Enter the temperature for day %d : ", day+1 );
10        scanf("%d", &t[day]);
11     }
12
13     sum = 0;
14     for( day = 0 ; day < 7 ; day++ )
15        sum = sum + t[day];
16
17     printf("Average temperature = %f\n", sum/7. );
18  }
```

```
Enter the temperature for day 1 : 22
Enter the temperature for day 2 : 20
Enter the temperature for day 3 : 18
Enter the temperature for day 4 : 19
Enter the temperature for day 5 : 23
Enter the temperature for day 6 : 24
Enter the temperature for day 7 : 26
Average temperature = 21.714286
```

在第四章中，我們曾經列出各種變數型態的指標，以上面的例子：



我們利用第 10 行 scanf("%d", &t[day]); 讀取資料，這裡用的是每一個陣列元素的位址 &t[day] ，day 由 0 到 6 。
我們也可以改用指標的方式： scanf("%d", t+day);
這裡的 t 是陣列變數的名稱，也就一個指標常數(constant pointer)，在宣告陣列變數的同時，就宣告了這個指標常數，只是這個指標常數的數值不是由你決定的，而是在程式執行時，它的數值才會確定，一但確定，就不會改變。
t + day 的 day 是 t 指標的增加值，假設 t 爲 (1000)h，
那 t + 1 是 (1001)h 嗎？
C 語言在作指標運算時，會依指標的型別不同，而改變遞增或遞減的數值。
以 t + 1 來說， t 是一個整數的指標常數，在 TC2 中 一個整數占有 2 bytes，所以 t + 1 的數值是 (1002)h ，而不是 (1001)h 。
同理，t + 2 的數值是 (1004)h 。

在上例第 17 行， printf("Average temperature = %f\n", sum/7.);
不知道你有沒有注意到 sum/7. 在 7 後面的一點 (.) ？
這並不是 Keyin 錯誤，而是故意的。因爲 sum 是一個整數變數，
若寫成 sum/7 表示整數除以整數，那結果還是整數。故意寫成 sum/7.
就變成整數除以實數，那結果會是實數。即然是平均值，通常會有小數，所以我們用 %f 來秀出資料。

◎ 字串

字串是一堆字元所組成的，它是一個字元陣列，只是字串它還要有個 '\0' 字元，作爲字串結束記號。例如，我們要用一個字串存 DOS 的檔名：
DOS 的檔名是主檔名最多 8 個字元，副檔名最多 3 個字元，中間以句號 (.) 分隔，所以需要 8 + 3 + 1 = 12 個字元來存檔名。但是別忘了字串要有 '\0' 的結束字元，所以，總共需要 12 + 1 = 13 個字元：

```
char filename[13];
```

你可以用以下的程式片斷來詢問檔名：

```
printf("Enter the file name : ");
scanf("%12s", filename );
```

scanf 是要變數的位址，而 filename[13] 這個變數的位址就是 filename ，你也可以用 &filename[0] 。另外，用 %12s 這個控制碼，表示讀入的字串只取前面 12 個字元再加上 '\0' 字元，傳給後面所指定的位址。如果你不加， 12 而用 %s 的話，當使用者輸入超過 13 個字元時，將會發生不可預期的後果。

以下的範例：讀取一個字串並作輸出。

string.c

```
1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
2  void main(void)
3  {
4    char name[13];
5
6    printf("Enter your name please : ");
7    scanf("%12s", name );
8    printf("Good day, Mr. %s.", name );
9  }
```

```
Enter your name please : Lee
Good day, Mr. Lee.
```

你可以將第 7 行的 %12s 改爲 %s ，並且在執行程式時，輸入一個較長的字串，看看會發生什麼事？

◎ 二維及多維陣列

C 語言除了一維陣列外，也可以依需要宣告二維或以上的變數陣列：

變數型別　變數名稱[第一維元素個數][第二維元素個數];

變數型別　變數名稱[第一維元素個數][第二維元素個數][第三維元素個數];

不過會用到三維陣列以上的機會蠻少的。
以計算學生成績爲例，用一維陣列　int score[50];　我們只能記錄一科的成績，用二維陣列　int score[50][7];　就能記錄七科的成績，或者是記錄單一科目的期中考、期末考、平常成績、作業成績等等。

以下是課本的範例(加上範圍判斷)：秀出使用者所指定座標的位置。

demo.c

```
1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
2  void main(void)
3  {
4     char matrix[5][10];
5     int x,y;
6
7     for( y=0 ; y<5 ; y++ )          /* 設定陣列初值 */
8        for( x=0 ; x<10 ; x++)
9           matrix[y][x] = '.' ;
10
11    printf("Enter the coordinate(0 0) - (9 4) : ");
12    scanf("%d %d", &x, &y );        /* 讀取指定座標 */
13
14    for( ; !( ( x>=0 || x<=9 ) && ( y>=0 || y<=4 ) ) ; )
15    {
16       printf("\aInvalid Value!!\n");
17       printf("Please enter the coordinate(0 0) - (9 4) : ");
```

```
18        scanf("%d %d", &x, &y );      /* 讀取指定座標 */
19     }
20     matrix[y][x] = '*' ;             /* 設定指定座標 */
21
22     for( y=0 ; y<5 ; y++ )           /* 秀出陣列值 */
23     {
24        for( x=0 ; x<10 ; x++)
25           printf("%c", matrix[y][x] );
26        printf("\n");
27     }
28  }
```

```
Enter the coordinate(0 0) - (9 4) :  5  3
..........
..........
..........
.....*....          ← 因為索引值是由 0 開始所以 * 在 6,4
..........
```

第 14 行到第 19 行是利用一個 for 迴圈來判斷輸入的數值是否在要求的範圍。在使用陣列時，最好要注意一下使用索引值的範圍，因為 C 不會做陣列邊界檢查，當你使用大於你所宣告的索引值時，在語法上，並不會有任何的錯誤。但是當你使用索引值超過你所宣告的的陣列時，你有可能改變到別的變數，甚至是程式碼。輕微的話，只是程式結果錯誤，嚴重的話，可能導致當機。

☆ 兩維陣列的指標 ☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆

以 int s[50][3]; 為例

位 址	記 憶 體	指 標			假設狀況
&s[0][0] →	s[0][0]	← *(s + 0) + 0	← s + 0	← s	(1000)h
&s[0][1] →	s[0][1]	← *(s + 0) + 1			(1002)h
&s[0][2] →	s[0][2]	← *(s + 0) + 2			(1004)h
&s[1][0] →	s[1][0]	← *(s + 1) + 0	← s + 1		(1006)h
&s[1][1] →	s[1][1]	← *(s + 1) + 1			(1008)h
&s[1][2] →	s[1][2]	← *(s + 1) + 2			(100A)h
&s[2][0] →	s[2][0]	← *(s + 2) + 0	← s + 2		(100C)h
	... ...				
&s[48][2] →	s[48][2]	← *(s +48) + 2	(30h*3+2)*2		(1124)h
&s[49][0] →	s[49][0]	← *(s +49) + 0	← s +49		(1126)h
&s[49][1] →	s[49][1]	← *(s +49) + 1			(1128)h
&s[49][2] →	s[49][2]	← *(s +49) + 2			(112A)h

s 是一個二維指標常數。
一個一維指標，我們必須在指標前加一個星號 (*) 才能取得一維陣列內的變數值，同樣地，在二維指標，我們必須在指標前加二個星號，才能取得二維陣列內的變數值。若是二維指標前的星號少於二個，例如，沒有星號或只有一個，那它所表示的就還是一個指標。另外，二維指標前如果有一個星號，那就相當是降了一維，而成爲一個一維指標。

○ 二維指標的遞增值

一個一維指標，如果它所指的變數型態占有 n 個 byte ，那這個一維指標的遞增值就是 n 。
一個二維陣列： vartype Array[X][Y]; 如果 vartype 占有 n 個 byte，那 Array 這個二維指標的遞增值就是 Y * n ，也就是 Y 個 vartype 所占的空間。
Array 是指向二維陣列的開端 &Array[0][0]，而 Array+1 正好指向 &Array[1][0] ，同理 Array+i 是指向 &Array[i][0]。

如前所述，在二維指標前加一個星號就會變成一維指標。所以， *Array 是個一維指標、*(Array+1) 是一維指標、...、*(Array+i) 是一維指標、...、*(Array+(X-1)) 是一維指標，而這些一維指標都是指向第二維陣列開始的位址。
其中的 0≦ i ≦(X-1)，如同決定 Array[X][Y] 中第一維(X)的位移量(Offset)。
這些一維指標的遞增值 *(Array+i)、...、*(Array+i)+j、...、*(Array+i)+(Y-1) 就指向第二維的每一個變數。
其中的 0≦ j ≦(Y-1)，如同決定 Array[X][Y] 中第二維(Y)的位移量(Offset)。
所以， *(Array + i) + j 就是指向 Array[i][j] 這個變數，同時
*(*(Array + i) + j) 就是 Array[i][j] 這個變數。

★ C 語言不會作陣列邊界(Boundary)檢查，例如，你宣告 int s[50][7];
可是你可以使用 s[0][8]、s[51][2] 或 s[100][200]。

○ 用指標的方式，修改上例，你看懂了嗎？

demoptr.c

```
 1  #include <stdio.h>     /* 宣告 printf(),scanf() 的原型 */
 2  void main(void)
 3  {
 4     char matrix[5][10];
 5     int i,x,y;
 6
 7     for( i=0 ; i<5*10 ; i++ )       /* 設定陣列初值 */
 8         *((*matrix)+i) = '.' ;
 9
10     printf("Enter the coordinate(0 0) - (9 4) : ");
11     scanf("%d %d", &x, &y );        /* 讀取指定座標 */
12
13     for( ; !( ( x>=0 || x<=9 ) && ( y>=0 || y<=4 ) ) ; )
14     {
15        printf("\aInvalid Value!!\n");
16        printf("Please enter the coordinate(0 0) - (9 4) : ");
17        scanf("%d %d", &x, &y );     /* 讀取指定座標 */
18     }
19     matrix[y][x] = '*' ;            /* 設定指定座標 */
20
21     for( y=0 ; y<5 ; y++ )          /* 秀出陣列值 */
22     {
23        for( x=0 ; x<10 ; x++)
```

```
24            printf("%c", *( *(matrix + y) + x) );
25         printf("\n");
26      }
27   }
```

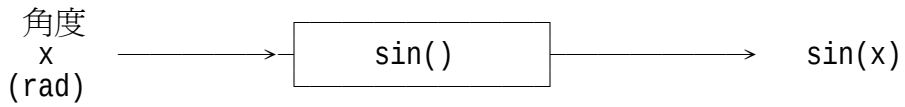
■ 第八章 函數與呼叫

C 語言的程式，是由一堆函數(Function)或函式(Routine)所組成的。

○ 什麼是函數？以 sin 這個函數爲例：

sin: sine function	正弦函數。
double sin(double x);	sin 的語法
Prototype in math.h	必須 #include<math.h>
x is in radians. Returns a value in the range -1 to 1.	x 的單位是徑度(rad) 傳回數值在 -1 到 1 之間。

A = sin(M_PI/2); 我們都知道 sin(π/2)=1.0 ，所以這一行敘述相當於 A=1.0;
我們可以把 sin() 當成一個黑箱子，給它角度(rad)，它會輸出相對的 sine 值。



在 C 語言中，函數的輸入可以多個，例如： pow(X, Y) 可以計算 X 的 Y 次方。

pow: power function, x to the y	乘冪函數，X 的 Y 次方。
double pow(double x, double y);	pow 的語法
Prototype in math.h	必須 #include<math.h>

但是函數的輸出值，也就是回傳值，只能有一個。

○ 什麼是函式？函式的英文爲 routine ，原意是「例行公事」，也就是一些一成不變做完甲這個事後，就做乙，再做丙...。
到現在爲止，我們常用的 printf() 、 scanf() 都算是函式，而且在每一個程式也都用建立了 main() 這個函式。通常稱 main() 爲主函式，而其他的都稱做副函式(sub-routine)。
在電腦語言中，函數、函式已經混在一起了，像教科書中的作者用函數，而筆者則是用慣了函式。

◎ 函式的宣告與定義

函式宣告與定義的格式如下：

```
傳回值型別  函式名稱(型別 參數1, 型別 參數2, ...)
{
    函式主體——包含宣告函式內部變數及指令群
}
```

其中，第一列 傳回值型別 函式名稱(型別 參數1, 型別 參數2, ...)
我們稱之爲函式的宣告，而大括號 { } 內則是函式的定義。

傳回值型別，表示函式傳回數值的資料型別，如果省略，則 C 語言會視爲是內定的整數型別。在函式中，我們可以用 return(回傳數值); 讓函式傳回數值，並且

結束函式的執行。
參數(或稱引數)可以使函式具有變化性，如果省略，則表示此函式不需要參數。

○ 養成好習慣

如果省略傳回值型別，等於定義了整數型別，那要是我們所設計的函式沒有傳回數值，應該怎麼辦呢？在 C 語言中，有一種資料型別叫 void ，void 表示「空」的資料型態，所以

```
void 函式名稱(型別 參數 1, 型別 參數 2, ...)
{
   ...
}
```

就表示這個函式沒有傳回任何數值。
同樣地，如果設計的函式不用參數，也可以用 void 來表示：

```
傳回值型別  函式名稱( void )
{
   ...
}
```

就表示這個函式不需要任何參數。

◎ 呼叫函式

C 語言在使用變數或是函式之前，都必須先宣告所使用的變數或是函式。
這也是爲什麼，我們所寫的程式都要 #include <stdio.h> ，因爲，
在 stdio.h 中，宣告了 printf() 、 scanf() 等函式的原型(prototype)。
同樣地，在使用自己所定義的函式前，也要先宣告。
前一節說過，在定義函式的同時，也做了宣告。如此，我們可以將自己寫的函式放在 main() 的前面，再由 main() 來呼叫。如同課本的範例：

以下是課本的範例：讓電腦發出「嗶」！

beep.c

```
 1 #include <stdio.h>     /* 宣告 printf() 的原型 */
 2 #include <conio.h>     /* 宣告 getche() 的原型 */
 3
 4 void beep(void)        /* 副函式 beep() 的宣告 */
 5 {                      /* 開始定義 beep() */
 6    printf("\a");       /* 「嗶」一聲 */
 7 }                      /* beep() 定義結束 */
 8
 9 void main(void)        /* 主函式 main() 的宣告 */
10 {                      /* 開始定義 main() */
11    beep();                                  /* 呼叫 beep() */
12    printf("Press any key to continue..."); /* 呼叫 printf() */
13    getche();                                /* 呼叫 getche() */
14    beep();                                  /* 呼叫 beep() */
15 }                      /* main() 定義結束 */
```

```
「嗶」！一聲
Press any key to continue...
「嗶」！一聲
```

void 既然是「空」的資料型態，所以在呼叫時就是「空的」。
如果，你還記得的話，getche() 會傳回使用者所按下的鍵值，但是在這裡，我們只是要使用者按下任意鍵，所以讀入的鍵值是多少我們並不在意，可以不管它，就如同呼叫不會傳回數值的 beep() 函式一樣。printf() 也是有傳回值的，它傳回輸出的 byte 數，而這個數值，我們通常也不會在意。

以下是課本的範例：以條形圖來比較數值的大小。

bar.c

```
 1  #include <stdio.h>      /* 宣告 printf() 的原型 */
 2
 3  void bar(int i)         /* 副函式 bar() 的宣告 */
 4  {                       /* 開始定義 bar() */
 5     int j;
 6
 7     for( j=0 ; j<i ; j++)  /* 秀出 i 個星號 */
 8        printf("*");
 9     printf("\n");
10  }                       /* bar() 定義結束 */
11
12  void main(void)
13  {
14     printf("Merry\t");
15     bar(30);                /* 輸入參數爲 30 */
16     printf("John\t");
17     bar(40);                /* 輸入參數爲 40 */
18     printf("Johnson\t");
19     bar(20);                /* 輸入參數爲 20 */
20     printf("Sposh\t");
21     bar(50);                /* 輸入參數爲 50 */
22  }
```

```
Merry   ******************************
John    ****************************************
Johnson ********************
Sposh   **************************************************
```

各位可能會發現，課本在宣告 bar() 函式時，好像跟老師所用的不一樣：

```
bar(i)
int i;          /* 宣告參數的資料型別 */
{
   ...
}
```

這種宣告參數的方式，是舊的 C 語言格式，目前的 TC 仍然可以使用。

在呼叫有參數的函式時，給定的參數會代入函式中相對的變數。
如： bar(30); 就是給定 30 作爲輸入參數，此例來說，就是先設定 i = 30 ，再開始執行第 5 行以後的指令。

以下是課本的範例：計算圓面積的副函式。

area.c

```
 1  #include <stdio.h>     /* 宣告 printf(), scanf() 的原型 */
 2
 3  float areas(float r)
 4  {                                /*              2   */
 5     return( 3.14159 * r * r );    /* 圓面積 = π r     */
 6  }
 7
 8  void main(void)
 9  {
10     float radius;
11
12     printf("Enter the radius = ");
13     scanf("%f", &radius);
14     printf("Area of this circle = %f\n", areas( radius ) );
15  }
```

```
Enter the radius = 6
Area of this circle = 113.097240
```

在第 14 行中， areas(radius) 表示以 radius 爲參數呼叫 areas() 函式，areas() 會傳回一個 float 型別的數值。我們可以把 areas(radius) 整個當成一個變數來看，而它的數值與 radius 有關。
例如，我們要計算兩個半徑分別爲 r1 及 r2 的圓面積總和：

```
totalarea = areas( r1 ) + areas( r2 ) ;
```

以下是課本的範例：求任意二個整數的最大值。

max.c

```
 1  #include <stdio.h>     /* 宣告 printf(), scanf() 的原型 */
 2
 3  int max(int i, int j)
 4  {
 5     if( i > j ) return i;     /* 若 i > j 則傳回較大的 i 值 */
 6     else        return j;     /* 否則是 j 比較大，就傳回 j 值 */
 7  }
 8
 9  void main(void)
10  {
11     int i, j;
12
13     printf("Enter 2 integers : ");
14     scanf("%d %d", &i, &j);
15     printf("Maximum of %d and %d is %d.\n", i, j, max(i,j) );
16  }
```

```
Enter 2 integers : 6 8
Maximum of 6 and 8 is 8.
```

事實上， TC2 有提供 max() 這個函式，你可以在 TC 的整合環境下的編輯視窗內輸入 max ，將游標移到 max 字上，按下 Ctrl+F1 就可以看到說明

```
Help
Macros: max, min
  These macros generate inline code to find
  the maximum or minimum value of two
  integers.

  max(a,b)  maximum of two integers a and b
  min(a,b)  minimum of two integers a and b

Defined in stdlib.h
```

巨集指令：max, min
以巨集的方式求得兩數的最大值或最小值。

求 a,b 的最大值
求 a,b 的最小值

必須 #include<stdlib.h>

由上可知， max() 是定義在 stdlib.h 內，所以我們可以改寫上例如下：

max1.c

```
 1  #include <stdio.h>     /* 宣告 printf(), scanf() 的原型 */
 2  #include <stdlib.h>    /* 定義 max() 的原型 */
 3
 4  void main(void)
 5  {
 6     int i, j;
 7
 8     printf("Enter 2 integers : ");
 9     scanf("%d %d", &i, &j);
10     printf("Maximum of %d and %d is %d.\n", i, j, max(i,j) );
11  }
```

```
Enter 2 integers : 6 8
Maximum of 6 and 8 is 8.
```

在這裡只是要告訴各位，TC 它有提供非常多的內建函式，有一些功能我們並不須要再去設計一次。如果要作練習，那就另當別論。建議有意要寫程式的人，多多參考「參考手冊」，大略知道所使用的語言提供了那些內建函式，要用時，只要依照它的格式來呼叫它就可以了。要不然，你可以花了許多時間在寫一個內建函式，而知道的人，只要 #include ??.h 就可以直接使用它。

◎ 整體變數(Global Variable)

在前面的例子中，所有的變數都只能在所宣告的函式內使用，函式與函式之間只能用傳回值或是參數來傳遞數值。除了用傳回值與參數外，在 C 語言中還可以使用「整體變數」。
什麼是「整體變數」？由字面上來看，就是適用於程式整體，都可以使用的變數。相對於整體變數的就是「區域變數(Local Variable)」。同樣地，由字面上來看，就是只適用於程式的某一個區域所能使用的變數。

我們要如何判斷變數可以使用的範圍呢？以下就來談談變數的生命週期。
以一個副函式來說：

```
void sub1(void)
{
   int i;      <------- 變數 i 的「生」
```

```
    ...
}     <———————————— 變數 i 的「滅」
```

我們所寫的函式，都會用一對大括號 { ... } 括起來，而在大括號內宣告的變數，就適用於大括號內。當這個函式執行結束時，在大括號內所宣告的變數，也就消逝不見。這種變數，就是區域變數，它能活動的區域，就是在所宣告的大括號內。

事實上，在 TC2.0 中，你可以在函式中任意加入一對大括號 { } ，在大括號中，前面可宣告變數，之後可以是程式碼。改寫前面 bar.c 爲例：

bar2.c

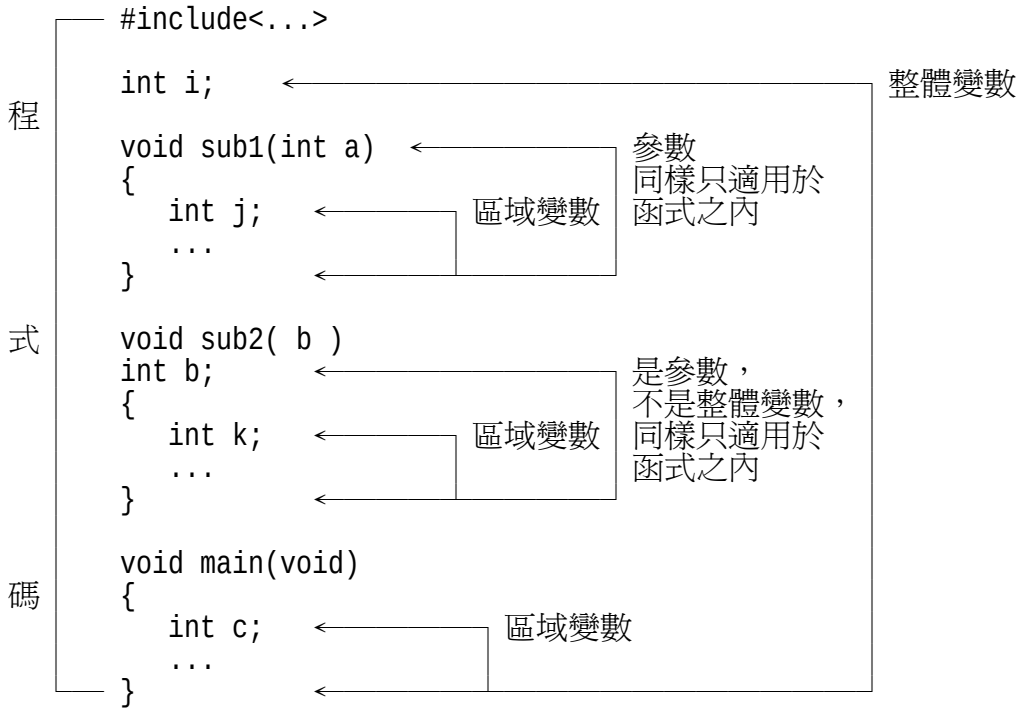
```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2
 3  void bar(int i)
 4  {                                          /* 開始定義 bar() */
 5     int j;
 6     for( j=0 ; j<i ; j++)  printf("*");      /* 秀出 i 個星號 */
 7     printf("\n");
 8  }                                          /* bar() 定義結束 */
 9
10  void main(void)
11  {
12     int i = 50;   <————————————————————————————————————————————
13     bar(i);             /* 使用外層的 i 爲 50 */                  |
14     {                                                             |
15        int i = 20;  <————————————————————————————————————————   |
16        bar(i);             /* 使用內層的 i 爲 20 */           |   |
17        i = i + 10;                                            |   |
18        bar(i);             /* 使用內層的 i 爲 30 */           |   |
19     }               <————————————————————————————————————————   |
20     i = i + 10;                                                   |
21     bar(i);             /* 使用外層的 i 爲 60 */                  |
22  }               <————————————————————————————————————————————

外層 i 的生命週期
內層 i 的生命週期
```

```
**************************************************
********************
******************************
************************************************************
```

在主程式中，故意宣告兩個同名的整數變數 i，它們分別在兩組巢狀的大括號內。在外層的 i 適用於整個 main() 函式。不巧的是，內層大括號也宣告了一個 i，使得在內層的程式碼只能使用內層宣告的 i。內層大括號執行結束後，內層的 i 就壽終正寢，外層的 i 就又開始有作用了。
如果，你在內層不另外宣告 i 這個變數，那在內層的程式也可以使用外層的 i，你可以把第 15 行的程式碼 remark 掉，看看結果有什麼不一樣。

總歸一句話，在大括號內所宣告的變數，就是區域變數。隨著大括號的結束，這一區的變數也就失去作用。等到下一次再執行到這段程式碼時，這一區的變數才會重生。

那整體變數要如何宣告呢？由上可知，只要是在大括號內宣告的就是區域變數，那整體變數，就是要宣告在大括號以外的地方。例：



☆ Call by Value v.s. Call by Address ☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆☆

○ Call by Value 傳值呼叫

前面我們所設計的副函式都是使用「傳值呼叫」，也就是副函式的參數變數不是指標型變數。以副函式 bar() 爲例，我們把它改寫如下，它的功能同上例：

bar3.c

```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2
 3  void bar(int i)
 4  {
 5     for(  ; i>0 ; i-- ) /* i 值，由傳進來的數值遞減至 0 */
 6        printf("*");
 7     printf("\n");
 8  }
 9
10  void main(void)
11  {
12     int i = 50;
13     int j = 30;
14
15     bar( 20 );          /* 以數值 20 呼叫 bar() 函式 */
16
17     bar( j );           /* 以 j 變數的數值 30 呼叫 bar() 函式 */
18     printf("j = %d\n", j );   /* 秀出 j 的數值 */
19
20     bar( i );           /* 以 i 變數的數值 50 呼叫 bar() 函式 */
21     printf("i = %d\n", i );   /* 秀出 i 的數值 */
22  }
```

```
********************
*****************************
j = 30
************************************************
i = 50
```

bar(20); 表示：把 20 這個數值傳給 bar() 函式做爲參數 i 的數值，
我們可以這樣看被呼叫的 bar() 函式：

```
void bar(...)
{
   int i = 20;
   for(  ; i>0 ; i-- )
      printf("*");
   printf("\n");
}
```

傳進來的數值是 20

變數 i 的滅亡

bar(j); 表示：把 j 這個變數的數值 30 傳給 bar() 函式做爲參數 i 的數值，
我們可以這樣看被呼叫的 bar() 函式：

```
void bar(...)
{
   int i = 30;
   for(  ; i>0 ; i-- )
      printf("*");
   printf("\n");
}
```

傳進來的數值是變數 j 的數值 30

變數 i 的滅亡

由於是傳入變數的數值，所以不論函式如何改變傳入的數值，都不會影響原來傳入的變數。請各位不要被變數名稱所矇騙，即使你使用與函式參數同名的變數，它一樣只是傳值，如範例中的

bar(i); 表示：把 i 這個變數的數值 50 傳給 bar() 函式做爲參數 i 的數值，
我們可以這樣看被呼叫的 bar() 函式：

```
void bar(...)
{
   int i = 50;
   for(  ; i>0 ; i-- )
      printf("*");
   printf("\n");
}
```

傳進來的數值是變數 i 的數值 50

變數 i 的滅亡

總結來說，傳值呼叫，就是不會改變所傳入變數的數值。
如果想要改變傳入的變數，就要使用「傳址呼叫」。

○ Call by Address 傳址呼叫

我們一直在使用的 scanf() 函式，就是一個傳址呼叫的函式，其一般型式如下：

```
scanf("控制字串" , &變數 1 , &變數 2 , ... );
```

我們傳給 scanf() 的參數是變數的位址，所以 scanf() 可以把讀入的字串依

控制字串的格式轉換成數值，存到變數的位址，這樣在結束 scanf() 之後，我們傳給 scanf() 的變數數值，就會是 scanf() 所讀入的數值。
前面說過，函式的傳回數值只有一個，而利用傳址呼叫的設計，就可以傳回更多的數值。以下就來看一個傳址呼叫的範例：將兩個整數值互換。

swap.c

```
 1  #include <stdio.h>     /* 宣告 printf() 的原型 */
 2
 3  void swap(int *a, int *b)
 4  {
 5     int backup = *a ;    /* 把 *a 的數值存到 backup 變數內   */
 6     *a = *b ;            /* 把 *b 的數值存到 *a              */
 7     *b = backup ;        /* 把 backup 的數值存到 *b 完成互換 */
 8  }
 9
10  void main(void)
11  {
12     int i = 50 , j = 30 ;
13
14     printf("Before swap(): i = %d   j = %d\n", i, j);
15
16     printf("Swapping i & j ...\n");
17     swap( &i , &j );                 /* 傳入 i 與 j 的位址 */
18
19     printf("After  swap(): i = %d   j = %d\n", i, j);
20  }
```

```
Before swap(): i = 50   j = 30
Swapping i & j ...
After  swap(): i = 30   j = 50
```

■ 第九章 檔案存取

前面幾章所寫的程式，都是將結果直接顯示在螢幕上，若有輸入的數值，則利用鍵盤輸入，想要知道程式執行的結果，就要再執行一次。 本章則是將程式執行的結果存成檔案，存好的檔案，你可以直接顯示出來看，或者是由程式直接讀取資料檔，當作是輸入的數值，經運算後，再另存檔案。

◎ 檔案的觀念

我們的資料及程式，都是以檔案的型式存在磁碟機中。每一種應用程式通常會使用自定的格式來存取它的資料檔，應用程式之間如果沒有互相支援，就不能讀取對方所產生的資料檔，例如：PEII 這個 DOS 下的文書處理器，就不能讀取 WORD 所編輯出來的檔案。
在磁碟機中的檔案可能很多，在使用時，我們必須先指定要處理檔案的名稱，作「開啓檔案」的動作，再依程式設計的格式存取檔案內的資料，處理完後，則要「關閉檔案」。
爲什麼要這麼麻煩呢？因爲，我們在作檔案存取時，是去呼叫系統程式所提供的程式模組，在「開啓檔案」後，系統會預置一些記憶體空間來存這個開啓的檔案。你在做檔案資料的處理時，系統並不一定會馬上將結果存回磁碟，它等預置的記憶體內的資料都處理完時，才會將資料存入磁碟。所以在使用完檔案之後，就要作「關閉檔案」的動作，這樣，系統才知道你已經不再使用這個檔案了，而將預置記憶體內尚未儲存的資料，存入磁碟，再收回預置的記憶體，供其他程式使用。如果，你不作「關閉檔案」的動作，那程式就會一直占用住那一塊記憶體，形成記憶體的浪費，此外，處理檔案不正常的結束，可能使新增的資料流失，還可能使磁碟機的檔案位置配置表(FAT,File Allocation Table)發生失去鏈結(Lost Chain) 或是交錯連結(Cross Link)的狀況。
所以，要養成好習慣：在程式中，只要有「開啓檔案」，就要有對應的「關閉檔案」。

◎ 檔案的開啓和關閉

要開啓檔案時，可以用 fopen 。其格式如下：

```
FILE *fp;
fp=fopen("檔案名稱","存取模式");
```

在這裡，fp 是一個指標，它指向 FILE 結構變數。FILE 結構內存著一些關於檔案的資訊，如：檔案位置指示，資料傳輸緩衝區的長度及其在記憶體中的位址等等，通常我們可以不用去理會這些數值，只要會使用 C 所提供的檔案處理函式，就可以了。FILE *fp; 中的 fp 通常稱爲檔案指標。在使用 C 所提供的檔案處理函式時，只要指定好檔案指標，就是對那個已開啓的檔案做相對應的處理。
fopen() 函式的兩個參數，第一個是"檔案名稱"，也就是要處理的檔案名稱，如果不在所執行的目錄，就要指定檔案的全名，也就是要包含路徑。
第二個參數是"存取模式"，有下列字串可供選擇：

存取模式	意義
r	開啓一存在檔案僅供讀取使用。當檔案不存在時，將傳回錯誤。
w	開啓檔案僅供寫入。當檔案不存在時，會產生新檔。
a	開啓一個檔案僅供增添資料。當檔案不存在時，會產生新檔。
r+	開啓一存在檔案供讀/寫使用。當檔案不存在時，將傳回錯誤。
w+	開啓一檔案供讀/寫使用。當檔案不存在時，會產生新檔。
a+	開啓一檔案供讀取及增添資料。當檔案不存在時，會產生新檔。

另外，還可以再加上 b ，表示以二進制的模式來存取資料檔。例如：

```
fp1 = fopen("letter.txt","w");
fp2 = fopen("c:\\work\\test.txt","a+");
fp3 = fopen("score.dat","r");
fp4 = fopen("database.dat","w+b");
```

要開啓檔案在使用完畢後，可以用 fclose 來關閉 。其格式如下：

```
fclose( 已開啓的檔案指標 );
```

如： fclose(fp);

對於已經關閉的檔案，如果還要再使用，則必須重新開啓。

◎ 檔案的寫入

要將資料輸出到檔案時，可以用 fprintf 。其格式如下：

```
fprintf( fp, "控制字串", 運算式 1, 運算式 2, ...);
```

fprintf() 的格式如同 printf() ，只是輸入參數中的第一個參數必須是檔案指標，也就是指定出 fprintf() 要輸出到那一個檔案。有關 printf() 請參考第四章。

以下是課本的範例：將使用者輸入的交易資料，編排成一份報表檔案。

record.c

```
 1 #include <stdio.h>     /* 宣告 printf, scanf, fopen,fprintf...*/
 2 #include <string.h>    /* 宣告 strcmp() 的原型 */
 3
 4 void line(FILE *fp)    /* 列出分隔線 */
 5 {
 6    int i;
 7    for( i=0 ; i<60 ; i++ )
 8       fprintf( fp, "-" );
 9    fprintf( fp, "\n" );
10 }
11
12 void main(void)
13 {
14    FILE *fp;           /* 檔案指標       */
15    char filename[20];  /* 輸出報表的檔名 */
16    char client[40];    /* 交易對象名稱   */
17    float amount;       /* 交易金額       */
18    float total=0;      /* 總金額         */
19
20    printf("File to record the amount : ");
21    scanf("%19s", filename );        /* 讀取輸出報表檔名 */
22    fp = fopen( filename, "w");      /* 開啓報表檔 */
23    if( fp == NULL )                 /* 判斷是否開啓成功 */
24       printf("\aCannot open %s for output!\n",filename);
25    else
26    {                                /* 開啓成功 */
27       line( fp );
28       printf("Client : ");
29       scanf("%39s", client);
```

```
30        for( ; strcmp(client, "end") != 0 ; ) /* clinet=="end" */
31        {                                     /* 時，迴圈結束  */
32           printf("Amount = ");
33           scanf("%f", &amount);
34           total = total + amount ;
35           fprintf( fp, "%-40s $%f\n", client, amount);
36
37           printf("Client : ");
38           scanf("%39s", client);
39        }
40        line( fp );
41        fprintf( fp, "%-40s $%f\n", "*****  Total", total );
42        line( fp );
43     }
44     fclose(fp);                        /* 關閉報表檔 */
45  }
```

```
File to record the amount : test
Client : Nanya_College
Amount = 9876.54
Client : ABC_Company
Amount = 1234.56
Client : IJK_lmn_...
Amount = 2323.23
Client : Xyz....
Amount = 2222.20
Client : zzzzzz
Amount = 1111.10
Client : end
```

test 檔案內容

```
------------------------------------------------------------
Nanya_College                              $9876.540039
ABC_Company                                $1234.560059
IJK_lmn_...                                $2323.229980
Xyz....                                    $2222.199951
zzzzzz                                     $1111.099976
------------------------------------------------------------
*****  Total                               $16767.630859
```

因爲我們是用 scanf() 來讀取字串的，而 scanf() 會把空白字元當成是資料的分隔，所以在輸入客戶名稱時以底線代替空白。（這也是 scanf() 的缺點之一）
第 30 行，迴圈以 strcmp(client, "end") != 0 作爲結束的條件判斷。其中，strcmp 是 TC 的內建函式，可用來比較兩個字串是否相等，若相等，則傳回 0 ，否則傳回非 0 值。所以當使用者輸入客戶名稱爲 "end" 時， strcmp 會傳回 0，而結束 for 迴圈。

◎ 檔案的讀取

要讀取檔案內的資料時，可以用 fscanf 。其格式如下：

```
fscanf( fp, "控制字串", &變數 1, &變數 2, ...);
```

fscanf() 的格式如同 scanf() ，只是輸入參數中的第一個參數必須是檔案指標，也就是指定出 fscanf() 要讀取那一個資料檔案。有關 scanf() 請參考第四章。

以下的範例：讀取檔案內容，將各個字元與 ASCII 值一起顯示。

ftoascii.c

```
 1  #include <stdio.h>     /* 宣告 printf, scanf, fopen,fprintf...*/
 2
 3  void main(void)
 4  {
 5     FILE *fp;           /* 檔案指標       */
 6     char filename[20];  /* 讀取的檔案名稱 */
 7     char ch=0;
 8
 9     printf("File name : ");
10     scanf("%19s", filename );        /* 讀取輸入檔名 */
11     fp = fopen( filename, "r");
12     if( fp == NULL )                 /* 判斷是否開啓成功 */
13        printf("\aCannot open %s !\n",filename);
14     else
15        for( ; ch != EOF ; )          /* ch == EOF 時，迴圈結束 */
16        {
17           fscanf( fp, "%c", &ch);    /* 讀取一個字元 */
18           printf("%c = %d\n", ch, ch);
19        }
20     fclose(fp);
21  }
```

```
File name : ftoascii.c
# = 35
i = 105
n = 110
... ...  (省略)
} = 125
= -1
```

第 15 行，迴圈以 ch != EOF 作爲結束的條件判斷。其中，EOF 是一個字元，表示檔案結束(End Of File)的字元。所以當 fscanf 讀取到檔案結束字元時，迴圈就會結束執行。

以下是課本的範例：讀取檔案內容，並將檔案行距加倍後，另存新檔。

2space.c

```
 1  #include <stdio.h>     /* 宣告 printf, scanf, fopen,fprintf...*/
 2
 3  void main(void)
 4  {
 5     FILE *fpi, *fpo;    /* 檔案指標       */
 6     char filename[20];  /* 讀取的檔案名稱 */
 7     char ch=0;
 8
 9     printf("File to be read : ");
10     scanf("%19s", filename );        /* 讀取輸入檔名 */
11     fpi = fopen( filename, "r");     /* 開啓輸入檔案 */
12     if( fpi == NULL )                /* 判斷是否開啓成功 */
13     {
14        printf("\aCannot open %s !\n",filename);
```

```
15          return ;                        /* 結束 main() 函式 */
16       }
17
18       printf("File to be written : ");
19       scanf("%19s", filename );           /* 讀取輸出檔名 */
20       fpo = fopen( filename, "w");        /* 開啓輸出檔案 */
21       if( fpo == NULL )                   /* 判斷是否開啓成功 */
22       {
23          printf("\aCannot open %s for output!\n",filename);
24          fclose(fpi);       /* 已開的 fpi 在結束前要先關閉 */
25          return ;                        /* 結束 main() 函式 */
26       }
27
28       for( ; ch != EOF ; )
29       {
30          fscanf( fpi, "%c", &ch);        /* 讀取一個字元 */
31          fprintf( fpo, "%c", ch);        /* 寫入一個字元 */
32          if( ch == '\n')                 /* 如果讀到跳行字元 */
33            fprinf( fpo, "\n");           /* 就再跳一行 */
34       }
35
36       fclose(fpo);                       /* 關閉輸出檔(後開先關) */
37       fclose(fpi);                       /* 關閉輸入檔(先開後關) */
38    }
```

```
File to be read : test
File to be written : doubled
```

double 檔案內容

```
-------------------------------------------------------------

Nanya_College                                $9876.540039

ABC_Company                                  $1234.560059

IJK_lmn_...                                  $2323.229980

Xyz....                                      $2222.199951

zzzzzz                                       $1111.099976

-------------------------------------------------------------

*****  Total                                 $16767.630859
```

```
                    Help
fopen: opens a stream

FILE *fopen(const char *filename,
            const char *mode);

Prototype in stdio.h

Returns a pointer to the newly open stream if
successful; else it returns NULL.

See also     fclose   creat    open
             dup      ferror   _fmode
             rewind   setbuf   setmode
```

開啓一個「檔案流」。

fopen 的語法

必須 #include<stdio.h>

若成功則傳回開啓檔案的指標
否則傳回 NULL (空指標)

相關指令

```
                    Help
fclose: closes a stream

int fclose(FILE *fp);

Prototype in stdio.h

Returns 0 on success; it returns EOF if any
errors are detected.

See also     fflush  flushall
             fopen   close   fcloseall
```

關閉一個「檔案流」。

fclose 的語法

必須 #include<stdio.h>

若關閉成功則傳回 0 ，
發生錯誤則傳回 EOF 字元。

相關指令

```
                    Help
fprintf: sends formatted output to a stream

int fprintf(FILE *fp, const char *format, ...);

Prototype in stdio.h

Uses the same format specifiers as printf,
but fprintf sends output to the specified
stream fp. fprintf returns the number of bytes
output.  In event of error, it returns EOF.

See also     putc    fscanf
```

將格式化的資料輸出到檔案流

fprintf 的語法

必須 #include<stdio.h>

使用方法同 printf()，
只是將結果輸出到指定的
檔案指標 fp 。

相關指令

```
                    Help
fscanf: performs formatted input from a stream

int fscanf(FILE *fp, const char *format, ...);

Prototype in stdio.h

Returns the number of input fields success-
fully scanned, converted, and stored; the
return value does not include unstored
scanned fields.

See also     getc    fprintf   scanf
```

由檔案流讀取格式化的資料

fscanf 的語法

必須 #include<stdio.h>

使用方法同 printf()，
只是由指定的檔案流 fp 讀入
格式化的資料。

相關指令

```
                        Help
strcmp: compares s2 to s1

int  strcmp(const char *s1, const char *s2);

Prototype in string.h

Returns a value that is < 0 if s1 is less than
s2; == 0 if s1 is the same as s2; > 0 if s1 is
greater than s2. Performs a signed comparison.
```

比較兩個字串

strcmp 的語法

必須 #include<string.h>

如果 s1 < s2 則傳回值 < 0
如果 s1 = s2 則傳回值 = 0
如果 s1 > s2 則傳回值 > 0

```
                        Help
gets: gets a string from stdin

char *gets(char *string);

Prototype in stdio.h

Collects input from stdin until a newline
character (\n) is found.  The \n is not
placed in the string.

Returns a pointer to the argument string.

See also     ferror     getc
             fopen      puts
             fread      scanf
```

由 stdin 讀取字串

gets 的語法

必須 #include<stdio.h>

由 stdin 讀取字串，直到遇到跳行字元 \n 。
\n 不包含在所讀入的字串中。

所傳回的指標指向參數字串。

相關指令