

## C 語言程式設計教學講義

陳佳杏 製作

<b>單元 1：程式語言簡介 .....</b>	<b>3</b>
一、程式設計的流程 .....	3
二、程式語言的分類 .....	3
【直譯器與編譯器的比較】 .....	3
三、依程式設計的方式分類 .....	4
四、常見的高階程式語言 .....	4
<b>單元 2：認識 C 語言 .....</b>	<b>5</b>
一、歷史發展 .....	5
二、C 程式的開發環境 .....	5
三、繪製流程圖 .....	6
四、結構化程式設計 .....	6
<b>單元 3：變數與資料型態 .....</b>	<b>7</b>
一、變數與常數 .....	7
二、基本資料型態 .....	7
三、變數的命名原則 .....	7
四、資料型態轉換 .....	7
<b>單元 4：格式化的輸入與輸出 .....</b>	<b>8</b>
一、標準輸出指令 .....	8
二、跳脫字元(Escape Sequence) .....	8
三、修飾子 .....	8
四、格式化輸出：使用修飾子 .....	9
五、標準輸入指令 .....	9
<b>單元 5：運算子與運算式 .....</b>	<b>10</b>
一、基本運算子 .....	10
二、其他常用的運算子 .....	11
<b>單元 6：選擇敘述句 .....</b>	<b>12</b>
一、if 敘述 .....	12
二、switch 敘述 .....	12
【補充】取亂數 .....	13
<b>單元 7：迴圈敘述句 .....</b>	<b>14</b>
一、比較 for / while / do...while .....	14
二、無窮迴圈 .....	14
三、空迴圈 .....	14
四、巢狀迴圈 .....	15
五、迴圈的跳離 .....	15
<b>單元 8：函式 .....</b>	<b>16</b>
一、為何要函式化？ .....	16
二、函式 (Function) 定義的格式 .....	16
三、函式原形 (Prototype) .....	16
四、函式的呼叫 .....	16

五、遞迴函式 (Recursive Function).....	16
六、儲存體類別 (Storage Classes).....	17
七、範圍規則 (Scope).....	17
【應用練習：使用遞迴函式】 .....	18
<b>單元 9：前置處理器 .....</b>	<b>19</b>
一、#define 前置處理器 .....	19
二、#include 前置處理器 .....	19
<b>單元 10：陣列.....</b>	<b>20</b>
一、一維陣列 .....	20
二、二維陣列 .....	20
三、陣列與函數 .....	21
<b>單元 11：字串.....</b>	<b>22</b>
一、字元陣列 .....	22
二、字串的輸入/輸出函數 .....	22
三、字串陣列 .....	23
<b>單元 12：指標.....</b>	<b>24</b>
一、指標變數 .....	24
二、指標運算子 .....	24
三、指標的運算 .....	25
【指標的簡潔運算式】 .....	25
四、指標與函數 .....	26
五、指標與陣列的關係 .....	26
六、指標陣列 .....	27
【比較】字串陣列 V.S. 指標陣列 .....	27
七、雙重指標 – 指向指標的指標 .....	27
八、動態配置記憶體 .....	28
<b>單元 13：結構與其他資料形態 .....</b>	<b>29</b>
一、結構 (Structure) .....	29
二、巢狀結構 .....	29
三、結構陣列 .....	30
四、結構指標 .....	30
五、結構與函數 .....	30
六、自訂型態 (typedef).....	31
<b>單元 14：檔案.....</b>	<b>32</b>
一、檔案儲存在記憶體的形式： .....	32
二、檔案存取模式 .....	32
三、檔案處理函數（有緩衝區） .....	33
四、命令列參數的使用 .....	34

#### 參考資料：

1. C 語言教學手冊（第二版）/ 洪維恩 編著 / 博碩文化（民 90）
2. C 程式設計藝術（第三版）/ 吳國樑 編譯 / 全華（民 90）

## 單元 1：程式語言簡介

### 一、程式設計的流程

1. Defining the program：定義問題
2. Planning the solution：設計解決方案（演算法）
3. Coding the program：撰寫程式
4. Testing the program：測試程式
5. Documenting the program：撰寫程式發展文件

### 二、程式語言的分類

1. **機器語言 (Machine Language)：**  
指硬體內部所使用的語言，也是電腦唯一能直接辨識的語言，通常是一連串的數字所組成 (0 或 1)。
2. **組合語言 (Assembly Language)：**  
最接近機器語言的一種低階語言，屬於符號式語言。組合語言必須經由組譯器 (assembler) 轉換成機器語言，才能在電腦上執行。
3. **高階語言 (High-level Language)：**  
又稱編譯語言，其文法接近日常英文用語，也包含一般常用的數學運算符號。高階語言的原始程式碼必須轉換為機器語言才能正確執行。轉換程式包含直譯器和編譯器兩種。
4. **非常高階語言 (Very High-level Language)：**  
又稱第四代語言，如 SQL (Structural Query Language，結構化查詢語言)。
5. **自然語言 (Natural Language)**

#### 【直譯器與編譯器的比較】

	說明	優	缺	例
直譯器 Interpreter	將原始程式的指令逐一翻譯並執行，不需要經過編譯。	<ul style="list-style-type: none"> <li>● 佔用的記憶體較少</li> <li>● 修改及除錯容易</li> </ul>	<ul style="list-style-type: none"> <li>● 每次執行前才翻譯，執行速度慢、效率較低</li> </ul>	BASIC、HTML...
編譯器 Compiler	將原始程式編譯後，先產生目的檔 (.obj)，再將其他要連結的程式連結後，再執行該程式。	<ul style="list-style-type: none"> <li>● 執行時不需要重複編譯，執行速度及效率高</li> </ul>	<ul style="list-style-type: none"> <li>● 原始程式經過修改就必須重新編譯</li> <li>● 較佔用記憶體空間</li> </ul>	C/C++、COBOL、PASCAL...

### 三、依程式設計的方式分類

1. 程序導向程式設計 (Procedure-Oriented Programming)
2. 物件導向程式設計 (Object-Oriented Programming)  
如：C++，Java，Visual BASIC

### 四、常見的高階程式語言

語言	說明	應用
FORTRAN	FORmula TRANslator (1954)	科學
COBOL	COmmon Business-Oriented Language (1959)	商業
BASIC	Beginner's All-purpose Symbolic Instruction Code	教育，商業
Pascal	named after French inventor Blaise Pascal (1971)	教育，系統，科學
Ada	named after Ada, the Countess of Lovelace (1980)	軍事，一般
C	evolved from the language B, from Bell Labs (1972)	系統，一般

## 單元 2：認識 C 語言

---

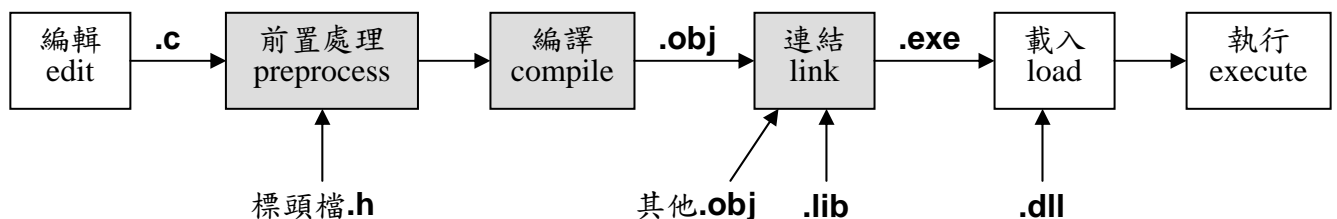
### 一、歷史發展

C 語言是由美國貝爾實驗室 (Bell Laboratory) 的 Dennis Ritchie 在 1972 年所發展出來的。C 語言的前身是 B 語言，原先用在 DEC PDP-11 電腦上。C 語言在各種平台上快速發展，之後出了許多版本，為了統一各版本，美國國家標準局 (ANSI) 提出了一套標準，並在 1989 年通過審查，而在 1999 年則進行了修訂。

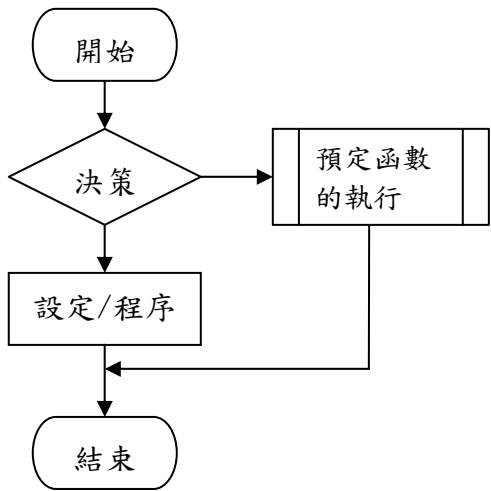
### 二、C 程式的開發環境

C 程式在執行前經過六個階段。

1. **編輯**：使用者可利用文書編輯器 (editor) 撰寫或修改 C 程式碼 (source code)。
2. **前置處理**：前置處理器 (preprocessor) 會在開始編譯前自動執行，依據程式碼中 # 所標示的指示 (preprocessor directives)，進行代換或插入等動作。例如：`#include <stdio.h>` 告訴編譯器在未編譯程式之前，先將程式庫中的標頭檔 `stdio.h` 插入該位置。
3. **編譯**：編譯器 (compiler) 將程式碼編譯為目的碼 (object code)。
4. **連結**：連結器 (linker) 將一個或多個目的檔 (.obj) 與靜態程式庫檔 (.lib) 連結，產生可執行檔 (.exe)。
5. **載入**：載入器 (loader) 將可執行檔 (.exe) 載入記憶體，並與動態程式庫檔 (.dll) 連結。動態程式庫可減少執行檔所佔的硬碟和記憶體空間。
6. **執行**：最後，電腦在 CPU 的控制下，開始執行所載入的程式。



三、繪製流程圖



四、結構化程式設計

	循序性結構 (Sequence Structure)	選擇性結構 (Selection Structure)	重複性結構 (Iteration Structure)
語法		<ul style="list-style-type: none"><li>if...else</li><li>switch...case</li></ul>	<ul style="list-style-type: none"><li>for</li><li>while</li><li>do...while</li></ul>
流程圖	<pre>graph TD; Start([開始]) --&gt; S1[敘述 1]; S1 --&gt; S2[敘述 2]; S2 --&gt; End([結束]);</pre>	<pre>graph TD; Entry(( )) --&gt; Decision{判斷}; Decision -- true --&gt; S1[敘述 1]; Decision -- false --&gt; S2[敘述 2]; S1 --&gt; S2; S2 --&gt; S3[敘述 3]; S3 --&gt; Exit(( ));</pre>	<pre>graph TD; Entry(( )) --&gt; Decision{判斷}; Decision -- true --&gt; S1[敘述 1]; S1 --&gt; Decision; Decision -- false --&gt; S2[敘述 2]; S2 --&gt; Exit(( ));</pre>

## 單元 3：變數與資料型態

### 一、變數與常數

變數 (variable) 是利用宣告的方式，將記憶體中的某個區塊配置給此變數，不管變數的值如何改變，它會一直佔用相同的記憶體空間。

例：`int i=3;`  
`float f=15.7;`  
`char ch='y';`

常數 (constant) 的值是固定的，如整數常數、字元常數等。

例：`const int max=65536;`

### 二、基本資料型態

	資料型態	位元組	範圍	備註
char	字元	1	0~255	用來儲存英文字母及 ASCII 碼
int	整數	2	-32768~32767	可在 int 之前加上修飾詞 (qualifier)， unsigned、short、long
float	浮點數	4	1.2e-38~3.4e38	可用小數點及指數型態表示。
double	倍精度浮點數	8	2.2e-308~1.8e308	可用小數點及指數型態表示。

### 三、變數的命名原則

所有的變數必須在使用前宣告。命名原則如下：

1. 不能使用關鍵字
2. 只有前 8 個字元為有效字元
3. 可使用英文字母、數字或底線
4. 變數名稱中間不可以有空白
5. 第一個字元不可為數字
6. 變數名稱要有意義，且長短適中
7. 大小寫有別

### 四、資料型態轉換

1. 指派轉換 (`x = 100;`)
2. 算術轉換 (`y = i * 5 + 7 / 23;`)
3. 模式轉換 (`i = (int)(x + 0.9);`)
4. 函數轉換 (`x = sum(a, b);`)

## 單元 4：格式化的輸入與輸出

### 一、標準輸出指令

```
printf("格式字串", var1, var2, ...);
```

### 二、跳脫字元(Escape Sequence)

\n 換行	\\" 雙引號	\x ASCII 碼 (16 進位)
\f 換頁	\' 單引號	\d ASCII 碼 (8 進位)
\t 跳格	\ / 斜線	
\b 倒退	\\ 反斜線	

例子	執行結果
<pre>printf("\tThis line begins with tab.\n"); printf("It\'s a \"C Tutorial\".\n"); printf("This is backslash: \\.\\.\n"); printf("\\101 is \101.\n"); printf("\\x41 is \x41.\n");</pre>	<pre>    This line begins with tab. It's a "C Tutorial". This is backslash: \. \101 is A. \x41 is A.</pre>

### 三、修飾子

-：向左靠齊 +：印出正負號
%c：字元 %s：字串 %d：十進位整數 %f：浮點數（小數點型式）
%l：長整數，加在 d、u...之前 %u：無號十進位整數 %e：浮點數（指數 e 型式）



## 四、格式化輸出：使用修飾子

資料	格式	結果									
12345	%10d						1	2	3	4	5
12345	%+d	+	1	2	3	4	5				
12345	%-10d	1	2	3	4	5					
12345	% d		1	2	3	4	5				
12345	%010d	0	0	0	0	0	1	2	3	4	5
123.456	%7.2f		1	2	3	.	4	6			
123.456	%010.3f	0	0	0	1	2	3	.	4	5	6
123.456	%+10.4f		+	1	2	3	.	4	5	6	0

## 五、標準輸入指令

```
scanf("格式化字串", &var1, &var2, ...);
```

1. &是位址運算子
2. 字元陣列不需要加上&位址運算子

## 【輸入格式】

%d：十進位整數 → int  
 %f：浮點數 → float, double  
 %c：字元 → char  
 %s：字串 → 字元陣列

例子	結果
<pre>int num1, num2; printf("Enter 2 numbers: "); scanf("%d,%d",&amp;num1,&amp;num2);</pre>	執行畫面： <b>Enter 2 numbers:</b>  輸入值格式： <b>103,227</b>

## 單元 5：運算子與運算式

### 一、基本運算子

#### 1. 算數運算子

運算子	意義	int a=9,b=4	運算結果
+	加法	a+b	13
-	減法	a-b	5
*	乘法	a*b	36
/	除法	a/b	2
%	取餘數	a%b	1

運算子	原式	簡潔運算式
+=	a=a+b	a+=b
-=	a=a-b	a-=b
*=	a=a*b	a*=b
/=	a=a/b	a/=b
%=	a=a%b	a%=b

#### 2. 關係運算子

運算子	意義	例子	運算結果
>	大於	2>3	false
>=	大於等於	2>=3	false
<	小於	2<3	true
<=	小於等於	2<=3	true
==	等於	2==3	false
!=	不等於	2!=3	true

#### 3. 邏輯運算子

運算子	意義
&&	AND, 且
	OR, 或
!	NOT, 否

a	b	a&& b	a    b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

## 二、其他常用的運算子

### 1. 遞增/減運算子

運算子	意義	int i=3; int a;	運算結果	
			i	a
++	變數值加 1	i++	4	
--	變數值減 1	i--	2	
		a = i++	4	3
		a = ++i	4	4
		a = i--	2	3
		a = --i	2	2

- **i++**：先執行整個敘述後，再將 i 的值加 1
- **++i**：先將 i 的值加 1，再執行整個敘述

### 2. 條件運算子

運算子(?:)	意義
條件判斷 ? 運算式 1 : 運算式 2	if(判斷條件) 運算式 1; else 運算式 2;
實例	
a = (x > 100) ? b : c;	if (x > 100) a = b; else a = c;
abs = (a > 0) ? a : -a;	if (a > 0) abs = a; else abs = -a;

## 單元 6：選擇敘述句

### 一、if 敘述

if 敘述	if...else	巢狀 if	if...else if...else
if (判斷條件) 敘述;	if (判斷條件) 敘述 1; else 敘述 2;	if (判斷條件 1) { 敘述主體 1;  if (判斷條件 2) { 敘述主體 2; } ... }	if (判斷條件) { 敘述主體 1; } else if (判斷條件) { 敘述主體 2; } ... else { 敘述主體 n; }
if (判斷條件) { 敘述 1; ... 敘述 n; }	if (判斷條件) { 敘述主體 1; } else { 敘述主體 2; }		

### 二、switch 敘述

switch (運算式)

```
{
    case 選擇值 1:
        敘述主體 1;
        break;
    case 選擇值 2:
        敘述主體 2;
        break;
    ...
    case 選擇值 n:
        敘述主體 n;
        break;
    default:
        敘述主體;
}
```

注意：

1. 選擇值只能是「字元」或「常數」
2. break 用於跳離 switch 區塊
3. 當選擇值皆不成立時預設處理程序放在 default 裡頭

### 【隨堂練習】

1. 輸入三個整數 a、b、c，印出此三數的最大值、最小值、總和及平均值。
2. 輸入兩整數的四則運算式 (a+b, a-b, a\*b, a/b, a%b)，印出其計算結果。
3. 利用 switch 敘述，將輸入的成績（分數）以下列方式分級。

90~100: A 級

80~89: B 級

70~79: C 級

60~69: D 級

0~59: E 級

### 【補充】取亂數

```
#include <stdlib.h>
#include <time.h>

srand((unsigned)time(NULL));    /* 函數 srand()：下亂數種子 */
int num = a + rand() % b;       /* 函數 rand()：取亂數 */

a：起始值
b：亂數個數
```

實例：

- (1) 數字 0~9 的亂數取法： $\text{rand()} \% 10$
- (2) 骰子點數 1~6 的亂數取法： $1 + \text{rand()} \% 6$
- (3) 整數 18~32 的亂數取法： $18 + \text{rand()} \% 15$
- (4) 五位數的亂數取法： $10000 + \text{rand()} \% 90000$

### 【練習題】

1. 數字加密：亂數產生四位數，將每個位數的數字加 7 後除以 10 取餘數，然後第一個數字和第三數字對調，第二個數字和第四個數字對調。最後印出加密過的數。
2. 數字解密：根據上題，讀入加密過的數字，然後解密為原來的四位數。

## 單元 7：迴圈敘述句

### 一、比較 for / while / do...while

for 敘述	while 敘述	do...while 敘述
<pre>for (初值; 判斷條件; 增減量) {     敘述 1;     敘述 2;     ...     敘述 n; }</pre>	<pre>設初值; while (判斷條件) {     敘述 1;     敘述 2;     ...     敘述 n;     設增減量; }</pre>	<pre>設初值 do {     敘述 1;     敘述 2;     ...     敘述 n;     設增減量; } while (判斷條件);</pre>
實例		
<pre>for (i=1,sum=0;i&lt;=9;i+=2) {     sum += i;     printf("i=%d\n", i);     printf("sum=%d\n",sum);     printf("\n"); }</pre>	<pre>i=1; sum=0; while(i&lt;=9) {     sum += i;     printf("i=%d\n", i);     printf("sum=%d\n",sum);     printf("\n");     i += 2; }</pre>	<pre>i=1; sum=0; do {     sum += i;     printf("i=%d\n", i);     printf("sum=%d\n",sum);     printf("\n");     i += 2; } while (i&lt;=9);</pre>

### 二、無窮迴圈

```
while (1)
{
    敘述主體;
}
```

### 三、空迴圈

```
for (設初值; 判斷條件; 設增減量)
{ }
```

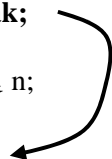

或

```
for (設初值; 判斷條件; 設增減量);
```

## 四、巢狀迴圈

for 之巢狀迴圈	while 之巢狀迴圈
<pre>for (初值 1; 條件 1; 增減量 1) {     for (初值 2; 條件 2; 增減量 2)     {         ...     }     ... }</pre>	<pre>初值 1; while (條件 1) {     初值 2;     while (條件 2)     {         ...         增減量 2;     }     ...     增減量 1; }</pre>

## 五、迴圈的跳離

break (跳出迴圈)	continue (回到迴圈開始處)
<pre>for (設初值; 判斷條件; 設增減量) {     敘述 1;     敘述 2;     ...     break;     ...     敘述 n; } ...</pre> 	<pre>for (設初值; 判斷條件; 設增減量) {     敘述 1;     敘述 2;     ...     continue;     ...     敘述 n; } ...</pre> 
備註：在 while / do while 迴圈中的用法相同	

## 【隨堂練習】

- 輸入正整數  $n$ 
  - 求  $1 + 2 + \dots + n$  的值
  - 求  $1 * 2 * \dots * n$  的值
  - 求  $1! + 2! + \dots + n!$  的值
- 輸入大於 2 的正整數  $n$ 
  - 求小於等於  $n$  的所有質數
  - 求小於  $n$  的所有質因數

## 單元 8：函式

---

### 一、為何要函式化？

1. 使程式發展容易管理
2. 軟體再使用（抽象化的技術, abstraction)
3. 避免重複撰寫相同的程式碼

### 二、函式 (Function) 定義的格式

回傳值型態 函式名稱(參數列) { 變數宣告; 陳述句; return 回傳值; }	<pre>int sum(int a, int b) {     int num;     num = a + b;     return num; }</pre>
-----------------------------------------------------------	------------------------------------------------------------------------------------

### 三、函式原形 (Prototype)

編譯器利用函式原形來驗證函式的呼叫，函式原形通常置於 main() 函式外，或是標頭檔內。函式原形的另一個功能是引數型態的強制轉換。

傳回值型態 函式名稱 (引數型態);	<pre>int sum(int, int); 或 int sum(int a, int b);</pre>
--------------------	--------------------------------------------------------

標頭檔 (Header File, .h) 含有函式原形，以及這些函式所需的各種「資料型態」和「常數」的定義。我們可以撰寫自己的標頭檔，以.h 為附檔名。在原始程式檔中，以 **#include** 將標頭檔含括進來。

### 四、函式的呼叫

1. 傳值呼叫 (call by value)：將引數「值」拷貝一份傳給函式，不會影響原來的變數值。
2. 傳參考呼叫 (call by reference)：將引數的「位址」傳給函式，會影響原來的變數值。

### 五、遞迴函式 (Recursive Function)

1. 自己呼叫自己，注意終止條件的設定
2. 用到大量的堆疊 (stack) 空間，容易造成記憶體不足
3. 可以改寫成迴圈形式



## 六、儲存體類別 (Storage Classes)

佔用期間	類別	說明
動態	自動變數 auto	<ul style="list-style-type: none"> <li>● auto 變數是在程式控制進入「所宣告的區塊」時才被產生出來，一離開此區塊，就會從記憶體中消失了。</li> <li>● 「區域變數」內定為自動變數 例：<code>auto float x,y;</code></li> </ul>
	暫存器變數 register	<ul style="list-style-type: none"> <li>● 宣告成 register 的變數會放到硬體暫存器中，減少運算時從記憶體載入的負擔。</li> <li>● register 變數只能用在「自動變數」上 例：<code>register int counter = 1;</code></li> </ul>
靜態	靜態變數 static	<ul style="list-style-type: none"> <li>● static 變數從程式開始執行時便配置好儲存體，並設定初值(數值預設為 0，指標預設為 NULL)</li> <li>● static 變數包含「內部靜態變數」與「外部靜態變數」 例：<code>static int count = 1;</code></li> </ul>
	外部變數 extern	<ul style="list-style-type: none"> <li>● 外部變數包含「全域變數」和「函式名稱」</li> <li>● 外部變數也屬於靜態變數，在程式開始執行時便佔有記憶體空間</li> </ul>

## 七、範圍規則 (Scope)

分類	說明	包含...
檔案範圍	可在整個檔案中使用	1. 全域變數 (Global Variable) 2. 函式定義 3. 函式原形
區塊範圍	可在區塊內使用，也就是{ }包起來的範圍	1. 區域變數 (Local Variable) 2. 巢狀區塊中的變數 3. 函式的參數
函式範圍	只能在函式範圍內使用	標名： 1. goto 陳述句的標名 2. switch 結構的 case
函式原形範圍		函式原形參數列中的識別字

## 【應用練習：使用遞迴函式】

## 1. 費氏數列 (Fibonacci)：

費氏數列中第  $n$  項的值等於前兩項的合，如 1 1 2 3 5 8 13 21 34 ...

設  $f(n)$  表示第  $n$  項，根據定義： $f(n) = f(n-1) + f(n-2)$

請利用遞迴函數，求出  $f(n)$

```
int f(int n)
{
    if (n==1 || n==2)
        return 1;
    else
        return f(n-1)+f(n-2);
}
```

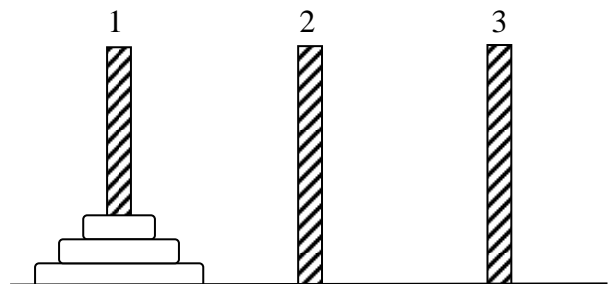
## 2. 河內塔 (Hanoi) 問題：

有三根柱子和  $n$  個大小不同的環（如圖）。

欲將  $n$  個環由 A 柱移到 B 柱上，移動規則如下：

- (1) 一次只能移動一個環
- (2) 在搬移的過程中，大環不能壓住小環。

問：最少移動次數及搬移過程為何？



```
void hanoi(int n, int src, int dst, int tmp)
{
    if (n==1)
        printf("%d → %d\n", src, dst);
    else
    {
        hanoi(n-1, src, tmp, dst);
        printf("%d ( %d\n", src, dst);
        hanoi(n-1, tmp, dst, src);
    }
}
```

[輸出畫面]  
 Input: (n,A,B)=(3,1,3)  
 Output: 1→3  
           1→2  
           3→2  
           1→3  
           2→1  
           2→3  
           1→3  
 移動次數:7

## 單元 9：前置處理器

---

### 一、#define 前置處理器

1. 增加程式的易讀性
2. 簡化修改常數或字串內容時的複雜度
3. 增加程式執行效率
4. 巨集 (Macro) 的使用可取代簡單的函數

```
#define MAX 32767
#define WORD "This is a test!"
#define AREA(w, h) ((w)*(h))
#define POWER(i) (i)*(i)*(i)
int main()
{
    ...
    printf("%d \n", MAX);
    printf(WORD);
    printf("area = %d \n", AREA(5, 3));
    printf("%d * %d * %d = %d \n", i+1, i+1, i+1, POWER(i+1));
    ...
}
```

### 二、#include 前置處理器

將標頭檔(.h)含括進程式中。

使用	說明	用法
< >	到系統設定的目錄尋找該檔案 如：C:\Dev-C++\Include\	#include <stdio.h>
" "	依指定的目錄尋找該標頭檔	#include "D:\myprog\area.h"

## 單元 10：陣列

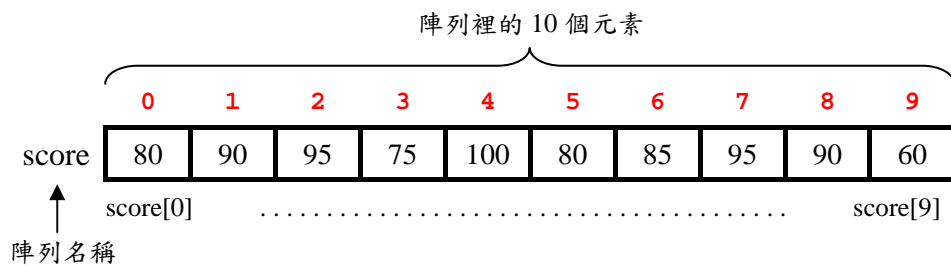
### 一、一維陣列

宣告格式：

型態 陣列名稱[個數];

範例

```
int score[10] = {80,90,95,75,100,80,85,95,90,60};
```



每個元素佔一個 int 型態大小(4 bytes)。

sizeof(score) → 4 \* 10 = 40 bytes

初值設定方式：

```
int data[5] = {1};           /* 將所有元素值都設為 1 */
int num[] = {60,75,48,92};   /* 依照初值設定的個數決定陣列的大小 */
int student[10] = {1,2,3,4,5}; /* 初值個數少於宣告元素個數時，剩餘空間填 0 */
```

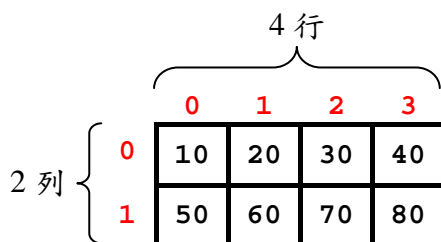
### 二、二維陣列

宣告格式：

型態 陣列名稱[列數][行數];

範例

```
int sale[2][4] = {{10,20,30,40}, {50,60,70,80}};
```



sale[0][0] → 10

sale[0][1] → 20

...

sale[1][0] → 50

sale[1][1] → 60

...

陣列中的元素個數 2\*4=8

sizeof(sale) → 4\*(2\*4)=32bytes

初值設定方式：

```
int temp[][4] = { {1,2,3,4},          /* 未定長度之二維陣列的初值設定 */
                  {5,6,7,8},
                  {9,10,11,12} };

```

多維陣列的宣告和初值設定方式依此類推。

### 三、陣列與函數

陣列可以當作引數傳遞到函數中，此時函數接收到的是陣列的『位址』，而非陣列的值。事實上，傳遞至函數中的也就是指向陣列位址的指標 (Pointer)。

範例

```
#include <stdio.h>
#define SIZE 5

void print_matrix(int A[]); /* 函式原型 */

int main(void)
{
    int data[SIZE] = {23,25,28,29,27};
    printf_matrix(data);
    return 0;
}

void print_matrix(int A[])
{
    int i;
    for (i=0; i<SIZE; i++)
        printf("%d ", A[i]);
    printf("\n");
    return;
}

```

Memory

In main()

data

0	23	0253FDB8
1	25	
2	28	
3	29	
4	27	

In print\_matrix()

A 為指標常數 0253FDB8

## 單元 11：字串

### 一、字元陣列

在 C 語言中並沒有字串的資料型態，要使用字串變數，就要宣告『字元陣列』。字串常數儲存在記憶體時，在最後面會加上字串結束字元\0做結尾。

宣告格式：

**char 字元陣列名稱[字串長度];**

範例

```
char name[15] = "David Chen";
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
name	D	a	v	i	d		C	h	e	n	\0				

比較下列三種宣告：

宣告	說明
<code>char a[] = "My friend";</code>	<code>sizeof(a) → 9 個字元 + 字串結束字元\0 = 10 bytes</code>
<code>char b = 'c';</code>	<code>sizeof(b) → 1 個字元 = 1 byte</code>
<code>char str[] = "c";</code>	<code>sizeof(str) → 1 個字元 + 加上字串結束字元\0 = 2 bytes</code>

### 二、字串的輸入/輸出函數

	(1)	(2)
語法	<code>scanf("%s", 字元陣列名稱);</code> <code>printf("%s", 字元陣列名稱);</code>	<code>gets(字元陣列名稱);</code> <code>puts(字元陣列名稱);</code>

說明	scanf()讀到 Enter 或空白時就結束讀取動作，同時在字串結尾處加上\0。	<ol style="list-style-type: none"> <li>1. gets()在讀到 Enter 時才結束讀取動作，並在字串結尾處加上\0。</li> <li>2. puts()會將\0 轉換成換行字元，在輸出字串時會自動換行。</li> </ol>
範例	<pre>#include &lt;stdio.h&gt; int main(void) {     char name[15];     printf("What's your name?\n");     scanf("%s", name);     printf("Hi! %s How are you?\n",            name);     return 0; }</pre>	<pre>#include &lt;stdio.h&gt; int main(void) {     char name[15];     puts("What's your name?");     gets(name);     puts("Hi! ");     puts(name);     puts(" How are you?");     return 0; }</pre>

### 三、字串陣列

宣告格式：

**char 字元陣列名稱[陣列大小][字串長度];**

範例

```
char name[3][10] = {"David", "Jane Wang", "Tom Lee"};
```

	0	1	2	3	4	5	6	7	8	9
name[0] 0253FDB8	D	a	v	i	d	\0				
name[1] 0253FDC2	J	a	n	e		W	a	n	g	\0
name[2] 0253FDCC	T	o	m		L	e	e	\0		

name[0] 與 name[1]相差 10 bytes

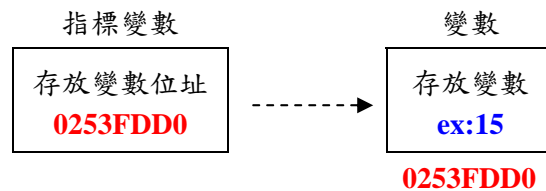
name[1] 與 name[2]相差 10 bytes。

常用的字串處理函數放在 string.h 中，請參見附錄 D-10 頁。

## 單元 12：指標

### 一、指標變數

指標 (Pointer) 是一種特殊的變數，用來存放變數在記憶體中的位址。在 Dev C++ 中，無論指標指向何種資料型態，指標變數本身均佔有 4 個位元組(bytes)。



宣告格式：

型態 \*指標變數;

範例

```

int *ptri;          /* 整數型態之指標變數 */
char *ptrch;        /* 字元型態之指標變數 */

sizeof(ptri) → 4 bytes
sizeof(ptrch) → 4 bytes
  
```

### 二、指標運算子

1. 位址運算子 &：用來取得變數或陣列元素在記憶體中的位址。
2. 依址取值運算子 \*：用來取得指標所指向的記憶體位址的內容。

範例

```

int a = 10, b;
int *p;

p = &a;
b = *p;
*p = 20;

結果：a = 20, b = 10
  
```



### 三、指標的運算

1. **設定運算**：將等號右邊的值設定給左邊的指標變數。
2. **加/減法運算**：針對各個資料型態的長度來處理位址的加減法運算。
3. **差值運算**：計算兩個指標之間的距離，其單位為資料型態的長度。

範例

```

int a=10, b=20;
int *p1, *p2;
char ch='a', *pch;

/* 設定運算 */
p1 = &a;           /* 將 a 的位址存放於 p1 */
p2 = &b;           /* 將 b 的位址存放於 p2 */
pch = &ch;         /* 將 ch 的位址存放於 pch */

/* 加減法運算 */
p1++;              /* 將 p1 中的位址值加上 4 bytes (int 型態的大小) */
pch--;             /* 將 pch 中的位址值減去 1 byte (char 型態的大小) */

/* 差值運算 */
sub = p1 - p2;     /* 計算 p1 和 p2 相差的距離 (以 int 為單位的距離) */

```

#### 【指標的簡潔運算式】

```

int X;
int A[5] = {10,20,30,40,50};
int *p = A + 2;

```

運算式	同義	所執行的敘述及順序	執行後	
			X	*p
X = *(p++);	X = *p++;	X = *p; p = p + 1;	30	40
X = *(++p);	X = *++p;	p = p + 1; X = *p;	40	40
X = (*p)++;		X = *p; *p = *p + 1;	30	31
X = ++(*p);		*p = *p + 1; X = *p;	31	31
X = *(p--);	X = *p--;	X = *p; p = p - 1;	30	20
X = *(--p);	X = *--p;	p = p - 1; X = *p;	20	20
X = (*p)--;		X = *p; *p = *p - 1;	30	29
X = --(*p);		*p = *p - 1; X = *p;	29	29

#### 四、指標與函數

函數的 return 敘述只能有一個回傳值，當程式需要傳遞兩個以上的值時，可以利用指標解決在函數間傳遞多個回傳值的問題。其做法是將指標當作引數傳入函數中，由於指標內的值是所指向變數的位址，因此不須經過 return 敘述即可更改變數的值。

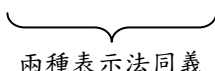
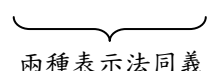
範例

```
void swap(int *, int *);      /* 函數原型，參數為兩個整數型態的指標變數 */
int main(void)
{
    int a=3, b=5;
    swap(&a, &b);            /* 傳遞 a 和 b 的位址 */
    return 0;
}
void swap(int *x, int *y)    /* 此函數用來交換 x、y 所指向的變數之值 */
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

#### 五、指標與陣列的關係

陣列可看成是指標的分身，陣列元素的排列可利用指標運算來存取。

```
int a[3] = {5,7,9};
```

指標的指向	陣列註標	陣列內容	記憶體位址	陣列元素位址	指標的移位
<code>*(a+0)</code>	<code>a[0]</code>	5	0253FDC8	<code>&amp;a[0]</code>	<code>a+0</code>
<code>*(a+1)</code>	<code>a[1]</code>	7	0253FDCC	<code>&amp;a[1]</code>	<code>a+1</code>
<code>*(a+2)</code>	<code>a[2]</code>	9	0253FDD0	<code>&amp;a[2]</code>	<code>a+2</code>
					

注意：陣列 a 以指標的方式表示時，a 會被視為指標常數，所以不可寫成 `a++`。

## 六、指標陣列

陣列中存放的變數為指標變數，即為指標陣列。

宣告格式：

型態 \*陣列名稱[個數];

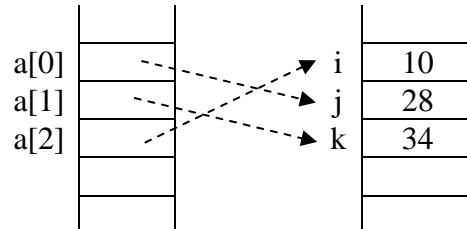
```
int i=10,j=28,k=34;
```

```
int* a[3];
```

```
a[0] = &j;
```

```
a[1] = &k;
```

```
a[2] = &i;
```



### 【比較】字串陣列 v.s. 指標陣列

(1) 字串陣列：`char name[3][10] = {"David", "Jane Wang", "Tom Lee"};`

name[0]	D	a	v	i	d	\0				
name[1]	J	a	n	e		W	a	n	g	\0
name[2]	T	o	m		L	e	e	\0		

(2) 指標陣列：`char *name[3] = {"David", "Jane Wang", "Tom Lee"};`

name[0]	D	a	v	i	d	\0				
name[1]	J	a	n	e		W	a	n	g	\0
name[2]	T	o	m		L	e	e	\0		

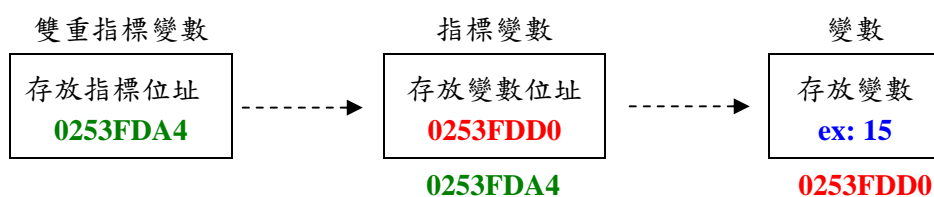
※ 利用指標陣列可節省浪費的記憶體空間。

## 七、雙重指標 - 指向指標的指標

指標變數中若是存放另一個指標變數的位址，這種指向指標的指標稱為雙重指標。

宣告格式：

型態 \*\*指標變數;

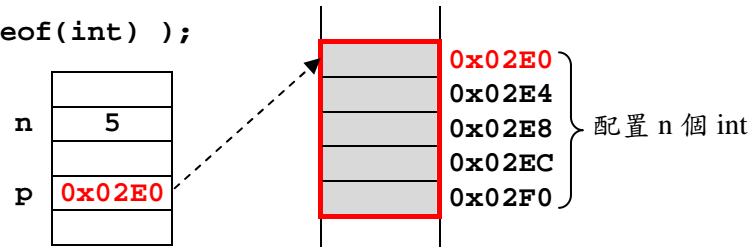


## 八、動態配置記憶體

一維配置

```
int n=5;
int* p;

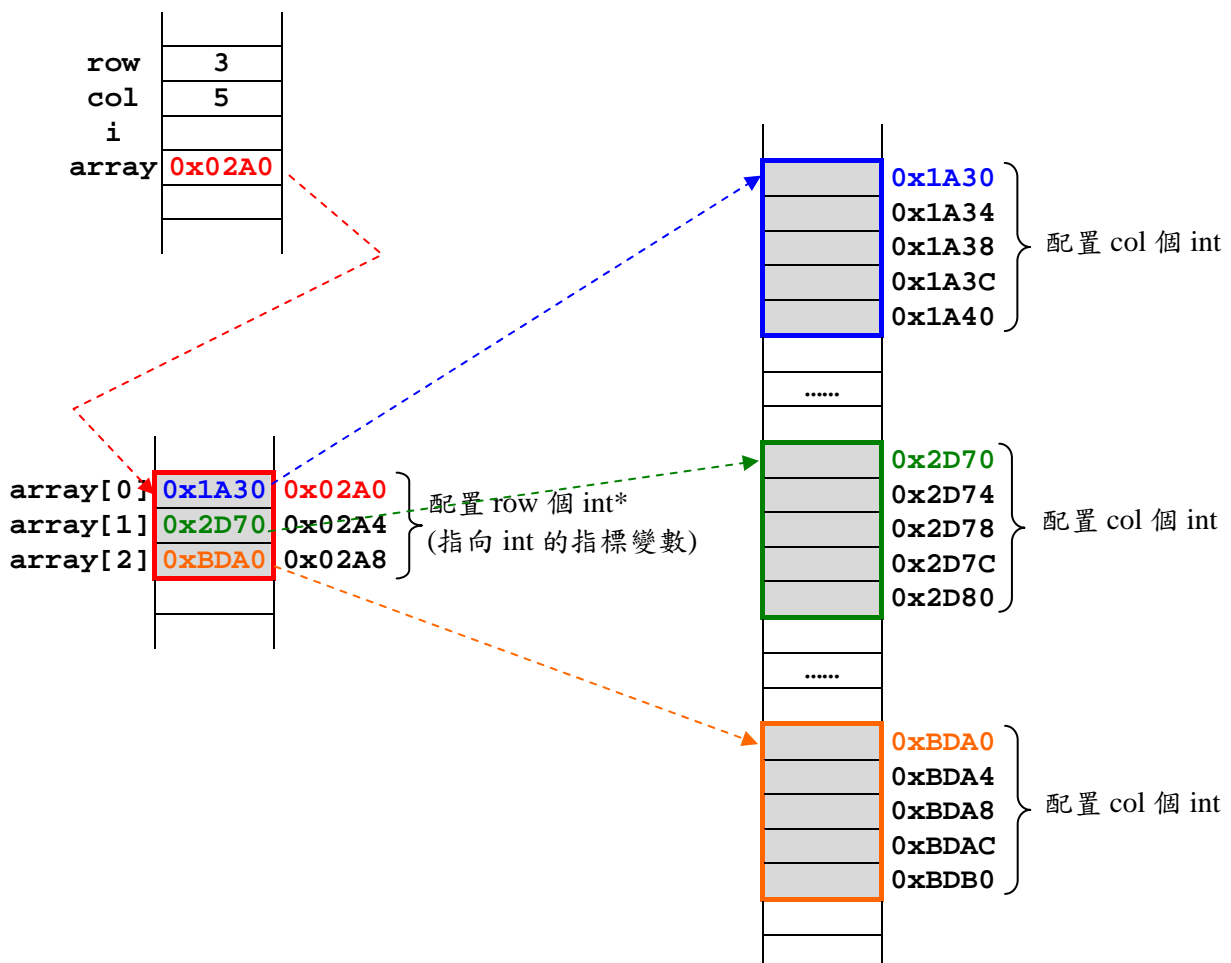
p = (int*) malloc( n * sizeof(int) );
```



二維配置

```
int row=3,col=5,i;
int** array;

array = (int**) malloc(row * sizeof(int*));
for (i=0; i<row; i++)
    array[i] = (int*) malloc(col * sizeof(int));
```



## 單元 13：結構與其他資料形態

### 一、結構 (Structure)

格式：

```
struct 結構名稱
{
    資料型態 欄位名稱 1;
    資料型態 欄位名稱 2;
    ...
};
```

#### 【結構的宣告及使用範例】

<pre>struct mydata          /* 結構名稱 */ {     char name[15];      /* 欄位 */     int score;          /* 欄位 */ };  /* 宣告結構變數並設定初值 */ struct mydata teacher = {"Apric", 90};</pre>	<pre>struct mydata          /* 結構名稱 */ {     char name[15];      /* 欄位 */     int score;          /* 欄位 */ } teacher = {"Apric", 90}; /* 宣告結構變數並設定初值 */</pre>
<pre>#include &lt;stdio.h&gt; /* 公用的結構通常定義為外部變數的型式 */  int main(void) {     struct mydata student;          /* 宣告結構變數 */      printf("Name: ");     scanf("%s", student.name);     printf("Score: ");     scanf("%d", &amp;student.score);     printf("%s got %d points! \n", student.name, student.score);     return 0; }</pre>	

### 二、巢狀結構

```
struct date          /* 結構名稱 */
{
    int month;
    int day;
};
```

範例

```

struct newdata          /* 結構名稱 */
{
    char name[15];
    struct date birthday; /* 結構變數 */
    int score;
};

int main(void)
{
    struct newdata student = {"Apric", {7, 10}, 90};

    printf("%s's birthday is %d/%d \n",
           student.name, student.birthday.month, student.birthday.day);
    printf("He/She got %d points! \n", student.score);
    return 0;
}

```

### 三、結構陣列

```

struct mydata student[10];

for(i=0; i<10; i++)
    printf("%s got %d points! \n", student[i].name, student[i].score);

```

範例

### 四、結構指標

```

struct mydata student[10];          /* 結構陣列 */
struct mydata *ptr = student;      /* 結構指標，初值為 student 陣列的起始位址 */

for (i=0;i<10;i++)
{
    printf("Name, Score:");
    scanf("%s, %d", (student+i)→name, &(student+i)→score);
    printf("%s got %d points!\n", ptr→name, ptr→score);
    ptr++;
}

```

範例

### 五、結構與函數

```

...
void get_data(struct mydata *p);          /* 函式原型，參數為結構指標 */
void print_data(struct mydata a);         /* 函式原型，參數為結構變數 */

int main(void)
{
    struct mydata student;

```

範例

```

    get_data(&student);
    print_data(student);
    return 0;
}

void get_data(struct mydata *p)
{
    printf("Name:");
    scanf("%s", p->name);
    printf("Score:");
    scanf("%d", &p->score);
}

void print_data(struct mydata a)
{
    printf("%s got %d !\n", a.name, a.score);
    return;
}

```

## 六、自訂型態 (typedef)

格式：

**typedef** 資料型態 識別字;

typedef	#define
由編譯器執行	由前置處理器主導
<pre>typedef int clock; clock hour, second;</pre>	<pre>#define CLOCK int CLOCK hour, second;</pre>

利用 typedef 自訂新的結構型態：

```

typedef struct
{
    int minite;
    float second;
} time;

time record = {3, 27.25};

```

## 單元 14：檔案

---

### 一、檔案儲存在記憶體的形式：

類型	儲存方式	儲存單位	檔案大小	檔案類型
<b>文字檔</b> <b>text file</b>	以 ASCII 碼儲存	每個字元皆佔有 1 個位元組， 如數值 132956 為 6 個字元	較大 (資料相同時)	文字資料
<b>二進位檔</b> <b>binary file</b>	以二進位的格式儲存	以資料型態的長度為儲存單位， 如整數 132956 在 Dev C++ 中佔有 4 個位元組	較小 (資料相同時)	圖形檔、聲音檔、影像檔

### 二、檔案存取模式

代碼	存取模式
<b>r</b>	讀取舊檔
<b>w</b>	覆寫新舊檔
<b>a</b>	附加新舊檔
<b>r+</b>	讀取、覆寫舊檔
<b>w+</b>	讀取、覆寫新舊檔
<b>a+</b>	讀取、附加新舊檔
<b>rb</b>	讀取二進位檔
<b>wb</b>	覆寫二進位檔
<b>ab</b>	附加二進位檔



## 三、檔案處理函數（有緩衝區）

變數宣告：

```
FILE *file;
char ch;
char buffer[128];
```

名稱	功能	使用範例
<b>fopen</b>	開檔	<code>file = fopen("C:\abc.txt", "r");</code>
<b>fclose</b>	關檔	<code>fclose(file);</code>
<b>getc</b>	讀取字元	<code>ch = getc(file);</code>
<b>putc</b>	寫入字元	<code>putc(ch, file);</code>
<b>fgets</b>	讀取字串	<code>fget(buffer, 128, file);</code>
<b>fputs</b>	寫入字串	<code>fputs(buffer, file);</code>
<b>fprintf</b>	格式化輸出	<code>fprintf(file, "%c \n", ch);</code>
<b>fscanf</b>	格式化輸入	<code>fscanf(file, "%c", &amp;ch);</code>
<b>fread</b>	區塊輸入	<code>fread(buffer, sizeof(char), 128, file);</code>
<b>fwrite</b>	區塊輸出	<code>fwrite(buffer, sizeof(char), 128, file);</code>
<b>feof</b>	檢查是否結束	<code>while( !feof(file) )     ch = getc(file);</code>
<b>ferror</b>	檢查錯誤	<code>if ( ferror(file) )     printf("error");</code>
<b>fseek</b>	移動檔案指標位置	<code>fseek(file, 128, SEEK_SET);</code>

函數的格式說明請參見課本 12-7, 12-8

## 四、命令列參數的使用

## ■ 語法

<pre>main(argc, argv) int argc; char *argv[]; {     ... }</pre>	<pre>main(int argc, char* argv[]) {     ... }</pre>
-----------------------------------------------------------------	-----------------------------------------------------

## ■ 參數說明

1. **argc** (argument count)：記錄參數個數
2. **argv** (argument value)：記錄參數值  
 argv[0] 記錄程式名稱，後面接續的參數依序指定給 argv[1]、argv[2]、...

## ■ 命令範例

```
type 123 abc.txt
```

```
argc = 3;
argv[0] = type (程式名稱)
argv[1] = 123
argv[2] = abc.txt
```

## ■ 程式範例

```
#include <stdio.h>
int main (int argc, char* argv[])
{
    int i;
    printf("The value of argc is %d \n", argc);
    for(i=0; i<argc; i++)
        printf("argv[%d]=%s \n", i, argv[i]);
    return 0;
}
```