

# 3

## CHAPTER

# Arduino C 語言介紹

## 本章單元

- C 語言資料型態與運算式
- C 語言指令操作
- C 語言函式庫操作
- 多個 C 語言程式的編譯
- Arduino 指令及函數庫

C 語言是國際上最通用的高階語言，可用來撰寫各種電腦的系統程式及一般的應用程式。除此之外，它能夠對電腦的硬體直接進行操作，同時對程式的表達及運算能力也較強，以往許多使用組合語言才能解決的問題，現在都可以用 C 語言來處理。可見它是一種高效率、高可讀性及高可攜性(可用於不同的 CPU)的程式語言。

Arduino IDE 所推出 C 語言編譯器(compiler)可應用於 C 及 C++具有 32-bit 變數及浮點數功能，和標準傳統的 ANSI C 也會有一些差異。

### 3-1 資料型態與運算式

C 語言的資料可分為常數(constant)及變數(variable)兩項，其中常數資料存放在 ROM，程式執行過程中不能改變。而變數資料存放在 RAM，程式執行過程中可任意改變。再加上運算式(expression)內的運算子(operators)及運算元(operand)，即可組成一個 C 語言程式。

#### 3-1.1 常數及變數資料

常數及變數的資料有整數(integer)、浮點數(floating point number)、字元(character)和字串(string)等。分別說明如下：

1. 整數(integer)：可分下列幾種格式，如下表(a)(b)所示：

表 3-1(a) 十進制/十六進制/二進制格式

整數常數	說明	範例
十進制整數	一般習慣用的數字	如 1234，-5678，0
十六進制整數	須在數字前加“0x”	如 0x0A，0x1A(相當於十進制數 10，26)
二進制整數	須在數字前加“0b”	如 0b00010111 (相當於十進制數 23)

表 3-1(b) 十進制/十六進制/二進制轉換

十進制	十六進制	二進制
0	0x0	0b0000
1	0x1	0b0001
2	0x2	0b0010
3	0x3	0b0011
4	0x4	0b0100
5	0x5	0b0101
6	0x6	0b0110
7	0x7	0b0111
8	0x8	0b1000
9	0x9	0b1001
10	0xA	0b1010
11	0xB	0b1011
12	0xC	0b1100
13	0xD	0b1101
14	0xE	0b1110
15	0xF	0b1111

## 2. 浮點數(floating point number)：浮點數有十進制和指數式。

- (1) 十進制式：由整數和小數點組成，如果整數或小數部分為 0 時，可以省略不寫，但小數點必須存在。例如：

0.1234	.1234	1234.1	1234.	0.0
--------	-------	--------	-------	-----

- (2) 指數式的格式：用 “e” 來代表 10 的幾次方，如  $1.2e3=1.2*10^3=1200$ ，格式如下：

[±] 數字 [.數字] e [±] 數字
-----------------------

其中 [ ] 內可有可無，範例如下：

合法的指數式浮點常數：112e3 、 6e5 、 -8.0e-4
不合法的型式：e4 、 2e2.3 、 e

- (3) 浮點數分為 32-bit 的單精度(float)為及 64-bit 的倍精度(double)。

## 3. 字元(character)：

- (1) 字元常數為單引號 ‘ ’ 內的字元，如 ‘A’、‘B’、‘a’、‘b’、‘0’、‘1’。
- (2) 字元常數必須設定初始值，否則編譯會產生錯誤。
- (3) 字元資料是以 ASCII(美國標準資訊交換碼)來表達，如下表所示：

表 3-2 常用 ASCII 字型表

低 4 位元	高 4 位元							
	0	1	2	3	4	5	6	7
0	\0			O	@	P	\	p
1			!	1	A	Q	a	q
2			”	2	B	R	b	r
3			#	3	C	S	c	s
4			\$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v



7	\a		,	7	G	W	g	w
8	\b	↑	(	8	H	X	h	x
9	\t	↓	)	9	I	Y	i	y
A	\n	→	*	:	J	Z	j	z
B	\v	ESC	+	:	K	[	k	{
C	\f	└	,	<	L	Y	l	「
D	\r		-	=	M	]	m	}
E			.	>	N	^	n	→
F			/	?	O	_	o	←

- (4) 若前面加上反斜線表示為控制字元，它用於輸出的格式設定及其它特殊功能，常用的控制字元如下表所示。

表 3-3 常用控制字元表

控制字元	動作	ASCII 碼
\0	空字元(NULL)	0x00
\a	嗶聲(bell)	0x07
\b	倒退(BS)	0x08
\t	水平跳格(HT)	0x09
\n	換行(LF)	0x0A
\v	垂直跳格(VT vertical tab)	0x0B
\f	換頁(FF)	0x0C
\r	歸位(CR)	0x0D

- (5) 字元與整數之間轉換：如下表所示。

表 3-4 字元與整數的轉換

字元轉換成數字	動作	範例
字元(0~9)→數字 0~9	字元(0~9)-'0'=0~9	'8'-'0'=0x38-0x30=8
字元(A~F)→數字 A~F	字元(A~F)-'0'-7=0x0A~0x0F	'B'-'0'-7=0x42-0x30-7=0x0B
字元(a~f)→數字 A~F	字元(a~f)-'0'-0x27=0x0A~0x0F	'b'-'0'-0x27=0x62-0x30-0x27=0x0B

- (a) 字元(0~9)的 ASCII 碼為 0x30~0x39，減 0x30(字元-'0')後成為數字

(0~9)。

(b) 字元(A~F)的 ASCII 碼為 0x41~0x46，減 0x37(字元(A~F)-'0'-0x37)後成為 16 進制數字(0x0A~0x0F)。

(c) 字元(a~f)的 ASCII 碼為 0x61~0x66，減 0x57(字元(a~f)-'0'-0x27)後成為 16 進制數字(0x0A~0x0F)。

(6) 字元與整數之間轉換範例程式(.\CH03\_C\C01)：開啓序列埠監控窗，來觀察其變數的動作，如下圖所示：

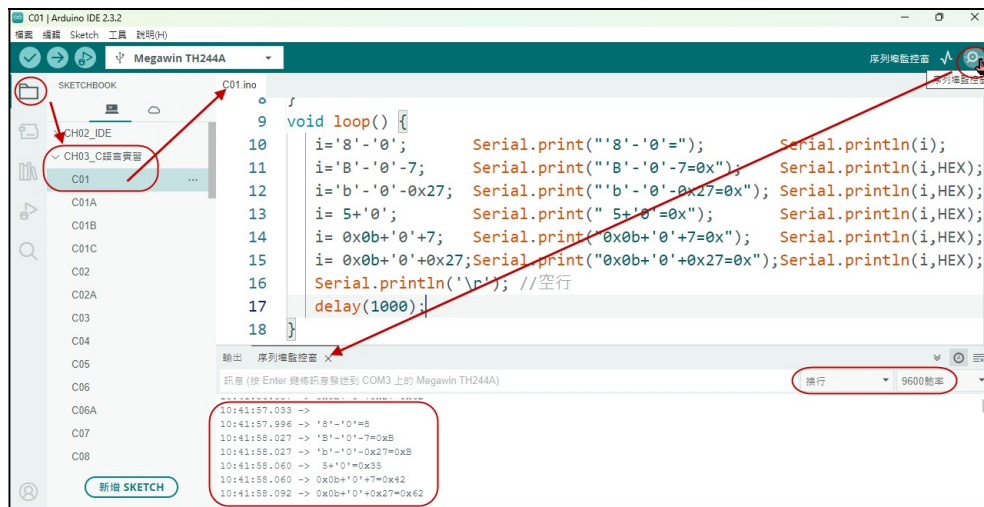


圖 3-1 字元與整數轉換範例

範例程式說明如下：

```

//*****C01 *****
//動作：字元與整數之間轉換
//*****

byte i;
void setup() {
  Serial.begin(9600); //啟動序列埠，速度為 9600-bps

```

```
    delay(2000); //延時
}
void loop() {
    i='8'-0;      //字元(8)轉換為數字(8)
    Serial.print("'8'-0="); Serial.println(i);

    i='B'-0-7; //字元(B)轉換為數字(0x0B)
    Serial.print("'B'-0-7=0x"); Serial.println(i,HEX);

    i='b'-0-0x27; //字元(b)轉換為數字(0x0B)
    Serial.print("'b'-0-0x27=0x"); Serial.println(i,HEX);

    i= 5+'0;      //數字(5)轉換為字元(5)，ASCII 碼=0x35
    Serial.print(" 5+'0'=0x"); Serial.println(i,HEX);

    i= 0x0b+'0'+7; //0x0B 轉換為字元(B)，ASCII=0x42
    Serial.print("0x0b+'0'+7=0x"); Serial.println(i,HEX);

    i= 0x0b+'0'+0x27; // 0x0B 轉換為字元(b)，ASCII=0x62
    Serial.print("0x0b+'0'+0x27=0x");Serial.println(i,HEX);

    Serial.println("\r"); //空行
    delay(1000); //延時
}
```

#### 4. 字串(string)：

- (1) 字串常數須在字串兩側加上雙引號(“字串“)，如”\$14ABCD”。
- (2) 當雙引號內沒有字元時，如” ”稱為空字串。
- (3) C 語言將字串常數作為「一組字元」來處理，在儲存字串常數時會自動在後面加 \0 表示為該字串常數的結束碼。

5.列舉常數(enum)：只能設定整數型態-128~127。

(1) 每個列舉常數可個別給予數值，如下：

```
enum {PORTA=2, PORTB=14, PORTC=57};
```

(2) 當只有設定第一個列舉常數值時，後續的列舉常數將依序加 1。若都不設定列舉常數值則由 0 開始，後續的列舉常數依序加 1，如下：

```
enum {ONE=1, TWO,THREE}; // ONE=1, TWO=2,THREE=3
```

(3) 列舉常數範例程式(\CH03\_C\C01A)。

### 3-1.2 常數及變數名稱

C 語言常用有意義的名稱來代表常數、變數或標記位址，此名稱可以使用英文字母、數字或底線，不過需要遵守下列規則，否則編譯時會產生錯誤。

1. 名稱第一個字元限制使用英文字母或底線 “\_”。
2. 名稱第二個字元以後，可使用英文字母、底線 “\_”或數字，不可有符號字元(如+、-、\*、/、%、^及!、@、#、\$、&等)。
3. 名稱的英文字母的大小寫會有區分，如 Loop 與 loop 不相同。
4. C 語言編譯器(compiler)的關鍵字用於指令或宣告，不可作為常數、變數或標記(lable)，這些關鍵字一般是 C 語言指令，關鍵字如下表所示。

表 3-5 C 語言關鍵字功能

關 鍵 字	用 途	說 明
auto	記憶種類宣告	預定值宣告區域變數
break	程式指令	退出最內層迴圈
byte	資料型態宣告	宣告 8-bit 無符號整數資料型態
case	程式指令	switch 指令中的選擇項
char	資料型態宣告	宣告 8-bit 字元資料型態
const	資料型態宣告	宣告常數
continue	程式指令	換下一個迴圈
default	程式指令	switch 指令中的“否”選擇項
do	程式指令	組成 do...while 迴圈程式
else	程式指令	組成 if...else 選擇程式
enum	列舉資料型態宣告	
extern	資料型態宣告	宣告變數在外部程式中
for	程式指令	組成 for 迴圈程式
goto	程式指令	跳躍程式

關 鍵 字	用 途	說 明
if	程式指令	組成 if...else 選擇程式
int	資料型態宣告	宣告 16-bit 整數資料
long	資料型態宣告	宣告 32-bit 長整數資料
return	程式指令	由函數返回
short	資料型態宣告	宣告 16-bit 短整數資料
signed	資料型態宣告	宣告有符號的變數
static	資料型態宣告	靜態變數
struct	資料型態宣告	結構型態資料
switch	程式指令	組成 switch 選擇程式
typedef	資料型態宣告	重新定義資料型態
union	資料型態宣告	聯合型態資料
unsigned	資料型態宣告	無符號資料
void	資料型態宣告	無回傳資料
volatile	資料型態宣告	宣告該變數在程式執行中無最佳化
while	程式指令	組成 while 和 do...while 迴圈程式
word	資料型態宣告	宣告 16-bit 無符號整數資料型態

4. Arduino C 語言的資料處理，如常數(Constants)、資料型態(Data types)、變數觀察及預處理(Varilabe scope & qualifiers)、資料轉換(Conversion)、utilities(公用)，如下表所示。

表 3-6 Arduino C 語言的資料處理

Constants (常數)	Data Types (資料型試)	Variable Scope & Qualifiers (變數觀察及預處理)
Floating Point Constants	array	const
HIGH   LOW	bool	scope
INPUT   INPUT_PULLUP   OUTPUT	boolean	static
Integer Constants	byte	volatile
LED_BUILTIN	char	
true   false	double	
	float	Utilities (公用)
	int	PROGMEM
Conversion (資料轉換)	long	sizeof()
(unsigned int)	short	
(unsigned long)	size_t	
byte()	string	
char()	String()	
float()	unsigned char	
int()	unsigned int	
long()	unsigned long	
word()	void	
	word	

### 3-1.3 變數的資料型態

變數在程式執行過程中，其資料會不斷的變化。此變數必須事先加以定義，以便編譯器能為它在 RAM 分配記憶空間。變數定義的格式如下：(△：空格)

```
[volatile] [變數型態] △資料型態△[記憶型態] △變數名稱；
```

其中[volatile]、[變數型態] 和[記憶型態]可省略。

1. volatile：程式中最好以 `volatile` 來宣告變數表示編譯時無最佳化，否則程式指令(如 `while`)執行時，變數的內容可能會有誤。
2. 資料型態(Data type)：它用來表示數值資料的大小範圍，如下所示：

```
資料型態 變數名稱；
```

- (1) 變數的資料型態這些變數會被編譯器規劃在 RAM 內，變數的資料型態的種類有 `bool`、`boolean`、`char`、`type`、`int`、`short`、`word`、`long`、`long long`、`float` 和 `double` 變數。

同時整數分為 `unsigned`(無正負符號)與 `signed`(有正負符號)，其中 `unsigned` 資料內所有位元均表示數值，而 `signed` 內最高 bit 表示正負值(0=正數，1=負數)，若是負數會以 2 的補數(2'S)來表達。

另外變數在 ARM 的資料型態：有 `int8_t`、`uint8_t`、`int16_t`、`uint16_t`、`int32_t`、`uint32_t`、`int64_t` 和 `uint64_t` 其中後面的數字為變數的 bit 數。變數的範例如下：

- (a) 定義 1-bit 變數：有 `bool` 及 `boolean`，數值限制 0 或 1。



```
bool      a=0;           //宣告 1-bit 變數 a=0
boolean   b=1;           //宣告 1-bit 變數 b=1
```

- (b) 有符號 8-bit 變數：表示字元或 ASCII 碼，資料範圍為-128~127。

```
signed char c=65; //宣告字元變數 c=65=字元 A
signed char c='A'; //宣告字元變數 c=字元 A
int8_t c=-5; //宣告有符號 8-bit 變數 c=-5
```

- (c) 無符號 8-bit 整數變數：表示資料範圍為 0~255。

```
unsigned char d=200;
byte d=200;
uint8_t d=200;
```

- (d) 有符號 16-bit 短整數變數：表示資料範圍為-32768~+32767。

```
signed short int e=-32760;
int16_t e=-32760;
```

- (e) 無符號 16-bit 短整數變數：表示資料範圍為 0~65535。

```
unsigned short int f=65530;
int16_t f=65530;
```

- (f) 有符號 32-bit 整數變數：資料範圍為-2147483648~+2147483647。

```
signed int g;
int32_t g;
```

- (g) 無符號 32-bit 整數變數：表示資料範圍為 0~0xFFFF\_FFFF。

```
unsigned int h=0xFFFFFFFF0;
long h=0xFFFFFFFF0;
```

```
word h=0xFFFFFFFF0;
uint32_t h=0xFFFFFFFF0;
```

(h) 有符號 64-bit 長整數變數：

```
signed long long count;
int64_t count;
```

(i) 無符號 64-bit 長整數變數：

```
unsigned long long count;
uint64_t count;
```

(j) float 變數：用來表示 32-bit 單精度浮點數資料。

(k) double 變數：用來表示 64-bit 倍精度浮點數資料。

(l) 變數具有循環特性，以 unsigned char 及 signed char 為例。如下圖所示：

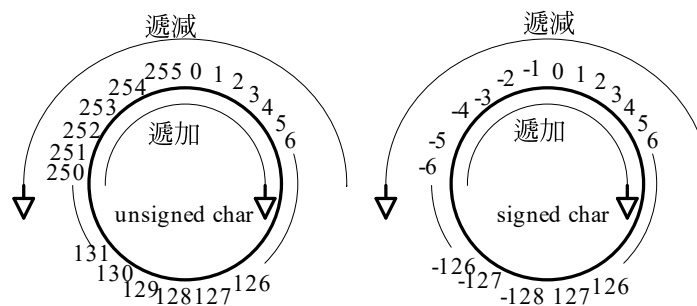


圖 3-2 變數的循環特性

unsigned char 的範圍為 0~255，若設定 unsigned char i=256 則實際內容為 i=0。signed char 的範圍為-128~127，若設定 signed char i=-129

則實際內容為  $i=127$ 。同時在程式中遞加或遞減運算時，也會有這種現象。

(m) 變數的宣告及應用範例程式(..\CH03\_C\C01B)。

(2) 重新定義變數的型態：

C 語言所定義變數的 bit 數，在不同的編譯器會有所差異，爲了避免影響 C 語言的可攜性，可以使用 `typedef` 重新定義變數的型態名稱。如下：

```
typedef unsigned char uint8; //定義 uint8 爲 8-bit 無符號變數
void main()
{
    uint8 i=0; //宣告 8-bit 無符號變數 i=0
    loop:      //標記
        i=i+1; //i 遞加
        goto loop; //跳到標記 loop 處
}
```

範例程式：重新定義變數的型態範例程式(..\CH03\_C\C01C)。

3. 變數型態，如下所示：

[變數型態] 資料型態 變數名稱；

變數型態表示變數存在的特性，它可設成四種：`auto`(自動)、`static`(靜態)和 `extern`(外部)。定義一個變數時，若省略[變數型態]時，該變數將預定爲 `auto` 變數。

(1) 自動(`auto`)變數：由編譯器來決定，預定爲動態變數，表示該變數的記憶空間不確定，在程式執行期間根據需要，動態地爲該變數分配記憶空間。每次進入函數時，都會重新宣告自動(動態)變數的數值，此動態變數在退出函數時會無作用，以便讓出 RAM 空間給其它函數使用，如

此可節省 RAM 的空間。動態變數範例程式(..\CH03\_C\C02)。

- (2) 靜態(static)變數：在變數之前若是加上 `static` 來宣告，表示該變數在程式執行期間其記憶空間為固定不變，且不得設定初始值：

```
static char i; //宣告靜態變數
```

靜態變數範例程式(..\CH03\_C\C02A)。

- (3) 外部(extern)函數及變數：若是在宣告函數或變數之前加上 `extern`，表示該函數或變數是在外部檔案，必須和外部該程式檔案連結後才有作用。

宣告如下：

```
extern void inc(void); //宣告函數在外面
extern char i;         //宣告外部變數
```

外部變數範例程式(..\CH03\_C\C03)。

#### 4. 區域變數(local variable)和全域變數(global variable)：

以變數的作用範圍來看，可分為區域變數和全域變數：

- (1) 區域變數是在函數內部或是大括號{ }內所定的變數，只在內部有效。
- (2) 全域變數表示是在函數以外(即大括號{ }以外)所定義的變數，若是在主函數 `main()` 上方所定義的全域變數，對於整個程式都有效，可提供程式中所有函數共同使用。而在各函數外面所定義的全域變數，只能對該處以後的各個函數有效。

- (3) 區域變數和全域變數的範例程式(..\CH03\_C\C04)。

#### 5. 修飾詞(qualifier)：其中 `const` 為記憶型態，如下所示：

```
[const] 資料型態 變數名稱
```

- (1) 記憶型態表示資料存放在 RAM 或 ROM，若省略時預定為 RAM。

(2) 加上 `const` 表示為常數放置於 **ROM** 內，且執行中不允許更動資料。它可以在函數內部宣告，且必須設定初始值。例如：

```
const long count=0x12345678; //宣告 32-bit 常數在 ROM 內
char const count=25;          //宣告 8-bit 常數在 ROM 內
```

#### 6. 陣列變數：

陣列變數將同一類型的資料整合在一起，以序號來表示儲存的空間。用序號來管理資料，對於大量資料的處理較為方便。資料型態如下：

◎一維陣列變數：容量為  $n$ 。

變數[0]	變數[1]	變數[2]	變數[3]	變數[...]	變數[n-1]
-------	-------	-------	-------	---------	---------

◎二維陣列變數：容量為  $(i) \times (j)$ 。

變數[0,0]	變數[0,1]	變數[0,2]	變數[0,3]	變數[0,i-1]
變數[1,0]	變數[1,1]	變數[1,2]	變數[1,3]	變數[1,i-1]
變數[2,0]	變數[2,1]	變數[2,2]	變數[2,3]	變數[2,i-1]
變數[j-1,0]	變數[j-1,1]	變數[j-1,2]	變數[j-1,3]	變數[j-1,i-1]

(1) 宣告陣列變數，如下：

```
byte TAB[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80}; //宣告陣列變數
```

陣列資料的指定存放位置，內容如下：

TAB[0]	TAB[1]	TAB[2]	TAB[3]	TAB[4]	TAB[5]	TAB[6]	TAB[7]
0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80

讀取陣列資料範例程式(..**CH03\_C**\C05)。

(2) 宣告陣列常數時，會將常數資料存放在 **ROM** 內，此時可將「記憶型態」設定在 `const`，如此可以有較大的空間來存放陣列資料，宣告時就要設定初始值。

- (3) 讀取陣列資料的範例程式(..\CH03\_C\C06)。
- (4) 將數值寫入陣列的範例程式(..\CH03\_C\C07)。

### 3-1.4 C 語言的運算式與運算子

C 語言的運算式分為運算元及運算子，可用來表示輸入與輸出的關係，如下所示：

運 算 式		
i	=	1
運算元(輸出)	運算子	運算元(輸入)

1. 設定運算子：以 “=” 來表示，如下：

```
i=9;    //將 9 送到變數 i
j=k=8;  //將 8 同時送到變數 j 和 k
```

2. 逗號運算子：以逗號 “,” 可將兩個以上的變數連在同一行，例如：

```
byte i,j,k;
```

3. 一元運算子：它只須要一個運算元，如下表所示：

表 3-7 一元運算子

一元運算子		範例	說明
+	正號	+2	
-	負號	-5	取數字的 2 補數(2'S)
!	NOT，否	bit=!bit (bit 反相處理)	令 bit 變數反相
~	取 1 的補數	temp=~i (資料反相輸出)	若 i=0x01，則 temp=0xFE

一元運算子的範例程式(..\CH03\_C\C08)。

4. 算術運算子：算術運算子的優先順序是：先正負→乘除→加減，若同級則由左向右運算，如下表所示：

表 3-8 算術運算子

優先	算術運算子	範例	浮點數運算結果	整數運算結果
1	- 負數運算	-5.3+1	-4.3	-4 (捨去小數)
2	* 乘運算	2.1*2	4.2	4 (捨去小數)
3	/ 除運算	5/3	1.6666	1 (捨去小數)
4	% 餘數運算	5%3	2	2 (取餘數)
5	+ 加運算	5.7+2	7.7	7 (捨去小數取最小值)
6	- 減運算	5.3-1	4.3	4 (捨去小數)

算術運算子的範例程式(..\CH03\_C\C09)。

5. 遞加和遞減運算子：只能用於變數，不能用於常數，如下表所示：

表 3-9 遞加和遞減運算子

遞加和遞減運算子	動作	說明
++j	i= ++j	先執行 j+1，結果再存入 i
j++	i= j++	j 先存入 i，再執行 j+1
--j	i= --j	先執行 j-1，結果再存入 i
j--	i= j--	j 先存入 i，再執行 j-1

遞加和遞減運算子的範例程式(..\CH03\_C\C10)。

6. 比較運算子：C 語言有 6 種比較運算子，可以用於 if、while 及 for 等指令的條件判斷，當條件符合時結果為 1，不符合時結果為 0。如下表所示：



表 3-10 關係運算子

優先	比較運算子		範例
高	>	大於	if(j>3) 如 j>3(不包括 3)則條件成立
高	<	小於	if(j<3) 如 j<3(不包括 3)則條件成立
高	>=	大於等於	if(j>=3) 如 j≥3 則條件成立
高	<=	小於等於	if(j<=3) 如 j≤3 則條件成立
低	==	等於	if(j==3) 如 j=3 則條件成立
低	!=	不等於	if(j!=3) 如 j≠3 則條件成立

比較運算子的範例程式(..\CH03\_C\C11)。

7. 邏輯運算子：C 語言中有 3 種邏輯運算子，大部份用於 if、while 及 for 等指令的條件比較判斷。如下表所示：

表 3-11 邏輯運算子

優先	邏輯運算子		範例
1	!	邏輯 NOT	if (!j) 相當於 if(j==0)，如 j=0 則條件成立
2	&&	邏輯 AND	if (j<6 && j>3)，如 j<6 及 j>3 (限 4,5)，條件成立
3		邏輯 OR	if (j>5    j<4)，如 j>5 或 j<4，(除排 4,5)則條件成立

邏輯運算子的範例程式(..\CH03\_C\C12)。

8. 位元邏輯運算子：能夠對 8、16 或 32-bit 的整數變數進行邏輯運算，但不會改變原有變數的值，它共有 6 種位元邏輯運算子，如下表所示：

表 3-12 位元邏輯運算子

優先	位元邏輯運算子		範例
1	~	邏輯 NOT	i=~j，將 j 的內容反相後，存入 i
2	<<	位元左移	i=j<<3，將 j 的內容左移 3 bit 後，存入 i
3	>>	位元右移	i=j>>3，將 j 的內容右移 3 bit 後，存入 i

4	&	邏輯 AND	$i = j \& 0x03$ ，將 j 的內容和 03 進行 AND 後，存入 i
5	^	邏輯 XOR	$i = j \wedge 0x03$ ，將 j 的內容和 03 進行 XOR 後，存入 i
6		邏輯 OR	$i = j   0x03$ ，將 j 的內容和 03 進行 OR 後，存入 i

邏輯運算動作如下表(a)所示。

表 3-13(a) 位元邏輯運算子

邏輯運算子		~B	A&B	A B	A^B
A 輸入	B 輸入	NOT 輸出	AND 輸出	OR 輸出	XOR 輸出
0	0	1	0	0	0
0	1	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0

綜合上述的動作可歸納特性如下表(b)所示。

表 3-13(b) 邏輯特性(x=1 或 0)

邏輯運算子		A&B	A B	A^B
A 輸入	B 輸入	AND 輸出	OR 輸出	XOR 輸出
0	x	0	x	x
1	x	x	1	x 反相

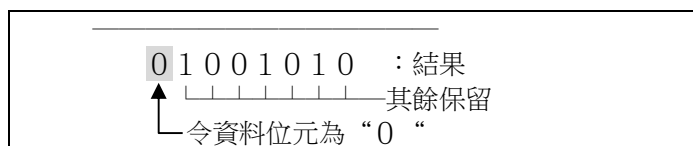
位元邏輯運算子(1)的範例程式(..\CH03\_C\C13)。

(1)NOT 指令：令位元組資料反相。

	1 1 0 0 1 0 1 0	: 資料
NOT	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
	0 0 1 1 0 1 0 1	: 結果

(2)AND 指令：和“0”做 AND 時，會令某位元為“0”

	1 1 0 0 1 0 1 0	: 資料
AND	↓	
	0 1 1 1 1 1 1 1	



(3)OR 指令：和“1”做 OR 時，會令某位元為“1”。



(4)XOR 指令：和“1”做 XOR 動作時，會令某位元反相，其餘保留。



(5)移位指令：有右移(>>)和左移(<<)兩種指令，無符號(unsigned)變數移位時會用 0(反白部份)遞補進來。如下圖所示。

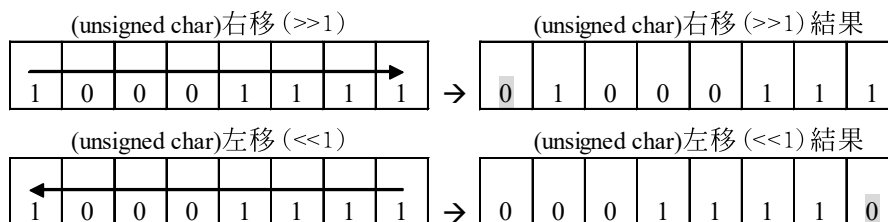


圖 3-3 無符號變數移位

有符號(signed)變數右移位時，會將符號位元往右遞補進來。有符號變

數左移位時，用 0 由右往左遞補進來與無符號相同，如下圖所示。

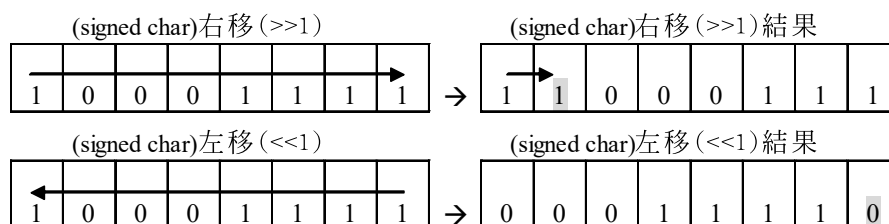


圖 3-4 有符號變數移位

位元邏輯運算子(2)範例程式(..\CH03\_C\C14)。

- (6)位元邏輯運算子可應用於 16-bit 資料高/低位元組的處理，例如 16-bit 的資料必須由兩組 8-bit 的 I/O 埠(如 PA、PB)來輸出。若陣列 16-bit 資料(0x1234)存入到 8-bit 的 P1 時，僅有低 8-bit(0x34)會寫入到 PA 內：

TABLE[i]	0x12	0x34
PA=TABLE[i]		0x34

- (7)若要將 16-bit 陣列資料(0x1234)的高 8-bit 資料(0x12)存入 P2 時，必須將 16-bit 陣列資料(0x1234)右移 8-bit 後成為(0x0012)，再寫入到 PB 內：

TABLE[i]	0x12	0x34
TABLE[i]>>8	0x00	0x12
PB = TABLE[i]>>8		0x12

9. 複合設定運算子：將設定運算子與其它運算子相結合，即可形成複合設定運算子，它可令程式較為簡潔及提高編譯的效率，如下表所示：

表 3-14 複合設定運算子

複合設定	運算子	範例	說明	複合設定	運算子	範例	說明
加法設定	+=	i+=a	i=i+a	左移設定	<<=	i<<=a	i=i<<a
減法設定	-=	i-=a	i=i-a	右移設定	>>=	i>>=a	i=i>>a

乘法設定	<code>*=</code>	<code>i*=a</code>	<code>i=i*a</code>	AND 設定	<code>&amp;=</code>	<code>i &amp;=a</code>	<code>i=i&amp;a</code>
除法設定	<code>/=</code>	<code>i/=a</code>	<code>i=i/a</code>	OR 設定	<code> =</code>	<code>i  =a</code>	<code>i=i a</code>
餘數設定	<code>%=</code>	<code>i%=a</code>	<code>i=i%a</code>	XOR 設定	<code>^=</code>	<code>i ^=a</code>	<code>i=i^a</code>

(1) 複合設定運算子(1)範例程式(.\CH03\_C\C15)。

(2) 複合設定運算子(2)範例程式(.\CH03\_C\C16)。

10. 指標與位址運算子：指標符號“\*”和變數位址符號“&”。

指標本身是一個變數，在這個變數內所存放的不是一般的資料，而是指向另一個變數的位址，指標變數表示方法如下：

```
*point; //表示 point 是一個字元型的指標變數
float *point; //表示 point 是一個浮點型的指標變數
```

指標變數所存放的內容是另一個變數的位址，而符號“&”則用來表示變數 RAM 的位址。這兩者配合使用可以對 RAM 以間接的方式來進行操作。

(1) 指標\*與&範例：表面上將計數值存入\*point，但實際上是存入\*point 所指向的變數 count 內。如下圖所示：

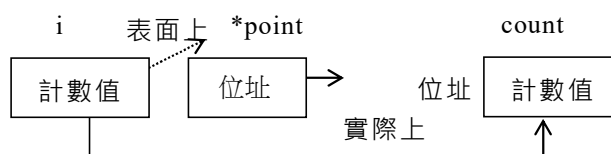


圖 3-5 指標資料存取的动作

指標 \* 與 & 範例程式(.\CH03\_C\C17)。

(2) 指標應用範例：將資料 0xFF 填入 RAM 區塊內，如下圖所示。

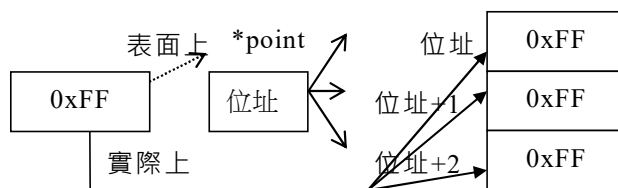


圖 3-6 資料填入 RAM 區塊動作

指標 \* 與 & 範例程式(..\CH03\_C\C18)。

11. `sizeof` 運算子：它並非一個函數，但可用來顯示得資料型態、變數及運算式的字元數，`sizeof` 運算子範例程式(..\CH03\_C\C19)。
14. 資料型態的轉換：當不同資料型態的變數一起運算時，它會強制性的加以轉換。轉換的方式以不流失資料為原則，如下：
  - (1) 較少 bit 變數和較多 bit 變數一起運算時，轉換為較多 bit 變數。
  - (2) 整數變數和浮點變數一起運算時，轉換為浮點變數。
  - (3) 有符號變數和無符號變數一起運算時，轉換為無符號變數。
  - (4) 強制轉換運算式型態的範例程式(..\CH03\_C\C20)。
  - (5) 可強制將有符號變數轉換為無符號變數，或 1-bit 與 8-bit 變數轉換，範例程式(..\CH03\_C\C21)。
15. 運算子的優先順序：將所有運算子的優先順序加以排序，如下表所示。其中「結合性(associativity)」有由左至右及由右至左兩種，它是表示當數個相同優先順序的運算子並列在一起時，可再區分其運算的先後順序。例如：
 

```
a=b/c*6; //算術運算子的結合性為由左至右，所以先 b/c，再*6
```

表 3-15 運算子的優先順序和結合性

優先	類別	運算子名稱	運算子	結合性
1	強制轉換 數組 結構，聯合	強制型態轉換 下標 存取結構或聯合成員	() [] -> 或 .	由左至右
2	邏輯(條件) 位元 遞加,遞減 指標 算術 長度計算	邏輯非(NOT) 反相(NOT) 遞增、遞減 位址運算、指標運算 單一運算元相減 長度計算	! ~ ++、-- &、* - sizeof	由右至左
3	算術	乘、除、取餘數	*、/、%	由左至右
4	算術	加、減	+、-	
5	位元	左移、右移	<<、>>	
6	比較	大於等於、大於 小於等於、小於	>=、> <=、<	
7		等於、不等於	==、!=	
8	位元	位元邏輯(AND)	&	
9		位元邏輯(XOR)	^	
10		位元邏輯(OR)		
11	邏輯(條件)	邏輯運算(AND)	&&	
12		邏輯運算(OR)		由左至右
13	條件	條件運算	?:	由右至左
14	設定	設定、複合設定	=、+=	
15	逗號	逗號運算	,	由左至右

## 3-2 C 語言指令實習

C 語言的指令大部份以迴圈為主，若能先設計流程圖，再來撰寫程式會有較佳的表現，在控制程式中常用的流程圖符號，如下圖所示：

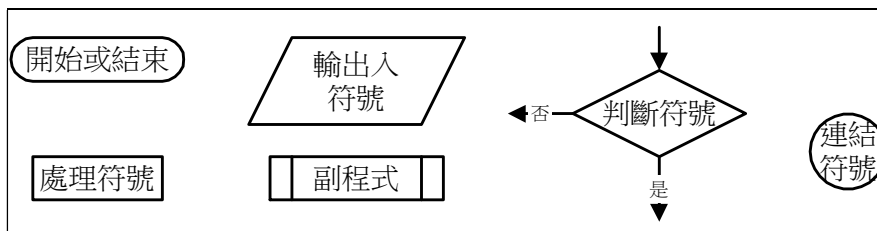


圖 3-7 常用的流程圖符號

### 3-2.1 if 指令實習

if 指令用於條件的判斷，它有幾種用法，如下：

1. if 指令的用法如下圖所示。

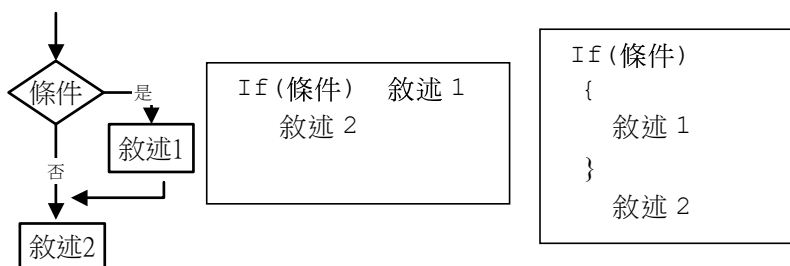


圖 3-8 if 指令用法

假如條件符合時會執行敘述 1 及敘述 2。如不符合則僅執行敘述 2。當敘



述不只一行時，必須加大括號。if 指令用於條件的判斷的範例程式(C22)。

2. if 指令可和 goto 指令配合，如下圖所示。

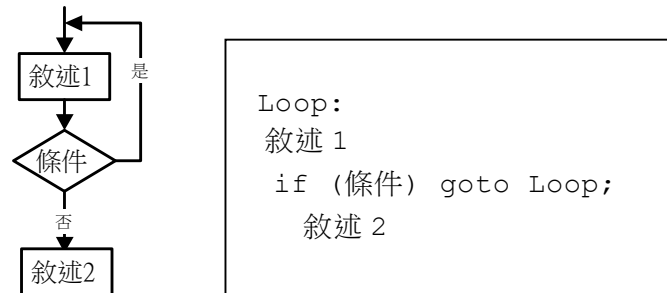


圖 3-9 if 指令和 goto 指令配合

假如條件符合時會重覆執行敘述 1，如不符合則敘述 1 及敘述 2 僅執行一次。if 指令和 goto 指令配合的範例程式(..\CH03\_C\C23)。

3. if 和 else 指令配合，如下圖所示。

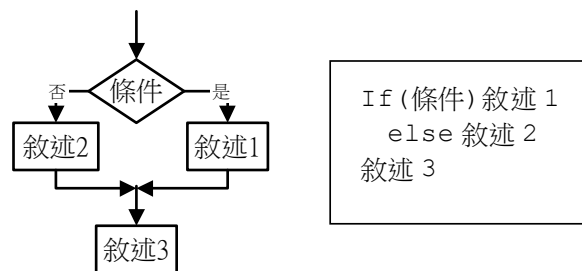


圖 3-10 if 和 else 指令配合

假如條件符合時會僅執行敘述 1，如不符合則僅執行敘述 2，最後都會執行敘述 3。if 指令和 else 指令配合的範例程式(..\CH03\_C\C24)。

4. 變數比較：兩變數比較，當條件成立時取變數 1，若條件不成立取變數 2，使用方式如下：

輸出= (變數 1 條件 變數 2) ? 變數 1: 變數 2

變數比較的範例程式(..\CH03\_C\C25)。

### 3-2.2 switch 指令實習

switch 指令適合於有數字順序的條件判斷，如下圖所示。

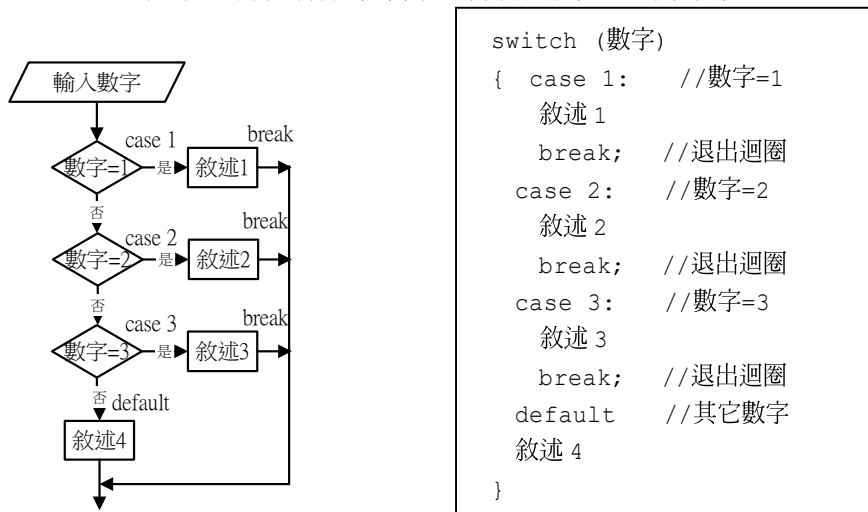


圖 3-11 switch 指令用法

它的用法是先輸入數字，假如條件 1 符合時會執行敘述 1，假如條件 2 符合時會執行敘述 2，如此依此類推。如果都不符合則執行敘述 4。它也可使用字元的順序，如'a'，'b'等，不過它是以字元的 ASCII 碼作為條件判斷的依據。switch 指令的範例程式(..\CH03\_C\C26)。

### 3-2.3 while 指令實習

while 是由條件來判斷程式是否要執行，如下圖所示。

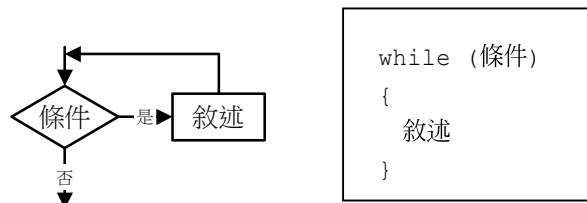


圖 3-12 while 指令用法

先判斷條件是否符合，若符合時會執行敘述，若不符合則跳過。它的用法有幾種：

1. 條件永久成立：若 `while(1)` 表示條件永久成立，在 `while(1)` 底下的程式會不斷的重覆執行，它可取代 `goto` 的功能。範例程式(..\CH03\_C\C27)。
2. 設定條件執行：若符合條件，即執行後面內的程式。假如程式僅有一行，則 `{}` 可省略，範例程式(..\CH03\_C\C28)。
3. 兩層 while 執行：令 LED 輸出霹靂燈，範例程式(..\CH03\_C\C29)。
4. 用 while 達成自我執行空轉的功能，範例程式(..\CH03\_C\C30)。
5. while 可用於位元(位元變數或單一接腳)的控制，範例程式(..\CH03\_C\C31)。

### 3-2.4 for 指令實習

for 指令同時可設定初值、條件及運算式。指令的格式如下：

```
for (初值; 條件; 運算式)
```

它由初值開始，若條件符合則進行大括號{}內的敘述，再將初值加以運算，再判斷條件，直到條件不符合，才往下執行，如下圖所示。

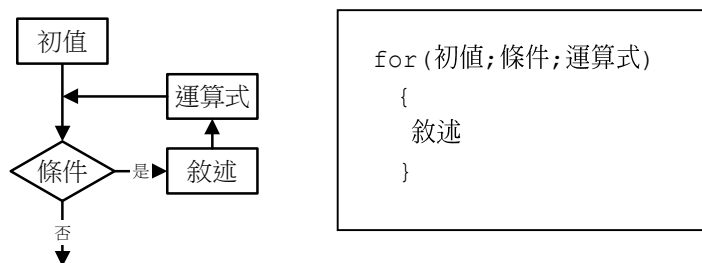


圖 3-13 for 指令用法

其中若無初值，則使用原來定義的初值或若無定義則內定為 0。如下：

```
for ( ; 條件; 運算式)
```

其中若無條件，則永久成立。如下：

```
for (初值; ; 運算式)
```

1. for 指令輸出由 1~8 範例，範例程式(..\CH03\_C\C32)。
2. for 指令計算累加範例，範例程式(..\CH03\_C\C33)。
3. for 指令輸出霹靂燈範例，範例程式(..\CH03\_C\C34)。
4. for 巢狀迴圈範例，範例程式(..\CH03\_C\C35)。
5. for 指令多重初值、條件及運算式範例：可設多重的初值(i=1, j=1)、條件(i<10 && j<8)及運算式(i=i+2, j++)，範例程式(..\CH03\_C\C36)。
6. for 指令中若無初值及條件，則由宣告變數的初值來定義，若變數無初值則

為 0。若無條件則永久成立，它會不停的執行運算子，範例程式(.\CH03\_C\C37)。

### 3-2.5 do-while 指令實習

do-while 指令的用法，如下圖所示。

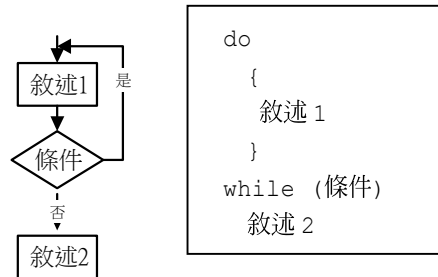


圖 3-14 do-while 指令用法

先執行敘述 1，再判斷條件是否符合，若符合時會重覆執行敘述 1，若不符合時則往下執行敘述 2。

1. do-while 指令範例(1):計數 1~7，若超過則停止，範例程式(.\CH03\_C\C38)。
2. do-while 指令範例(2):設定累加計數最大值，範例程式(.\CH03\_C\C39)。

### 3-2.6 break 與 continue 指令實習

1、break(中斷)的作用是可以由程式迴圈中跳出來。

- (1) 由 while 程式迴圈中斷，範例程式(.\CH03\_C\C40)。
- (2) 由 do-while 程式迴圈中斷，範例程式(.\CH03\_C\C41)。

(3) 由 for 程式迴圈中斷，範例程式(..\CH03\_C\C42)。

2、執行 continue(繼續)指令時，會跨過後面的所有程式，直接跳到前面 “{ “，再由函數的開始位置來執行，範例程式(..\CH03\_C\C43)。

### 3-3 C 語言函數庫實習及假指令

C 語言中有許多的函數及假指令來提供使用者應用，如此可令程式較為精簡，其中函數可分為自定函數及內部的函數庫。

#### 3-3.1 自定函數

使用自定函數時，要注意的事項如下：

- ◎在自定函數中的 `void` 表示無資料傳遞。
- ◎函數定義時，同時也要宣告變數的型式。
- ◎自訂函數一般須放置在主程式前面。
- ◎若要將自訂函數放在程式後面時，必須一開始就宣告函數。
- ◎傳入函數的引數值，和被傳入函數的引數值，兩者變數的型別須相同。

函數的格式如下表所示：

```
int Delay(int count)
回傳      傳入引數
```

表 3-16 函數的格式

函數格式	說 明
<code>void Delay(void)</code>	無傳入引數，無回傳資料(無入無回)，如下表(a)
<code>void Delay(int count)</code>	有傳入整數，無回傳資料(有入無回)，如下表(b)
<code>int Delay(void)</code>	無傳入引數，有回傳整數(無入有回)，如下表(c)
<code>int Delay(int count)</code>	有傳入整數，有回傳整數(有入有回)，如下表(d)

表 3-17(a) 無傳入引數，無回傳資料

<pre>void main() {   Delay(); }</pre>	<pre>void Delay(void) {   int count=100;   temp=count; // temp =100 }</pre>
---------------------------------------	---

表 3-17(b) 有傳入整數，無回傳資料

<pre>void main() {   Delay(100); }</pre>	<pre>void Delay(int count)// count=100 {   temp =count; // temp =100 }</pre>
--	--

表 3-17(c) 無傳入引數，有回傳整數

<pre>void main() {   int i;   i=Delay();//i=100 }</pre>	<pre>int Delay(void) {   int count=100; //count=100   return count; //回傳 count }</pre>
---	--

表 3-17(d) 有傳入整數，有回傳整數

<pre>void main() {   int i;   i=Delay(100);//i=101 }</pre>	<pre>int Delay(int count) //count=100 {   count++; //count=101   return count ; //回傳 count }</pre>
--	--

1. 自訂函數一般須放置在主程式前面。
2. 若要將自訂函數放在程式後面時，必須一開始就宣告函數。以



`void Delay(int count)` 函數為例，它有傳入整數資料，但不回傳資料，且傳入函數的引數值，和被傳入函數的引數值，其兩者變數的型別相同。

- 若要回傳變數，則在函數內最後須加 `return 變數`，範例程式 (..\CH03\_C\C44)。
- 在函數中可以包含位元型態，函數回傳也可以用位元型態，範例程式 (..\CH03\_C\C45)。

### 3-3.2 前置處理假指令

前置處理假指令用於協助編譯器來編譯程式碼，此假指令類似一個巨集處理器，在編譯原始程式之前先進行某些動作。

前置處理假指令是以“#”符號為開頭，且一般是不會產生執行碼。如下表所示：

表 3-18 前置處理假指令

前置處理假指令	功能
<code>#asm</code> 、 <code>#endasm</code>	可內嵌組合語言指令
<code>#define</code>	定義符號或前置處理的巨集指令
<code>#elif</code>	<code>#else #if</code> 的縮寫
<code>#error</code>	產出錯誤訊息
<code>#if</code> 、 <code>#else</code> 、 <code>#endif</code>	定義條件成立或不成立 所執行的設定
<code>#ifdef</code>	如果前置處理符號已定義，則執行其程式
<code>#ifndef</code>	如果前置處理符號未定義，則執行其程式
<code>#include</code>	將標頭檔的內容含入
<code>#pragma</code>	編譯器的特殊選項
<code>#undef</code>	取消先前定義的符號或前置處理巨集

1. 含入檔案假指令(`#include`)：會將指定的檔案的內容插入到原始檔案中。

(1) 使用`#include <檔名>`時，編譯器會從環境變數 `INCLUDE` 所指定的目錄中找尋檔案，如果沒有定義 `INCLUDE`，編譯器會由檔名中的路徑搜尋檔案。

(2) 使用`#include "檔名"`時，編譯器會搜尋所指定的檔案，如果沒有指定目錄，則會從目前所在的目錄中找尋檔案。

語法：	範例：
<code>#include &lt;檔名&gt;</code>	<code>#include &lt;msp430g2553.h&gt;</code>
<code>#include "檔名"</code>	<code>#include "../..\config.h"</code>

2. 定義假指令(`#define`)：用於定義字串常數來代表某些數值，如此可增加程式的可讀性與維護性。語法如下：

```
#define 名稱 數值或字串
#define 名稱 []數值或字串
```

其中如果無法在一行中寫完數值或字串，可以使用反斜線(`\`)表示還有更多的數值或字串。 範例如下：

```
#define i 40 //i=40
#define NAME "Henry" //NAME=字串"Henry"
#define SWAP(j,k) //定義j 及 k 內容交換
do { int tmp;
    tmp=j;
    j=k;
    k=tmp; }
while (0);
```

3. 條件編譯假指令(`#if`、`#else` 及`#endif`)：

- (1) `#if` 和 `#endif` 一起用來作為條件編譯程式的假指令，而編譯時由“條件”來決定編譯那些程式。
- (2) `#else` 假指令提供二選一的編譯方式，可省略。
- (3) 如果條件成立，編譯程式 1，否則編譯程式 2。語法如下：

```
#if 條件 //判斷條件是否成立
    原始程式 1 //若條件成立，編譯及執行程式 1
    [#else 原始程式 2] //若條件不成立，編譯及執行程式 2(可省略)
#endif
```

範例：

```
#define i 2 //定義 i=2
#if i > 0 //判斷是否 i>0
    #define j 1 //假如 i>0, 編譯定義 j=1
#else
    #define j 2 //假如不是 i>0, 編譯定義 j=2
#endif
```

#### 4. 錯誤假指令：`#error`

`#error` 會產生使用者所定義的錯誤訊息(message)。語法如下：

```
#error "message-string" //錯誤訊息
```

範例：

```
#if i > 10 //若 i>10 顯示錯誤訊息
    #error "Too many count." //錯誤訊息
#endif
```

5. 條件編譯假指令(`#ifdef` 及 `#else`)：`#ifdef` 類似 `#if`，但它用於檢查所指定的符號(symbol)是否已經被定義。如果符號(symbol)已經被定義則原始程式 1 將被編譯，否則編譯原始程式 2。

`#else` 假指令提供二選一的編譯方式，可省略。語法如下：

```
#ifdef 符號(symbol) //檢查符號(symbol)是否被定義
    原始程式 1
[#else 原始程式 2]
#endif //結束 if 判斷
```

範例：

```
#ifdef MODE //檢查 MODE 是否被定義
    #define count 100 //若是 count=100
#endif //結束 if 判斷
```

6. 條件編譯假指令(`#ifndef`)：相當於反相的`#ifdef`。
7. 條件編譯假指令(`#elif`)：要配合`#if`一起使用，它提供第三種條件編譯方式。

語法如下：

```
#if 條件 1
    原始程式 1
#elif 條件 2
    原始程式 2
[#else 原始程式 3]
#endif //結束 if 判斷
```

範例：

```
#if i==1 //若 i=1
    #define j 1 //若 i=1，則定義 j=1
#elif i==2 //若 i≠1，但 i=2
    #define j 2 //若 i=2，則定義 j=2
    #else #define j 3 //若 i≠1 及 i≠2，則定義 j=3
#endif //結束 if 判斷
```

8. 條件編譯假指令(`defined`)：`defined` 可以使用在`#if` 或`#elif` 中。其中“`#ifdef` 符號(symbol)”相當於“`#if defined 符號(symbol)`”，而“`#ifndef` 符號

(symbol) “則等於 “`#if !defined 符號(symbol) “。語法如下：`

```
#if defined 符號(symbol)
    原始程式 1
[#else 原始程式 2 ]    //可省略
#endif    //結束 if 判斷
```

範例：

```
#if defined i //檢定符號 i 是否有定義
    #define j 50 //若有 j=50
#endif    //結束 if 判斷
```

9. 條件編譯假指令(`#undef`)：將已定義的符號(symbol)清除，相當於此符號沒有被定義。語法如下：

```
#undef 符號(symbol)
```

範例：

```
#define i 100 //定義 i=100
#undef i      //清除 i 的定義
#define i 50  //重新定義 i=50
```

10. 在 `wiring_constants.h` 內含常數的定義，如下：

```
#ifndef min    //取變數的最小值

    #define min(a,b) ((a)<(b)?(a):(b))

#endif // min
```

```
#ifndef max//取變數的最大值

#define max(a,b) ((a)>(b)?(a):(b))

#endif // max
```

```
#define abs(x) ((x)>0?(x):- (x)) //取變數的絕對值

#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))

#define round(x) ((x)>=0?(long)((x)+0.5):(long)((x)-0.5)) //取亂數

#define radians(deg) ((deg)*DEG_TO_RAD)

#define degrees(rad) ((rad)*RAD_TO_DEG)

#define sq(x) ((x)*(x)) //取變數的均方值
```

```
#define interrupts() __enable_irq()

#define noInterrupts() __disable_irq()
```

```
#define lowByte(w) ((uint8_t) ((w) & 0xff))

#define highByte(w) ((uint8_t) ((w) >> 8))
```

```
#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) :
bitClear(value, bit))
#define bit(b) (1UL << (b))
```

## 3-4 Arduino 函數庫

Arduino 內含許多軟硬體函數，透過函數即可控制軟硬體，如此可令程式較為精簡。如下表(a)(b)所示：

表 3-19(a) Arduino 系統函數式(1)

Constants	Data Types	Variable Scope & Qualifiers
Floating Point Constants	array	const
HIGH   LOW	bool	scope
INPUT   INPUT_PULLUP   OUTPUT	boolean	static
Integer Constants	byte	volatile
LED_BUILTIN	char	
true   false	double	
	float	<b>Utilities</b>
	int	PROGMEM
<b>Conversion</b>	long	sizeof()
(unsigned int)	short	
(unsigned long)	size_t	
byte()	string	
char()	String()	
float()	unsigned char	
int()	unsigned int	
long()	unsigned long	
word()	void	
	word	



表 3-19(b) Arduino 系統函數式(2)

<b>Digital I/O</b>	<b>Math</b>	<b>Bits and Bytes</b>
digitalRead()	abs()	bit()
digitalWrite()	constrain()	bitClear()
pinMode()	map()	bitRead()
	max()	bitSet()
	min()	bitWrite()
<b>Analog I/O</b>	pow()	highByte()
analogRead()	sq()	lowByte()
analogReadResolution()	sqrt()	
analogReference()		
analogWrite()	<b>Trigonometry</b>	<b>External Interrupts</b>
analogWriteResolution()	cos()	attachInterrupt()
	sin()	detachInterrupt()
<b>Advanced I/O</b>	tan()	digitalPinToInterrupt()
noTone()		
pulseIn()	<b>Characters</b>	<b>Interrupts</b>
pulseInLong()	isAlpha()	interrupts()
shiftIn()	isAlphaNumeric()	noInterrupts()
shiftOut()	isAscii()	
tone()	isControl()	<b>Communication</b>
	isDigit()	Print
<b>Time 計時</b>	isGraph()	Serial
delay()	isHexadecimalDigit()	SPI
delayMicroseconds()	isLowerCase()	Stream
micros()	isPrintable()	Wire
millis()	isPunct()	
<b>Random Numbers</b>	isSpace()	<b>USB</b>
random()	isUpperCase()	Keyboard
randomSeed()	isWhitespace()	Mouse

### 3-4.1 Arduino 常用系統函數式

Arduino 系統常用函數式(Functions)如下：

1. 輸出入(I/O)函數式：如下表所示：

表 3-20 輸出入(I/O)函數式

數位輸出入(Digital I/O)	類比(Analog)輸出入	進階(Advanced)輸出入
pinMode() digitalWrite() digitalRead()	analogReference() analogRead() analogWrite() - <i>PWM</i>	tone()、noTone() shiftOut()、shiftIn() pulseIn()

2. 算術(Math)函數式：如下表所示：

表 3-21 算術(Math)函數式

算術(Math)	亂數(Random)	三角(Trigonometry)
<a href="#">min()</a> 、 <a href="#">max()</a> 、 <a href="#">abs()</a> 、 <a href="#">sqrt()</a> <a href="#">constrain()</a> 、 <a href="#">map()</a> 、 <a href="#">pow()</a>	<a href="#">randomSeed()</a> 、 <a href="#">random()</a>	<a href="#">sin()</a> 、 <a href="#">cos()</a> 、 <a href="#">tan()</a>

3. 延時(Timer)及位元/位元組(Bit/Byte)函數式：如下表所示：

表 3-22 計時及位元/位元組函數式

計時(Time)	位元(Bits)及位元組(Bytes)	
<a href="#">millis()</a> 、 <a href="#">micros()</a> 、 <a href="#">delay()</a> <a href="#">delayMicroseconds()</a> 、 <a href="#">sleep()</a>	<a href="#">lowByte()</a> 、 <a href="#">highByte()</a> <a href="#">bitRead()</a> 、 <a href="#">bitWrite()</a>	<a href="#">bitSet()</a> 、 <a href="#">bitClear()</a> 、 <a href="#">bit()</a>

4. 中斷(External)及通訊(Communication)函數式：如下表所示：

表 3-23 中斷及通訊函數式

外部(External)中斷	中斷(Interrupts)	通訊(Communication)
attachInterrupt() detachInterrupt()	interrupts() noInterrupts()	Serial Stream

### 3-4.2 Arduino 常用系統函數式範例

在 Arduino 系統下達各種輸出入命令的常數時，必須使用開發板 TH244A 的接腳名稱或編號，如下：

1. 在 Arduino 常用的數位輸出入函數式如下表所示：

表 3-24 輸出入(I/O)函數式

數位輸出入	語法	常數
pinMode()	pinMode(pin, mode)	接腳 pin : 0~46、A0~A15
digitalWrite()	digitalWrite(pin, value)	模式 mode : OUTPUT、INPUT、INPUT_PULLUP
digitalRead()	digitalRead(pin)	數值 value : 1、0、HIGH、LOW

例如：設定 D13(LED)腳為輸出，如下：

```
pinMode(13, OUTPUT) //設定D13(LED)為輸出腳
digitalWrite(13, HIGH) 或 digitalWrite(13, 1) //令輸出D13(LED)=1
digitalWrite(13, LOW) 或 digitalWrite(13, 0) //令輸出D13(LED)=0
```

2. 在 Arduino 常用的計時函數如下表所示：

表 3-25 計時(Time)函數式

延時函數	語法	常數
<a href="#">delay()</a>	delay(ms)	延時 ms = unsigned long
<a href="#">delayMicroseconds()</a>	delayMicroseconds(us)	延時 us = unsigned int

設定延時範例如下：

```
delay(1000) //設定原地空轉延時1秒鐘
delay(1000*60) //設定原地空轉延時1分鐘
delay(1) //設定原地空轉延時1mS
delayMicroseconds (1000) //設定原地空轉延時1mS
```

