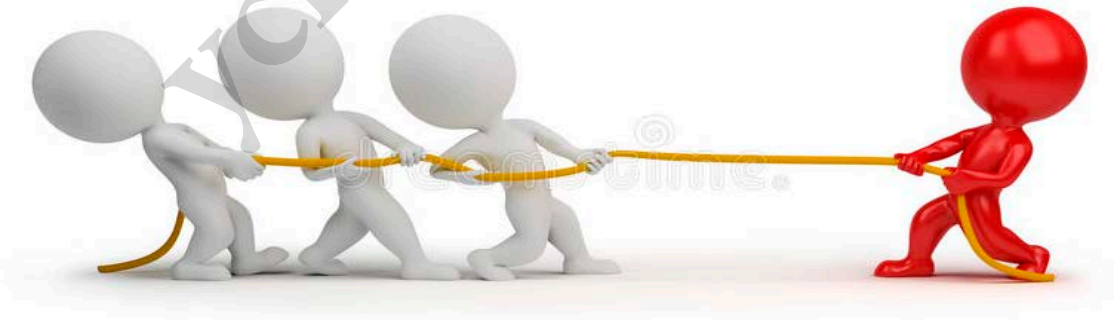
Chap 3
Arduino程式語言

3.1 Arduino程式的基本認識

- 3.1. Arduino程式的基本認識
  - 3.1.1. 註解說明
  - 3.1.2. 以分號；為結尾的敘述（statement）
  - 3.1.3. 使用大括號{...}來定義程式區塊或範圍
  - 3.1.4. Arduino程式的框架
  - 3.1.5. 第一支Arduino程式：在PC上秀出結果



3.1 Arduino程式的基本認識

- 因為創始 Arduino 所堅持的核心概念，就是要非資訊相關科系的學生或使用者皆能輕鬆入門、簡單上手，所以Arduino程式語言基本上繼承了 C/C++ 語言的語法跟概念，具有簡潔靈活但運算能力強大的特性，學習起來並不會太困難。

- 一般我們會用 program 這個單字來代表程式，可是在 Arduino 的習慣用語中卻是將程式稱之為 sketch，我們可翻譯成「腳本」、「草稿」、「素描」或是「速寫」。

3.1.1. 註解說明

- 如果要讓你的程式碼有重複利用的價值，請記得一定要留下必要的註解與說明。Arduino程式的註解說明分成單行註解與多行註解二種，

(1) //：單行註解，在//之後到該行結尾的文數字都會被當成註解不會被編譯。

(2) /*...*/：多行註解，在/*與*/之間所有的文數字都會被當成註解不會被編譯。

```
//單行註解
t=t+1; //t是代表時間變數

/*
多行註解使用在詳細的解釋，
時間: 2017/08/01
作者: David
說明:此段程式碼是快速排序法
*/

/*這樣也可以*/
```

3.1.2. 以分號；為結尾的敘述（statement）

- 與C/C++程式語言一樣，敘述（statement）是Arduino程式最基本的執行單位，每個敘述都是以分號；為結尾。

(1) 根據定義，最簡單的敘述就是只有一個分號的空敘述，相當於NOP（No Operation）的作用，表示不執行任何動作。

(2) 除了在字串中的空格之外，敘述中的任何空格不分數量都是可以省略的。

(3) 在程式碼中可允許多個敘述寫在同一行，但為了程式具有較好的可讀性與可除錯性，還是建議一行一個敘述比較單純，請記住行數的增加並不會影響程式碼的大小（code size）。

(4) 如果一個敘述太長必須分成多行來寫，有二種方式可以採用：1. 只要不是斷行在字串、數字、變（常）數名稱或函式名稱的中間，都可任意分成多行來寫。 2. 沒有任何限制，在敘述中的任一個位置都可以使用反斜號 \ 來斷行，因為在編譯時 \ 後面的換行符號將被忽略而當成一行來處理。

3.1.2. 以分號；為結尾的敘述（statement）

<範例 3.2>

```
;  //空的敘述
int i=0;       //宣告整數變數 i
m=10;          //設定變數 m 的值
delay(1000);   //呼叫延遲函式，輸入參數 1000
i=1; i=f1(i); i=f2(i);  //一行有 3 個敘述

total=a*1000+b*100+c*10+d;
//同一個敘述寫成四行
total=
a*1000
+b*100+c
*10+d;

total=a*1000+b*10      //錯誤的斷行
0+c*10+d;

total=a*1000+b*10     //使用反斜號¥來斷行
0+c*10+d;
```

3.1.3. 使用 {...} 定義程式區塊或範圍

- 在Arduino的程式中我們會使用大括號 {...}來定義函式的範圍、迴圈的範圍和條件成立/不成立的程式區塊範圍，使用上有下列二點要特別注意：

(1) 大括號的使用一定要成雙成對符合對稱性，。

(2) 大括號的配對原則是屬於堆疊（stack）結構。

3.1.4. Arduino程式的框架

- Arduino的程式基本上可以分成五個部分，除了setup() 和loop() 這二個不可缺少也不可更改名稱的函式之外，其餘的部分皆是可有可無，可以視程式的需要增加或是省略。

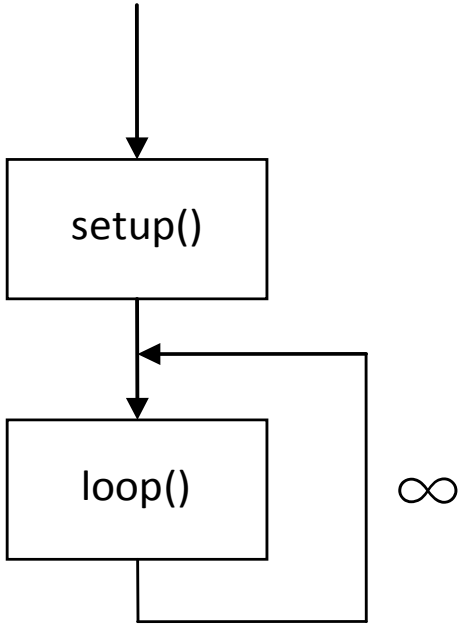
(1) 第一部分是前置處理部分。

(2) 第二部分是常數或全域變數的宣告。

(3) 第三部分是必要的初始設定函式setup( )

(4) 第四部分是必要的無窮迴圈函式loop( )

(5) 第五部分是自訂函式的部分



3.1.4. Arduino程式的框架

<範例 3.4>

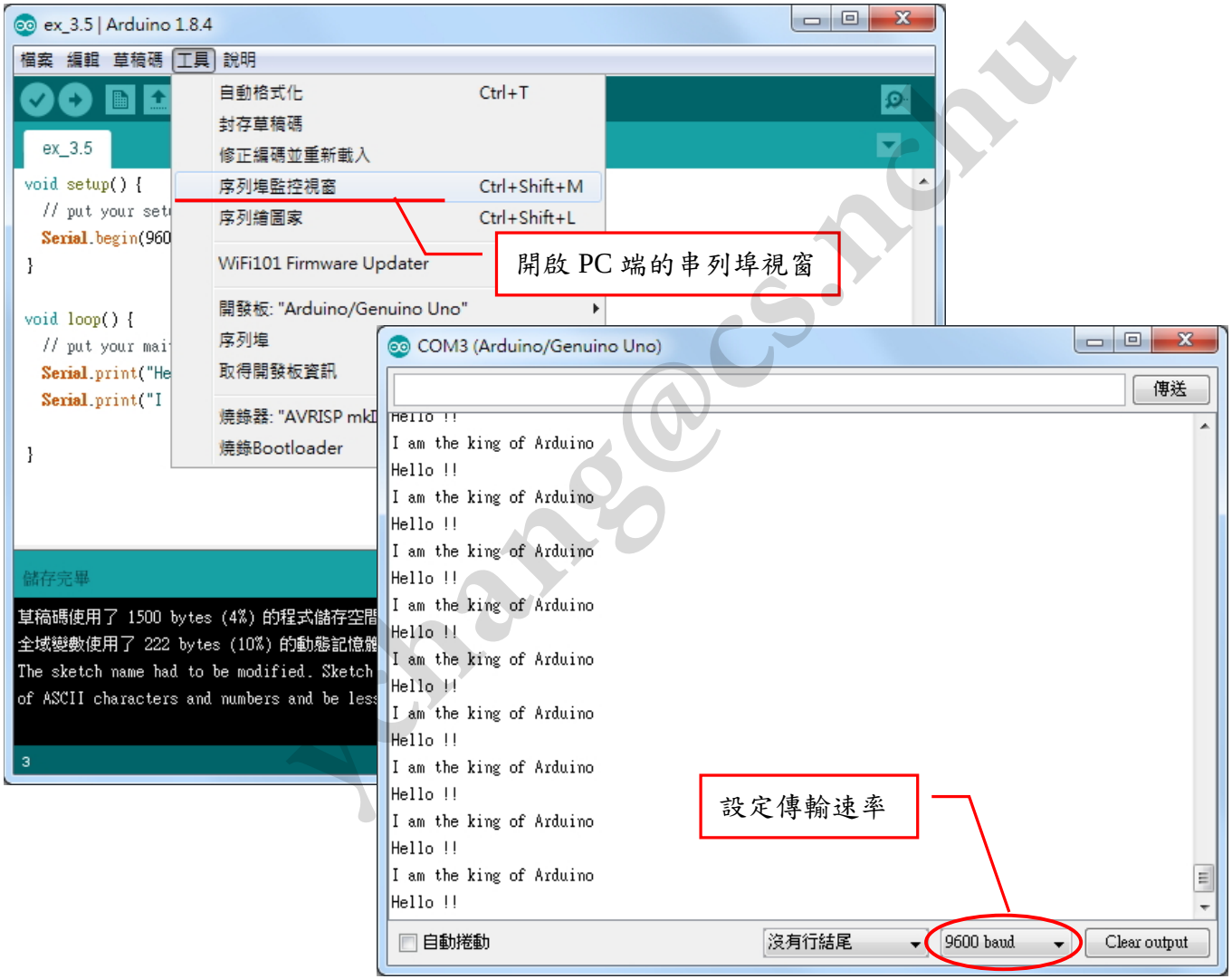
```
//1.匯入標頭檔，與字串/巨集的定義（可有可無）
  #include <wire.h>
  #define LED_pin 13;

//2.宣告常數與全域變數（可有可無）
  const float PI=3.14;
  int index;

//3.初始設定函式（必要，不可缺少也不可更改名稱）
void setup()
{
}

//4.無窮迴圈函式（必要，不可缺少也不可更改名稱）
void loop()
{
}

//5.其它自訂函式（可有可無）
func1() {...}
func2() {...}
```

3.1.5. 第一支Arduino程式

<範例 3.5>

```
void setup()
{
   Serial.begin(9600);  //以 9600 bps 傳輸速率初始化串列埠
}

void loop()
{
   Serial.print( “Hello !!\n” );  //  ‘\n’ 是換行控制代號
   Serial.print( “I am the king of Arduino.\n” );  //印出 I am...字串並且換行
}
```

3.1.5. 第一支Arduino程式



3.2 常數

- 3.2. 常數

常數（constant）就是指恆常不會變動的數值，一般而言常數的使用有助於提高程式碼的可讀性，在Arduino的程式碼中使用的常數可以分成二類：第一類是Arduino預先定義好的內定常數，第二類是使用者自行定義的常數。此外為了凸顯常數的專用性與獨特性，常數名稱通常使用大寫，但這只是習慣，並非語法規則。。

- 3.2.1. Arduino內定的常數
- 3.2.2. 使用者自行定義的常數
- 3.2.3. 十進位以外的數值表示

3.2.1. Arduino內定的常數

常數類別	常數名稱
布林常數	true，false
接腳模式常數	INPUT，INPUT_PULLUP，OUTPUT
電壓準位常數	HIGH，LOW

3.2.1. Arduino內定的常數

常數類別	常數名稱	說明
布林常數	true	邏輯判斷的”真”，一般都是定義成數值 “1” 來表示，但其實只要是非 0 的整數都可以代表 true，不限定只是數值 “1”，數值-5 或 100 都是代表 true 的意思。
	false	邏輯判斷的”假”，定義比 true 明確而且唯一，只有數值 “0” 能表示 false。

```
//以下 4 個敘述同為邏輯判斷的 true
if (true) { }
if (1) { }
if (-2) { }
if (100) { }

//以下 2 個敘述同為邏輯判斷的 false
if (false) { }
if(0) { }
```

```
int true;     //會發生編譯錯誤
true=5;     //會發生編譯錯誤

int TRUE;   //因為區分大小寫，不會有錯誤
TRUE=false;   //沒有錯誤
TRUE=100;     //沒有錯誤
```

3.2.1. Arduino內定的常數

常數類別	常數名稱	說明
接腳模式常數	使用在 pinMode()函式中，設定接腳訊號的模式，也就是訊號的方向是輸入（INPUT）到 Arduino 開發板，還是輸出（OUTPUT）到週邊設備。	
	INPUT	規劃數位接腳的模式為輸入模式，訊號的方向是從週邊設備輸入到 Arduino 開發板，對應的動作函式為 digitalRead()。為了避免浮接狀態（floating）下讀取接腳訊號會呈現無法預測的值，一般都會外接一個適當的上拉/下拉（pull-up / pull-down）電阻。
	INPUT_PULLUP	規劃數位接腳的模式為輸入模式，並且啟用接腳內建的 20KΩ 上拉電阻，在浮接狀態下可以讀取到穩定的 HIGH，省去使用者必需自行外接一個上拉電阻的麻煩。
	OUTPUT	規劃數位接腳的模式為輸出模式，訊號的方向是從 Arduino 開發板輸出到週邊設備，對應的動作函式為 digitalWrite()。相較之下 OUTPUT 模式沒有浮接的問題，所以不需要外接上拉/下拉電阻。

3.2.1. Arduino內定的常數

常數類別	常數名稱	說明
電壓準位常數	根據電壓準位的高低範圍來定義數位接腳的訊號值，因為是使用在數位接腳所以只定義 LOW 與 HIGH 二種值，也就是二進位中的 0 與 1，相關函式有 digitalRead()與 digitalWrite()。	
	HIGH	(1) 讀取時，電壓範圍符合下列條件者為 HIGH 5V 的開發板 ：訊號的電壓值>3V 3.3V 的開發板：訊號的電壓值>2V (2) 寫入時，訊號為 HIGH 的電壓值 5V 的開發板 ：訊號的電壓值=5V 3.3V 的開發板：訊號的電壓值=3.3V
	LOW	(1) 讀取時，電壓範圍符合下列條件者為 LOW 5V 的開發板 ：訊號的電壓值<1.5V 3.3V 的開發板：訊號的電壓值<1.0V (2) 寫入時，訊號為 LOW 的電壓值 5V 的開發板 ：訊號的電壓值=0V 3.3V 的開發板：訊號的電壓值=0V

3.2.2. 使用者自行定義的常數

const 變數型態 變數名稱=常數值;

- 其概念就是先宣告一個變數，然後使用const這個修飾詞（qualifier）把變數設定成唯讀（read only）的模式，同時指定常數值之後，就再也不能改變其值，任何要修改自定常數的敘述都會導致編譯錯誤。

<範例 3.xx>

```
const float myPI = 3.14;  //宣告浮點常數 myPI，並給定常數值為 3.14
float x;                  //宣告浮點數 x

x = myPI * 2;   //正確無誤
myPI = 7;       //會出現編譯錯誤，因為要修改常數值
```

3.2.3. 十進位以外的數值表示法

- 不同的進位表示法並不會改變原來的數值，只是顯示出來的格式數字有所不同。Arduino程式語言中可接受的數值表示方法共有10進位、2進位、8進位與16進位四種。

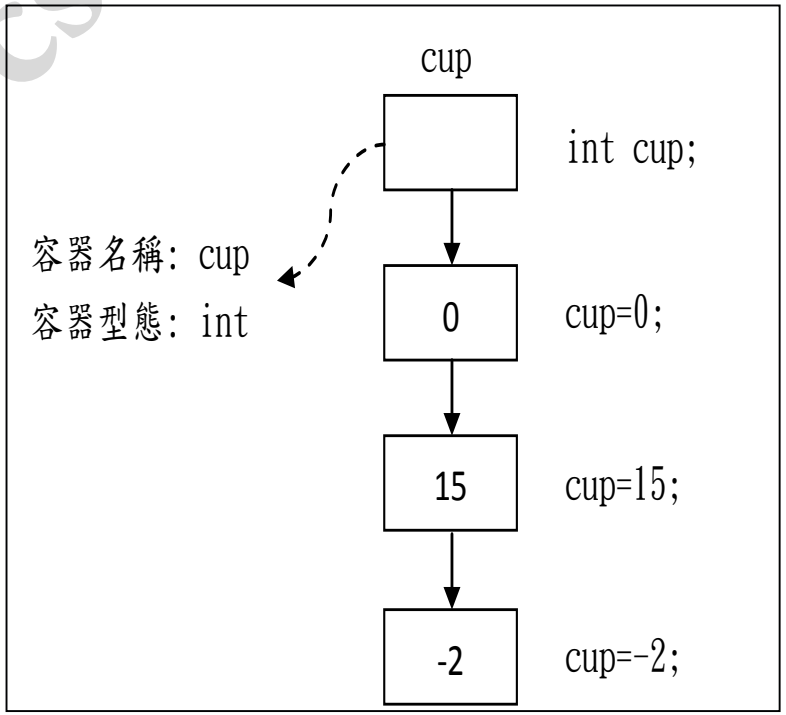
數值表示法	格式	範例	說明
10 進位 （Decimal）	數字 0~9	101	$10^2+1=101_{10}$
2 進位 （Binary）	B/0b+數字 0/1	B101 或 0b101	$2^2+1=5_{10}$
8 進位 （Octal）	0+數字 0~7	0101	$8^2+1=65_{10}$
16 進位 （Hexadecimal）	0x/0X+數字 0~F/f	0x101 或 0X101	$16^2+1=257_{10}$

3.3 變數

3.3 變數

另一類與常數不同的數值就是變數（variable），從字面上的解釋變數就是可變動的數，是Arduino程式中使用最多的一類數值，與數學中變數的觀念類似，Arduino的變數也是使用某個名稱來代替某個數值，方便運算式的撰寫。

- 3.3.1 變數名稱
- 3.3.2 變數型別（Type）
- 3.3.3 變數的有效範圍（Scope）
- 3.3.4 變數的修飾子（Qualifier）



3.3.1 變數名稱

- 在Arduino程式語言中，變數在使用前一定要先進行宣告（declare），宣告的方式很簡單，就是變數型態後面接空格，然後是變數的識別名稱，格式如下：

  變數型態 變數名稱; //例如: int apple;

- 變數的宣告主要有二個目的：(1) 讓編譯器在進行程式編譯時能識別這個變數名稱，不會有錯誤產生。(2) 讓編譯器知道變數的資料型態，為此變數配置足夠的記憶體空間。

3.3.1 變數名稱

- 變數的名稱可以是任何大小寫英文字母（含底線_）或數字，而且字母與數字可以混合使用，但是必須遵從下列幾點命名的規則：

(1) 英文字母的大小寫是有區分的。。

(2) 名稱中不可以有空格。

(3) 不可以全部是數字，或者用數字當變數名稱的第一個字元。

(4) 不可使用字母或數字以外的文字符號，包括中文。

(5) 不可以是Arduino程式語言的保留字。

(6) 變數的名稱最好有意義，這樣才能提高程式碼的可讀性，例如若有一變數為某人的年齡可命名為「age」、體重可命名為「weight」等。

<範例 3.xx>

```
//以下為合法的變數名稱
int aBc;
int A2bc3;
int char_5b;
int _0_1_2;
int m_;
int Case;
```

```
//以下為錯誤的變數名稱
int 2nd;      //第一個字元為數字
int 123;      //全為數字
int he is;    //名稱中有空格
int case;     //Arduino 保留字
int a_人數;   //使用中文
int you&me;   //使用字母數字外的符號
```

3.3.2 變數型別（Type）

- 我們可以把變數想像成是一個可以放任何數值的「容器」，可是容器形狀各異，尺寸有大有小，我們不希望容器太小放不進數值，也不希望容器太大而浪費空間，所以選擇一個形狀正確尺寸大小最適合的「容器」就是宣告變數的第一步，也就是變數型態（variable type）的選擇。

變數型態	佔用記憶體空間（byte）	數值種類	可儲存數值範圍
boolean	1	整數	只有 true 和 false 二種值，也就是 1 與 0
char	1	整數	-128~127
unsigned char	1	無號整數	0~255
byte	1	整數	0~255，與 unsigned char 一樣
int	2 (Uno+) 4 (Due*)	整數	-32,768~32,767 -2,147,483,648~2,147,483,647
unsigned int	2 (Uno+) 4 (Due*)	無號整數	0~65,535 0~4,294,967,295

3.3.2 變數型別（Type）

word (unsigned int)	2 (Uno+) 4 (Due*)	整數	0~65,535 0~4,294,967,295
long	4	整數	-2,147,483,648~2,147,483,647
unsigned long	4	無號整數	0~4,294,967,295
short	2	整數	-32,768~32,767
float	4	浮點數	-3.4028235E+38~3.4028235E+38
double	4 (Uno+) 8 (Due*)	浮點數	-3.4028235E+38~3.4028235E+38 ?
string	-		char array，詳見說明(6)
String	-		詳見說明(7)

Uno+：Uno 和 ATMega 系列的開發板 (請參閱圖 2.5)

Due*：Due 和 ATSAMD 系列的開發板 (請參閱圖 2.5)

3.3.3 變數的有效範圍（Scope）

變數有效範圍	說明
全域變數	程式中任何一個函式都能存取到的變數即為全域變數，一般會定義在所有的函式之前，變數名稱具有唯一性不能重複，例如範例 3.xx 中的變數 gVIP。優點是變數存取方便，缺點是不易除錯。
區域變數	定義在函式之內，只有在這個函式的範圍內才能被存取的變數，在同一個程式中，區域變數的名稱只要不是在同一個函式之內都可以重複使用，不會互相干擾，例如範例 3.xx 中的變數 var，同時使用在 loop 與 func1 函式中。
for 迴圈變數	定義在 for 迴圈敘述的第一個欄位，此種變數的有效範圍只侷限在此 for 迴圈內，例如範例 3.xx 函式 loop 中的變數 j。

3.3.3 變數的有效範圍（Scope）

```
int gVIP;  //全域變數，程式中所有的函式都能存取

void setup()
{
  func1();
}

void loop()
{
  int var=1;    //var 為區域變數，只有 loop 函式能存取。

  for (int j=0; j<100; j++) { //j 為 for 迴圈變數，只有在此 for 迴圈內才能被存取
    j=j+1;
  }

  j=var+1;       //會出現編譯錯誤
  gVIP=var-1;
}

void func1()
{
  int var=11;   //var 為區域變數，只有 func1 函式能存取，
                //即使與 loop 中的 var 名稱相同也不會互相干擾。
  gVIP=var-1;
}
```

3.3.4. 變數的修飾子（Qualifier）

- 變數的宣告除了在名稱前一定要加上變數型態之外，我們也可以在變數型態之前選擇性的加上修飾子（qualifier），賦予變數更多樣的特性，讓程式設計者使用變數的時候更靈活更有彈性，格式如下：

  變數宣告格式：　**[修飾子]　變數型態　變數名稱;**　//例如: const int apple;

- Arduino程式語言中可使用的變數修飾子有const，static 和 volatile 三種。

變數的修飾子	說明
const	將變數設定成唯讀（read only）的特性。
static	將區域變數配置在靜態區（static），而非堆疊區（stack）的記憶體。
volatile	指定載入變數的時候是直接從記憶體讀取而來，而不是從暫存器讀取的。

表 3.xx 變數的修飾子

3.3.4. 變數的修飾子: const

- (1) const：const 修飾子是constant常數的縮寫，顧名思議其作用就是將一個變數設定成唯讀（read only）的特性，也就是變成固定不變的常數，在給定初始值之後，任何要改變其值的敘述都會導致編譯錯誤的發生。

- 參考範例3.xx，因為 const 可使用在全域變數，也可使用在區域變數，它並不會改變變數的有效範圍，比起使用#define 定義出來的常數，只有全域範圍的效更有彈性，所以相較之下大部分的設計者還是會選擇採用const 的方式來定義常數。

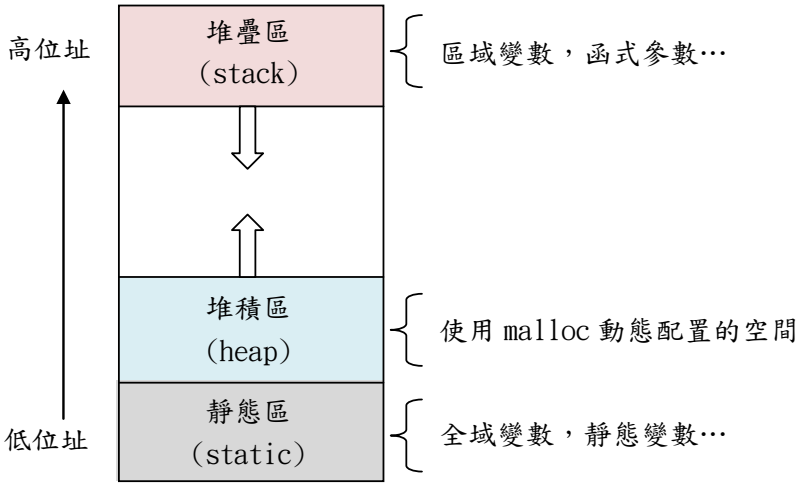
```
#define V1 11
const int V2=22;  //全域常數
int M;

void setup()
{
 const int V3=33;  //區域常數
 M=V2+V3;          //M=22+33=55
}
void loop()
{
 const int V3=44;
   M=V2+V3;          //M=22+44=66
}
```

3.3.4. 變數的修飾子: static

- (2) static：static 修飾子是使用在函式內區域變數的宣告，加上static的變數即為靜態變數，其作用是將變數配置在記憶體的靜態區。

- 一般編譯器都會把函式內的區域變數配置在堆疊區（stack）中以達到記憶體動態利用的效果，在呼叫函式時才會將記憶體配置給這個函式的區域變數，一旦函式執行結束就會摧毀這些區域變數的值收回其記憶體空間。

- 但是使用static修飾子可將變數配置在靜態區（static），其最大的特色是在函式第一次執行時才配置記憶體而且設定其初始值，之後函式執行結束並不會收回此靜態變數的記憶體空間，其值可以持續保留到下一次函式執行時仍然可以使用。



3.3.4. 變數的修飾子: static

- 如範例3.xx中 loop 的區域變數 V2，其值會隨著 loop 函式不斷的執行而產生累加的效果；相反的，一般區域變數 V1 在每一次 loop 函式執行時都會重新配置，不會有累加的效果。

```
void setup()
{
Serial.begin(9600);
}

void loop()
{
int V1=0;        //一般區域變數，初始值=0
static int V2=0; //靜態區域變數，初始值=0

  V1=V1+1;
V2=V2+1;
Serial.print(“V1=”); Serial.println(V1);
Serial.print(“V2=”); Serial.println(V2);
}
```

```
執行結果...
 V1=1 V2=1
 V1=1 V2=2
 V1=1 V2=3
 V1=1 V2=4
 V1=1 V2=5
  :   :
```

3.3.4. 變數的修飾子: volatile

- (3) volatile：volatile 修飾子的作用是在解決可能會發生資料不一致（data inconsistent）的問題，也就是程式在執行過程中讀取到的是舊的資料，而不是最新的資料，只要變數加上 volatile 就可以指定編譯器不會對該變數做任何的最佳化，而且在載入該變數的值的時候是直接從記憶體讀取而來，而不是從暫存器讀取的，如此就可以解決資料一致性的問題。

- 一般而言資料不一致的問題會發生在 (1) 編譯器的最佳化所造成的錯誤 (2) 在多線程（multi-thread）的程式中共用資料所衍生的一致性問題。

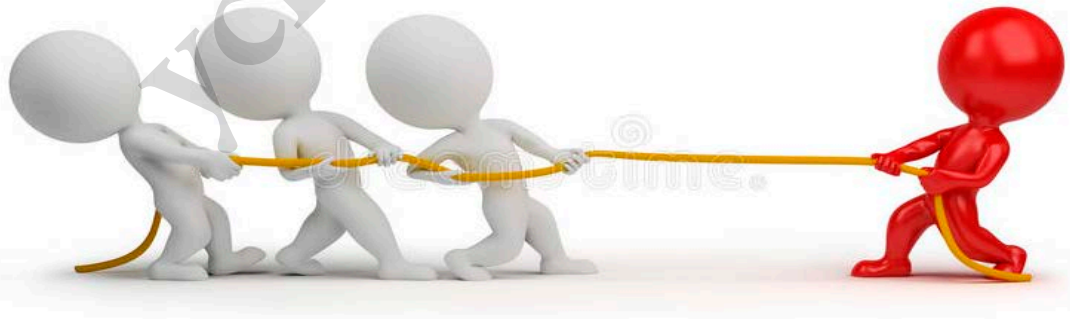
3.3.4. 變數的修飾子: volatile

- 但是以Arduino程式而言，唯一有可能會發生資料不一致的狀況是在使用中斷服務程式（interrupt service routine, ISR）的時候，所以只要將中斷服務程式中有可能改變數值的變數加上 volatile 修飾子就可以避免資料不一致的問題發生，例如範例3.xx中的變數state。

```
//當中斷接腳的狀態改變時會觸發LED亮暗狀態的反轉
int pin=13;
volatile int state=LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  //表示在第0腳位啟用外部中斷，當狀態有改變CHANGE的時候，就執行myISR函式
  attachInterrupt(0, myISR, CHANGE);
}
void loop()
{
  digitalWrite(pin, state);
}
void myISR()
{
  state = !state;
}
```

3.4 Arduino的資料運算

- 繼承C/C++的特色，Arduino擁有豐富而且多樣的資料運算能力，包括
  - 算術運算（Arithmetic）
  - 比較運算（Comparison）
  - 邏輯運算（Boolean）
  - 位元運算（Bitwise）
  - 複合運算（Compound）
  - 指標存取運算（Pointer Access）



3.4.1. 算術運算（Arithmetic）

- 如表3.xx所列，Arduino共有6種算術運算，除了最基本的加減乘除四則運算外，還有指定運算跟取餘數運算，其中要特別注意浮點數的除法運算，只有在運算元符合規定的格式下，才會有正確的浮點數的商。

運算子	名稱	說明
=	指定運算	將右邊運算元的值指定給左邊的變數，切記不要跟比較運算的相等"=="混淆了。例如： x=100; x=y; x=func(); x=y=z=100;
+	加法運算	對運算元進行加法運算。 例如：x=1+2+3; x=y+5; x=func()+y+z;
-	減法運算	對運算元進行減法運算。
*	乘法運算	對運算元進行乘法運算。

/	除法運算	對運算元進行除法運算，特別注意若要執行浮點數的除法，至少要有一個運算元為浮點數，才會正確的產生有小數點的商。例如：
		int x=100/8; //x=12
		float y=100/8; //y=12
		float y=(float)100/8; float y=100.0/8; //y=12.5

%	取餘數運算	對運算元執行取餘數的運算。例如：
		x=11%3; //x=2
		x=(a+b)%c; //將(a+b)對c取餘數的結果指定給變數x

表 3.xx Arduino的算術運算

3.4.2. 比較運算（Comparison）

- 比較運算固定有二個運算元，主要目的就是在比較二個運算元值的大小，其運算結果是boolean型態的true或false，一般會使用在條件式的判斷，以控制程式的執行流程。

運算子	名稱	說明
>	大於	判斷左邊運算元是否大於右邊運算元。 例如：if (x>0) x=x-1; else x=x+1;
<	小於	判斷左邊運算元是否小於右邊運算元。
==	等於	判斷左邊運算元是否等於右邊運算元。 例如：while (x==y) { c=100; };
>=	大於等於	判斷左邊運算元是否大於等於右邊運算元。
<=	小於等於	判斷左邊運算元是否小於等於右邊運算元。
!=	不等於	判斷左邊運算元是否不等於右邊運算元。

3.4.3. 邏輯運算（Logic）

- Arduino有3種邏輯運算，與比較運算一樣，邏輯運算的結果也是boolean型態的true或false，一般會用在條件式的判斷，達到控制程式執行流程的目的。

運算子	名稱	說明
&&	邏輯and	適用在2個或2個以上的運算元。只有在所有運算元皆為true的時候其結果才會為true，只要有一個以上的運算元為false，則結果就為false。 例如：if (x>0 && y<0 && z==0) a=a+1;
||	邏輯or	適用在2個或2個以上的運算元。只有在所有運算元皆為false的時候，其結果才會為false，只要有一個以上的運算元為true，則結果就為true。 例如：if (x==y || m || func()) a=a+1;
!	邏輯not	只有一個運算元，其作用就是將運算元的邏輯值反相，即為結果 例如：while (!(x+y)) a=a+1;

3.4.4 位元運算（Bitwise）

運算子	名稱	說明

&	位元 AND	適用在 2 個或 2 個以上的運算元，其作用是將運算元的每一個位元做個別的 AND 運算。例如：
		char A=55; // 00110111_2
		char B=81; // 01010001_2
		char C=A & B; // $00010001_2 = 17_{10}$

|	位元 OR	適用在 2 個或 2 個以上的運算元，其作用是將運算元的每一個位元做個別的 OR 運算。例如：
		char A=55; // 00110111_2
		char B=81; // 01010001_2
		char C=A | B; // $01110111_2 = 119_{10}$

^	位元 XOR	適用在 2 個或 2 個以上的運算元，其作用是將運算元的每一個位元做個別的 XOR 運算。例如：
		char A=55; // 00110111_2
		char B=81; // 01010001_2
		char C=A ^ B; // $01100110_2 = 102_{10}$

3.4.4 位元運算（Bitwise）

~	位元 NOT	只有 1 個運算元，其作用是將運算元的每一個位元做 NOT 運算。例如：
		char A=55; // 00110111_2
		char B=~A; // $11001000_2 = 200_{10}$

<<	位元左移	將運算元中所有的位元向左移動 n 個位元，等同乘上 2^n 倍，向左移動後，超出儲存範圍的數字捨去，右邊位元則補上 0。例如：
		char A=23; // 00010111_2
		char B=A<<2; // $01011100_2 = 92_{10}$，將 A 左移 2 個位元

>>	位元右移	將運算元中所有的位元向右移動 n 個位元，等同除以 2^n，向右移出的位元就直接捨去，而左邊多出的位元就補上 0。例如：
		char A=23; // 00010111_2
		char B=A>>3; // $00000010_2 = 2_{10}$，將 A 右移 3 個位元

3.4.5 複合運算（Compound）

複合運算子	範例	運算結果等同
++	a++;	a=a+1; a+=1;
--	a--;	a=a-1; a-=1;
+=	a+=5; a+=b;	a=a+5; a=a+b;
-=	a-=5; a-=b;	a=a-5; a=a-b;
*=	a*=2; a*=a;	a=a*2; a=a*b;
/=	a/=2; a/=b;	a=a/2; a=a/b;

3.4.5 複合運算（Compound）

%=	a%=2; a%=b;	a=a%2; a=a%b;
&=	a&=2; a&=b;	a=a&2; a=a&b;
|=	a|=2; a|=b;	a=a|2; a=a|b;

3.4.6 運算子的優先順序

優先順序　高 → 低									
1	2	3	4	5	6	7	8	9	10
( )	! ~ ++ −−	* / %	+ −	<< >>	> >= < <=	== !=	& | ^	&& ||	=

3.5 Arduino的流程控制

- 在Arduino程式語言中流程控制敘述可以分成二大類，分別為無條件跳躍（unconditional jump）與條件式控制（conditional control）：

條件式控制	無條件跳躍
if if…else switch…case while do…while for	break continue return goto

3.5.1 if 條件式控制

```
if (條件)
敘述 1;   //條件式執行
敘述 2;
敘述 3;
…
```

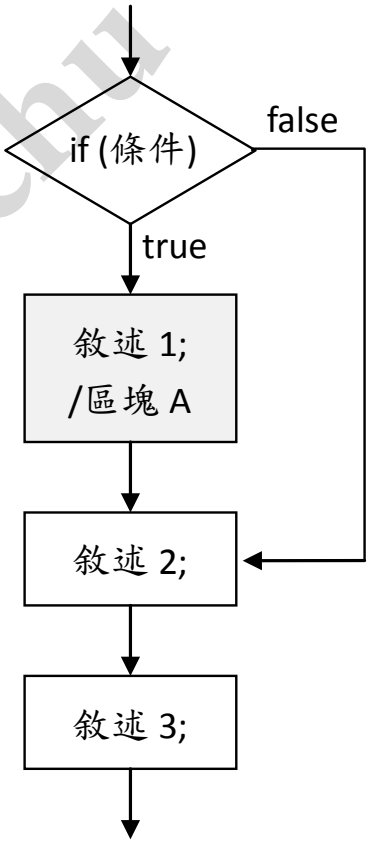
(a)

```
if (條件) 敘述 1;
敘述 2;
敘述 3;
…
```

(b)

```
if (條件)
{
    區塊 A
}
敘述 2;
敘述 3;
…
```

(c)



(d)

3.5.1 if...else 條件式控制

```
if (條件)
    敘述 1;
else
    敘述 2;
敘述 3;
```

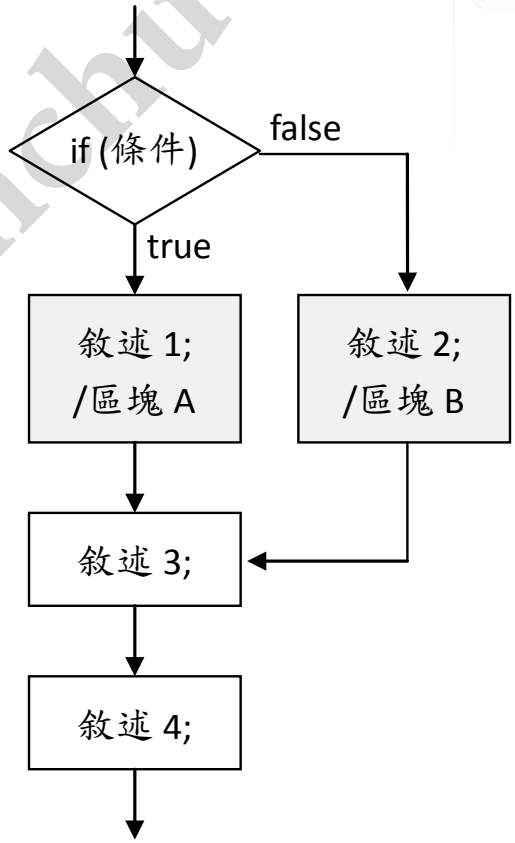
(a)

```
if (條件) 敘述 1;
else 敘述 2;
敘述 3;
…
```

(b)

```
if (條件)
{
    區塊 A
}
else
{
    區塊 B
}
敘述 3;
敘述 4;
…
```

(c)



(d)

3.5.1 if...else 條件式控制

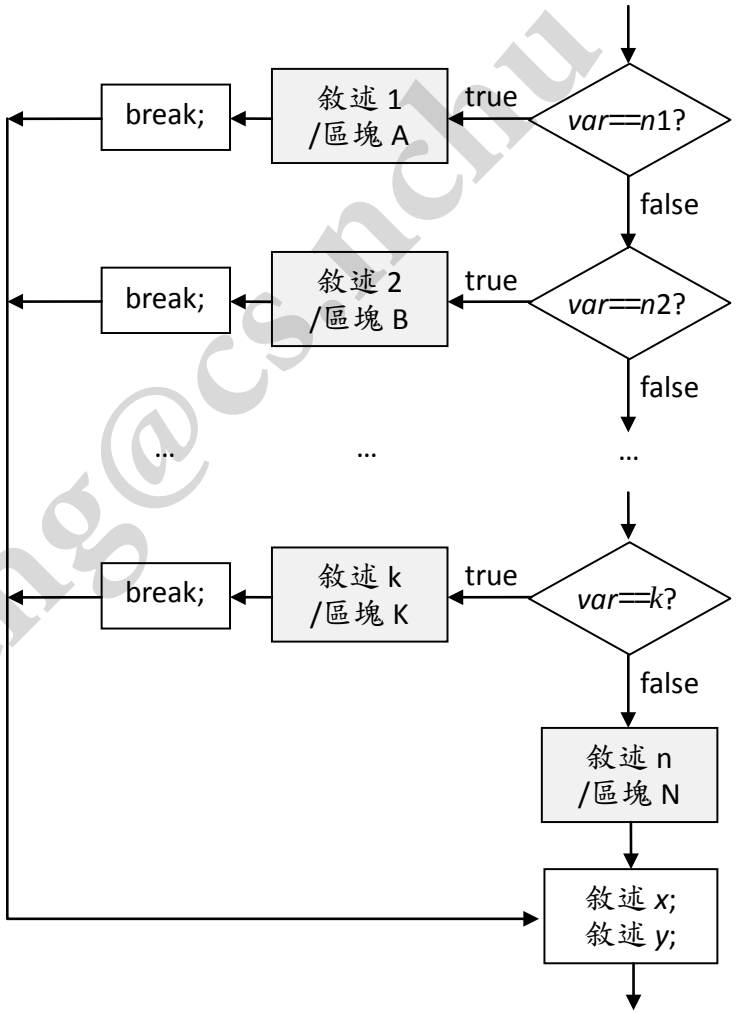
<範例 3.xx> ok

```
if(a==0) b++;          //判斷 a 是否為 0
if(a>0 && a<0) b++;    //判斷 0<a<5 ?
if(a==2 && (b==5 || b==7)) c++;  //要同時滿足 a 為 2，而且 b 為 5 或 7，才為 true
a=2; if(a) b++;        //只要 a 不為 0 都視為 true
a=0; if(!a) b++;       //a 為 0 視為 false，所以!a 為 true
a=5; if(!a) b++;       //只要 a 不為 0，!a 為 false，不可使用位元 NOT，~a≠!a
if(a=2) b++;  //若(a==2)錯寫成(a=2)，不會出現編譯錯誤，但會造成恆為 true 的效果
if(a=0) b++;  //若(a==0)錯寫成(a=0)，不會出現編譯錯誤，但會造成恆為 false 的效果
```

3.5.2 switch…case多重分支控制

```
switch (var) {
   case n1:
        敘述 1/區塊 A
        break;   //一定要寫
   case n2:
        敘述 2/區塊 B
        break;   //一定要寫
…
…
   case k:
        敘述 k/區塊 K
        break;   //一定要寫
   default:  //預設
        敘述 n/區塊 N
        break;   //可省略
}
敘述 x;
敘述 y;
```

(a)



(b)

3.5.3 do…while 迴圈控制

- 如果條件式執行的程式碼只有執行一次的需求，那if (if…else)和switch…case的控制敘述就已適用，可是當條件式執行的程式碼需要多次甚至是重複不斷的執行時，則我們就必須選用while迴圈控制才可達到目的。

```
do
敘述 1;  //條件式執行
while (條件)
敘述 2;
敘述 3;
```

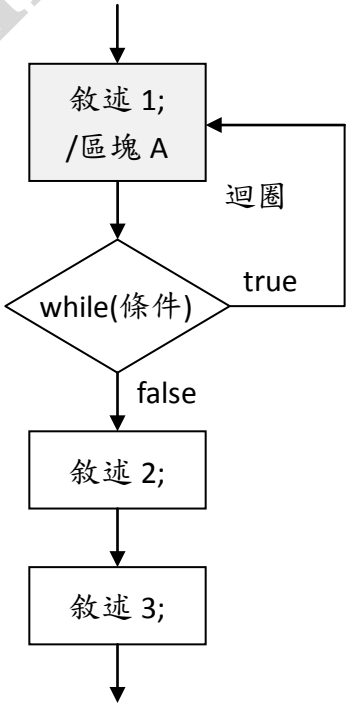
(a)

```
do 敘述 1;
while (條件)
敘述 2;
敘述 3;
```

(b)

```
do
{
   區塊 A
} while (條件)
敘述 2;
敘述 3;
```

(c)



(d)

3.5.3 while 迴圈控制

```
while (條件)
敘述 1;   //條件式執行
敘述 2;
敘述 3;
…
```

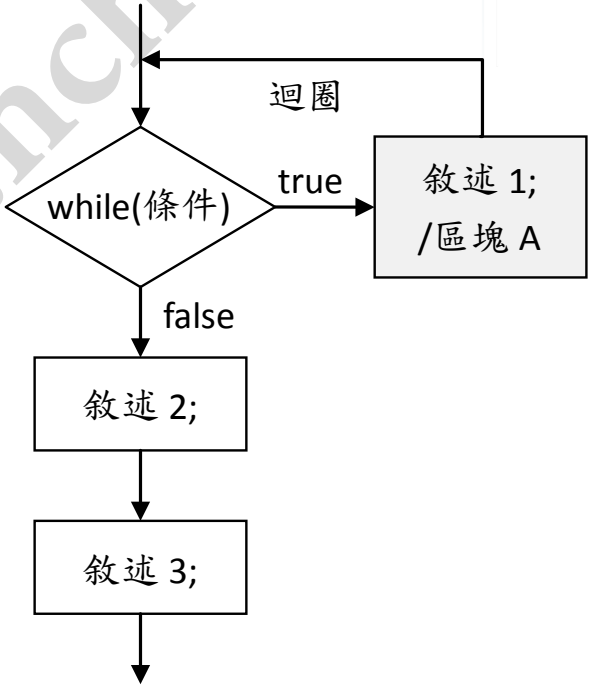
(a)

```
while (條件) 敘述 1;
敘述 2;
敘述 3;
…
```

(b)

```
while (條件)
{
    區塊 A
}
敘述 2;
敘述 3;
…
```

(c)



(d)

3.5.4 for迴圈控制

for(第一運算式；條件；第二運算式) { ... }

(1) for迴圈控制固定由三個欄位所組成，分別是第一運算式、條件與第二運算式，每個欄位之間以「；」做為間隔。。

(2) for迴圈的三個欄位都是非必要的，可選擇性的使用或是省略，但是省略的時候間隔符號「；」一定要保留。如果三個欄位都省略的時候就相當於while的無窮迴圈，範例如下：for(;;) = while(1)

(3) 第一運算式是在進入for迴圈前所執行的運算，執行次數只有一次，通常是迴圈控制變數的初始設定，但不以此為限。

(4) 第二個欄位為條件運算式，其結果為布林常數（true/false），只有0值為false，非0值則皆為true，控制著要繼續執行迴圈或是要結束迴圈。

(5) 第二運算式是在迴圈區塊結束後所執行的運算，每一次迴圈內容結束後都會先執行第二運算式再回到條件的判斷。

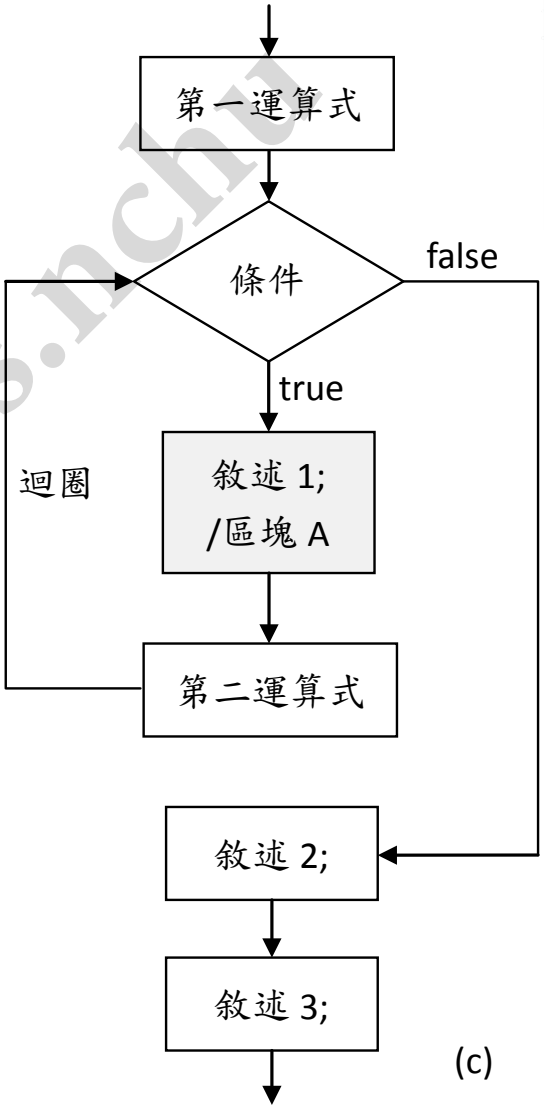
3.5.4 for迴圈控制

```
for(第一運算式；條件；第二運算式) 敘述 1;
敘述 2;
敘述 3;
…
```

(a)

```
for(第一運算式；條件；第二運算式)
{
    區塊 A
}
敘述 2;
敘述 3;
…
```

(b)



(c)

3.5.5 無條件跳躍 break

- break使用在for、while、do…while迴圈結構的強制離開，程式遇到break會立即停止迴圈的執行，並且跳到迴圈區塊之外的下一行敘述往下執行，這時候迴圈並不是正常的結束。

<範例 3.xx> ok

```
a=11;
While(1) {
  b=a%2; Serial.println(b);
  a=a/2;
  if(a==0) break;  //當 a/2 的商值為 0 時，則 break 迴圈
}
```

3.5.5 無條件跳躍 continue

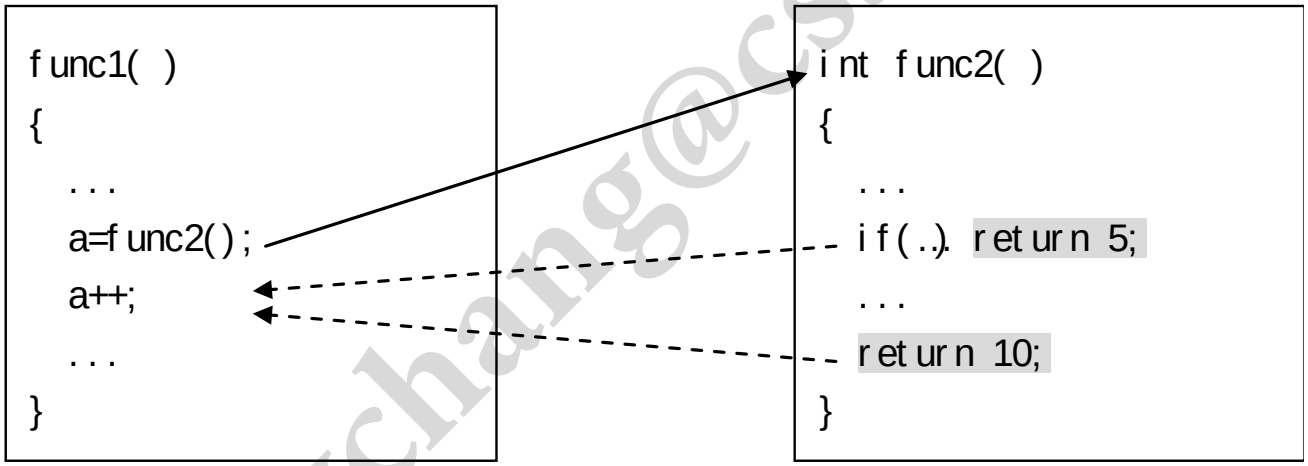
- continue也是使用在for、while、do…while迴圈結構的控制，與break不同的地方，continue並不會跳出迴圈的執行，而是省略迴圈剩下的敘述，直接跳回到迴圈條件的判斷，進行迴圈下一回合（next iteration）的執行，特別注意若是使用在for迴圈，continue之後會先執行第二運算式，再進行條件判斷。

<範例 3.xx> ok

```
For(i=0;i<=10;i++) {
  if(i%2==0) continue;  //若 i 為偶數,則 continue
  n=i*2;
  Serial.println(n);
}
```

3.5.5 無條件跳躍 return

- return是使用在函式的結束返回，會返回到上一層呼叫此函式的下一行敘述繼續執行，如有必要return還具有回傳數值的功能，依據使用者的需求return的返回值可有可無。



3.5.5 無條件跳躍 goto

- goto的使用必須搭配label的觀念，label是標籤的意思，使用在程式碼中具有標示定位的功能，要設定一個標籤非常的簡單，只要在一個非保留字的名稱後加上「：」就是一個標籤名稱；而goto後面必須要接著一個標籤名稱，才會強制執行流程轉移到標籤名稱的地方繼續執行

```
goto LB_b; //跳到LB_b標籤處執行
...
LB_a: //設定LB_a標籤
...
LB_b: //設定LB_b標籤
```

- 在這裡要特別強調，goto只能跳到同一個函式內部的label位置，無法跨函式的跳躍。因為goto的使用會破壞程式碼的結構化，讓程式碼的執行流程沒有規則不容易追蹤，因此大部分的教科書或是有經驗的程式設計師都不建議使用goto，但是有時候又必須借助goto強大的威力才能大幅的降低程式控制的複雜度，所以goto就像是一把雙面刃，使用的時候要特別小心。