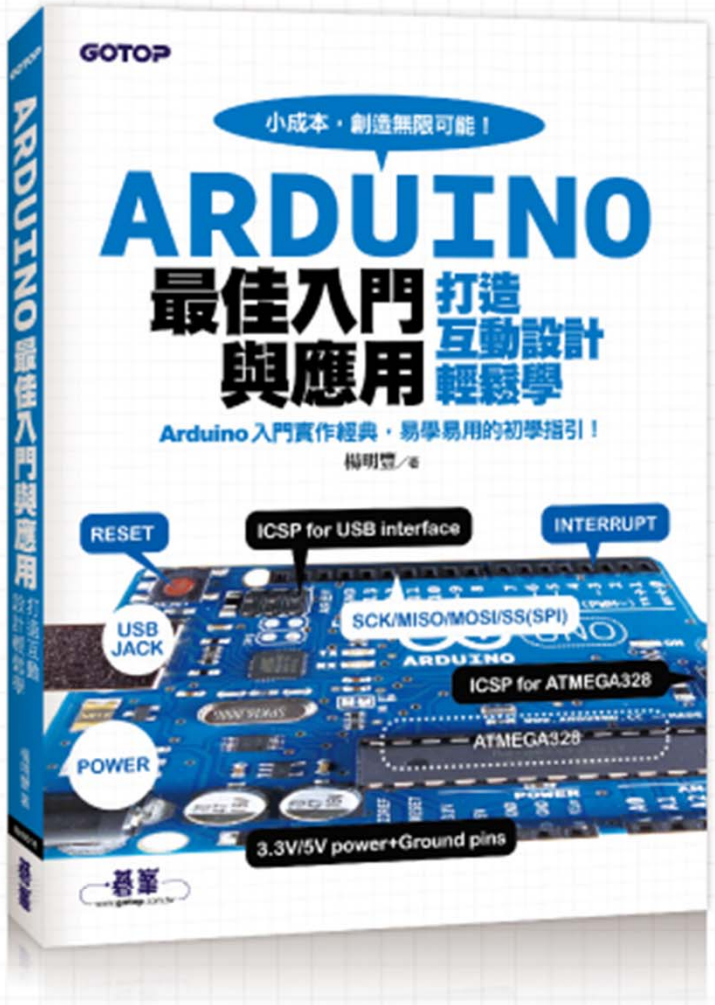

小成本，創造無限可能！

ARDUINO 最佳入門與應用 打造互動設計輕鬆學

Arduino入門實作經典，易學易用的初學指引！

第3章 Arduino語言基礎

碁峰資訊

- C語言是一種常用的高階語言，由函式（function）所組成，所謂函式是指執行某一特定功能的程式集合。
- 當我們在設計C語言程式時，首先會依所需的功能先寫一個函式，然後再由主程式或函式去呼叫執行另一個函式。
- 在函式名稱後面會接一組小括號“ ( ) ”，通知編譯器此為一個函式，而不是變數，而在小括號內也可包含引數，引數是用來將主程式中的變數數值或變數位址傳至函式中來運算。

```
main()                                  //主函式名稱，括號內可包含引數。
{                                       //函式開始。
    printf(“I love C language”);        //一個敘述，並以分號結束敘述。
}                                       //函式結束。
```

圖 3-1 C 語言結構

- Arduino程式與C語言程式很相似，但語法更簡單而且易學易用，完全將微控制器中複雜的暫存器設定寫成函式，使用者只需輸入參數即可。
- Arduino程式主要由結構（structure）、數值(values)及函式（functions）等三個部份組成。
  - Arduino程式的結構（structure）包含setup( )及loop( )兩個函式。
  - 數值(values)包含常數及變數 (變數又分為外部變數及內部變數)。
  - 函式（functions）分成(1)公用函式 (2)自訂函式
- setup( )函式由其名稱“setup”暗示執行 “設定” 的動作，用來初始化變數、設定接腳模式為輸入（INPUT）或輸出（OUTPUT）等。在每次通電或重置 Arduino 電路板時，setup( )函數只會被執行一次。
- loop()函式由其名稱"loop"暗示執行"迴圈"的動作，用來設計程式控制Arduino電路板執行所需的功能，並且重覆執行。

```
void setup()                    //初始化變數、設定接腳模式等。
{}
void loop()                     //迴圈。
{}
```

- 在Arduino程式中常使用變數（variables）與常數（constants）來取代記憶體的實際位址，好處是程式設計者不需要知道那些位址是可以使用的，而且程式將會更容易閱讀與維護。

- 一個變數或常數的宣告是為了保留記憶體空間給某個資料來儲存，至於是安排那一個位址，則是由編譯器統一來分配。

3-3-1 變數名稱

- Arduino語言的變數命名規則與C語言相似，必須是由英文字母、數字或底線符號“_”之後，再緊接著字母或數字，並且第一個字元不可以是數字。因此我們在命名變數名稱時，應該以容易閱讀為原則，例如col、row代表行與列，就比i、j更容易了解。

3-3-2 資料型態

表3-1　資料型態

資料型態	位元數	範圍
boolean	8	true（定義為非 0），false（定義為 0）
char	8	–128~ + 127
unsigned char	8	0~ 255
byte	8	0~ 255
int 註 1	16	–32,768~ + 32,767
unsigned int 註 2	16	0~ 65,535
word	16	0~ 65,535
long	32	–2,147,483,648~ + 2,147,483,647
unsigned long	32	0~ 4,294,967,295
short	16	–32,768~ + 32,767
float	32	–3.4028235E+ 38~ + 3.4028235E+ 38
double 註 3	32	–3.4028235E+ 38~ + 3.4028235E+ 38

註 1：在 Arduino Due 板為 32 位元，其餘為 16 位元。

註 2：在 Arduino Due 板為 32 位元，其餘為 16 位元。

註 3：在 Arduino Due 板為 64 位元，其餘為 32 位元。

3-3-3 變數宣告

- int ledPin=10; //宣告整數變數ledPin，初始值為10。
- char myChar='A'; //宣告字元變數myChar，初始值為' A'。
- float sensorVal=12.34 //宣告浮點數變數sensorVal，初始值為12.34。
- int year=2013,moon=7,day=11; //宣告整數變數year、moon、day及其初值。

3-3-4 變數的生命週期

- 全域變數
- 全域變數被宣告在任何函式之外。
- 當執行Arduino程式時，全域變數即被產生並且配置記憶體空間給這些全域變數，直到程式結束執行時，才會釋放這些佔用的記憶體空間。
- 全域變數並不會禁止與其無關的函式作存取的動作，因此在使用上要特別小心，避免變數數值可能被不經意地更改。因此除非有特別需求，否則還是儘量使用區域變數。

- 區域變數
- 區域變數又稱為自動變數，被宣告在函式的大括號“{ }”內。
- 當函式被呼叫使用時，這些區域變數就會自動的產生，系統會配置記憶體空間給這些區域變數，當函式結束時，這些區域變數又自動的消失並且釋放所佔用的記憶體空間。

```
int     total;                          //全域變數total在所有函數內皆有效。
void setup()
{
        //...
}
void loop()
{
        int i;                          //區域變數i只有在loop()函數內才有效。
        for(int j=0;j<100;j++)          //區域變數j只有在for迴圈內才有效。
        {
                //...
        }
}
```

3-3-5 變數型態的轉換

- 在Arduino程式中可以使用char(x)、byte(x)、int(x)、word(x)、long(x)、float(x)等資料型態轉換函式來改變變數的資料型態，引數x可以是任何型態的資料。

- 電腦除了能夠儲存資料之外，還必須具備運算的能力，而在運算時所使用的符號，即稱為運算子（operator）。
- 常用的運算子可分為算術運算子、關係運算子、邏輯運算子、位元運算子與指定運算子等三種。
- 當敘述中包含不同運算子時，Arduino微控制器會先執行算術運算子，其次是關係運算子，最後才是邏輯運算子。
- 我們也可以使用小括號( )來改變運算的順序。

表3-2　算術運算子

算術運算子	動作	範例	說明
+	加法	a+ b	a 內含值與 b 內含值相加。
-	減法	a-b	a 內含值與 b 內含值相減。
*	乘法	a* b	a 內含值與 b 內含值相乘。
/	除法	a/b	取 a 內含值除以 b 內含值的商數。
%	餘數	a%b	取 a 內含值除以 b 內含值的餘數。
+ +	遞增	a+ +	a 的內含值加 1，即 a= a+ 1。
– –	遞減	a– –	a 的內含值減 1，即 a= a-1。

- 程式範例

```
void setup()
{
}
void loop()
{
    int a=20,b=3;                    //宣告整數變數 a、b 並指定變數初值。
    int c,d,e,f;                     //宣告整數變數 c、d、e、f。
    c=a+b;                           //加法運算。
    d=a-b;                           //減法運算。
    e=a/b;                           //除法運算。
    f=a%b;                           //餘數運算。
    a++;                             //遞增。
    b--;                             //遞減。
}
```

- 運算結果

```
c=23,d=17,e=6,f=2,a=21,b=2
```

表3-3　關係運算子

比較運算子	動作	範例	說明
= =	等於	a= = b	a 等於 b? 若為真，結果為 true，否則為 false。
!=	不等於	a!= b	a 不等於 b? 若為真，結果為 true，否則為 false。
<	小於	a< b	a 小於 b? 若為真，結果為 true，否則為 false。
>	大於	a> b	a 大於 b? 若為真，結果為 true，否則為 false。
< =	小於等於	a< = b	若 a 小於或等於 b，結果為 true，否則為 false。
> =	大於等於	a> = b	若 a 大於或等於 b，結果為 true，否則為 false。

- 程式範例

```
void setup()
{
}
void loop()
{
    int x=200;                          //宣告整數變數 x，初始值為 200。
    if(x>100)                           //x 大於 100?
       digitalWrite(LEDpin,HIGH);       //若 x 大於 100 則點亮 LED。
    else                                //x 小於或等於 100。
       digitalWrite(LEDpin,LOW);        //x 小於或等於 200，則關閉 LED。
}
```

- 運算結果

點亮 LED。

表3-4　邏輯運算子

邏輯運算子	動作	範例	說明
&&	AND	a&&b	a 與 b 兩變數執行邏輯 AND 運算。
||	OR	a|| b	a 與 b 兩變數執行邏輯 OR 運算。
!	NOT	!a	a 變數執行邏輯 NOT 運算。

- 程式範例

```
void setup()
{
}
void loop()
{
    boolean a=true,b=false,c,d,e;     //宣告布林變數 a、b、c、d、e。
    c=a&&b;                           //a、b 兩變數作邏輯 AND 運算。
    d=a||b;                           //a、b 兩變數作邏輯 OR 運算。
    e=!a;                             //a 變數作邏輯 NOT 運算。
}
```

表3-5　位元運算子

位元運算子	動作	範例	說明
&	AND	a&b	a 與 b 兩變數的每一相同位元執行 AND 邏輯運算。
|	OR	a| b	a 與 b 兩變數的每一相同位元執行 OR 邏輯運算。
^	XOR	a^ b	a 與 b 兩變數的每一相同位元執行 XOR 邏輯運算。
~	補數	~ a	將 a 變數中的每一位元反相(0、1 互換)。
< <	左移	a< < 4	將 a 變數內含值左移 4 個位元。
> >	右移	a> > 4	將 a 變數內含值右移 4 個位元。

- 程式範例

```
void setup()
{
}
void loop()
{
    char a=0b00100101;          //宣告字元變數 a=0b00100101(二進值)。
    char b=0b11110000;          //宣告字元變數 b=0b11110000(二進值)。
    char c=0x80;                //宣告有號數字元變數 c=0x80(十六進值)。
    unsigned char d,e,f,l,m,n;  //宣告無號數字元變數 d、e、f、l、m、n。
    d=a&b;                      //a、b 兩變數執行位元 AND 邏輯運算。
    e=a|b;                      //a、b 兩變數執行位元 OR 邏輯運算。
    f=a^b;                      //a、b 兩變數執行位元 XOR 邏輯運算。
    l=~a;                       //a 變數執行位元反 NOT 邏輯運算。
    m=b<<1;                     //b 變數內容左移 1 位元。
    n=c>>1;                     //c 變數內容右移 1 位元。
}
```

- 運算結果

```
d=0x20,e=0xf5,f=0xd5,l=0xda,m=0xe0,n=0x40
```

表3-6　複合運算子

複合運算子	動作	範例	說明
+ =	加	a+ = b	與 a= a+ b 運算相同。
-=	減	a-= b	與 a= a-b 運算相同。
*=	乘	a*= b	與 a= a*b 運算相同。
/=	除	a/= b	與 a= a/b 運算相同。
%=	餘數	a%= b	與 a= a%b 運算相同。
&=	位元 AND	a&= b	與 a= a&b 運算相同。
| =	位元 OR	a| = b	與 a= a| b 運算相同。
^ =	位元 XOR	a^ = b	與 a= a^ b 運算相同。

- 程式範例

```
void setup()
{}
void loop()
{
    int x=2;                          //宣告整數變數 x，設定初值為 2。
    char a=0b00100101;                //宣告字元變數 a=0b00100101(二進值)。
    char b=0b00001111;                //宣告字元變數 b=0b00001111(二進值)。
    x+=4;                             //x=x+4=6。
    x-=3;                             //x=x-3=3。
    x*=10;                            //x=x*10=30。
    x/=2;                             //x=x/2=15。
    x%=2;                             //x=x%2=1。
    a&=b;                             //a=a&b=0b00000101。
    a|=b;                             //a=a|b=0b00001111。
    a^=b;                             //a=a^b=0b00000000。
}
```

- 運算結果

```
x=1,a=0b00000000,b=0b00001111
```

表3-7　運算子的優先順序

優先順序	運算子	說明
1	( )	括號
2	~ ！	補數、NOT 運算
3	+ + 、– –	遞增、遞減
4	* ， / ，%	乘法，除法，餘數
5	+ ，–	加法，減法
6	< < ，> >	移位
7	< > ，< = ，> =	關係
8	= = ，!=	相等、不等
9	&	位元 AND 運算
10	^	位元 XOR 運算
11	|	位元 OR 運算
12	&&	邏輯 AND 運算
13	||	邏輯 OR 運算
14	*= ，/= ，%/，+ = ，–= ，&= ，^ = ，| =	複合運算

3-5-1 迴圈控制指令—for迴圈

- for 迴圈

  for 迴圈是由初值運算式、條件運算式與增量或減量運算式三部份組成，彼此之間以分號隔開。

  (1) 初值：設定初值，可由任何數值開始。

  (2) 條件：若條件為真，則執行括號"{ }"中的敘述，否則離開迴圈。

  (3) 增量(或減量)：每執行一次迴圈內的動作後，依增量(減量)遞增(遞減)。

- 指令格式

```
for(初值; 條件; 增量或減量)
{
    //敘述;
}
```

3-5-1 迴圈控制指令—for迴圈

- 程式範例

```
void setup()
{}
void loop()
{
    int i,s=0;                          //宣告整數變數。
    for(i=0;i<=10;i++)                  //當 i<=10 時，執行迴圈。
        s=s+i;                          //s=1+2+…+10。
}
```

- 執行結果

```
s=55
```

3-5-1 迴圈控制指令—while迴圈

- while 迴圈

  while 為先判斷型迴圈，當條件式為真時，則執行大括號"{ }"中的敘述，直到條件式為假不成立時，才結束 while 迴圈。在 while 條件式中沒有初值運算式及增量(或減量)運算式，因此必須在敘述中設定。

- 指令格式

```
while( 條件式 )                    //先判斷型迴圈。
{
    //敘述;
}
```

3-5-1 迴圈控制指令—while迴圈

- 程式範例

```
void setup()
{
}
void loop()
{   int i=0,s=0;                    //宣告整數變數。
    while(i<=10)                    //當 i 小於或等於 10 時，執行迴圈。
    {
       s=s+i;                       //s=1+2+3+…+10。
       i++;                         //i 遞增。
    }
}
```

- 執行結果

```
s=55
```

3-5-1 迴圈控制指令—do-while迴圈

- do-while 迴圈

  do-while 為後判斷型迴圈，會先執行大括號"{ }"中的敘述一次，然後再判斷條件式，當條件式為真時，則繼續執行大括號"{ }"中的動作，直到條件式為假時，才結束 do-while 迴圈。因此 do-while 迴圈至少執行一次。

- 指令格式

```
do  {                                    //先執行敘述一次。
        //敘述;
    }
while( 條件式 )                          //後判斷型迴圈。
```

3-5-1 迴圈控制指令—do-while迴圈

- 程式範例

```
void setup()
{}
void loop()
{
    int i=0,s=0;                    //宣告整數變數。
    do {
       s=s+i;                       //s=1+2+3+…+10。
       i++;                         //i 遞增。
    }
    while(i<=10)                    //當 i 小於或等於 10 時，繼續執行迴圈。
}
```

- 執行結果

```
s=55
```

3-5-1 迴圈控制指令—do-while迴圈

- 程式範例

```
void setup()
{}
void loop()
{
    int i=0,s=0;                    //宣告整數變數。
    do {
       s=s+i;                       //s=1+2+3+…+10。
       i++;                         //i 遞增。
    }
    while(i<=10)                    //當 i 小於或等於 10 時，繼續執行迴圈。
}
```

- 執行結果

```
s=55
```

3-5-2 條件控制指令—if敘述

- if 敘述

  if 敘述會先判斷條件式，若條件式為真時，則執行一次大括號{}中的敘述，若條件式為假時，則不執行。if 敘述內如果只有一行敘述時，可以不用加大括號"{ }"，但如果有一行以上敘述時，一定要加上大括號"{ }"，否則在 if 敘述內只會執行第一行敘述，其餘敘述則視為在 if 敘述之外。

- 指令格式

```
if ( 條件式 )
{
    //敘述;
}
```

3-5-2 條件控制指令—if敘述

- 程式範例

```
void setup()
{}
void loop()
{
    int a=2,b=3,c=0;                    //宣告整數變數。
    if(a>b)                             //a 大於 b?
    {
       c=a;                             //若 a 大於 b，則 c=a。
    }
}
```

- 執行結果

```
c=0
```

3-5-2 條件控制指令—if-else敘述

- if-else 敘述

  if-else 敘述會先判斷條件式，若條件式為真時，則執行敘述 1，若條件式為假時，則執行敘述 2。在 if 敘述或 else 敘述內，如果只有一行敘述時，可以不用加大括號"{ }"，但如果有一行以上敘述時，一定要加上大括號"{ }"。

- 指令格式

```
if ( 條件式 )                          //條件式為真，執行敘述1。
{
    //敘述1;
}
else                                   //條件式為假，執行敘述2。
{
    //敘述2;
}
```

3-5-2 條件控制指令—if-else敘述

- 程式範例

```
void setup()
{}
void loop()
{
    int a=3,b=2,c=0;                    //宣告整數變數。
    if(a>b)                             //a 大於 b?
    {
       c=a;                             //若 a 大於 b，則 c=a。
    }
    else                                //a 小於或等於 b
    {
       c=b;                             //c=b。
    }
}
```

- 執行結果

```
c=3
```

3-5-2 條件控制指令—巢狀if-else敘述

- 巢狀 if-else 敘述

  使用巢狀 if-else 敘述時，必須注意 if 與 else 的配合，其原則是 else 要與最接近且未配對的 if 配成一對，通常我們都是以 Tab 定位鍵或空白字元來對齊配對的 if-else，才不會有錯誤動作出現。在 if 敘述或 else 敘述內，如果只有一行敘述時，可以不用加大括號{}，但如果有一行以上敘述時，一定要加上大括號"{ }"。

- 指令格式

```
if ( 條件１ )
    if ( 條件２ )
       if ( 條件３ )                    //條件１、２、３皆成立時，執行敘述１。
          {//敘述１;}
       else                             //條件１、２成立且條件３不成立，執行敘述２。
          {//敘述２;}
    else
       {//敘述３;}                       //條件１成立且條件２不成立時，執行敘述３。
else
    {//敘述４; }                         //條件１不成立時，執行敘述４。
```

3-5-2 條件控制指令—巢狀if-else敘述

- 程式範例

```
void setup()
{}
void loop()
{
    int score=75;
    char grade;
    if(score>=60)                    //成績大於或等於60分?
       if(score>=70)                 //成績大於或等於70分?
          if(score>=80)              //成績大於或等於80分?
             if(score>=90)           //成績大於或等於90分?
                grade='A';           //成績大於或等於90分，等級為A。
             else                    //成績在80~90分之間。
                grade='B';           //成績在80~90分之間，等級為B。
          else                       //成績在70~80分之間。
             grade='C';              //成績在70~80分之間，等級為C。
       else                          //成績在60~70分之間。
          grade='D';                 //成績在60~70分之間，等級為D。
    else                             //成績小於60分。
       grade='E';                    //成績小於60分，等級為E。
}
```

3-5-2 條件控制指令—if-else-if敘述

- if-else if 敘述

  使用 if-else if 敘述時，必須注意 if 與 else if 的配合，其原則是 else if 要與最接近且未配對的 if 配成一對，通常我們都是以 Tab 定位鍵或空白字元來對齊配對的 if-else，才不會有錯誤動作出現。在 if 敘述或 else 敘述內，如果只有一行敘述時，可以不用加大括號{}，但是一行以上敘述時，一定要加上大括號"{ }"。

- 指令格式

```
if( 條件1 )                          //條件1成立?
{                                    //條件1成立，執行敘述1。
    //敘述1;
}
else if( 條件2 )                     //條件2成立?
{                                    //條件2成立，執行敘述2。
    //敘述2;
}
else if( 條件3 )                     //條件3成立?
{                                    //條件3成立，執行敘述3。
    //敘述3;
}
else                                 //條件1、2、3皆不成立。
{                                    //條件1、2、3皆不成立，執行敘述4。
    //敘述4;
}
```

3-5-2 條件控制指令—if-else-if敘述

- 程式範例

```
void setup()
{}
void loop()
{
    int score=75;                       //成績。
    char grade;                         //等級。
    if(score>=90 && score<=100)         //成績大於或等於 90 分?
       grade='A';                       //成績大於或等於 90 分，等級為 A。
    else if(score>=80 && score<90)      //成績在 80~90 分之間?
       grade='B';                       //成績在 80~90 分之間，等級為 B。
    else if(score>=70 && score<80)      //成績在 70~80 分之間?
       grade='C';                       //成績在 70~80 分之間，等級為 C。
    else if(score>=60 && score<70)      //成績在 60~70 分之間?
       grade='D';                       //成績在 60~70 分之間，等級為 D。
    else                                //成績小於 60 分。
       grade='E';                       //成績小於 60 分，等級為 E。
}
```

- 執行結果

```
grade='C'
```

3-5-2 條件控制指令—switch-case敘述

switch-case 敘述與 if-else if 敘述類似，但 switch-case 敘述的格式較清楚而且有彈性。if-else if 敘述是二選一的程式流程控制指令，而 switch-case 則是多選一的程式流程控制指令。switch 以條件式運算的結果與 case 所指定的條件值比對，若與某個 case 中的條件值比對相同，則執行該 case 所指定的敘述，若所有的條件值都不符合，則執行 default 所指定的敘述，在 switch 內的條件式運算結果必須是整數或字元。如果要結束 case 中的動作，可以使用 break 敘述，但是一次只能跳出一層迴圈，如果要一次結束多個迴圈，可以使用 goto 指令，但程式的流程將變得更凌亂，所以應儘量少用或不用 goto 指令。

3-5-2 條件控制指令—switch-case敘述

- 指令格式

```
switch ( 條件式 )
{
    case 條件值1:
        { 敘述1;}                    //條件式等於條件值1時，執行敘述1。
        break;
    case 條件值2:
        { 敘述2;}                    //條件式等於條件值2時，執行敘述2。
        break;
    default:
        { 敘述n;}                    //所有的條件值都不等於條件式時，執行敘述n。
}
```

3-5-2 條件控制指令—switch-case敘述

- 程式範例

```
void setup()
{}
void loop()
{
    int score=75;                    //成績。
    int value;                       //數值。
    char grade;                      //等級。
    value=score/10;                  //取出成績十位數值
```

3-5-2 條件控制指令—switch-case敘述

```
    switch(value)                    //以成績十位數值作為判斷條件。
    {
        case 10:                     //成績為100分。
            grade='A';               //成績為100分，等級為A。
            break;                   //結束迴圈。
        case 9:                      //成績大於或等於90分?
            grade='A';               //成績大於或等於90分，等級A。
            break;                   //結束迴圈。
        case 8:                      //成績在80~90分之間?
            grade='B';               //成績在80~90分之間，等級B。
            break;                   //結束迴圈。
        case 7:                      //成績在70~80分之間?
            grade='C';               //成績在70~80分之間，等級C?
            break;                   //結束迴圈。
        case 6:                      //成績在60~70分之間?
            grade='D';               //成績在60~70分之間，等級為D。
            break;                   //結束迴圈。
        default:                     //成績在小於60分。
            grade='E';               //成績在小於60分，等級為E。
            break;                   //結束迴圈。
    }
}
```

- 執行結果

```
grade='C'
```

3-5-3 無條件跳躍指令—goto敘述

- 指令格式

```
goto 標記名稱
```

- 程式範例

```
void setup()
{}
void loop()
{
    int i,j,k;                          //宣告整數變數 i,j,k。
    for(i=0;i<1000;i++)                 //i 迴圈。
       for(j=0;j<1000;j++)              //j 迴圈。
          for(k=0;k<1000;k++)           //k 迴圈。
             if(analogRead(0)>500)      //類比接腳 0 讀值大於 500?
                goto exit;              //類比接腳 0 讀值大於 500，結束 i,j,k 迴圈。
    exit:                               //標記 exit。
}
```

- 執行結果

若類比接腳 A0 讀值大於 500，則結束迴圈。

3-6-1 函式原型

所謂函式原型就是指定傳至函式引數的資料型態與函式傳回值的資料型態，函式原型的宣告包含函式名稱、傳至函式的引數資料型態及函式傳回值的資料型態。當被呼叫的函式必須傳回數值時，函式的最後一個敘述必須使用 return 敘述。使用 return 敘述有兩個目的，一是將控制權轉回給呼叫函式，另一是將 return 敘述後面小括號" ( ) "中的數值傳回給呼叫函式。return 敘述只能從函式傳回一個數值。

- 宣告格式

```
傳回值型態 函數名稱(引數1型態 引數1, 引數2型態 引數2,…引數n型態 引數n)
```

- 宣告範例

```
void func1(void);                    //函式無引數，無傳回值。
void func2(char i);                  //函式有 char 型態引數 i，無傳回值。
char func3(void);                    //函式無引值，有 char 型態的傳回值。
char func4(char i);                  //函式有 char 型態的引數 i 及傳回值。
```

3-6-1 函式原型

- 程式範例

```
void setup()
{}
void loop()
{
    int x=5,y=6,sum;                    //宣告整數變數 x,y,sum。
    sum=area(x,y);                      //呼叫 area 函數。
}
int area(int x,int y)                   //計算面積函式 area()。
{
    int s;
    s=x*y;                              //執行 s=x*y 運算。
    return(s);                          //傳回面積 s 值。
}
```

- 執行結果

```
sum=30。
```

所謂陣列（array）是指存放在連續記憶體中的一群相同資料型態的集合，陣列也如同變數一樣需要先宣告，編譯器才會知道陣列的資料型態及大小。陣列的宣告陣列包含四個項目：

1. 資料型態：在陣列中每個元素的資料型態皆相同。
2. 陣列名稱：命令規則與變數宣告方法相同。
3. 陣列大小：陣列可以是多維的，但必須指定其大小，編譯器才能為陣列配置記憶體空間。
4. 陣列初值：與變數相同，可以事先指定初值或不指定。

3-7-1 一維陣列

- 宣告格式

資料型態　陣列名[陣列大小 n]={初值 0,初值 1,…,初值 n-1};

- 程式範例

```
void setup()
{}
void loop()
{
    int n;                          //陣列註標。
    int a[5]={1,2,3,4,5};           //宣告一維整數陣列。
}
```

- 執行結果

```
a[0]=1,a[1]=2,a[2]=3,a[3]=4,a[4]=5
```

3-7-2 二維陣列

- 宣告格式

```
資料型態 陣列名[m][n]=
    {   {初值0,初值1,…初值n-1},        //第0列。
        {初值0,初值1,…初值n-1},        //第1列。
                  :
        {初值0,初值1,…初值n-1} };      //第m-1列。
```

- 程式範例

```
void setup()
{}
void loop()
{   int m,n;                           //陣列註標。
    int a[2][3]=                       //宣告二維整數陣列。
       {  {0,1,2},                     //第0列。
          {3,4,5} };                   //第1列。
}
```

- 執行結果

```
a[0][0]=0,a[0][1]=1,a[0][2]=2,a[1][0]=3,a[1][1]=4,a[1][2]=5
```

3-7-3 以陣列傳引數

- 當傳遞陣列給函式時，並不會將此陣列複製一份給函式，只是傳遞陣列的位址給函式，函式再利用這個位址與註標去存取原來在主函式中的陣列。

- 程式範例

```
void setup()
{}
void loop()
{   int result;                          //宣告整數變數 result。
    int a[5]={1,2,3,4,5};                //宣告整數陣列 a[5]。
    int size=5;                          //宣告整數變數 size。
    result=sum(a,size);                  //傳址呼叫函式 sum。
     Serial.println(result);
}
int sum(int a[],int size)                //函數 sum。
{
    int i;                               //宣告整數變數 i。
    int result=0;                        //宣告整數變數 result。
    for(i=0;i<size;i++)
       result=result+a[i];               //計算陣列中所有元素的總和。
    return(result);                      //傳回計算結果。
```

- 執行結果

```
result=15
```

3-8-2 #include前置命令

- 指令格式

```
#include <標頭檔>
#include "標頭檔"
```

- 程式範例

```
#include <Servo.h>                    //載入Servo.h標頭檔案。
Servo myservo;                        //定義Servo物件。
int pos = 0;                          //伺服器轉動角度。
void setup()
{
    myservo.attach(9);                //servo連接至數位接腳9。
}
void loop()
{
    for(pos =0; pos<180; pos+=1)      //由0°~180°每次轉動1°。
    {
       myservo.write(pos);            //伺服器轉動至指定的角度。
       delay(15);                     //延遲15ms。
    }
    for(pos=180; pos>=1; pos-=1)      //由180°~0°每次轉動1°。
    {
       myservo.write(pos);            //伺服器轉動至指定的角度。
       delay(15);                     //延遲15ms。
    }
}
```

3-8-1 #define前置命令

- 指令格式

```
#define  巨集名稱  字串
```

- 程式範例

```
#define PI 3.14159                          //定義巨集 PI=3.14159。
#define AREA(x) PI*x*x                      //定義巨集 AREA(X)=PI*x*x。
void setup()
{}
void loop()
{
    float result=AREA(2);                   //計算圓面積。
}
```

- 執行結果

```
result=12.57
```