Chapter 2

# 函式

## 本章大綱



## 2-1 自訂函式

前面資訊科技已經介紹一些公用函式，若公用函式無使用者功能，就必須自己寫此程式片段，但若此程式片段需要在同一程式或不同一個程式重複出現很多次，如果這些程式片段都要在每個地方寫一次，那是一件非常浪費時間的事，且會使程式變得冗長而不易閱讀；更糟的是，萬一要調整此程式片段的部分功能，更要至不同的地方修改，若未完全修改，則會造成執行結果的不一致，程式維護當然困難，此時可透過函式解決以上困難。

函式的使用方式是將此常用的程式片段賦予一個名稱，當程式設計者需要使用此程式片段時，只要使用此名稱即可呼叫此程式片段，此程式片段即稱爲函式，又稱爲副程式。其次，函式完成之後可交由不同的程式呼叫使用，以節省程式設計者的時間。例如：已在前面章節所介紹過的數值函式、字元函式及字串函式等等，這些稱爲公用內儲函式，若公用內儲函式無程式設計者要求的功能，此時就可使用本章的自訂函式。其次，自訂函式同樣可個別存檔，以便讓別的使用者使用。

## 函式之原型宣告與呼叫

在資訊科技一書已經介紹了基本資料型態、陣列型態，本節將介紹函式型態。同理，使用函式前一樣要宣告函式型態，但函式的宣告較前面幾種複雜，它包含有函式的傳回值型態、函式名稱、參數型態與參數名稱，所以另稱爲函式原型（Prototype）宣告，而不稱爲函式宣告。（雖然若說函式宣告，別人也知道指的是函式原型宣告，但較少人使用函式宣告）。函式的使用有 3 個步驟，步驟一：函式原型宣告；步驟二：實作函式本體；步驟三：呼叫函式。分別說明如下：

### ☞ 步驟一：函式原型宣告

函式原型宣告之簡要語法如下：

```
傳回值型態    函式名稱  ([ 參數型態1 參數名稱1, ...]....);
```

以上語法說明如下：

1. 傳回值型態爲函式所要傳回的資料型態，可以是任一基本資料型態（int、char 及 double 等）、結構及指標等，若無傳回值，則用 void 表示。
2. 函式名稱的命名同識別字命名規則。
3. 參數型態與參數名稱必須成對出現。參數爲主程式與函式傳遞資料或回傳執行結果的窗口。若無資料傳遞則括號內空白即可。
4. 函式原型宣告的位置，亦影響其有效範圍，其有效範圍同一般變數的宣告。
5. 以下程式片段爲函式原型宣告，其意義是函式傳回一個整數型態，函數名稱爲 add，且傳遞兩個整數型態給函式本體。

```
int add (int a, int b);
```

其次，函式原型宣告的變數名稱可省略，例如：以上敘述亦可簡寫如下：

```
int add (int , int );
```

### ☞ 步驟二：實作函式本體

實作函式本體的簡要語法如下：

```
傳回值型態    函式名稱  ( [ 參數 1, 參數 2...] ){
[ 敘述區塊;]
[ return ( 運算式);]
[ 敘述區塊;]
}
```

以上語法說明如下：

1. 傳回值型態與函式名稱須與函式原型宣告相同。
2. 每個函式可放在 main 函式前面或後面均可，若放在 main() 前面則函式原型宣告還可省略，其次，每個函式的地位均相同，所以不可以在函式中再定義其它函式。
3. 函式原則上從左大括號『{』執行到右大括號『}』，但若中途有特殊原因要離開函式，可用 return 提早離開。
4. 參數（Parameter）有些書亦稱爲引數（Argument）。函式呼叫的參數名稱與實作函式本體的參數名稱可相同或不同都可以，但其資料具有傳遞性，也就是函式呼叫的參數將會傳遞給函式本體，至於函式本體的運算結果是否回傳至主程式，那就得依參數的傳遞方式了，請看下一節。
5. 以下程式片段，可將所傳來的兩個參數相加並傳回。

```
int add (int a, int b){
    int c;
    c=a+b;
```

```
    return (c);
}
```

### ☛ 步驟三：呼叫函式

呼叫函式的簡要語法如下：

```
[ 傳回值 =]   函式名稱 ([ 參數 1, 參數 2, ...]);
```

以上語法說明如下：

1. 函式若無傳回值，也就是其傳回值型態爲『void』，則函式名稱前的傳回值可省略。
2. 主程式呼叫函式與函式原型宣告的參數稱爲實際參數（Actual Parameter），函式本體的參數稱爲形式參數（Formal Parameter），且兩者的名稱可以不同也可相同。實際參數會將參數值傳給形式參數，而形式參數是否將值傳回，則要看參數傳遞方式，參數的傳遞方式有傳值、傳址及傳參考，請看下一節。
3. 以下程式片段可呼叫 add 函式，傳回值皆爲 8：

```
c=add (6, 2);
```

或

```
m=6; n=2;
c=add (m, n);
```

以上三個步驟，全部程式如下：

```
#include <stdio.h>
#include <stdlib.h>
//1.函數原型宣告
int add (int a, int b);
int main()
```

```
{
    int m=6,n=2;
    int c;
    //3.呼叫函式
    c=add(m,n);
    printf("%d\n",c);
    c=add(6,2);
    printf("%d\n",c);
    return 0;
}
//2.實作函式實體
int add (int a, int b){
    int c;
    c=a+b;
    return (c);
}
```

若將函數實體放在 main() 前面，函數原型宣告可省略，程式如下：

```
#include <stdio.h>
#include <stdlib.h>
//函式放在前面，可省函式原型宣告
//1.實作函式實體
int add (int a, int b){
    int c;
    c=a+b;
    return (c);
}
int main()
{
    int m=6,n=2;
    int c;
    //2.呼叫函式
    c=add(m,n);
    printf("%d\n",c);
    c=add(6,2);
    printf("%d\n",c);
    return 0;
}
```

## 自我練習

1. 以上範例，已經完成 add 函式，請自行完成 sub、mul、div、mod 等函式。

### 範例2-1a

試寫一程式，計算 $C_n^m$ 的值。

提示：$C_n^m = \frac{m!}{n!(m-n)!}$

## 程式列印

1. 以下程式未使用函式，階乘共需重複寫三次，若要修改階乘的程式內容，則要到 3 個地方修改，若沒有修改到，則造成程式執行的不一致。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int m=5, n=3, m1, m2, m3, m4, i;
    m1=1;
    for(i=1; i<=m; i++)
        m1=m1*i;
    m2=1;
    for(i=1; i<=n; i++)
        m2=m2*i;
    m3=1;
    for(i=1; i<=m-n; i++)
        m3=m3*i;
    m4=m1/(m2*m3);
    printf("%d",m4);
    return 0;
}
```

2. 以下以函式改寫如下，萬一要修改函式，則是在同一個地方修改。

```
#include <stdio.h>
#include <stdlib.h>
// 1. 實作函式實體
int sum(int m){
    int m1=1;
    int i;
    for(i=1; i<=m; i++)
        m1=m1*i;
    return(m1);
}
int main()
{
    int m=5, n=3, m1, m2, m3, m4;
    m1=sum(m);// 1. 函式呼叫
    m2=sum(n);// 1. 函式呼叫
    m3=sum(m-n);// 1. 函式呼叫
    m4=m1/(m2*m3);
    printf("%d",m4);
    return 0;
}
```

### 自我練習

1. 試寫一程式，使用函式完成 1! + 2! + 3! + 4! + ... + n!（n 可由使用者輸入）
2. 試寫一程式，使用函式完成 1/1!+1/2!+1/3!+1/4!+../n!（n 可由使用者輸入）

## 2-2　參數的傳遞

參數的傳遞有三種情形，一種是傳值（Call by value）、另一種是傳址（Call by address）、第三種是傳參考（Call by reference）。參數的傳遞若是傳值，則是一種單向的傳遞方式，參數僅能由主程式傳至函式本體，函式本體並不能藉由參數，將運算結果傳回。也就是使用傳值傳遞，運算結果至多能用函式名稱傳回一個值，若要傳回兩個或以上的值，就要使用傳址或傳參考。

以傳值方式傳遞參數時，被傳遞的變數值通常只是被複製到其對應的參數，在函式執行過程，主程式中的變數之值並不會受到影響，其優點是保護主程式的值不會隨便被更動。然而，傳址與傳參考參數是一種雙向的參數傳遞，因爲傳址與傳參考都是共用記憶體位址，主程式不僅可以將參數傳至函式本體，函式本體內若參數的內容有所變動，亦可將變動的結果傳回主程式。

**範例2-2a**

示範傳值呼叫。

## 執行結果

1. 本例的實際參數 x1=3，x2=4，將值傳至形式參數 p 和 g，所以印出 p=3，g=4。
2. 函式中 p、g 交換，所以印出 4、3。
3. 函式交換後的值，並不傳回主程式的實際參數，所以 x1、x2 還是 3、4，如下圖。


程式列印 ( 本例使用 C++, 請留意除了輸出改爲 cout，其餘和 C 都相同 )

```
//C++
#include <iostream>
using namespace std;
void swap(int, int);
int main() {
    int x1=3, x2=4;
    swap(x1, x2);
    cout<<"3: "<< x1<<" , "<< x2<<endl;
    return 0;
```

```
}
void swap(int p, int q){
    int t;
    cout<<"1: "<< p<<" , "<< q<<endl;
    t=p; p=q; q=t;
    cout<<"2: "<< p<<" , "<< q<<endl;
}
```

## 範例2-2b

示範傳址呼叫

### 執行結果

參數的運算結果可傳回主程式，所以最後主程式印出的值，已是形式參數交換的結果 4、3。

```
1: 3 , 4
2: 4 , 3
3: 4 , 3
```

### 程式列印

```
#include <iostream>
using namespace std;
void swap(int *p, int *q){
    int t;
    cout<<"1: "<< *p<<" , "<< *q<<endl;
    t=*p; *p=*q; *q=t;
    cout<<"2: "<< *p<<" , "<< *q<<endl;
}
int main() {
    int x1=3, x2=4;
    swap(&x1, &x2);
    cout<<"3: "<< x1<<" , "<< x2<<endl;
    return 0;
}
```

## 程式說明

本例函式呼叫爲：

```
swap (&x1, &x2);
```

函式本體如下：

```
void  swap(int  *p,  int  *q)
```

實際參數與形式參數對照如下 *p=&x1, *q=&x2; 其意爲 *p 是 x1 的別名，所以其值可傳回。

## 範例2-2c

示範傳參考呼叫

## 執行結果

傳參考的執行結果與傳址想同，都是共用位址，其值當然相同，但是傳參考只是參數列加上取址運算子（&），其餘程式的撰寫與傳值相同，這一樣可達到傳址的功能，又不會造成修改一大堆程式碼的困擾。

```
1: 3 , 4
2: 4 , 3
3: 4 , 3
```

## 程式列印

```
#include <iostream>
using namespace std;
void swap(int &, int &);
int main() {
    int x1=3, x2=4;
    swap(x1, x2);
    cout<<"3: "<< x1<<" , "<< x2<<endl;
    return 0;
```

```
}
void swap(int &p, int &q){
    int t;
    cout<<"1: "<< p<<" , "<< q<<endl;
    t=p; p=q; q=t;
    cout<<"2: "<< p<<" , "<< q<<endl;
}
```

## 陣列的傳遞

陣列的名稱是指向陣列的開始位址，所以呼叫函式時，只要將陣列名稱傳給函式即可，且是一種傳址呼叫，陣列的執行結果將可回傳主程式。若爲確保陣列在函式中不被修改，則可在函式本體與函式原型宣告的陣列型態前加 const。呼叫函式的語法如下：

```
函式名稱（陣列名稱）；
```

以下式子是傳遞 a 陣列至函式本體。

```
int a[5], n=5;
getarray(a, n);//請留意不要中括號
```

函式本體的語法如下：

```
傳回值型態　函式名稱（陣列型態　陣列名稱[ ]）{
    敘述區塊；
}
```

以下敘述是函式本體的實例。

```
void getarray (int a[], int n){
}
```

函式原型的宣告同函式本體的實作，但同一般變數，可省略陣列名稱如下：

```
傳回值型態　函式名稱（[ 陣列型態]）；
```

以下敘述是傳遞陣列的函式原型宣告實例。

```
void getarray (int [], int);
```

## 範例2-2d

請完成以下陣列處理函式。

(1) 列印陣列。

(2) 內部排序。

### 輸出結果

```
Before sort:
   99   88   77   66   55
After sort:
   55   66   77   88   99
```

### 程式列印

```
#include <stdio.h>
#include <stdlib.h>
void print(const int a[],int n){//加const則爲傳值
    int i;
    for (i=1;i<=n;i++){
        printf("%5d",a[i]);
    }
    printf("\n");
}
void swap(int &p, int &q){//傳參考，p,q值才能傳回
  int t;
  t=p; p=q; q=t;
}
void sortarray(int a[],int N){
    int i,j;
    for(i=1; i<=N-1; i++)
        for(j=1; j<=N-i; j++)
            if(a[j]>a[j+1])
            swap(a[j], a[j+1]);
}
```

```
int main()
{
    int a[]={0, 99, 88, 77, 66, 55};
    int i, j; const int N=5;
    printf("Before sort:\n");
    print(a,N);
    //Sort
    sortarray(a,N);
    //output
    printf("After sort:\n");
    print (a,N);
}
```

## 補充說明

1. 陣列是傳『參考』，效果同『傳址』，這樣資料是雙向傳遞，也就是函式中若有改變陣列內容，其值均可傳回，如本例的 sortarray 函式；若要改爲傳值，也就是函式中陣列的運算結果不傳回，則應加上 const，如本例的 print 函式。
2. 變數則預設『傳值』，也就是資料的傳遞是單向，資料於函式中若有改變，並不會傳回。但若一定要讓資料傳回，請於變數前加上取址符號『&』，例如，本例的：

   ```
   void swap(int &p, int &q)
   ```

   請嘗試將取址符號去掉，如以下敘述，並再執行一次，觀察結果。

   ```
   void swap(int p, int q)
   ```

## 自我練習

1. 請以傳址重做以上範例。
2. 請完成以下陣列處理函式。
   (1) 列印陣列。(2) 求極大值。(3) 將全部元素乘以 2。

## 二維陣列的傳遞

二維陣列的傳遞原則上與一維陣列相近，唯在函式原型宣告與函式本體的實作僅可省略最高維度的大小。例如，以下敘述爲傳遞一個二維陣列的實例，此函式名稱爲 getarray，陣列第二維的大小可省略。其次，c 則是一個常數，代表陣列的第一維大小。

```
void  getarray (int [ ][c], int);
```

呼叫函式的敘述如下：

```
getarray (a, stdnum);
```

函式本體實作如下：

```
void  getarray (int a[ ][c],  int)
{
    敘述區塊；
}
```

### 範例10-2f

示範二維陣列的傳遞

### 執行結果

```
 NO   CHI   ENG  MATH
  1    41    85    72
  2    38    80    69
  3    65    68    96
  4    22    49    67
  5    51    61    63
```

### 程式列印

```
#include <iostream>
#include <stdlib.h>
#define s 5  //stdnum
#define c 4  //column
```

```
using namespace std;
int main(int argc, char** argv) {
    void getarray(int [][c+1], int);
    void printarray(int [][c+1], int);
    char *title[]={" ","    NO", "   CHI", "   ENG", "  MATH"};
    int a[s+1][c+1];
    for(int i=1; i<=c; i++){
        cout<< title[i];
    }
    cout<<endl;
    getarray(a, s);
    printarray(a, s);
    return(0);
}
void getarray(int a[][c+1], int n){
    for(int i=1; i<=n; i++) {
        a[i][1]=i;
        for(int j=2; j<=c; j++)
            a[i][j]=rand()%101;
    }
}
void printarray(int a[][c+1],int n){
    for(int i=1;i<=n;i++){
        for (int j=1;j<=c;j++){
            cout.width(6);
            cout<<a[i][j];
        }
        cout<<endl;
    }
}
```

## 程式說明

1. 本例故意沒有用 srand()，所以每次執行，每次亂數都相同。
2. C 語言強調執行速度，所以有『指標』型態，以下是字串的指標用法：

```
char *title[]={" ","    NO", "   CHI", "   ENG", "  MATH"};
```

以上用法同下：

```
char title[][8]={" ","    NO", "   CHI", "   ENG", "  MATH"};
```

### 遞迴

函式可以自己呼叫自己，此稱爲遞迴，例如，以下程式可用遞迴計算階層，遞迴在第四章分而治之，才會詳細介紹。

```
#include <stdio.h>
#include <stdlib.h>
int f(int x){
    if(x>=1)
        return(x*f(x-1));
    else
        return(1);
}
int main()
{
    int f(int x);
    int x=6, d;
    d=f(x);
    printf("%d",d);
    return 0;
}
```

main() 也可以自己呼叫自己，請鍵入以下程式，並觀察執行結果。

```
#include <stdio.h>
#include <stdlib.h>
int a=5;
int main()
{
    a=a-1;
    printf("a=%d\n",a);
    if (a>=1)
        main();
    return 0;
}
```