C++程式設計

單元：函數及巨集

函數(一)

- 是一段程式的集合，並給予此段程式一個名稱來代替。
- 通常具有某些特定之功能，可供主程式重覆呼叫使用。
- 依設計者的規劃，可具有參數傳入或結果回傳的功能。
- 函數的使用可增加程式的可讀性、精減程式碼及便於維護的特性。

函數(二)

- 在C++中，函數可分為系統本身提供的標準函數及使用者自行定義函數兩種。
- 使用標準函數時，只要將該函數所屬的標頭檔於程式開頭include進來即可。

自訂函數宣告(一)

- 宣告語法格式

```
回傳值資料型態　函數名稱（參數1,參數2,…){
        程式區塊;
        return 回傳值;
}
```

- **回傳資料型態：**

  表示函數執行完後要回傳給呼叫端的值，可為各種資料型態，若不需回傳則可填 void 。

自訂函數宣告(二)

- **函數名稱：**

  由設計者依函數特性命名，命名規則如同變數命名一般。

- **參數：**

  參數是呼叫函數時，所需傳遞的值，數量可以是0也可以是多個，若不需要代入參數給函數時，直接空白即可。

- **回傳值：**

  函數執行完後，要傳回給呼叫端的結果，若無回傳值，此行可省略。

函式參數及傳回值的組合

傳入參數	傳回參數	宣告型式
不要	不要	void Area ( )
要	不要	void Area (int x)
不要	要	int Area ( )
要	要	int Area (int x)

函式參數的傳遞方式

函數參數的傳遞，可以依據傳遞和接收的方式分成下列三種：

- **傳值呼叫：**

  是指主程式在呼叫函數時，系統會把要傳遞的參數值複製一份給函數，因此，在函數執行過程，並不會影響到原來的變數內容。

- **傳址呼叫、傳參考呼叫：**

  是指主程式在呼叫函數時，系統會把要傳遞的**參數記憶體位址**，直接指定給對應的參數，所以，在執行過程會影響到原變數的內容。

三種函數參數傳呼組合範例

傳遞方式	函數定義	主程式呼叫方式
傳值	int gcd(int a,int b)	gcd(a , b)
傳址	void swap(int *a,int *b) void sort(int a[],int k) void sort(int *a,int k) void sort(int a[][行數]) void sort(int *a)	swap(&a , &b) sort(a , b) sort(a , b) sort(a[][行數]) sort(&a[0][0])
傳參考	int swap(int &a,int &b)	add(a , b)

傳值呼叫之變數觀察(一)

```
#include <iostream>
using namespace std;
void test(int ,int);
main(){
	int a=10,b=5;
	cout<<"呼叫函數前在主程式中變數內容-----\n";
	cout<<"a的位址為:"<<&a<<"、a的內容為: "<<a<<endl;
	cout<<"b的位址為:"<<&b<<"、b的內容為: "<<b<<endl<<endl;
	test(a,b);
	cout<<"呼叫函數後在主程式中變數內容-----\n";
	cout<<"a的位址為:"<<&a<<"、a的內容為: "<<a<<endl;
	cout<<"b的位址為:"<<&b<<"、b的內容為: "<<b<<endl<<endl;
	return 0;
}
```

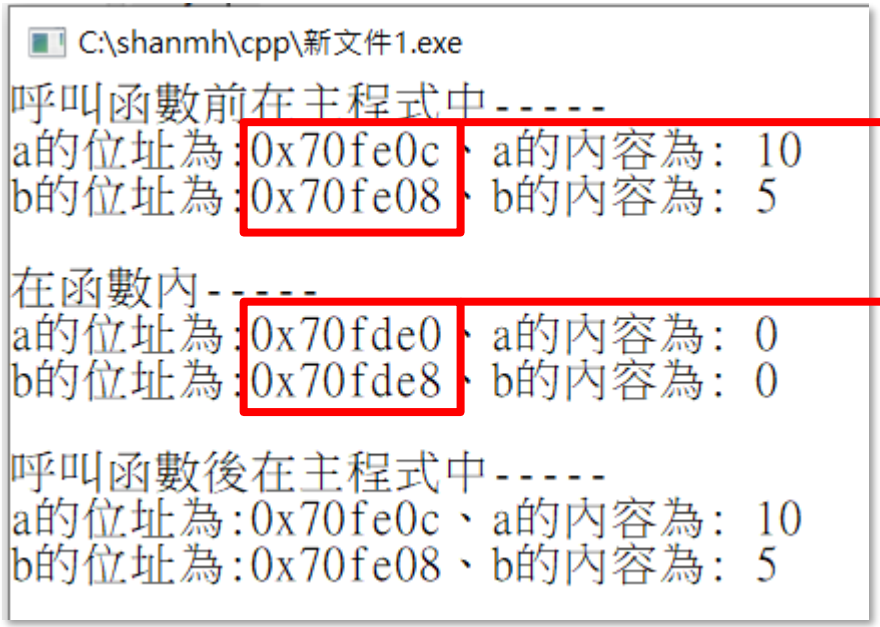
函數之形式宣告及主程式

傳值呼叫之變數觀察(二)

```
void test(int a,int b){
        a=0;
        b=0;
        cout<<"在函數內之變數內容-----\n";
        cout<<"a的位址為:"<<&a<<"、a的內容為: "<<a<<endl;
        cout<<"b的位址為:"<<&b<<"、b的內容為: "<<b<<endl<<endl;
}
```

函數內容

傳值呼叫之變數觀察(三)



由此可觀察到函數內外的變數雖然都是 a 及 b ，但所配置的位址不同，因此，雖然在函數中有改變a、b之值，但返回主程式之後，在主程式中的變數內含值仍維持不變

傳值呼叫範例-最簡化分數(一)

化簡分數時，可先求得分子和分母兩數的最大公因數後，再和分子分母相除。

```
#include <iostream>
using namespace std;
Int  gcd(int ,int);   //先做求最大公因數的函數之形式宣告
main(){
        int a,b,c;
        do{
                cout<<"請依序輸入分數的分子及分母（輸入0 0後結束程式)：";
                cin>>a>>b;
                if(a==0 && b==0) break;  //若輸入為0 0時，跳出迴圈
                c=gcd(a,b);                    //呼叫gcd函數，以求得兩數之公因數
                cout<<"原分數為："<<a<<"/"<<b<<"、約分後為："<<a/c<<"/"<<b/c<<endl;
        } while(1);
        return 0;
}
```

傳值呼叫範例-最簡化分數(二)

```
//以輾轉相除法求最大公因數
int gcd(int a,int b){    //傳值呼叫，不影響原程式之a、b內容
        int t;

        while(a%b!=0){
                t=b;
                b=a % b;
                a=t;
        }

        return b;  //將結果傳田回
}
```

傳址及傳參考呼叫之變數觀察(一)

```
#include <iostream>
using namespace std;
void test(int * ,int &);
main(){
	int a=10,b=5;
	cout<<"呼叫函數前在主程式中變數內容-----\n";
	cout<<"a的位址為:"<<&a<<"、a的內容為: "<<a<<endl;
	cout<<"b的位址為:"<<&b<<"、b的內容為: "<<b<<endl<<endl;
	test(&a,b);
	cout<<"呼叫函數後在主程式中變數內容-----\n";
	cout<<"a的位址為:"<<&a<<"、a的內容為: "<<a<<endl;
	cout<<"b的位址為:"<<&b<<"、b的內容為: "<<b<<endl<<endl;
	return 0;
}
```

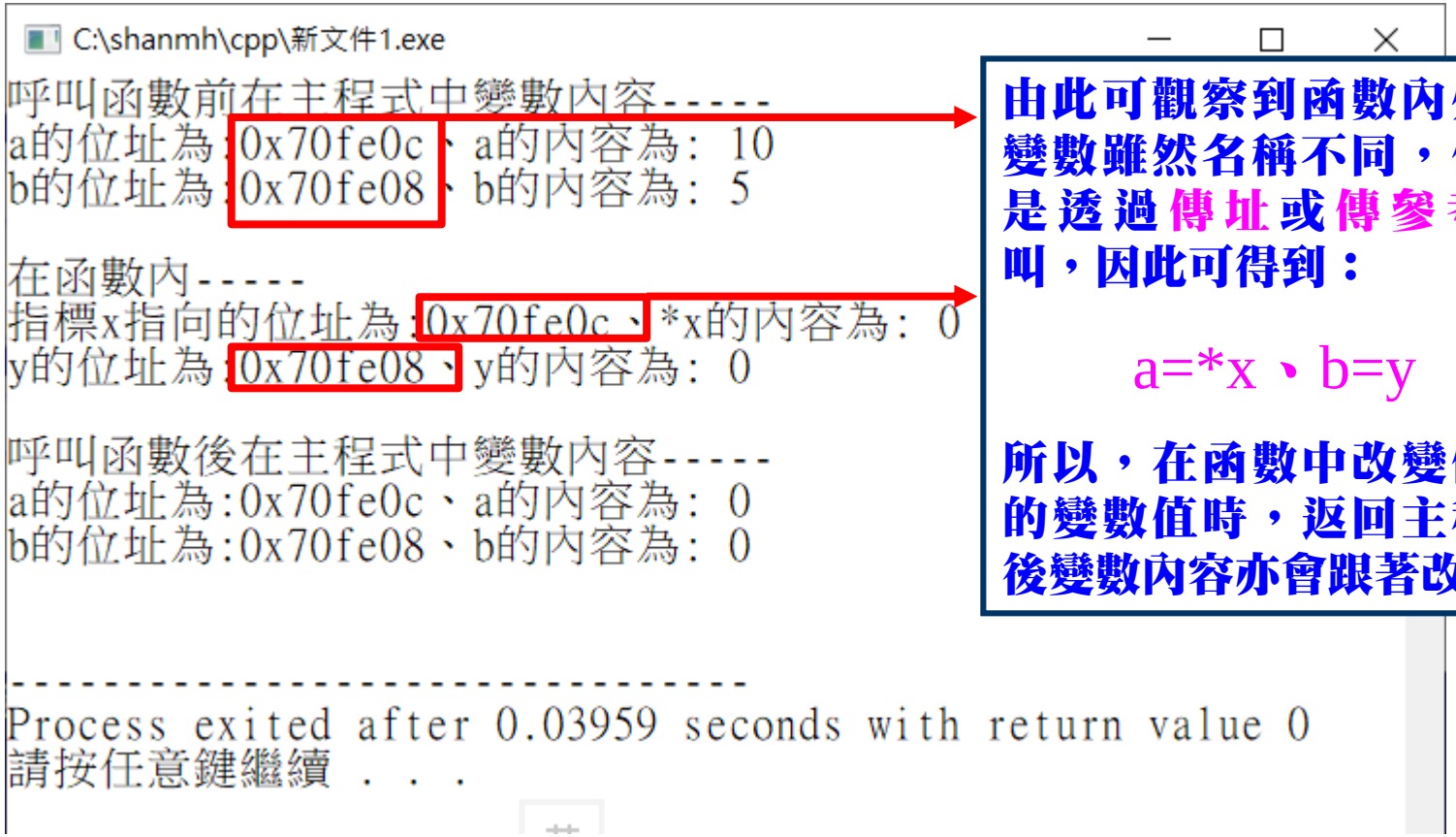
函數之形式宣告及主程式

傳址及傳參考呼叫之變數觀察(二)

```
void test(int *x,int &y){
        *x=0;
        y=0;
        cout<<"在函數內-----\n";
        cout<<"指標x指向的位址為:"<<x<<"、*x的內容為: "<<*x<<endl;
        cout<<"y的位址為:"<<&y<<"、y的內容為: "<<y<<endl<<endl;
}
```

函數內容

傳址及傳參考呼叫之變數觀察(三)



由此可觀察到函數內外的變數雖然名稱不同，但因是透過傳址或傳參考呼叫，因此可得到：

a=*x、b=y

所以，在函數中改變傳遞的變數值時，返回主程式後變數內容亦會跟著改變

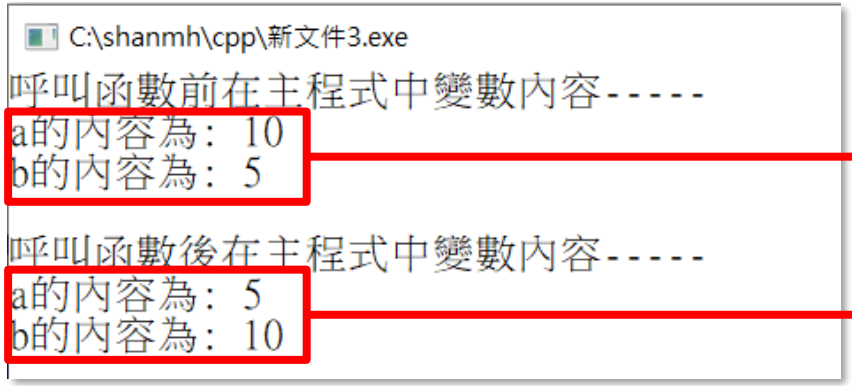
傳址呼叫範例-兩變數交換(一)

```
#include <iostream>
using namespace std;
void swap(int * ,int *);
main(){
	int a=10,b=5;
	cout<<"呼叫函數前在主程式中變數內容-----\n";
	cout<<"a的內容為: "<<a<<endl;
	cout<<"b的內容為: "<<b<<endl<<endl;
	swap(&a,&b);
	cout<<"呼叫函數後在主程式中變數內容-----\n";
	cout<<"a的內容為: "<<a<<endl;
	cout<<"b的內容為: "<<b<<endl;
	return 0;
}
```

傳址及傳參考呼叫之變數觀察(二)

```
void swap(int *x,int *y){
        int t;
        t=*x;
        *x=*y;
        *y=t;
}
```

函數內容



呼叫函式後，已完成兩變數內容交換

陣列傳遞範例-排序(一)

```
#include <iostream>
using namespace std;
void sort(int * ,int);  //第一個參數為陣列位址，第二個為陣列最大索引
void swap(int *,int *);
main(){
        int k,i;
        cout<<"請輸入欲排序之數字個數:";cin>>k;
        int a[k];
        cout<<"請輸入 "<< k <<" 個數字:" ;
        for(i=0;i<k;i++)  cin>>a[i];

        sort(a,k-1);   //以傳址的方式將陣列位址傳給函數
        for(i=0;i<k;i++)  cout<<a[i]<<" ";

        return 0;
}
```

陣列傳遞範例-排序(二)

```
void sort(int *arr,int arr_max){    //氣泡排序法
        int i,j;

        for(i=0;i<arr_max;i++)
                for(j=0;j<arr_max-i;j++)
                        if(*(arr+j)>=*(arr+j+1))
                                swap((arr+j),(arr+j+1));   //接收端為指標，因此將指標內的位址傳出去
}

void swap(int *x,int *y){   //兩數交換
        int t;
        t=*x;
        *x=*y;
        *y=t;
}
```

函數的多載

函數多載為C++新增的功能，可藉此功能來讓同一個函數名稱依其不同的參數形態來區分不同的主體。

```
int Data(int a){
        return a*10;
}

float Data(float a){
        return a*2
}
```

同為Data名稱，依所傳入的函數資料型態來決定要執行那一個主體

```
int area(int len,int width){
        return len*width;
}

float area(int r){
        return r*3.14;
}
```

同為area名稱，依所傳入的函數的數量來決定要執行那一個主體

函數的多載範例

```
#include<iostream>
using namespace std;
int Data(int);
float Data(float);
int area(int ,int );
float area(int r);
main(){
        int a=3,l=5,w=4;
        float b=3.1;

        cout<< Data(a) <<endl;
        cout<< Data(b) <<endl;
        cout<< area(l,w) <<endl;
        cout<< area(a) <<endl;
   return 0;
 }
```

```
int Data(int a){
        return a*10;
}

float Data(float a){
        return a*2;
}

int area(int len,int width){
        return len*width;
}

float area(int r){
        return r*3.14;
}
```

main函式的傳入值

main函式的傳入值指的是接數系統所給予的命令參數；常用的兩個main函數的使用及參數說明如下：

```
main(int argc,char** argv){


}
```

- 第一個參數 argc：

  代表作業系統一共傳入多少參數，而指令本身也是一個，因此，此參數之最小值為1。

- 第二個參數 argv：

  字串陣列型態，存放系統執行時所讀取的參數資料，即執行程式式，在其檔名後面所加之文字。

main函式的傳入值使用範例

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
        int i;
        cout<<"參數個數："<<argc<<endl;
        cout<<"參數列表：\n"

        for (i=0;i<argc;i++)
                cout<<argv[i]<<endl;

        return 0;
}
```

巨集

巨集，和函式有些類似，但巨集是將某些程式段或數值先定義好，再由前置處理器處理，在程式編譯前進行程式碼的置換；因此，編譯階段是看不到任何巨集存在。

由於是程式的置換，因此，程式在編譯後會變大，但執行起來比使用函式來的快；但也因為是進行程式置換，因此，編譯後檔案會變的比較大。

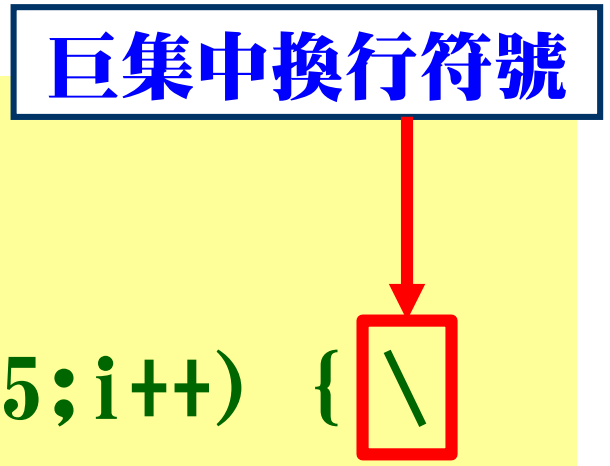
巨集指令(一)

- 語法格式(一)

```
#define 巨集名稱　替換內容
```

- 範例

```
#define PI 3.14159
#define e 2.71828
#define loop5 for(i=0;i<=5;i++) { \
                    cout<<PI<<endl;}
```



巨集指令(二)

- 語法格式(二)

```
#define 巨集名稱(參數列)　替換內容
```

- 範例

```
#define sum(a,b) a+b
#define div(x,y) (x)/(y)
```

以巨集取代常數

```
#include<iostream>
#define PI 3.14159
using namespace std;

main(){
	float area,r;
	cout<<"輸入半徑："; cin>>r;
	area=PI*r*r;
	cout<< "圓面積為："<<area<<endl;

	return 0;
}
```

原程式

```
main(){
	float area,r;
	cout<<"輸入半徑："; cin>>r;
	area=3.14159*r*r;
	cout<< "圓面積為："<<area<<endl;

	return 0;
}
```

編譯程式所看到的主程式碼

以巨集取代字串

```
#include<iostream>
#define H "Hello World\n"
using namespace std;

main(){

        cout<<H;

        return 0;
 }
```

```
main(){



cout<<"Hello World\n“;

                return 0;
 }
```

原程式

編譯程式所看到的主程式碼

以巨集取代程式片段

```
#include<iostream>
#define H "Hello World\n"
#define loop5 for(i=0;i<5;i++) { \
              cout<<H; \
              cout<<endl; \
        }
using namespace std;
main(){
       int i;
       loop5;
       return 0;
 }
```

原程式

```
main(){
       int i;

      for(i=0;i<5;i++) {
             cout<<H;
             cout<<endl;
       }


       return 0;
}
```

編譯程式所看到的主程式碼

以巨集取代函數

```
#include<iostream>
#define sum(a,b)  a+b
using namespace std;

main(){

      int a,b;

      cout<<"輸入兩數：";cin>>a>>b;
      cout<<sum(x,y)<<endl;

      return 0;
}
```

原程式

```
main(){

      int a,b;

      cout<<"輸入兩數：";cin>>a>>b;
      cout<<x+y<<endl;

      return 0;
}
```

編譯程式所看到的主程式碼

以巨集內容包含運算式的問題

```
#include<iostream>
#define Mul(a,b)  a*b
using namespace std;
main(){

      int a,b;

      cout<<“輸入兩數：";cin>>a>>b;
      cout<<Mul (x+y,x+y)<<endl;

      return 0;
}
```

原程式

```
main(){

      int a,b;

      cout<<"輸入兩數：";cin>>a>>b;
      cout<<x+y*x+y<<endl;

      return 0;
}
```

各別代入後會產生運算順序問題

編譯程式所看到的主程式碼

問題更正

定義時直接加括號

```
#include<iostream>
#define Mul(a,b)  (a)*(b)
using namespace std;
main(){

      int a,b;

      cout<<“輸入兩數：";cin>>a>>b;
      cout<<Mul (x+y, x+y)<<endl;

      return 0;
}
```

原程式

```
main(){

      int a,b;

      cout<<"輸入兩數：";cin>>a>>b;
      cout<<(x+y)*(x+y)<<endl;

      return 0;
}
```

編譯程式所看到的主程式碼

巨集的條件編譯指令

巨集的條件編譯指令類似程式中if指令的功能，主要是在程式中判斷前置處理器中是否有定義某個巨集名稱，若該巨集存在，則去執行某段和該巨集有關之程式，否則就略過該程式段。

巨集的條件編譯指令組合

巨集的條件編譯指令有下列三種組合：

```
#ifdef 巨集名稱

    執行敘述

#endif
```

```
#ifdef 巨集名稱
    執行敘述1
#else
    執行敘述2
#endif
```

```
#ifdef 巨集名稱
    執行敘述1
#elif 巨集名稱2
    執行敘述2
#elif 巨集名稱3
    執行敘述3
#endif
```

巨集的條件編譯指令範例

```
#include<iostream>
#define PI 3.14
using namespace std;
main(){

    #ifdef PI
        cout<<"PI="<<PI;
    #else
        float PI;
        cout<<"請輸入PI值：";cin>>PI;
        cout<<"PI="<<PI;
    #endif
  return 0;
}
```

若有定義巨集PI，則執行此段程式

```
#include<iostream>

using namespace std;
main(){

    #ifdef PI
        cout<<"PI="<<PI;
    #else
        float PI;
        cout<<"請輸入PI值：";cin>>PI;
        cout<<"PI="<<PI;
    #endif
  return 0;
}
```

沒有定義巨集PI，則執行此段程式

變數生命週期（一）

變數依宣告的位置會有不同的作用範圍，一般可分為下列幾種情形：

- 全域變數：

  宣告於所有函數之外，在宣告指令以下的所有函數或程式區塊都可使用，且其生命週期會從宣告開始一直到程式結束。

- 一般區域變數：

  宣告於所有函數內，只有在該函數內可以使用，當該函數結束後，此變數亦會消失（即所占用的記憶體會被釋放）；另外，當全域變數和區域變數擁有相同名稱時，在函數內會以區域變數為優先。

變數生命週期（二）

- **靜態區域變數：**

  和一般區域變數相同，但宣告時在資料型態前需加上 static；其作用範圍僅限於函數中，不同的是，靜態區域變數不會因為函數結束而消失，相反的其值會被保留下來，等到下一次函數再次被執行時，前一次的值會被取出使用。

- **區塊變數：**

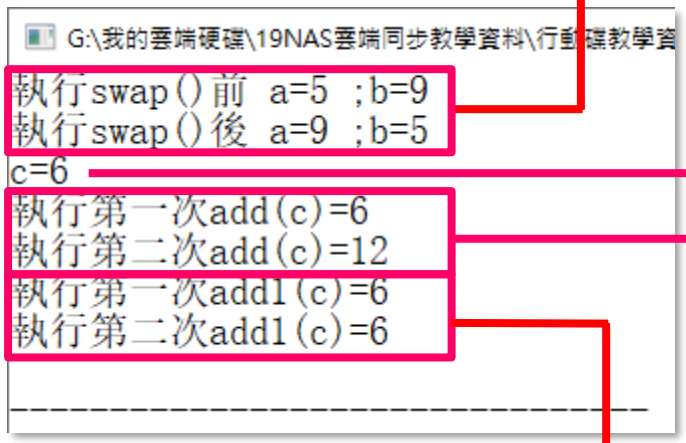
  被宣告在程式區塊中（即在｛｝之間），也是區域變數的一種，只是其作用範圍更小，僅限於區塊中使用，當區塊結束，該變數亦會消失。

各種變數宣告範例

```
#include<iostream>
using namespace std;
void swap();
int add(int);
int add1(int);
int a=5,b=9,c=3;
main(){
   int c=6;
   cout<<"執行swap()前 a="<<a<<" ;b="<<b<<endl;
   swap();
   cout<<"執行swap()後 a="<<a<<" ;b="<<b<<endl;
   cout<<"c="<<c<<endl;
   cout<<"執行第一次add(c)="<<add(c)<<endl;
   cout<<"執行第二次add(c)="<<add(c)<<endl;
   cout<<"執行第一次add1(c)="<<add1(c)<<endl;
   cout<<"執行第二次add1(c)="<<add1(c)<<endl;
   return 0; }
```

```
void swap(){
   int t;
   t=a; a=b; b=t;
}
int add(int n){
   static int sum=0;
   sum+=n;
   return sum;
}
int add1(int n){
   int sum=0;
   sum+=n;
   return sum;
}
```

各種變數宣告範例執行結果



a、b為全域變數，swap()函式中預設執行a、b兩變數交換

全域變數c=3、main函數中之區域變數c=6，故在此執行印出c的內容時，以區域變數為主

add(c)為執行c之累加，在此函數中儲存累加之變數sum為靜態區域變數，故經兩次執行，其值會累計

add(c)為執行c之累加，在此函數中儲存累加之變數sum為一般區域變數，故經兩次執行，其值不會累計