

# 程式設計 (一)

## CH6. 函式 (上)

Ming-Hung Wang 王銘宏

tonymhwang@cs.ccu.edu.tw

Department of Computer Science and Information Engineering  
National Chung Cheng University

Last Semester, 2021

# 本章目錄

1. 程式模組
2. 定義函式
3. 函式原型 (function prototype)
4. 範圍規則
5. 傳值呼叫 (call by value)
6. 標準函式庫的標頭 (header)
7. `<math.h>` 數學函式庫
8. `<stdlib.h>` 公用函式庫 - 產生亂數

# 程式模組

大部分用來解決真實世界問題的電腦程式，都要比我們在前幾章中所介紹的程式大很多。

發展和維護大型程式的最好方法，便是以一些較小的單元或模組 (module) 來建構整個程式。這些小單元要比整個大程式好管理多了。

# 程式模組

- C 語言的模組稱為函式 (function)
- 函式有 2 種：
  1. 位於 C 標準函式庫 (C standard library) 中，事先寫好的函式，例如 `scanf`、`pow...` 等
  2. 自己編寫的新函式

函式經由函式呼叫 (function call) 的方式調用 (invoked) 呼叫時需指明欲調用之函式的名稱，並提供所需的引數 (argument) 給受呼叫函式，以執行其工作。

例如我們常使用的 printf:

```
printf("hello world\n");
```

函式名    引數(argument)

# 定義函式

# 定義函式

## 定義函式的格式：

```
return_value_type function_name(parameter_list) {  
    definitions  
    statements  
}
```

每個函式在使用前，都要先將其定義。  
我們平常所撰寫的主程式 `int main(){...}`  
其實就是在定義一個名稱為 `main` 的函式。



## 回傳值

任何函式在執行結束時，都回傳一個值 (比如 `printf` 運行結束時會回傳印出的字元數，`pow` 會回傳計算次方後的結果)。而那個回傳值的型態要跟定義函式時的 `return_value_type` 相同。

我們會使用 `return` 敘述來回傳數值並結束函式。

```
int maximum(int a, int b) {  
    if (a > b){  
        return a;  
    }  
    return b;  
}
```

## void 回傳型態

return\_value\_type 為 void 的話，表示此函示沒有傳  
回值，其 return 敘述不用寫上數值。若要在函式尾  
端回傳，則可以省略 return 敘述。

```
void printMaximum(int a, int b) {  
    if (a > b){  
        printf("%d", a);  
        return;  
    }  
    printf("%d", b);  
}
```

## main 函式的回傳值

main 有個 `int` 回傳型態，main 的回傳值式用來指出程式是否正確地執行。

在 main 的結尾回傳 0 代表程式執行成功。  
在 C 語言標準裡，若省略了此一敘述，則會預設回傳 0。

# 定義函式

## 呼叫自行定義的函式

被呼叫的函式需被定義在呼叫者的前面

```
1 // Function
2 #include <stdio.h>
3
4 int maximum(int a, int b){
5     if (a > b)
6         return a;
7     return b;
8 }
9
10 int main(){
11     int val1 = 5, val2 = 7;
12     printf("%d and %d, %d is bigger\n",
13           val1, val2, maximum(val1, val2));
14 }
15
```

D:\codeblocks\Function.exe  
5 and 7, 7 is bigger

# 函式原型 (function prototype)

## 函式原型 (function prototype)

有時編寫 C 語言時，所編寫的函式篇幅較長，若全部寫在 main 函式的前面，可能會導致難以閱讀。這時，我們可以先定義函式原型，之後在定義函式內容。

# 函式原型 (function prototype)

函式原型的定義方式為函式的標頭再加上分號：

```
void printMaximum(int a, int b);
```

也可以連參數名稱都省略

```
void printMaximum(int, int);
```

注意函式原型的回傳型態與參數型態與個數  
都要與函式定義時相同。

# 函式原型 (function prototype)

先在呼叫者 (main) 之前定義函式原型，即可將函式寫在呼叫者 (main) 後面

```
1 // Function
2 #include <stdio.h>
3
4 int maximum(int, int); //function prototype
5
6 int main(){
7     int val1 = 5, val2 = 7;
8     printf("%d and %d, %d is bigger\n",
9           val1, val2, maximum(val1, val2));
10 }
11
12 int maximum(int a, int b){
13     if (a > b)
14         return a;
15     return b;
16 }
```



# 函式原型 (function prototype)


若有兩函式要互相呼叫，則必須使用函式原型  
下方是一個簡易的對戰程式

```
1 // Function
2 #include <stdio.h>
3
4 const int aATK = 9; //global variable
5 const int bATK = 11;
6 int attackAtoB(int aHP, int bHP);
7 int attackBtoA(int aHP, int bHP);
8
9 int attackAtoB(int aHP, int bHP){
10     bHP -= aATK;
11     printf("a -> b, aHP = %2d, bHP = %2d\n",
12           aHP, bHP);
13     if (bHP > 0)
14         return attackBtoA(aHP, bHP);
15     return 1; //A win
16 }
17
```

```
18 int attackBtoA(int aHP, int bHP){
19     aHP -= bATK;
20     printf("b -> a, aHP = %2d, bHP = %2d\n",
21           aHP, bHP);
22     if (aHP > 0)
23         return attackAtoB(aHP, bHP);
24     return 2; //B win
25 }
26
27 int main(){
28     int aHP = 60, bHP = 50, result;
29     printf("SET: aHP = %2d, bHP = %2d\n",
30           aHP, bHP);
31     result = attackAtoB(aHP, bHP);
32     if (result == 1)
33         printf("A WIN!\n");
34     else if (result == 2)
35         printf("B WIN!\n");
36 }
```

# 函式原型 (function prototype)

下圖是上頁程式碼的運行結果

 D:\codeblocks\Function.exe

```
SET: aHP = 60, bHP = 50
a -> b, aHP = 60, bHP = 41
b -> a, aHP = 49, bHP = 41
a -> b, aHP = 49, bHP = 32
b -> a, aHP = 38, bHP = 32
a -> b, aHP = 38, bHP = 23
b -> a, aHP = 27, bHP = 23
a -> b, aHP = 27, bHP = 14
b -> a, aHP = 16, bHP = 14
a -> b, aHP = 16, bHP = 5
b -> a, aHP = 5, bHP = 5
a -> b, aHP = 5, bHP = -4
A WIN!
```

# 範圍規則

# 範圍規則

在先前所學的判斷與迴圈中，應該有發現在大括號內部所宣告的變數，到大括號外面會變得不可使用，這就是一種範圍規則。

# 範圍規則

範圍規則包含 4 種識別字的範圍 (scope of an identifier):

- 函式範圍 (function scope)
- 檔案範圍 (file scope)
- 區塊範圍 (block scope)
- 函式原型範圍 (function-prototype scope)

※ 識別字 (identifier):  
程式語言中自行定義的名稱,  
例如變數名稱、函數名稱... 等

# 範圍規則

- 函式範圍：**標籤** (在識別字後加上冒號，如 `start:`，用於 `goto` 敘述式) 是唯一具有函式範圍的識別字，標籤可在函式的任何位置使用，不過出了這個函式的本體，便不能參用這些標籤。
- 檔案範圍：宣告在任何函式之外的識別字都具有檔案範圍，從這種識別字宣告的位置開始，一直到整個檔案結束，所有的函式中都會知道它的存在。**全域變數**、**函式定義**，和放在函式之外的**函式原型**都具有檔案範圍。

# 範圍規則

- 區塊範圍：宣告在區塊之內的識別字都具有區塊範圍。區塊範圍終止的位置在此區塊的結束右大括號 (})。  
宣告在大括號內 (如函式、if 敘述式、迴圈敘述式內) 的變數、函式的參數、for 迴圈標頭宣告的變數，都是具有區塊範圍的**區域變數**。
- 函式原型範圍：在函式原型參數列中的識別字。前面提到，函式原型的參數列不需要參數名稱，若填寫了名稱，編譯器也會將其忽略。

## 全域變數與區域變數

在範圍規則中提到，全域變數屬於檔案範圍的變數，自從宣告後，在整個程式檔內都能夠使用；  
區域變數屬於區塊範圍的變數，只能在宣告所在的區塊 (block) 內使用。



## 練習

請判斷下列程式碼，哪幾行不符合範圍規則？

```
1 //scope of variables
2 #include <stdio.h>
3
4 int a = 0;
5 int funcA(int);
6
7 int main() {
8     int b = 1;
9     for (int i = 0; i < 10; i++) {
10         int c = b + i;
11         printf("%d\n", funcA(i));
12     }
13     {
14         int d = a + b;
15     }
```

```
16     printf("%d\n", i);
17     printf("%d\n", a);
18     printf("%d\n", b);
19     printf("%d\n", c);
20     printf("%d\n", d);
21 }
22
23 int funcA(int p) {
24     if (a == 0) {
25         a += c;
26     }
27     return a + p;
28 }
```

傳值呼叫 (call by value)

# 傳值呼叫 (call by value)

大多數的程式語言用來調用函式的方式分為兩種：

- 傳值呼叫 (call by value)
- 傳參考呼叫 (call by reference)

當以**傳值呼叫**來傳遞引數時，此引數值的一份複製將會傳給受呼叫的函式 (代入函式參數)。對此複製所做的修改並不會影響到呼叫者原來變數的值。

# 傳值呼叫 (call by value)

例如以下程式碼，add 函式中的參數值的更動，並不會影響 main 函式的變數值 (儘管變數名稱相同)

```
1 // call by value
2 #include <stdio.h>
3
4 int add(int a, int b){
5     a += b;
6     return a;
7 }
8
9 int main(){
10     int a = 5, r;
11     r = add(a, 10);
12     printf("a = %d\nr = %d\n", a, r);
13 }
14
```

"D:\codeblocks\call by value.exe"

a = 5  
r = 15

# 標準函式庫的標頭 (header)

# 標準函式庫的標頭 (header)

每個標準函式庫都有一個相對應的標頭 (header)，它含有此函式庫中所有函式的函式原型，以及這些函式所需之各種資料型別和常數的定義。

若要使用標準函式庫所定義的函式，需用前置處理器命令 `#include` 含括進該標準函式庫的標頭。

# 標準函式庫的標頭 (header)

## 標準函式庫標頭檔 (1/2)

<code>&lt;assert.h&gt;</code>	內含一些用來幫助程式偵錯的巨集和資訊。
<code>&lt;ctype.h&gt;</code>	內含一些檢測字元特性及大小寫字元轉換等函式的原型。
<code>&lt;errno.h&gt;</code>	定義了一些用來回報錯誤狀況的巨集。
<code>&lt;float.h&gt;</code>	內含此系統中對浮點數大小的限制。
<code>&lt;limits.h&gt;</code>	內含此系統中對整數大小的限制。
<code>&lt;locale.h&gt;</code>	內含一些能夠使程式區域化的函式原型與資訊。區域化的表示方式讓電腦系統能處理世界各地各種不同的資料 ( 如日期, 時間, 金額及大的數目 ) 表示習慣。
<code>&lt;math.h&gt;</code>	內含數學函式庫的函式原型。

# 標準函式庫的標頭 (header)

## 標準函式庫標頭檔 (2/2)

<code>&lt;setjmp.h&gt;</code>	內含改變正常函式呼叫與回傳順序的函式原型。
<code>&lt;signal.h&gt;</code>	內含處理程式執行中各種狀況的函式原型和巨集。
<code>&lt;stdarg.h&gt;</code>	定義一些處理不確定型別及個數之引數列的函式。
<code>&lt;stddef.h&gt;</code>	內含一些在 C 執行運算時所常用到的型別。
<code>&lt;stdio.h&gt;</code>	內含標準輸出 / 入函式庫的函式原型以及所需的資訊。
<code>&lt;stdlib.h&gt;</code>	內含一些數字與文字間轉換，記憶體配置，亂數，及其他公用函式的原型。
<code>&lt;string.h&gt;</code>	內含字串處理函式的原型。
<code>&lt;time.h&gt;</code>	內含處理時間與日期的函式原型。



## <math.h> 數學函式庫

數學函式庫讓我們能夠方便的執行某些常用的數學運算，例如平方根、次方、三角函數等。

## 常用的數學函式庫函式 (1/2)

<code>sqrt( x )</code>	$x$ 的平方根	<code>sqrt( 900.0 )</code> is 30.0 <code>sqrt( 9.0 )</code> is 3.0
<code>exp( x )</code>	指數函式 $e^x$	<code>exp( 1.0 )</code> is 2.718282 <code>exp( 2.0 )</code> is 7.389056
<code>log( x )</code>	$x$ 的自然對數 (底為 $e$ )	<code>log( 2.718282 )</code> is 1.0 <code>log( 7.389056 )</code> is 2.0
<code>log10( x )</code>	$x$ 的對數 (底為 10)	<code>log10( 1.0 )</code> is 0.0 <code>log10( 10.0 )</code> is 1.0 <code>log10( 100.0 )</code> is 2.0
<code>fabs( x )</code>	$x$ 的絕對值	<code>fabs( 13.5 )</code> is 13.5 <code>fabs( 0.0 )</code> is 0.0 <code>fabs( -13.5 )</code> is 13.5

(表中的函式參數與回傳值皆為 double)

## 常用的數學函式庫函式 (2/2)

<code>ceil( x )</code>	不小於 $x$ 的最小整數	<code>ceil( 9.2 )</code> is 10.0 <code>ceil( -9.8 )</code> is -9.0
<code>floor( x )</code>	不大於 $x$ 的最大整數	<code>floor( 9.2 )</code> is 9.0 <code>floor( -9.8 )</code> is -10.0
<code>pow( x, y )</code>	$x$ 的 $y$ 次方 ( $x^y$ )	<code>pow( 2, 7 )</code> is 128.0 <code>pow( 9, .5 )</code> is 3.0
<code>fmod( x, y )</code>	$x/y$ 的浮點餘數	<code>fmod( 13.657, 2.333 )</code> is 1.992
<code>sin( x )</code>	$x$ 的正弦值 ( $x$ 的單位為弧度)	<code>sin( 0.0 )</code> is 0.0
<code>cos( x )</code>	$x$ 的餘弦值 ( $x$ 的單位為弧度)	<code>cos( 0.0 )</code> is 1.0
<code>tan( x )</code>	$x$ 的正切值 ( $x$ 的單位為弧度)	<code>tan( 0.0 )</code> is 0.0

(表中的函式參數與回傳值皆為 double)

# <stdlib.h> 公用函式庫 - 產生亂數

(虛擬亂數, pseudo-random number)

## <stdlib.h> 公用函式庫 - 產生亂數

我們可以用 C 標準函式庫中，標頭檔 <stdlib.h> 中的 rand 函式來模擬機會的運行

```
int rand (void);
```

rand 函式會產生一個介於 0 和 RAND\_MAX(值為 32767，定義在 <stdlib.h> 標頭檔中) 之間的整數。

## <stdlib.h> 公用函式庫 - 產生亂數

實際上，rand 函式所產生的是虛擬亂數 (pseudo-random numbers)。重複呼叫 rand 所產生的數列，看起來也是隨機的。只不過每次執行時，都會出現相同的數列。

不過，我們可以使用 srand 函式，藉由每次輸入不同的 seed 來使 rand 函式的結果隨機化 (randomizing)。

```
void srand (unsigned int seed);
```

# <stdlib.h> 公用函式庫 - 產生亂數

srand 會依照傳入的 seed 初始化亂數表

```
1 // random
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main(){
6     int seed;
7     scanf("%d", &seed);
8     srand(seed);
9     for (int i = 0; i < 10; i++){
10         printf("%d ", rand());
11     }
12     printf("\n");
13 }
```

D:\codeblocks\random.exe

```
10
71 16899 3272 13694 13697 18296 6722 3012 11726 1899
```

D:\codeblocks\random.exe

```
20
103 26079 18073 24951 18538 24795 5078 6508 13002 5955
```

D:\codeblocks\random.exe

```
50
201 20851 29710 25954 296 11525 148 16994 16830 18121
```

## <stdlib.h> 公用函式庫 - 產生亂數

我們可以藉由 <time.h> 所定義的 time 函式取得執行時的時間 (秒數) 當作 seed 來初始化亂數表。

```
time_t time (time_t* timer);
```

time\_t 是 <time.h> 所定義的整數型態，該整數值表示自 UTC 1970 年 1 月 1 日 00:00 起經過的秒數。

而 time 函式的引數可使用 NULL 或 0 即可。



## <stdlib.h> 公用函式庫 - 產生亂數

使用 time 函式取得時間作為 seed

```
1 // random
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int main(){
7     srand(time(NULL));
8     for (int i = 0; i < 10; i++)
9         printf("%d ", rand());
10    printf("\n");
11 }
```

D:\codeblocks\random.exe

22381 22813 18681 30480 7800 21283 13484 20710 15210 31031

### 比例化和位移

- 直接由 rand 所產生的值一定落在  
 $0 \leq \text{rand}() \leq \text{RAND\_MAX} (= 32767)$
- 我們可以藉由調整比例與平移，將數值調整到想要的範圍

### 比例化和移位公式

```
n = a + rand() % b;
```

其中 a 代表**位移值** (shifting value)，b 代表比例因子

# <stdlib.h> 公用函式庫 - 產生亂數

## 模擬骰 10 次 6 面骰子程式

```
1 // random
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int main(){
7     srand(time(NULL));
8     for (int i = 0; i < 10; i++){
9         int r = 1 + rand() % 6;
10        printf("%d ", r);
11    }
12
13    printf("\n");
14 }
15
```

D:\codeblocks\random.exe

4 6 2 6 3 1 6 1 4 1

參考資料： Deitel, H. M., & Deitel, P. J. (2015). C: How to program.  
Upper Saddle River, N.J: Prentice Hall.