

程式設計

Ch05. Function

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

Spring Semester, 2024

Outline

- 1 模組化程式設計 (Modular Programming)
- 2 函式定義與函式原型 (Function Definition and Function Prototype)
- 3 傳值呼叫 (Call by Value)
- 4 識別字的範圍規則 (Scope Rules of Identifier)
- 5 變數的儲存類別 (Storage Classes of Variable)
- 6 函式呼叫堆疊 (Function Call Stack)
- 7 遞迴函式 (Recursive Function)
- 8 標準函式庫的標頭 (Headers of Standard Library)
- 9 產生亂數 (Generate Random Number)

模組化程式設計 Modular Programming

模組化程式設計 (Modular Programming)

- 在撰寫大型程式時，程式通常由許多小功能組合而成，有些功能還有可能會被重複使用。
- 如何開發與維護這些功能，最好的方式就是將其模組化。
- 以模組 (module) 為單位開發程式會便於編寫主程式的流程控制，也便於編修那些重複使用的功能。

模組化程式設計 (Modular Programming)

- 在程式語言中，我們稱除了主程式之外的模組為副程式 (subprogram)，須經由呼叫 (call) 的方式來調用 (invoked)。而在 C 語言中，模組被稱之為函式 (function)。
- 函式可簡單分為：
 - main 函式：作為程式的進入點 (entry point)
 - C 標準函式庫 (C standard library) 提供的函式，如 scanf、pow...
 - 自己定義的新函式

函式定義與函式原型 (Function Definition and Function Prototype)

函式定義與函式原型

Function Definition and Function Prototype

函式定義與函式原型 (Function Definition and Function Prototype)

定義函式

- 每個函式在使用前，都要先將其定義。定義函式語法如下：

```
1 RETURN_TYPE functionName(PARAMETER_LIST)
2 {
3     STATEMENTS;
4     return RETURN_VALUE;
5 }
```

- 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。

函式定義與函式原型 (Function Definition and Function Prototype)

- 呼叫時需指明欲調用之函式的名稱，並提供與參數列對應的引數。
- 以下為計算 a^n 函式實作範例：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int power(int a, unsigned int n)
5  {
6      int result = 1;
7      while (n--)
8          result *= a;
9      return result;
10 }
11
12 int main()
13 {
14     printf("5^3 = %d\n", power(5, 3));
15 }
```

參數列(parameter list)

引數列(argument list)

C:\Projects\ch04_code\power x + v

5^3 = 125

函式定義與函式原型 (Function Definition and Function Prototype)

回傳 (Return)

- 任何函式在執行結束時，會回傳一個值 (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)。
- 回傳值的型態要跟定義函式時的型態相同。使用 `return` 敘述式來回傳數值並結束函式。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int maximum(int a, int b)
5  {
6      if (a > b) return a;
7      return b;
8  }
9
10 int main()
11 {
12     printf("max(5, 3) = %d\n", maximum(5, 3));
13 }
```

C:\Projects\ch04_code\maximr x + v

max(5, 3) = 5

函式定義與函式原型 (Function Definition and Function Prototype)

- 若回傳型態為 **void**，表示此函式沒有傳回值。若要在函式尾端回傳，則可以省略 `return` 敘述。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void printMaximum(int a, int b)
5  {
6      if (a > b)
7      {
8          printf("%d\n", a);
9          return;
10     }
11     printf("%d\n", b);
12 }
13
14 int main()
15 {
16     printf("max(5, 3) = ");
17     printMaximum(5, 3);
18     printf("max(2, 7) = ");
19     printMaximum(2, 7);
20 }
```

```
C:\Projects\ch04_code\printW  x  +  v
max(5, 3) = 5
max(2, 7) = 7
```

函式定義與函式原型 (Function Definition and Function Prototype)

main 函式的回傳值

- main 函式的回傳型態為 int，用來指出程式是否正確執行。
- 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。

函式定義與函式原型 (Function Definition and Function Prototype)

函式原型 (function prototype)

- 有時編寫 C 語言，副程式篇幅較長時，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，之後再定義函式內容。

```
1 RETURN_TYPE functionName(PARAMETER_LIST);
```

- 函式原型與函式定義的第一行是一樣的。

函式定義與函式原型 (Function Definition and Function Prototype)

- 範例：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // define the prototype of the function
5  int maximum(int, int); //parameter names can be omitted
6
7  int main()
8  {
9      printf("max(5, 3) = %d\n", maximum(5, 3));
10 }
11
12 // implement the function exactly
13 int maximum(int a, int b)
14 {
15     if (a > b) return a;
16     return b;
17 }
```

函式定義與函式原型 (Function Definition and Function Prototype)

- 若有多個函式要互相呼叫，則必須使用函式原型

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void foo(int);
5  void bar(int);
6
7  int main()
8  {
9      foo(30);
10 }
11
12 void foo(int n)
13 {
14     printf("%2d / 3 = ", n);
15     printf("%2d\n", n /= 3);
16     bar(n);
17 }
18
```

```
19 void bar(int n)
20 {
21     if (n == 0) return;
22     printf("%2d * 2 = ", n);
23     printf("%2d\n", n *= 2);
24     foo(n);
25 }
```

C:\Projects\ch04_code\protot

```
30 / 3 = 10
10 * 2 = 20
20 / 3 = 6
6 * 2 = 12
12 / 3 = 4
4 * 2 = 8
8 / 3 = 2
2 * 2 = 4
4 / 3 = 1
1 * 2 = 2
2 / 3 = 0
```

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。
- 另外，在 C 語言中沒有傳參考呼叫，而是使用傳值呼叫傳遞位址 (address) 代替，此方法會在第七章介紹。

傳值呼叫 (Call by Value)

- 以下範例的 add 函式中更動參數的值，並不會影響 main 函式中的引數的值：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int add(int a, int b)
5  {
6      a += b;
7      return a;
8  }
9
10 int main()
11 {
12     int a = 5, b = 10, c;
13     c = add(a, b);
14     printf("a = %d, c = %d\n", a, c);
15 }
```

C:\Projects\ch04_code\call_by x + v

a = 5, c = 15

識別字的範圍規則 Scale Rules of Identifier

識別字的範圍規則 (Scale Rules of Identifier)

- 在先前所學的判斷與迴圈中，在大括號區塊內部所宣告的變數，到大括號外面會變得不可使用，這就是一種範圍規則。
- 範圍規則決定了識別字的作用域，也就是識別字能被參考的範圍，包含 4 種識別字範圍 (scope of an identifier)：
 - 區塊範圍 (block scope)
 - 函式範圍 (function scope)
 - 函式原型範圍 (function-prototype scope)
 - 檔案範圍 (file scope)

識別字的範圍規則 (Scale Rules of Identifier)

- 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**(或稱區塊變數)。

- 函式範圍：

- **標籤**(在識別字後加上冒號，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。

識別字的範圍規則 (Scale Rules of Identifier)

- 函式原型範圍：

- 在函式原型參數列中的識別字。
- 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。

- 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。

識別字的範圍規則 (Scale Rules of Identifier)

思考練習

- 請判斷右方程式碼，哪幾行不符合範圍規則？

```
1  #include <stdio.h>
2
3  void foo(int p);
4  int a = 0;
5
6  int main()
7  {
8      int b = 1;
9      for (int i = 0; i < 10; i++)
10     {
11         int c = b + i;
12         foo(c);
13     }
14     printf("a = %d\n", a);
15     printf("b = %d\n", b);
16     printf("c = %d\n", c);
17     printf("i = %d\n", i);
18 }
19
20 void foo(int p)
21 {
22     if (a) a += c;
23     return a += p;
24 }
```

變數的儲存類別 Storage Classes of Variable

變數的儲存類別 (Storage Classes of Variable)

- 截至目前為止，我們已經會使用識別字來作為變數名稱以及函式名稱，也討論過識別字當變數名稱時的範圍規則。但識別字還有一些其他特性，就是儲存類別 (storage class)。
- 識別字的儲存類別會影響 **生命週期** (lifetime，儲存佔用期間)，以及 **作用域** (scope，能被參考的範圍) 等特性。

變數的儲存類別 (Storage Classes of Variable)

- C 提供了以下儲存類別指定詞 (storage class specifiers) :
 - auto : 自動變數 (Automatic Variable)
 - register : 暫存器變數 (Register Variable)
 - static : 靜態變數 (Static Variable)
 - extern : 外部變數 (External Variable)

auto 自動變數

- 自動變數 (Automatic variable) 或稱為內部變數，只可以是區域變數 (只能宣告在函式的參數列或函式區塊本體)。
- 自動變數的有效範圍是由變數的宣告處開始，一直到離開區塊時，自動變數將釋放掉所佔用的記憶體空間，等到下次呼叫函式或進入區塊時，再重新配置記憶體位址給該變數使用，而無法保留其舊值。

變數的儲存類別 (Storage Classes of Variable)

- 關鍵字 **auto** 用來明確指定宣告為自動變數。
- 函式的區域變數 (包括宣告在參數列或函式體的變數) 的儲存類別都預設為 **auto**。

```
1 #include <stdio.h>
2
3 auto int a; // ERROR, Global variable cannot be automatic
4 void foo( auto int p ) // OK, "auto" can be omitted
5 {
6     auto int local; // OK, "auto" can be omitted
7 }
8 int main()
9 {
10     auto int local; // OK, "auto" can be omitted
11 }
```

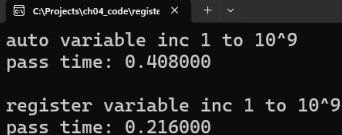
變數的儲存類別 (Storage Classes of Variable)

register 暫存器變數

- 暫存器變數與自動變數都是區域變數，差別在於暫存器變數會配置於 CPU 的 register 中，自動變數會配置在 RAM 中。
- 下方程式碼可以比較運算的速度。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main()
6 {
7     const int count = 1000000000; // 10^9
8     clock_t start, delta;
9
10    printf("auto variable inc 1 to 10^9\n");
11    start = clock();
12    for (auto int i = 0; i < count; i++);
13    delta = clock() - start;
14    printf("pass time: %f\n\n",
15           (double)delta / CLOCKS_PER_SEC);
```

```
16    printf("register variable inc 1 to 10^9\n");
17    start = clock();
18    for (register int i = 0; i < count; i++);
19    delta = clock() - start;
20    printf("pass time: %f\n\n",
21           (double)delta / CLOCKS_PER_SEC);
22 }
```



```
auto variable inc 1 to 10^9
pass time: 0.408000

register variable inc 1 to 10^9
pass time: 0.216000
```

變數的儲存類別 (Storage Classes of Variable)

static 靜態變數

- 靜態變數可以是區域變數或全域變數，屬於靜態儲存類別，與函式一樣都是從程式開始執行至程式結束都存在 (只會被宣告及初始化一次)。但因為範圍規則，這些變數並不是在程式的各個角落都可以被使用。
- 區域靜態變數只會被宣告一次，並在下次呼叫時延續之前的值

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int count()
5  {
6      static int c = 0;
7      return ++c;
8  }
9
10 int main()
11 {
12     int i;
13     for (i = 0; i < 10; i++)
14         printf("%d ", count());
15     printf("\n");
16 }
```

C:\Projects\ch04_code\static_ x + v

1 2 3 4 5 6 7 8 9 10

變數的儲存類別 (Storage Classes of Variable)

extern 外部變數

- 外部變數屬於全域變數。全域變數與函式若沒有指明儲存類別，會被預設為 `extern`。
- `extern` 與 `static` 同屬於靜態儲存類別，但不同的地方是當在 `extern` 變數的範圍規則之外的函式內或是同專案的不同檔案，可藉由宣告同名的外部變數來取用。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void foo()
5  {
6      extern int a; // out of scope, so define again
7      printf("a = %d\n", a);
8  }
9
10 int a = 10; // global/external variable a
11
12 int main()
13 {
14     foo();
15 }
```

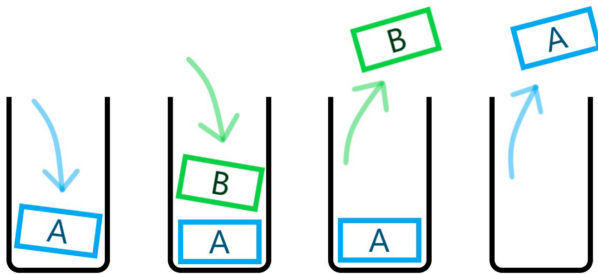
C:\Projects\ch04_code\extern x + v

a = 10

函式呼叫堆疊 Function Call Stack

函式呼叫堆疊 (Function Call Stack)

- 堆疊 (stack) 是一種後進先出 (last-in first-out, LIFO) 的結構。最後加入堆疊的項目會最先從堆疊取出。而函式的呼叫就是一種堆疊的機制。



堆疊示意圖

函式呼叫堆疊 (Function Call Stack)

- 以右方程式碼為例，執行流程會是：

- ① 進入 main()
- ② 進入 square(5)
- ③ 結束 square(5)
- ④ 結束 main()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int square(int a)
5 {
6     return a * a;
7 }
8
9 int main()
10 {
11     int a = square(5);
12     printf("%d\n", a);
13 }
```

函式呼叫堆疊 (Function Call Stack)

- 就算函式呼叫再複雜，也符合堆疊「後進先出」的概念。

- 以右方程式碼為例：

- ❶ 進入 main()
- ❷ 進入 isRight(3, 4, 5)
- ❸ 進入 square(a)
- ❹ 離開 square(a)
- ❺ 進入 square(b)
- ❻ 離開 square(b)
- ❼ 進入 square(c)
- ❽ 離開 square(c)
- ❾ 離開 isRight(3, 4, 5)
- ❿ 離開 main()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int square(int a)
5 {
6     return a * a;
7 }
8
9 int isRight(int a, int b, int c)
10 {
11     return square(a) + square(b) == square(c);
12 }
13
14 int main()
15 {
16     int a = isRight(3, 4, 5);
17     printf("%d\n", a);
18 }
```

遞迴函式 Recursive Function

遞迴

- 在程式設計中重複執行程式的方法分為遞迴 (recursion) 與迭代 (iteration)。遞迴是指在函數中直接或間接呼叫自身的方法，迭代則是使用迴圈敘述式重複運行程式區塊的方法。
- 使用遞迴可以設計出比迭代變化更多的流程控制。

遞迴函式 (Recursive Function)

- 在設計遞迴函式時，須將一個問題分為基本情況 (base case) 與遞迴情況 (recursive case)。
- 在「遞迴情況」中，會將問題概念性的分為「知道如何解決」與「不知道如何解決但類似於原先的問題」這兩個部分，並將不知道如何解決的部分呼叫自身來解決。

遞迴函式 (Recursive Function)

- 例如下方是一個階乘的數學遞迴式：

$$n! = \begin{cases} 1, & \text{if } n = 0 \text{ or } 1 \\ n \times (n-1)!, & \text{if } n > 1 \end{cases}$$

當 $n = 0$ 或 $n = 1$ 時為基本情況， $n > 1$ 時是遞迴情況。

- 而在遞迴情況中，只要將 $(n-1)!$ 乘以 n 就能求得 $n!$ ，雖然 $(n-1)!$ 暫時無法求得，但類似於原先的問題，可以透過呼叫自身來解決。

遞迴函式 (Recursive Function)

- 將階乘數學遞迴式寫成程式碼：

```
1 unsigned int factorial(unsigned int n)
2 {
3     if (n == 0 || n == 1)
4         return 1;
5     return n * factorial(n - 1);
6 }
```

標準函式庫的標頭 Headers of Standard Library

標準函式庫的標頭 (Headers of Standard Library)

- 每個標準函式庫都有一個相對應的標頭檔 (header file)，它含有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<>` 將標準函式庫的標頭檔的內容包括進程式碼裡。

標準函式庫標頭檔 (1/2)

assert.h	定義了用來除錯的 assert 巨集
ctype.h	定義了判斷字元的種類與轉換大小寫的函式
errno.h	用於測試錯誤代碼
float.h	定義了表示浮點數的極限值的常數
limits.h	定義了表示整數的極限值的常數
locate.h	定義了地區資訊的設定與取得的函式
math.h	定義了數學相關的函式
setjmp.h	定義了非局部跳躍用的函式

標準函式庫標頭檔 (2/2)

signal.h	定義了訊號處理的常數與函式
stdarg.h	定義了處理不確定個數的引數列的函式
stddef.h	定義了幾個常見的類型與巨集
stdio.h	定義了輸入輸出相關的函式
stdlib.h	定義了一些數字與文字的轉換、記憶體配置、亂數，以及其他公用函式
string.h	定義了處理字串與位元組串列的函式
time.h	定義了處理時間與日期相關的函式

產生亂數 Generate Random Number

產生亂數 (Generate Random Number)

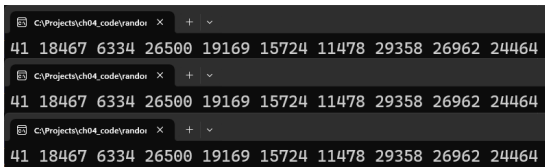
```
1 int rand();
```

- `<stdlib.h>` 定義了 `rand` 函式產生 0 到 `RAND_MAX` 之間的整數。
`RAND_MAX` 定義在 `<stdlib.h>` 中，其值可能根據編譯器而有所不同，於 GNU-GCC 中其值為 32767。
- 實際上，`rand` 函式所產生的是虛擬亂數 (pseudo-random numbers)。重複呼叫 `rand` 函式所產生的數列看起來是隨機的，其實是使用亂數演算法生成的數列。因此每次執行程式時，都會出現相同的數列。

產生亂數 (Generate Random Number)

- 每次執行程式時，rand 都會產生相同的數列。

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



C:\Projects\ch04_code\randor x + v
41 18467 6334 26500 19169 15724 11478 29358 26962 24464

C:\Projects\ch04_code\randor x + v
41 18467 6334 26500 19169 15724 11478 29358 26962 24464

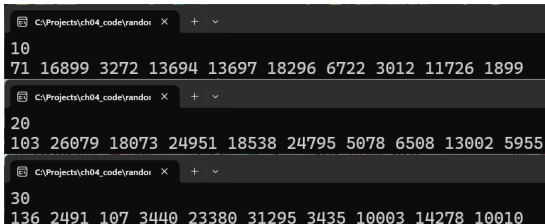
C:\Projects\ch04_code\randor x + v
41 18467 6334 26500 19169 15724 11478 29358 26962 24464

產生亂數 (Generate Random Number)

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

- 使用 srand 函式設定種子 (seed) 以初始化亂數生成器。
- 藉由每次執行程式時設定不同的 seed 來使 rand 函式的結果隨機化 (randomizing)。

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



Three screenshots of a Windows command prompt showing the output of the rand() function for different seed values:

- Seed 10: 10, 71, 16899, 3272, 13694, 13697, 18296, 6722, 3012, 11726, 1899
- Seed 20: 20, 103, 26079, 18073, 24951, 18538, 24795, 5078, 6508, 13002, 5955
- Seed 30: 30, 136, 2491, 107, 3440, 23380, 31295, 3435, 10003, 14278, 10010

產生亂數 (Generate Random Number)

- 若要在每次執行程式時自動設定不同的 seed，最簡單的方式就是使用當前時間作為 seed。藉由 `<time.h>` 所定義的 `time` 函式取得執行時的時間 (秒數) 作為 seed 來初始化亂數生成器。

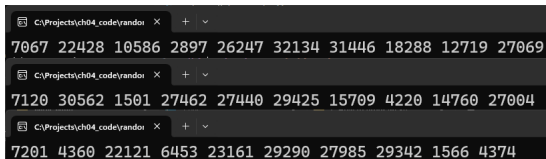
```
time_t time( time_t* timer );
```

- `time` 函式的回傳值型態 `time_t` 是 `<time.h>` 所定義的整數型態，回傳值為當前時間自 UTC1970 年 1 月 1 日 00:00 起經過的秒數。
- 另外，此函式為傳址呼叫函式 (於第七章介紹)，引數若不為 `NULL`，此函式也會把當前時間存入 `timer` 位址。

產生亂數 (Generate Random Number)

- 使用當前時間作為 seed :

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



```
C:\Projects\ch04_code\random x + v
7067 22428 10586 2897 26247 32134 31446 18288 12719 27069
C:\Projects\ch04_code\random x + v
7120 30562 1501 27462 27440 29425 15709 4220 14760 27004
C:\Projects\ch04_code\random x + v
7201 4360 22121 6453 23161 29290 27985 29342 1566 4374
```

比例化和位移

- 因為由 `rand` 所產生的值一定落在 $0 \leq \text{rand}() \leq \text{RAND_MAX}$ ，需藉由調整比例與平移，將數值調整到想要的範圍。
- 使用以下公式將隨機範圍調整為 a 到 $a + b - 1$ ，其中 b 不可大於 `RAND_MAX`：

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

產生亂數 (Generate Random Number)

- 模擬骰 10 次 6 面骰子：

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

C:\Projects\ch04_code\dice.e) X + v

3 4 2 6 3 1 2 1 2 6

Q & A