GLOBAL EDITION

Computer Programming and Practice (Ⅰ)
計算機程式設計與實習(一)
- Ch05 -

110年上學期

國立臺南大學 電機工程系

梁家銘

GLOBAL EDITION

5 函　式

5.1 簡介

- 電腦程式用於解決真實問題，通常程式會較龐大。
  - 開發和維護大型程式的最佳方法，是將程式分割成數個較好管理的小單元或**模組(module)**，較有組織性、易於管理。
  - 此方式稱為**分治法 (divide and conquer)**。
- 本章將介紹C程式設計、實作、操作和維護大型程式方面的關鍵方式。

5.2 C語言中的程式模組(Modularizing)

- C程式的模組稱為**函式 (function)**，共分為兩種：
  - **C標準函式庫 (C standard library)**中的函式
  - 開發者自訂的新函式
- 以上兩者結合便構成程式，本章將討論此兩種函式。

- 函式透過**函式呼叫 (function call)** 方式**調用 (invoked)**。
- 函式指明了欲調用之函式名稱，再提供所需資訊 (稱為**引數, argument**) 給受呼叫函式，以便執行工作。
  - 如同：老闆 (**呼叫的函式**或**呼叫者，calling function**或**caller**) 請求某位員工 (**被呼叫的函式，called function**) 去執行某項工作，並在工作完成後回報 (如下圖)。

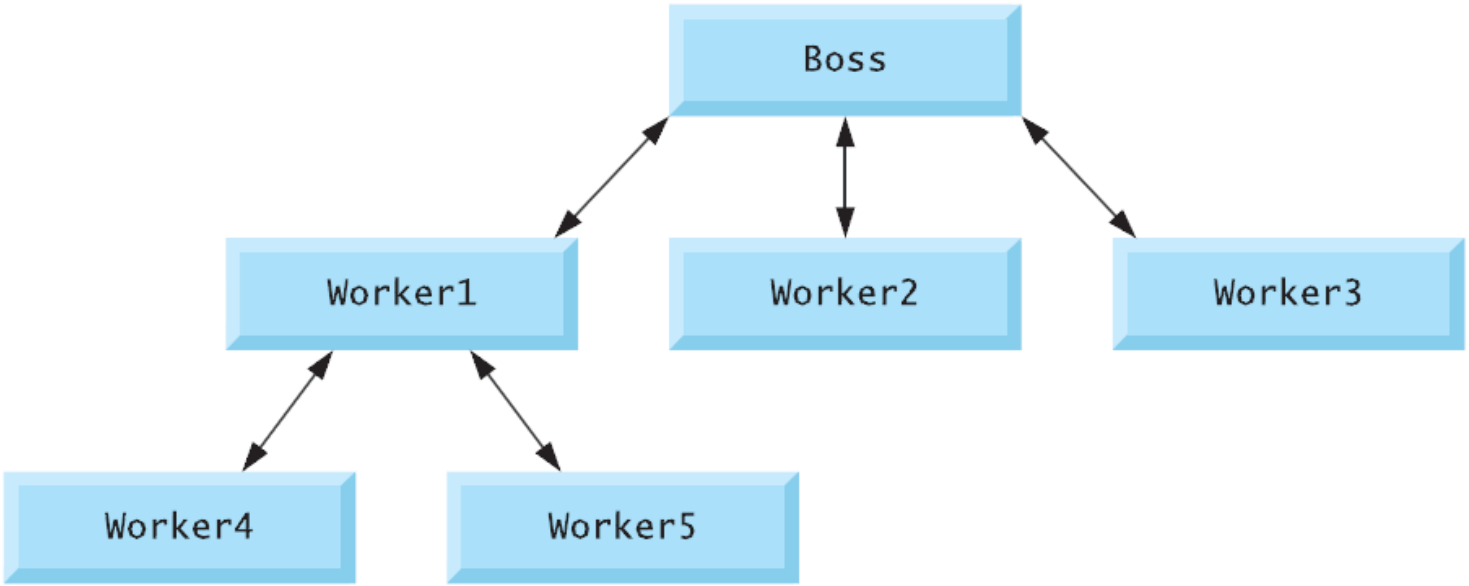


圖5.1　階層式的老闆函式/員工函式關係圖

5.3 數學函式庫函式

- 數學函式庫函式能執行某些常用數學計算。
  - 函式呼叫時，會寫成：函式名稱(引數1, 引數2, ...)。
  - 註：**引數(argument)**由逗號分隔，可以是：常數、變數或運算式。
- 右圖整理一些C程式的數學函式庫函式。
  - 此圖中，變數**x**和**y**型別都是**double**型態。

函式	說明	範例
sqrt(x)	x的平方根	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
exp(x)	指數函式 e^x	exp(1.0) is 2.718282 exp(2.0) is 7.389056
log(x)	x的自然對數 (底爲 e)	log(2.718282) is 1.0 log(7.389056) is 2.0
log10(x)	x的對數 (底爲 10)	log10(1.0) is 0.0 log10(10.0) is 1.0 log10(100.0) is 2.0
fabs(x)	x的絕對值	fabs(13.5) is 13.5 fabs(0.0) is 0.0 fabs(-13.5) is 13.5
ceil(x)	不小於 x的最小整數 (上高斯)	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
floor(x)	不大於 x的最大整數 (下高斯)	floor(9.2) is 9.0 floor(-9.8) is -10.0
pow(x, y)	x的 y次方 (x^y)	pow(2, 7) is 128.0 pow(9, .5) is 3.0
fmod(x, y)	x/y的浮點餘數	fmod(13.657, 2.333) is 1.992
sin(x)	x的正弦值 (x的單位爲弧度)	sin(0.0) is 0.0
cos(x)	x的餘弦值 (x的單位爲弧度)	cos(0.0) is 1.0
tan(x)	x的正切值 (x的單位爲弧度)	tan(0.0) is 0.0

圖5.2　常用的數學函式庫函式

5.4 函式(Functions)

- 函式能夠將一個程式模組化。
  - 宣告在函式定義裡的變數都是**區域變數 (local variable)**只限在函式內能使用。
  - 大部分函式都有**參數/引數列(parameter)**。
  - 參數/引數可傳遞給被呼叫的函式資訊。
  - 函式之參數也屬於區域變數
- **main**函式主要負責呼叫函式來完成程式的工作。
  - 右圖程式使用了一個稱為**square**函式來計算1~10之整數平方。

(練習Trace code)

程式碼

```
1  // Fig. 5.3: fig05_03.c
2  // Creating and using a programmer-defined function.
3  #include <stdio.h>
4
5  int square( int y ); // function prototype
6                       //宣告函式
7  // function main begins program execution
8  int main( void )
9  {
10    int x; // counter
11
12    // loop 10 times and calculate and output square of x each time
13    for ( x = 1; x <= 10; ++x ) {
14       printf( "%d  ", square( x ) ); // function call
15    } // end for            //呼叫函式
16
17    puts( "" ); //換行
18 } // end main
19
20 // square function definition returns the square of its parameter
21 int square( int y ) // y is a copy of the argument to the function
22 {                   //函式定義
23    return y * y; // returns the square of y as an int
24 } // end function square
```

```
1  4  9  16  25  36  49  64  81  100
```

圖5.3 建立並使用自訂函式

5.5 函式定義(Function Definitions)

1. 函式定義的格式如下

```
                    //回傳型態      //函式名稱    //引數/參數列
return-value-type function-name(parameter-list)
{
    statements
}
                                                  //合法: 由英文字母、數字組成
                                                  ( 數字不能在前、英文有區分大小寫)
```

- 函式名稱 (function-name)：可以是任何合法的識別字(Identifier)。
- 回傳型態 (return-value-type)：傳回給呼叫者之結果的資料型別。
  - 若為**void**，表示無回傳值。
  - **return-value-type**、**function-name**及**parameter-list**亦稱為函式**標頭 (function header)**。
- 引數/參數列(parameter-list)：是一串以逗號分隔的變數宣告(如: a, b, c …)，用於傳遞給此函式。
  - 若函式不需要任何引數，可寫**void**，比如： int hello (void);

2. main的回傳型態

- main具有整數回傳型態，其回傳值是用來表示程式是否正確執行。

```
return 0;
```

  - 在main的結尾，return 0; 代表程式執行成功。
  - 若省略此敘述，則隱含預設為0 (本書範例皆如此)。

3. maximum函式

- 下圖為第二個例子：三個數找最大值。
  - 此程式使用一個自訂函式**maximum**，用來傳回三個整數中的最大數值。

程式碼

```
1  // Fig. 5.4: fig05_04.c
2  // Finding the maximum of three integers.
3  #include <stdio.h>
4
5  int maximum( int x, int y, int z ); // function prototype
6                                      //宣告函式
7  // function main begins program execution
8  int main( void )
9  {
10    int number1; // first integer entered by the user
11    int number2; // second integer entered by the user
12    int number3; // third integer entered by the user
13
14    printf( "%s", "Enter three integers: " );
15    scanf( "%d%d%d", &number1, &number2, &number3 );
16
17    // number1, number2 and number3 are arguments
18    // to the maximum function call
19    printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
20 } // end main                  //呼叫函式
21
22 // Function maximum definition
23 // x, y and z are parameters
24 int maximum( int x, int y, int z )//函式定義
25 {
26    int max = x; // assume x is largest
27
28    if ( y > max ) { // if y is larger than max,
29       max = y; // assign y to max
30    } // end if
31
32    if ( z > max ) { // if z is larger than max,
33       max = z; // assign z to max
34    } // end if
35
36    return max; // max is largest value
37 } // end function maximum
```

```
Enter three integers: 22 85 17
Maximum is: 85
```

```
Enter three integers: 47 32 14
Maximum is: 47
```

```
Enter three integers: 35 8 79
Maximum is: 79
```

圖5.4　取得三個整數中的最大值

5.6 函式原型 (類似宣告)

- 函式原型 (function prototype)是參考C++的功能。
  - 函式原型是提供編譯器驗證函式呼叫(function calling)是否錯誤或不正確呼叫。
  - 前圖例中 (第5行)，函式**maximum**的函式原型為

```
int maximum(int x, int y, int z); // function prototype
```

  - 其指出**maximum函式**共有三個**int**型別的引數，並傳回**int**型別的數值結果。
  - 註1：函式原型和**maximum**函式定義的第一行是相同的。
  - 註2：編譯器會忽略寫在函式原型裡的變數名稱。

//某些編譯器允許原型不寫變數，如： int max (int, int, int);

1. 編譯錯誤

- 若不符合函式原型描述的函式呼叫 (比如：參數個數不同)，將導致編譯錯誤。
  - 若函式原型與函式定義不一致，也會造成錯誤。
  - 舉例來說：若圖5.4的函式原型改成

```
void maximum(int x, int y, int z);
```

```
int maximum( int x, int y, int z )
{                                   //函式定義(標頭)
   int max = x; // assume x is largest

   if ( y > max ) { // if y is larger than max,
      max = y; // assign y to max
   } // end if

   if ( z > max ) { // if z is larger than max,
      max = z; // assign z to max
   } // end if
   return max; // max is largest value
} // end function maximum
```

  - 編譯時將產生錯誤，由於函式原型是**void**回傳型別，與函式標頭中的**int**回傳型別不同。

2. 強制參數與常用算數轉換規則

- 函式原型的另一項功能為：**強制引數型別轉換 (coercion of arguments)**，能強迫引數轉成合適型別。
  - 比如：double sqrt (double x); //函式宣告
  - printf("%f", sqrt(2)); //函式呼叫：自動將2(整數)轉換成double型態
  - 算數轉換規則，會自動應用到含有兩種(或更多)資料型別之數值運算式也稱為**混合型別運算式 (mixed-type expressions)**，會自動由編譯器處理
  - 至少包含一個浮點數值的混合型別運算式，常見算數轉換規則為：

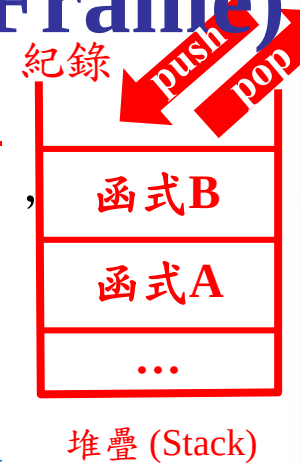
- 若一數值為**long double**(較高精準度)，則其他值都轉換成**long double**。
- 若一個數值為**double**(較高精準度)，則其他值都轉換成**double**。
- 若一個數值為**float**(較高精準度)，則其他值都轉換成**float**。

- 下圖列出了所有資料型別(由最高型別至最低型別)，及每一型別的**printf**和**scanf**轉換指定方式。

資料型別	printf 的轉換指定詞	scanf 的轉換指定詞
Floating-point types		
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
Integer types		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

圖5.5　算數資料型別與轉換規則

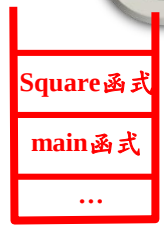
5.7 函式呼叫堆疊與堆疊框架(Stack Frame)



- **堆疊(stack)**是一種用於存取的資料結構方式，透過**後進先出 (last-in first-out，LIFO)**存取概念，以加入(Push)方式放在堆疊最上層，而取出(Pop)方式從最上層拿取。
  - 被呼叫的函式，可在堆疊上層找到其回傳資訊及返回位址(即為堆疊框架, stack frame)
  - 舉例來說：若函式A呼叫了函數B，新的函數B之堆疊框架(含函式資訊，如：變數及位址)將會推入堆疊之上層。

1. 函數呼叫堆疊

- 右圖探討：**main**函式呼叫**square**函數的堆疊運作。
  - 1) 首先，作業系統呼叫**main函式**
  - 2) 將**main函式**的堆疊框架推入(push)堆疊
  - 3) 堆疊框架會紀錄**main函式**返回作業系統的回傳位址R1及**main函式**的變數資訊 (即a = 10)。




```
1  // Fig. 5.6: fig05_06.c
2  // Demonstrating the function call stack
3  // and stack frames using a function square.
4  #include <stdio.h>
5
6  int square( int ); // prototype for function square
7
8  int main()
9  {
10    int a = 10; // value to square (local automatic variable in main)
11
12    printf( "%d squared: %d\n", a, square( a ) ); // display a squared
13 } // end main
14
15 // returns the square of an integer
16 int square( int x ) // x is a local variable
17 {
18    return x * x; // calculate square and return result
19 } // end function square
```

```
10 squared: 100
```

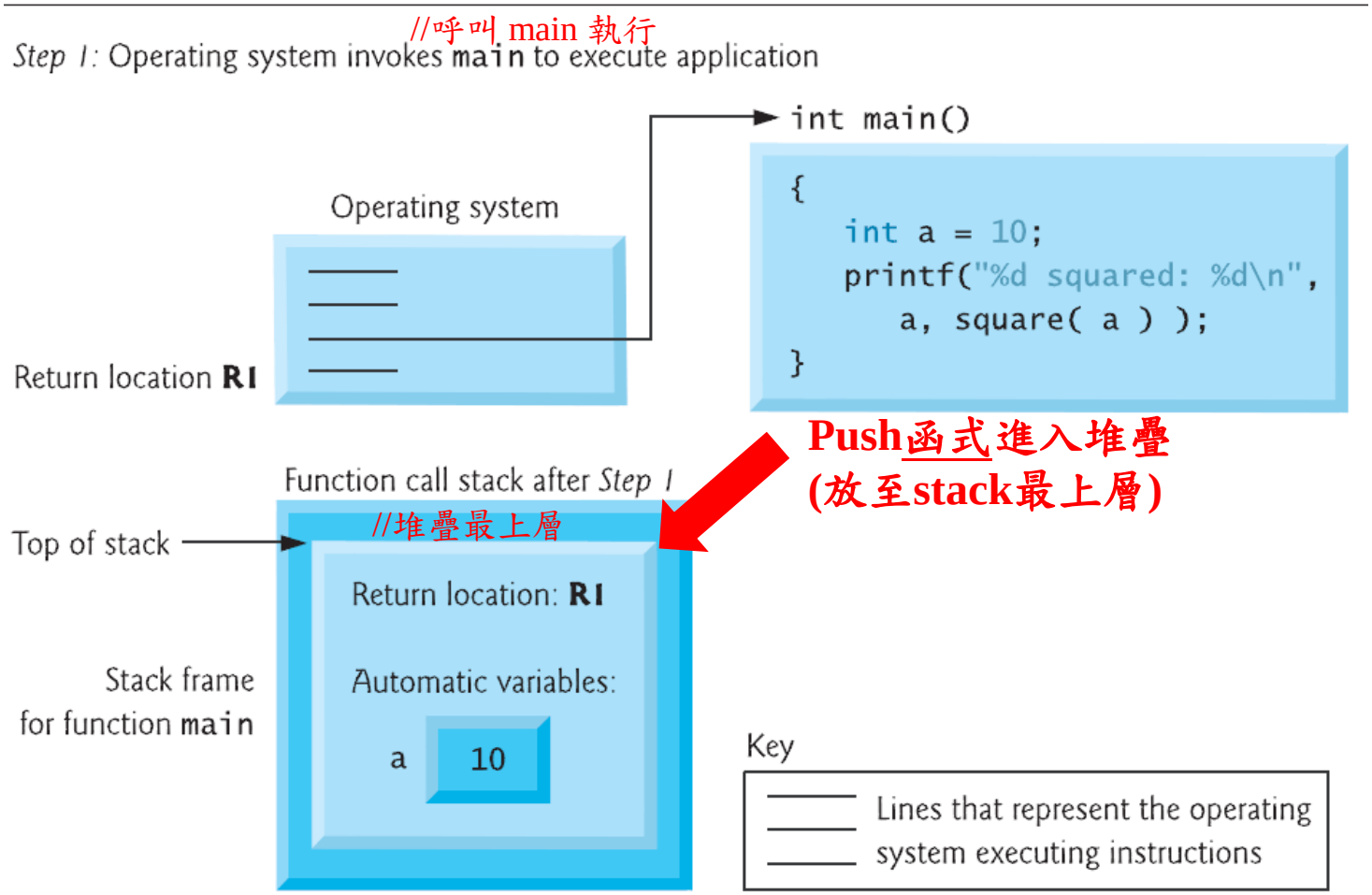


圖5.6 使用square函數展示函數呼叫堆疊與堆疊框架

2. 呼叫堆疊 (Call Stack)

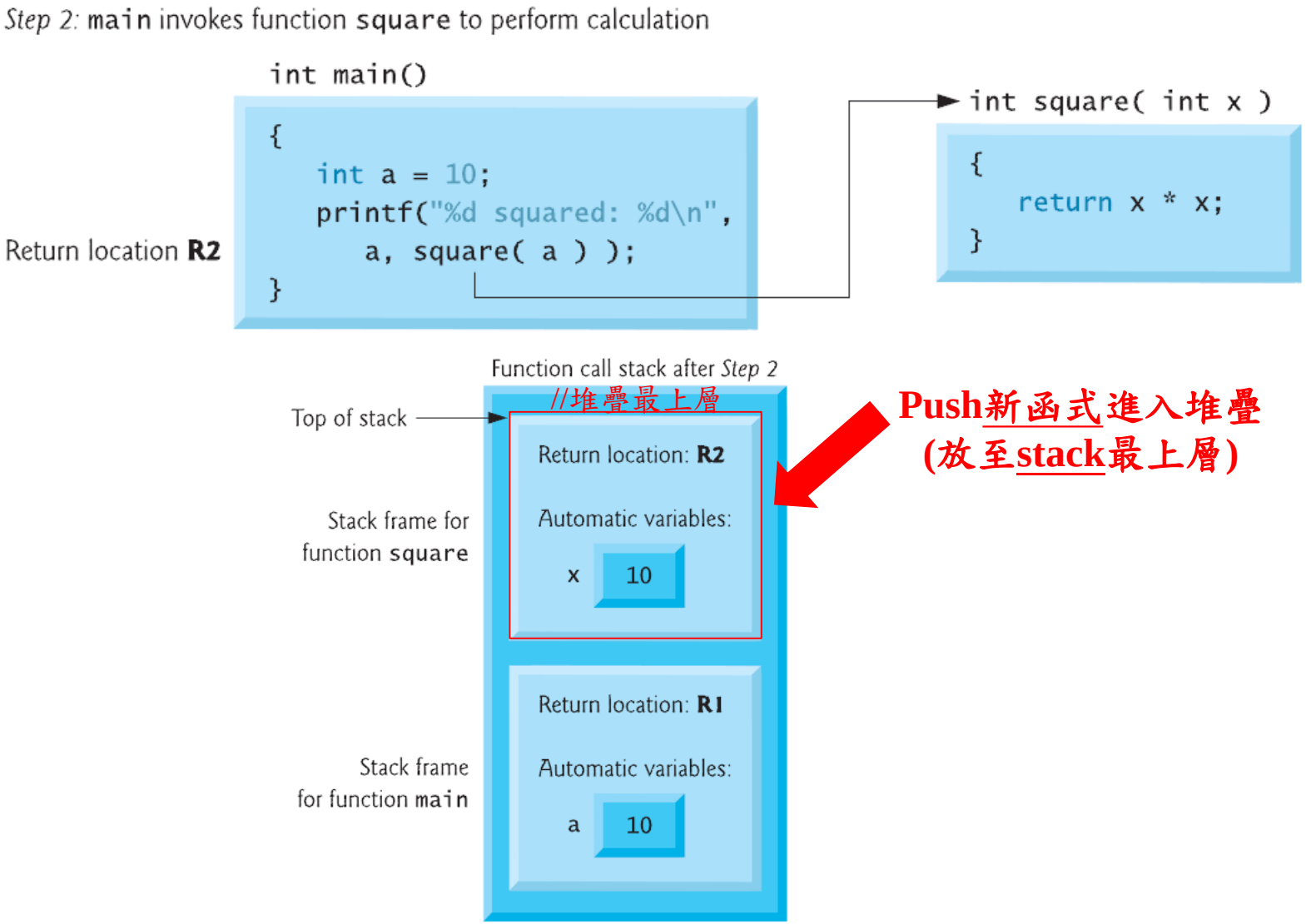


圖5.8 main函數呼叫square函數進行計算後的函數呼叫堆疊

- **main函式**—在回傳至作業系統前，於圖5.6(第12行)呼叫了**square函式**，使得**square函式**的堆疊框架（第16-19行）推入至函數呼叫堆疊中（圖5.8）。
- 下圖表示在**square**堆疊框架被取出後的函數呼叫堆疊。

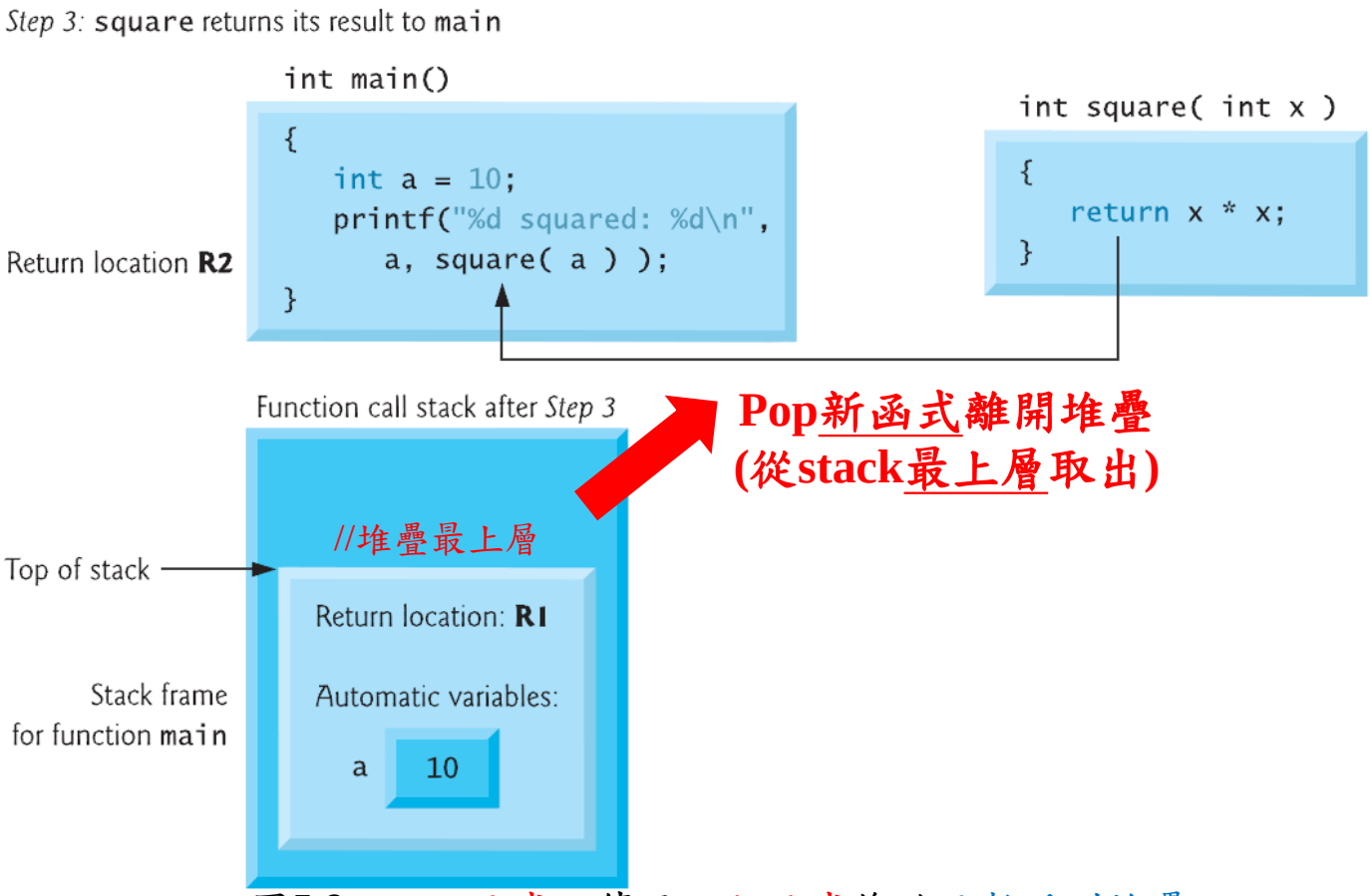


圖5.9 square函式回傳至main函式後的函數呼叫堆疊

5.8 標頭(Headers)

- 每個標準函式庫，都有一個對應的**標頭檔(header, 如: *.h)**
  - 描述此函式庫的函式原型及函式所需的資料型別和變數定義。
  - 右圖為常見的標準函式庫之標頭檔。
- 開發者也可以自定標頭檔。
  - 須以**.h**做為副檔名。
  - 透過前置處理器命令**#include**及雙引號"..."，引入自定義的標頭檔，例如：

    #include "mylib.h"

標準函式庫標頭	說明
<assert.h>	內含一些用來幫助程式偵錯的巨集和資訊。
<ctype.h>	內含一些檢測字元特性及大小寫字元轉換等函式的原型。
<errno.h>	定義了一些用來回報錯誤狀況的巨集。
<float.h>	內含此系統中對浮點數大小的限制。
<limits.h>	內含此系統中對整數大小的限制。
<locale.h>	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。
<math.h>	內含數學函式庫的函式原型。
<setjmp.h>	內含改變正常函式呼叫與回傳順序的函式原型。
<signal.h>	內含處理程式執行中各種狀況的函式原型和巨集。
<stdarg.h>	定義一些處理不確定型別及個數之引數列的函式。
<stddef.h>	內含一些在 C 執行運算時所常用到的型別。
<stdio.h>	內含標準輸出 / 入函式庫的函式原型以及所需的資訊。
<stdlib.h>	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。
<string.h>	內含字串處理函式的原型。
<time.h>	內含處理時間與日期的函式原型。

圖5.10 標準函式庫之標頭檔

5.9 由值與參考傳遞參數

- **傳值呼叫 (pass by value)** 和**傳參考呼叫 (pass by reference)** 為程式語言用來呼叫函式的兩種傳遞值方式。
  1. 以傳值呼叫(pass by value)來傳遞引數時，會複製此值並傳給被呼叫函式，因此不會修改原來變數值。
  2. 以傳參考(pass by reference)來傳遞引數時，是傳遞變數位址給被呼叫函式，因此可以修改原來變數值 (以上兩者視情況運用)。

5.10 亂數產生(Random Number Generation)

- 以下介紹程式應用：模擬擲骰子程式。
  - 利用標準函式庫中標頭檔 **<stdlib.h>**的**rand()**函式，來模擬隨機數的產生，如下敘述：

```
i = rand();
```

  - 其中**rand();**函式將產生一個介於 0 ~ **RAND_MAX**的整數(**RAND_MAX**是殊特常數，定義在**<stdlib.h>**標頭檔中， C程式標準規定**RAND_MAX** 的值至少需為 32767)。

```
//不同平台,如:win/linux不太相同
```

1. 投擲六面的骰子

- 以下程式模擬投擲具有1~6點的骰子20次，並印出每次擲的值。
  - 下方隨機數的結果(1~6之間)，會因不同編譯器而有所差異。

(練習Trace code)


```
1  // Fig. 5.11: fig05_11.c
2  // Shifted, scaled random integers produced by 1 + rand() % 6.
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  // function main begins program execution
7  int main( void )
8  {
9     unsigned int i; // counter //非負整數
10
11    // loop 20 times
12    for ( i = 1; i <= 20; ++i ) {
13
14       // pick random number from 1 to 6 and output it
15       printf( "%10d", 1 + ( rand() % 6 ) );
16
17       // if counter is divisible by 5, begin new line of output
18       if ( i % 5 == 0 ) {
19          puts( "" ); //換行
20       } // end if
21    } // end for
22 } // end main
```

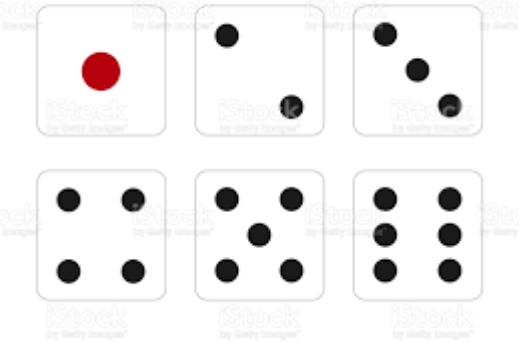


圖5.11(a) 六面骰子

```
         6         6         5         5         6
         5         1         1         5         3
         6         6         2         4         2
         6         2         3         4         1
```

圖5.11　利用**1+rand()%6**產生的平移、縮放後的隨機數

2. 投擲一個6面的骰子6,000,000次

- 當擲骰子的次數增加，這些隨機數的機率將會相近
  - 下圖程式模擬投擲 6,000,000次骰子結果
  - 預期結果：1 ~ 6的整數次數，都將接近1,000,000次左右。


```
1  // Fig. 5.12: fig05_12.c
2  // Rolling a six-sided die 6,000,000 times.
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  // function main begins program execution
7  int main( void )
8  {
9     unsigned int frequency1 = 0; // rolled 1 counter
10    unsigned int frequency2 = 0; // rolled 2 counter
11    unsigned int frequency3 = 0; // rolled 3 counter
12    unsigned int frequency4 = 0; // rolled 4 counter
13    unsigned int frequency5 = 0; // rolled 5 counter
14    unsigned int frequency6 = 0; // rolled 6 counter
15
16    unsigned int roll; // roll counter, value 1 to 6000000
17    int face; // represents one roll of the die, value 1 to 6
18
19    // loop 6000000 times and summarize results
20    for ( roll = 1; roll <= 6000000; ++roll ) {
21       face = 1 + rand() % 6; // random number from 1 to 6
22
23       // determine face value and increment appropriate counter
24       switch ( face ) {
26          case 1: // rolled 1
27             ++frequency1;
28             break;
29
30          case 2: // rolled 2
31             ++frequency2;
32             break;
33
34          case 3: // rolled 3
35             ++frequency3;
36             break;
37
38          case 4: // rolled 4
39             ++frequency4;
40             break;
41
42          case 5: // rolled 5
43             ++frequency5;
44             break;
45
46          case 6: // rolled 6
47             ++frequency6;
48             break; // optional
49       } // end switch
50    } // end for
```

//練習看註解，有助於了解程式

圖5.12　投擲一顆6面骰子6,000,000次

```
52      // display results in tabular format
53      printf( "%s%13s\n", "Face", "Frequency" );
54      printf( "    1%13u\n", frequency1 );
55      printf( "    2%13u\n", frequency2 );
56      printf( "    3%13u\n", frequency3 );
57      printf( "    4%13u\n", frequency4 );
58      printf( "    5%13u\n", frequency5 );
59      printf( "    6%13u\n", frequency6 );
60   } // end main
```

程式碼

```
Face    Frequency
   1       999702
   2      1000823
   3       999378
   4       998898
   5      1000777
   6      1000422
```

//各面結果接近1,000,000次左右

圖5.12　投擲一顆6面骰子6,000,000次

3. 對亂數產生器進行隨機化

- 當每次執行圖5.11擲骰子程式時，竟得到相同結果(如下), why~?!

```
      6       6       5       5       6
      5       1       1       5       3
      6       6       2       4       2
      6       2       3       4       1
```

- 事實上，**rand函式**所產生的是**虛擬亂數(pseudorandom numbers)**。
  - 重複呼叫**rand**所產生的隨機數列，都會出現相同的隨機數列。
  - 若想每次執行產生不同隨機數列，可以透過標準函式庫**srand(放入種子)**函式來進行，此過程稱為**隨機化 (randomizing)**。
  - 註：在debug階段，最好先不用srand，避免讓程式太複雜(導致debug困難)

- 右圖5.13將示範srand(seed)函式用法。
  - 當放入不同seed值時，每次產生亂數順序便不同。
  - 若想每次出現不同隨機序列，可使用srand (time(NULL));。

- 若要產生不同範圍隨機數，利用縮放(scaling)和平移(shifting)方式：
  - 以下為例，可產生1~6的隨機數。

```
face = 1 + rand() % 6;
```

- 因此，指定範圍隨機數：

  n = a + rand() % b ;

  - 其中，**a**表示**平移值(shifting value)**，即隨機數的下界。
  - **b**代表**縮放因子(scaling factor)**，即隨機數的範圍。
  - Ex: 5~55，則 n = 5 + rand()% 51;

程式碼

```
1  // Fig. 5.13: fig05_13.c
2  // Randomizing the die-rolling program.
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  // function main begins program execution
7  int main( void )
8  {
9     unsigned int i; // counter
10    unsigned int seed; // number used to seed the random number generator
11
12    printf( "%s", "Enter seed: " );
13    scanf( "%u", &seed ); // note %u for unsigned int
14
15    srand( seed );  // seed the random number generator
16
17    // loop 10 times
18    for ( i = 1; i <= 10; ++i ) {
19
20       // pick a random number from 1 to 6 and output it
21       printf( "%10d", 1 + ( rand() % 6 ) );
22
23       // if counter is divisible by 5, begin a new line of output
24       if ( i % 5 == 0 ) {
25          puts( "" );
26       } // end if
27    } // end for
28 } // end main
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

```
Enter seed: 867
         2         4         6         1         6
         1         1         3         6         2
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

5.11 程式範例：機會遊戲 (採用enum方式)

- 以下示範「crap」的擲骰子遊戲，遊戲規則如下：
  - 玩家共投擲兩顆骰子，每顆骰子共6個面 (分別為1、2、3、4、5和6點)。
  - 若第一輪投擲加總點數為：7點或11點，則判定玩家「贏」。
  - 若第一輪擲出加總點數為：2點、3點或12點，則玩家「輸」(莊家贏)，這些點數稱為「crap」。
  - 反之，若第一輪擲出其他點數：4點、5點、6點、8點、9點或10點，則紀錄此數，成為玩家的「目標點數」。
  - 之後，須繼續投擲這兩顆骰子直到「目標點數」才算贏；但若擲出7點，則判定玩家「輸」。

```
1  // Fig. 5.14: fig05_14.c
2  // Simulating the game of craps.   (練習Trace code)
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h> // contains prototype for function time
6
7  // enumeration constants represent game status
8  enum Status { CONTINUE, WON, LOST }; //新型態定義
9
10 int rollDice( void ); // function prototype
11                         //函式宣告
12 // function main begins program execution
13 int main( void )
14 {
15    int sum; // sum of rolled dice
16    int myPoint; // player must make this point to win
17
18    enum Status gameStatus; // can contain CONTINUE, WON, or LOST
19                            //gameStatus 則有三種數值可能
20    // randomize random number generator using current time
21    srand( time( NULL ) ); //以時間作為隨機數種子
22
```

```
                                  //第一輪擲骰子
23    sum = rollDice(); // first roll of the dice
24                                  //函式呼叫
25    // determine game status based on sum of dice
26    switch( sum ) {
27
28       // win on first roll
29       case 7: // 7 is a winner
30       case 11: // 11 is a winner
31          gameStatus = WON; // game has been won
32          break;
33
34       // lose on first roll
35       case 2: // 2 is a loser
36       case 3: // 3 is a loser
37       case 12: // 12 is a loser
38          gameStatus = LOST; // game has been lost
39          break;
40
41       // remember point
42       default:
43          gameStatus = CONTINUE; // player should keep rolling
44          myPoint = sum; // remember the point
45          printf( "Point is %d\n", myPoint );
46          break; // optional
47    } // end switch
48
```

```
//第二輪擲骰子
49    // while game not complete
50    while ( CONTINUE == gameStatus ) { // player should keep rolling
51       sum = rollDice(); // roll dice again
52
53       // determine game status
54       if ( sum == myPoint ) { // win by making point
55          gameStatus = WON; // game over, player won
56       } // end if
57       else {
58          if ( 7 == sum ) { // lose by rolling 7
59             gameStatus = LOST; // game over, player lost
60          } // end if
61       } // end else
62    } // end while
63
64    // display won or lost message
65    if ( WON == gameStatus ) { // did player win?
66       puts( "Player wins" );
67    } // end if
68    else { // player lost
69       puts( "Player loses" );
70    } // end else
71 } // end main
72
73 // roll dice, calculate sum and display results
74 int rollDice( void ) //函式定義
75 {
76    int die1; // first die
77    int die2; // second die
78    int workSum; // sum of dice
79
80    die1 = 1 + ( rand() % 6 ); // pick random die1 value
81    die2 = 1 + ( rand() % 6 ); // pick random die2 value
82    workSum = die1 + die2; // sum die1 and die2
83
84    // display results of this roll
85    printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
86    return workSum; // return sum of dice
87 } // end function rollRice
```

Player wins on the first roll

```
Player rolled 5 + 6 = 11
Player wins
```
//第一輪出現：7或11點，玩家贏

Player wins on a subsequent roll

```
Player rolled 4 + 1 = 5
Point is 5
Player rolled 6 + 2 = 8
Player rolled 2 + 1 = 3
Player rolled 3 + 2 = 5
Player wins
```
//第一輪出現其他(5點)，紀錄成「目標點數」

//再次出現目標點數(5點)，玩家贏

Player loses on the first roll

```
Player rolled 1 + 1 = 2
Player loses
```
//第一輪出現：(2、3、12點)，玩家輸

Player loses on a subsequent roll

```
Player rolled 6 + 4 = 10
Point is 10
Player rolled 3 + 4 = 7
Player loses
```
//第一輪出現其他(10點)，紀錄成「目標點數」

//下一輪出現點數(7點)，玩家輸

圖5.15「Crap」遊戲的執行範例

1. 列舉(enum)

- 此遊戲利用宣告變數**gameStatus**為新定義型別**enum Status**，來記錄目前狀態{繼續, 贏, 輸}。
  - 第8行的關鍵字**enum**稱為**列舉(enumeration)**型別，是新定義的型別，表示特定識別字的整數常數集合。
  - 舉例如下：

    enum Status {CONTINUE, WON, LOSE};
  - 其中CONTINUE代表常數0， WON代表常數1，LOSE代為常數2。
- **列舉常數(Enumeration constants)** 有時稱為符號常數。
  - 列舉的識別字必須唯一

2. 當第一輪投擲遊戲後

- 當第一輪投擲後，若遊戲已結束，**while迴圈**將因**gameStatus**不等於**CONTINUE**而跳過 (第50-62行)。
  - 程式接著執行第65-70行的**if...else**敘述式。
  - 若**gameStatus**為1的話，則印出**"Player wins"**，否則印出**"Player loses"**。

3. 當連續投擲遊戲後

- 當第一輪投擲後，若遊戲還未分勝負，便將**sum**紀錄到**myPoint**變數。
  - 此時，**gameStatus**等於**CONTINUE**，程式接著執行**while**敘述。
  - **while**每次會呼叫**rollDice()**來產生新的隨機數**sum**。
  - 若**sum**和**myPoint**相同，則**gameStatus**設為WON，表示玩家贏了，接著**while**的條件檢查失敗，程式跳出**while**迴圈繼續**if...else**結構的執行。
  - **if...else**結構印出**"Player wins"**後，便結束程式。
  - 若**sum**等於7(第58行)，**gameStatus**將設為LOST，表示玩家輸了，接著跳出**while**迴圈，然後**if...else**敘述式印出**"Player loses"**後，便會結束執行。

4. 控制結構

- 本程式控制結構，使用兩個函式 (**main**和**rollDice**)及**switch**、**while**、巢狀**if...else**和巢狀**if**等敘述式。
  - 補充：習題會有更多有趣遊戲介紹。

5.12 儲存類別 (Storage Classes)

- C程式提供**儲存類別詞 (storage class specifiers)**：`auto`、`register`、`extern`和`static`，用來區分識別字(或變數)的儲存佔用期間、範圍和連結等特性。
  - **儲存佔用期間 (storage duration)** 是指此識別字(或變數)存在記憶體中的時間。
  - 補充：因為有些識別字(或變數)只需要短暫存在，而有些須要一直存在。
  - 識別字(或變數)的**範圍 (scope)**，是指此在程式中能夠被使用的範圍，比如：在全部程式中、僅在子函式中、僅在括弧中{...}。

1. 區域變數 (Local Variables)

- 只有變數才能具有自動儲存佔用期間(automatic storage duration)。
  - 函式中的變數 (包含：參數或函式中宣告的變數)，屬於區域變數，具有自動儲存佔用期間
  - 變數也可以用`auto`用來宣告為自動儲存佔用期間。 `//補充：auto 區域內使用`
  - 區域變數預設具有自動儲存佔用期間，因此`auto`很少用。
  - 以下，我們會將自動儲存佔有期間的變數，簡稱為**自動變數 (automatic variables)**。

2. **全域變數（Global variable）**是將變數宣告在任何函式定義之外來產生，他們在整個程式 執行的過程都一直保留著。

2. 靜態儲存類別 (Static Storage Class)

- 關鍵字**extern**和**static**，可用來將變數或函式宣告為具有靜態儲存佔用期間，允許在程式開始執行至程式結束都存在。
  //補充：extern 關鍵字，用來表示此變數已經在別處定義(definition)，以後用到後會比較熟悉
  - 這些變數可以在整個程式中使用，比如：全域變數和函式名稱(其預設為**extern**儲存類別)及**static**宣告的區域變數。
- 宣告成static的區域變數，在函式之間呼叫時會保留其值，並只會初始化一次，下次呼叫函式時，會保存上次數值，不再做初始化。
  - 若未指定初始值，則預設為0。
  - Ex:
```
static int count = 1;
```

5.13 範圍規則(Scope Rules)

- **變數的使用範圍 (scope of an identifier)** 是指可存取此變數的程式範圍。
  - 比如：在函式中宣告一個變數(區域變數)時，只能在函式中使用。
- **變數**使用範圍共分為四種：
  - **檔案範圍 (file scope, 跨整個檔案, 如：全域變數)**、**函式範圍 (function scope)**、**區塊範圍 (block scope)**及**函式原型範圍 (function-prototype scope)**。
  - 其中，**檔案範圍 (file scope)**是指宣告在所有函式外的變數，從宣告位置開始，到整個檔案結束，都能使用，如：全域變數、函式定義、函式原型等都具有檔案範圍。
- 標籤 (識別字再加一個冒號，如**start:**)是唯一具有**函式範圍 (function scope)**的識別字。
  - 標籤可在出現的函式中任何位置使用，不能超出函式本體。

//補充：標籤常配合goto使用，表示跳到此標籤(如: start:)執行

//重要常考

```c
// Fig. 5.16: fig05_16.c
// Scoping.
#include <stdio.h>

void useLocal( void ); // function prototype
void useStaticLocal( void ); // function prototype
void useGlobal( void ); // function prototype

int x = 1; // global variable

// function main begins program execution
int main( void )
{
   int x = 5; // local variable to main

   printf("local x in outer scope of main is %d\n", x );

   { // start new scope
      int x = 7; // local variable to new scope

      printf( "local x in inner scope of main is %d\n", x );
   } // end new scope

   printf( "local x in outer scope of main is %d\n", x );

   useLocal(); // useLocal has automatic local x
   useStaticLocal(); // useStaticLocal has static local x
   useGlobal(); // useGlobal uses global x
   useLocal(); // useLocal reinitializes automatic local x
   useStaticLocal(); // static local x retains its prior value
   useGlobal(); // global x also retains its value

   printf( "\nlocal x in main is %d\n", x );
} // end main

// useLocal reinitializes local variable x during each call
void useLocal( void )
{
   int x = 25; // initialized each time useLocal is called

   printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
   ++x;
   printf( "local x in useLocal is %d before exiting useLocal\n", x );
} // end function useLocal

// useStaticLocal initializes static local variable x only the first time
// the function is called; value of x is saved between calls to this
// function
void useStaticLocal( void )
{
   // initialized once before program startup
   static int x = 50;

   printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
   ++x;
   printf( "local static x is %d on exiting useStaticLocal\n", x );
} // end function useStaticLocal

// function useGlobal modifies global variable x during each call
void useGlobal( void )
{
   printf( "\nglobal x is %d on entering useGlobal\n", x );
   x *= 10;
   printf( "global x is %d on exiting useGlobal\n", x );
} // end function useGlobal
```

//全域變數(宣告於函式之外)

//區域變數(限制在main函式內使用)

//區塊變數(限制在區塊內使用)

//區域靜態變數(可重覆在函式useStaticLocal 內使用)

//沒有宣告，直接的是全域變數x

//區域變數(限制在函式useLocal內使用)

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```

圖5.16　範圍(Scope)的例子

- **區塊範圍 (block scope)**：宣告在區塊{....}內的變數，僅能在右大括號}結束前使用 (註：多層括號時，需注意是哪一層 {..{..}..})
- **函式原型範圍 (function-prototype scope)**，是指函式原型參數列中的變數，僅能在函式中使用。
  - 註：函式原型的參數列，不一定需要寫變數名稱，僅需放型別即可。
  - Ex: int sqrt (int);
- 上頁示範：全域變數、自動區域變數及**static**區域變數的範圍界定差異。

5.14 遞迴 (Recursion)

- **遞迴函式(recursive function)** 是一種直接或間接呼叫自己的函式。
  - 將問題分成兩種概念性小塊：一塊是函式已知如何處理，另一塊則是比原問題較為簡單的小塊，直到化簡至最基本情況(base case)藉以簡化問題。

1. 用遞迴方法計算階乘

- 計算非負整數的階乘n，寫成$n!$ (亦讀作「n階乘」)，如下乘積：

$$n \cdot (n-1) \cdot (n-2) \cdot \cdot 1$$

  - 根據定義：1!等於1，0!等於1。

```
//想法：n! = n * (n-1)!
改寫成：f(n) = n * f(n-1);
```

//想法：n! = n * (n-1)!

- 舉例來說：5! 計算，如下所示。

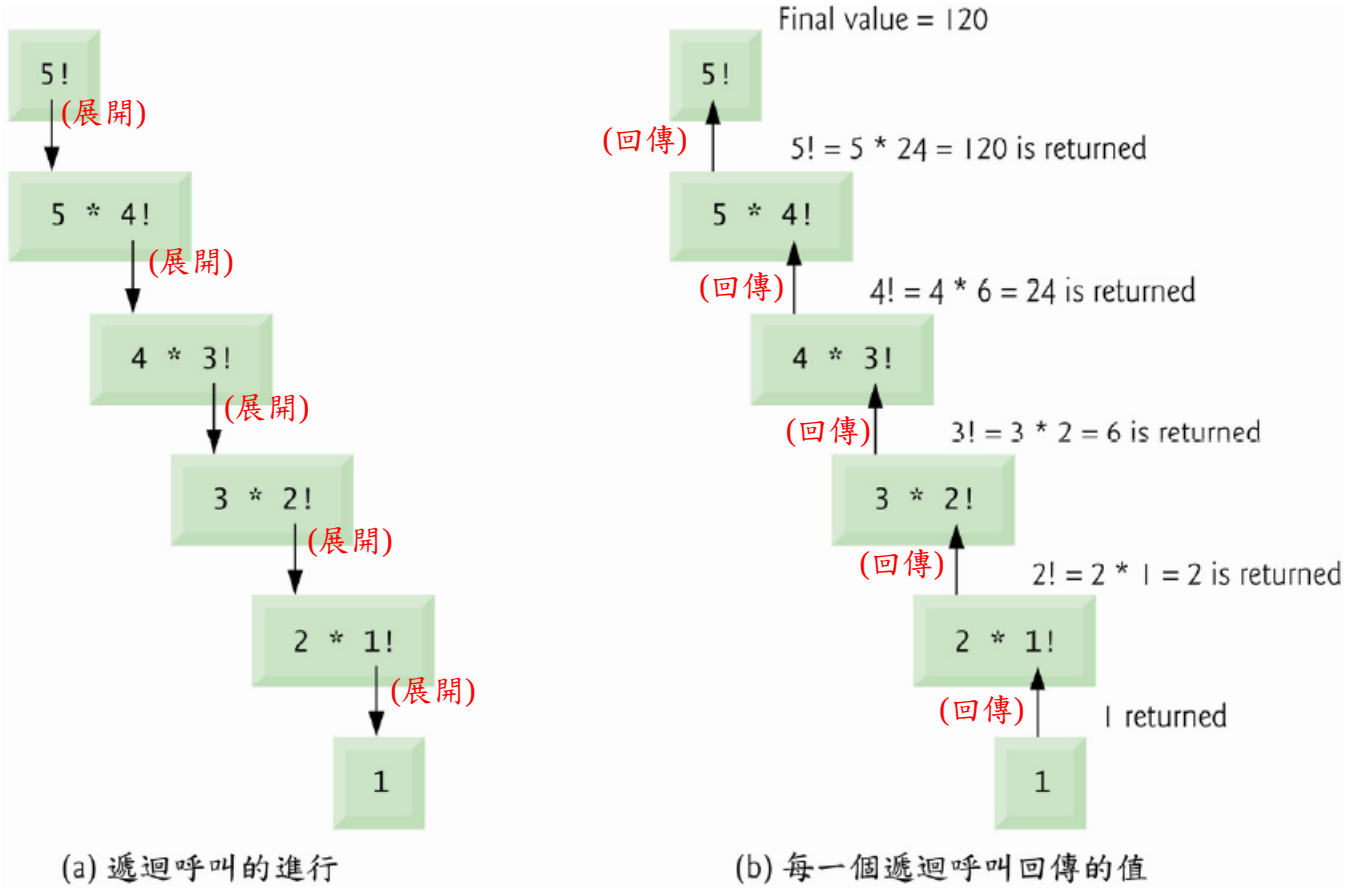


圖5.17　5! 的遞迴求值法

- 下圖利用遞迴來計算，並印出0~21之整數階乘值。

程式碼

```
1  // Fig. 5.18: fig05_18.c
2  // Recursive factorial function.
3  #include <stdio.h>
4
5  unsigned long long int factorial( unsigned int number );
6
7  // function main begins program execution
8  int main( void )
9  {
10    unsigned int i; // counter
11
12    // during each iteration, calculate
13    // factorial( i ) and display result
14    for ( i = 0; i <= 21; ++i ) {
15       printf( "%u! = %llu\n", i, factorial( i ) );
16    } // end for
17 } // end main
18
19 // recursive definition of function factorial
20 unsigned long long int factorial( unsigned int number )
21 {
22    // base case
23    if ( number <= 1 ) { //1) 基本case
24       return 1;
25    } // end if
26    else { // recursive step                    //2) 遞迴case
27       return ( number * factorial( number - 1 ) );
28    } // end else                               //呼叫自己(的一部分)
29 } // end function factorial
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 14197454024290336768
```

圖5.18　以遞迴函式計算階乘

5.15 使用遞迴的例子：Fibonacci費式數列

1. 費伯納斯數列 0, 1, 1, 2, 3, 5, 8, 13, 21,

- 從0,1開始，之後每個Fibonacci數，都是前兩個Fibonacci數之和。
  - 費伯那斯數列可透過遞迴方式定義，如下：

fibonacci(0) = 0
fibonacci(1) = 1 } // 1) 基本情況(base case)
fibonacci(*n*) = fibonacci(*n* -1) + fibonacci(*n*-2) // 2)遞迴規則

- 下圖程式利用函式**Fibonacci**，遞迴計算出第n個Fibonacci數：

程式碼

```
1  // Fig. 5.19: fig05_19.c
2  // Recursive fibonacci function
3  #include <stdio.h>
4
5  unsigned long long int fibonacci( unsigned int n ); // function
6                                                      //函式宣告
7  // function main begins program execution
8  int main( void )
9  {
10    unsigned long long int result; // fibonacci value
11    unsigned int number; // number input by user
12
13    // obtain integer from user
14    printf( "%s", "Enter an integer: " );
15    scanf( "%u", &number );
16
17    // calculate fibonacci value for number input by user
18    result = fibonacci( number ); //函式呼叫
19
20    // display result
21    printf( "Fibonacci( %u ) = %llu\n", number, result );
22 } // end main
23
24 // Recursive definition of function fibonacci
25 unsigned long long int fibonacci( unsigned int n )
26 {                                                  //函式定義
27    // base case
28    if ( 0 == n || 1 == n ) {
29       return n;              //初始項(定義n==0和n==1)
30    } // end if
31    else { // recursive step
32       return fibonacci( n - 1 ) + fibonacci( n - 2 );
33    } // end else              //遞迴項(描述n-1項)
34 } // end function fibonacci
```

圖5.19 遞迴產生Fibonacci數 //想法：f(n) = f(n-1) + f(n-2);

```
Enter an integer: 0
Fibonacci( 0 ) = 0
```

```
Enter an integer: 1
Fibonacci( 1 ) = 1
```

```
Enter an integer: 2
Fibonacci( 2 ) = 1
```

```
Enter an integer: 3
Fibonacci( 3 ) = 2
```

```
Enter an integer: 10
Fibonacci( 10 ) = 55
```

```
Enter an integer: 20
Fibonacci( 20 ) = 6765
```

```
Enter an integer: 30
Fibonacci( 30 ) = 832040
```

```
Enter an integer: 40
Fibonacci( 40 ) = 102334155
```

圖5.19　遞迴產生Fibonacci數

- 下圖示範**Fibonacci**函式如何計算**Fibonacci(3)**。

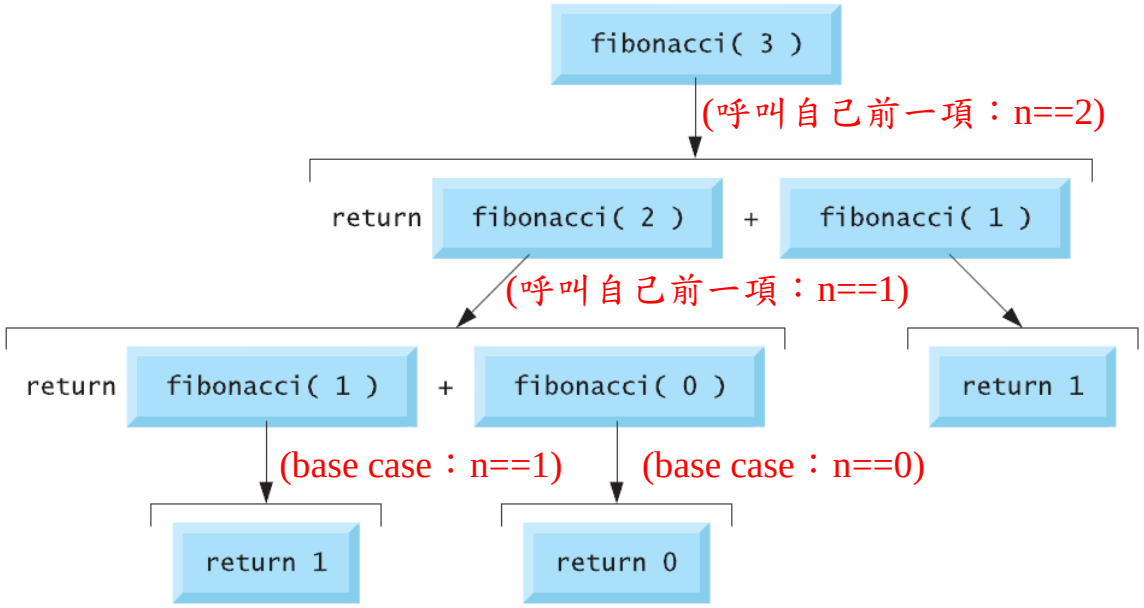


圖5.20　呼叫**fibonacci(3)**的遞迴呼叫

2. 指數複雜度(Exponential Complexity)

- 遞迴須注意其複雜度 (執行次數)。
  - 舉例來說：**fibonacci函式**中，每一次遞迴都會呼叫2次子遞迴，如下：

    ```
    return Fibonacci(n-1)+Fibonacci(n-2);
    ```

  - 當計算第n個**Fibonacci**數時，需要執行2^n次遞迴呼叫，當計算第20個**Fibonacci**數，便需要約2^{20}次函式呼叫，當計算第30個**Fibonacci**數，便需要約2^{30}次函式呼叫，也就是十億個函式呼叫。
- 這種現象稱為指數等級複雜度 (exponential complexity)
  - 因此，遞迴結構簡單，但需注意時間複雜度。

5.16 遞迴(Recursion) vs 迭代(Iteration)

- 一般程式可以用遞迴或迭代(迴圈)實作，以下將比較這兩種方法。
  - **兩者**皆以控制結構為基礎：迭代 - 使用迴圈結構；遞迴 - 使用選擇結構。
  - **兩者**皆含有重覆性：迭代 - 明確使用重複描述；遞迴 - 使用重複函式呼叫。
  - **兩者**皆含有終止檢測：迭代 - 在迴圈繼續條件不符時結束；遞迴 - 到達基本情況(base case)時結束。
  - 迭代 - 使用計數器控制重複結構；遞迴 - 逐步接近終止
  - 迭代 - 持續改變計數器的值，直到迴圈繼續條件不符為止；遞迴 - 持續將原問題簡化，直到問題縮小成基本情況(base case)為止。
  - **兩者**皆可能產生無窮迴圈：迭代 - 迴圈繼續條件永遠為真(True)時；遞迴 - 無法將問題收斂到基本情況(base case)時。
- 缺點方面：
  - 遞迴 - 重複呼叫函式，造成多餘執行時間和記憶體空間負擔，每次遞迴會產生函式變數，需要可觀的記憶體空間。
- 優點方面：
  - 遞迴 - 結構簡單，容易理解及debug；
  - 迭代 - 通常在同一個函式內進行，避免重複呼叫造成的額外記憶體負擔。

遞迴的例子和習題

第 5 章	第 7 章
Factorial 函式	迷宮尋訪
Fibonacci 函式	第 8 章
最大公因數	反向印出輸入的字串
乘以兩個整數	第 12 章
將整數升冪爲整數次冪	搜尋連結串列
河內塔	反向印出連結串列
遞迴 main	二元樹插入
視覺化的遞迴	二元樹的前序遍歷
第 6 章	二元樹的中序遍歷
加總一個陣列的元素	二元樹的後序遍歷
印出陣列	印出樹
印出反向陣列	附錄 D
印出反向字串	選擇排序
檢查字串是否爲回文	快速排序
陣列中的最小值	附錄 E
線性搜尋	Factorial 函式
二元搜尋	
八皇后問題	

圖5.21　本書中關於遞迴的範例

[工商服務]

FB分享：學長姐發展消息

Jia-Ming Liang 覺得開心——和 **Kris Hung** 及**其他 2 人**，在**桃園市桃園區**。

2020年5月8日・桃園市桃園區・

恭喜~😄🎉 語歆 (UC Irvine)、和寰 (Rice university)、德融 (NYU Tandon)~錄取美國研究所~!! 🎊🎊🎊

擔任你們的推薦人，老師也感到非常光榮哦~!! 希望未來能有機會回國貢獻所學~! 😀👍👍... 查看更多

May. 08. 2020

Jia-Ming Liang——和**蔡宛凌**及**其他 4 人**，在**聯華神通集團總部**。

4月16日・台北市・

恭喜 ~!! 宛凌、志豪、胤恩、香伶~ 🥳

我們專題「智慧水族箱」獲得 2021育秀盃 優選~! 🤩

#開心做專題
#學生棒棒... 查看更多

FB分享：學長姐專題競賽成果

Jia-Ming Liang——和**賴怡誠**及**其他 4 人**。

2020年11月6日 ·

恭喜~ 怡誠, 冠綸, 峻瑋, 淵翔~! 😀🎉🥳

我們專題『智慧行動商店服務』，獲得2020全國大專校院智慧創新競賽 ~ 第2名耶🏆~!! 🥰🤩

#謝謝評審
#獎金5萬元
#繼續努力
#遲來的祝福

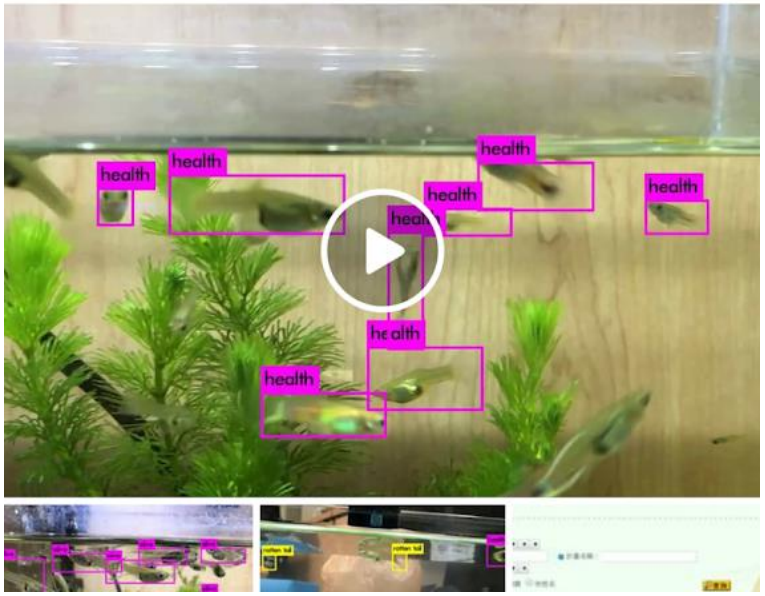
Jia-Ming Liang 覺得開心——和**蔡宛凌**及**其他 4 人**。

2020年6月21日 ·

恭喜~ 宛凌丶志豪丶香伶丶胤恩~! 😀🎉

我們的水寶貝~開心水族系統 🐟🐳🐡... 入選科技部大專生計畫~! 😊🥳

#養魚趣
#魚類健康辨識... 查看更多



【工商服務】加入電機系粉絲頁 (點擊加入)

- 國立臺南大學電機工程學系
- 最新消息(粉絲頁)
  - https://www.facebook.com/%E5%9C%8B%E7%AB%8B%E8%87%BA%E5%8D%97%E5%A4%A7%E5%AD%B8%E9%9B%BB%E6%A9%9F%E5%B7%A5%E7%A8%8B%E5%AD%B8%E7%B3%BB-206370602753119/

最新消息