Chapter 5 - Functions

Outline

Objectives

- In this chapter, you will learn:
  - To understand how to construct programs modularly from small pieces called functions..
  - To introduce the common math functions available in the C standard library.
  - To be able to create new functions.
  - To understand the mechanisms used to pass information between functions.
  - To introduce simulation techniques using random number generation.
  - To understand how to write and use functions that call themselves.

5.1 Introduction

```
/* 未用函數時的範例 */
#include <stdio.h>

int main()
{
   printf("**************\n"); /* 印出13個星號 */
   printf("歡迎使用C語言\n");
   printf("**************\n"); /* 印出13個星號 */

   return 0;
}
```

5.1 Introduction

```
/* 簡單的函數範例 */
#include <stdio.h>

void star(void);                    /* star()函數的原型 */

int main()
{
   star();                          /* 呼叫star函數 */
   printf("歡迎使用C語言\n");
   star();                          /* 呼叫star函數 */

   return 0;
}
```

注意後面必須加分號「;」
也可用 void star();


```
void star(void)
{
   printf("**********\n"); /* 印出13個星號 */
   return;
}
```

注意後面不可加分號「;」
也可用 void star()

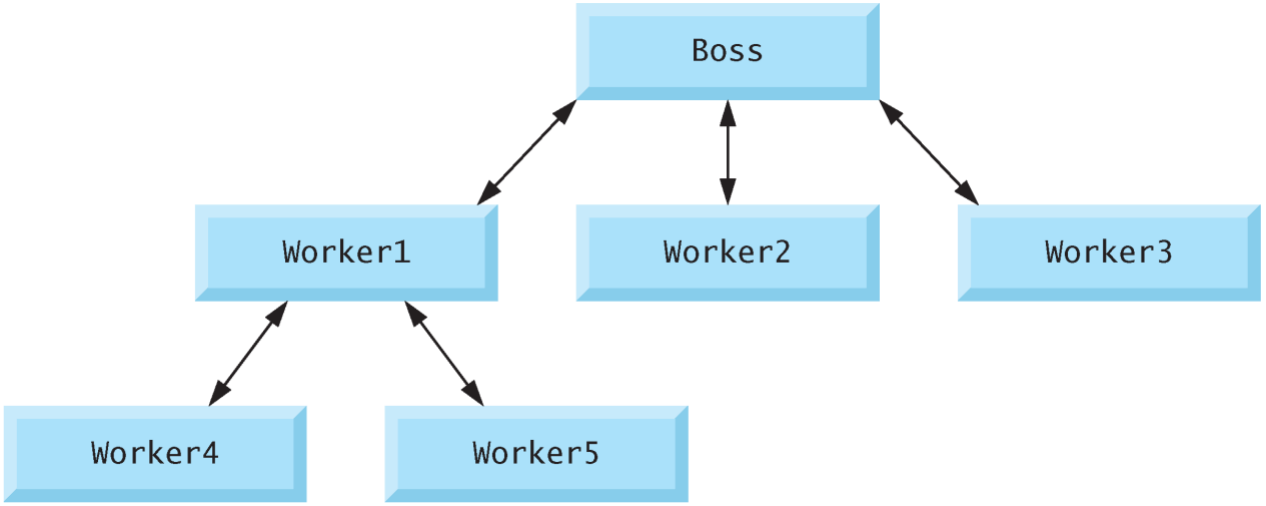
- Divide and conquer (各個擊破)
  - Construct a program from smaller pieces or components
    - These smaller pieces are called modules (模組)
  - Each piece more manageable than the original program

5.2 Program Modules in C

- Functions
  - Modules in C
  - Programs combine user-defined functions with library functions
    - C standard library has a wide variety of functions
- Function calls
  - Invoking functions
    - Provide function name (函數名稱) and arguments (data，引數、參數)
    - Function performs operations or manipulations
    - Function returns results (傳回值)
  - “Function call” analogy:
    - Boss asks worker to complete task
      - Worker gets information, does task, returns result
      - Information hiding: boss does not know details

5.2 Program Modules in C

Fig. 5.1 Hierarchical boss function/worker function relationship.



5.3 Math Library Functions

- Math library functions (內定的數學函式)
  - perform common mathematical calculations
  - `#include <math.h>`
- Format for calling functions
  - `FunctionName( argument );`
    - If multiple arguments, use comma-separated list
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Calls function `sqrt`, which returns the square root of its argument
    - All math functions return data type `double`
  - Arguments may be constants, variables, or expressions

5.3 Math Library Functions

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`exp( x )`	exponential function e^x	`exp( 1.0 )` is `2.718282` `exp( 2.0 )` is `7.389056`
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`

`log10( x )`	logarithm of x (base 10)	`log10( 1.0 )` is `0.0`
		`log10( 10.0 )` is `1.0`
		`log10( 100.0 )` is `2.0`

`fabs( x )`	absolute value of x	`fabs( 13.5 )` is `13.5`
		`fabs( 0.0 )` is `0.0`
		`fabs( -13.5 )` is `13.5`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`

Fig. 5.2 | Commonly used math library functions. (Part 1 of 2.)

5.3 Math Library Functions

Function	Description	Example
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 )` is `128.0` `pow( 9, .5 )` is `3.0`
`fmod( x, y )`	remainder of x/y as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is `0.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is `1.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is `0.0`

Fig. 5.2 | Commonly used math library functions. (Part 2 of 2.)

5.4 Functions

- Functions
  - Modularize a program
  - All variables defined inside functions are local variables (區域變數)
    - Known only in function defined
  - Parameters
    - Communicate information between functions
    - Local variables
- Benefits of functions
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoid code repetition

5.5 Function Definitions

- Function definition format

  return-value-type function-name(parameter-list)

  ```
  {
    declarations and statements
  }
  ```

  – Example:

  ```
  void star(void)
  {
     printf("**********\n"); /* 印出13個星號 */
     return;
  }
  ```

  – Function-name: any valid identifier

  – Return-value-type: data type of the result (default `int`)

  - `void` – indicates that the function returns nothing

  – Parameter-list: comma separated list, declares parameters

  - A type must be listed explicitly for each parameter unless, the parameter is of type `int`

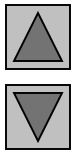
5.5 Function Definitions

- Function definition format (continued)

  return-value-type function-name(parameter-list)
  {
  declarations and statements
  }

  - Definitions and statements: function body (block)
    - Variables can be defined inside blocks (can be nested)
    - Functions can not be defined inside other functions
  - Returning control
    - If nothing returned
      - `return;`
      - or, until reaches right brace
    - If something returned
      - `return` *expression*`;`

Programmer-Defined **square** function



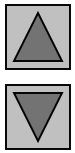
Outline

fig05_03.c (Part 1 of 2)

```
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
7  /* function main begins program execution */
8  int main()
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 */
16
17    printf( "\n" );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
```

自訂函數時，必須要把函數的原型放到主函數前

呼叫函數 **square** ，把 **x** 的值利用函數的 argument 傳到函數中。



Outline

```
23 /* square function definition returns square of an integer */
24 int square( int y ) /* y is a copy of argument to function */
25 {
26    return y * y; /* returns square of y as an int */
27
28 } /* end function square */
```

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

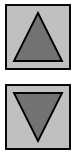
fig05_03.c (Part 2 of 2)

Function `square`

Program Output

本函數接受經由 argument 傳進來的數值 (**`int y`**) 然後把計算完的結果 (**`y*y`**) 經由函數名稱 **`square`** (整數型態) 傳回。

Programmer-Defined **maximum** function



Outline

```
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
7  /* function main begins program execution */
8  int main()
9  {
10    int number1; /* first integer */
11    int number2; /* second integer */
12    int number3; /* third integer */
13
14    printf( "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
21    return 0; /* indicates successful termination */
22
23 } /* end main */
24
```

fig05_04.c (Part 1 of 2)

此處亦可使用
`int maximum( int, int, int );`

```
25 /* Function maximum definition */
26 /* x, y and z are parameters */
27 int maximum( int x, int y, int z )
28 {
29    int max = x;      /* assume x is largest */
30
31    if ( y > max ) { /* if y is larger than max, assign y to max */
32       max = y;
33    } /* end if */
34
35    if ( z > max ) { /* if z is larger than max, assign z to max */
36       max = z;
37    } /* end if */
38
39    return max;       /* max is largest value */
40
41 } /* end function maximum */
```

x, y, z 已在此宣告為整數，function 中區域內不必再宣告！

把 max 的值藉由函數名稱 maximum 傳回主程式

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

Outline

fig05_04.c (Part 2 of 2)

Function `maxium`

Program Output

5.6 Function Prototypes

- Function prototype (函數原型)
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default **`int`**)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program (習慣上副程式(函數)是放在主程式後，如果把副函數放到主程式前，就不用在主程式前加 prototype，不過在實務上還是先放主程式並在主程式前放副函數的 function prototype 為宜，尤其當主程式和副函數在不同的檔案時必須放 prototype)
  - The function with the prototype

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

    - Takes in 3 `int`s
    - Returns an `int`
- Promotion rules and conversions
  - Converting lower types to higher type in the expression
    - example: sqrt(4) ⇒ sqrt(4.0)
  - Converting to lower types can lead to errors

5.6 Function Prototypes

Data type	printf conversion specification	scanf conversion specification
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
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

Fig. 5.5 | Promotion hierarchy for data types.

Typical Size and Range of Data Types

For Borland Compiler

Data Type	Size Bytes	Min Value	Max Value
char	1	-128	127
unsigned short int	2	0	65535
short int	2	-32768	32767
unsigned int	4	0	4294967295
int	4	-2147483648	2147483647
unsigned long int	4	0	4294967295
long int	4	-2147483648	2147483647
float	4	1.17549e-38	3.40282e+38
double	8	2.22507e-308	1.79769e+308
long double	10	3.3621e-4932	1.18973e+4932

For Visual C++ and C Compiler

Data Type	Size Bytes	Min Value	Max Value
char	1	-128	127
unsigned short int	2	0	65535
short int	2	-32768	32767
unsigned int	4	0	4294967295
int	4	-2147483648	2147483647
unsigned long int	4	0	4294967295
long int	4	-2147483648	2147483647
float	4	1.17549e-038	3.40282e+038
double	8	2.22507e-308	1.79769e+308
long double	8	2.22507e-308	1.79769e+308

5.7 Header Files

- Header files
  - Contain function prototypes for library functions
  - `<stdlib.h>` , `<math.h>` , etc
  - Load with **`#include <filename>`**

    `#include <math.h>`
- Custom header files
  - Create file with functions
  - Save as `filename.h`
  - Load in other files with **`#include "filename.h"`**
  - Reuse functions

5.7 Header Files

Header	Explanation
`<assert.h>`	Contains macros and information for adding diagnostics that aid program debugging.
`<ctype.h>`	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
`<errno.h>`	Defines macros that are useful for reporting error conditions.
`<float.h>`	Contains the floating-point size limits of the system.
`<limits.h>`	Contains the integral size limits of the system.
`<locale.h>`	Contains function prototypes and other information that enables a program to be modified for the current locale on which it's running. The notion of locale enables the computer system to handle different conventions for expressing data like dates, times, dollar amounts and large numbers throughout the world.
`<math.h>`	Contains function prototypes for math library functions.
`<setjmp.h>`	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.

Fig. 5.6 | Some of the standard library headers. (Part 1 of 2.)

5.7 Header Files

Header	Explanation
`<signal.h>`	Contains function prototypes and macros to handle various conditions that may arise during program execution.
`<stdarg.h>`	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
`<stddef.h>`	Contains common type definitions used by C for performing calculations.
`<stdio.h>`	Contains function prototypes for the standard input/output library functions, and information used by them.
`<stdlib.h>`	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers, and other utility functions.
`<string.h>`	Contains function prototypes for string-processing functions.
`<time.h>`	Contains function prototypes and types for manipulating the time and date.

Fig. 5.6 | Some of the standard library headers. (Part 2 of 2.)

5.8 Calling Functions: Call by Value and Call by Reference

- Call by value (傳值呼叫)
  - Copy of argument passed to function
  - Changes in function do not effect original (不會改變傳進來的變數值)
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference or Call by address(傳址呼叫)
  - Passes original argument
  - Changes in function effect original (會改變傳進來的變數值)
  - Only used with trusted functions
  - Will be discussed in chapters 6 and 7
- For now, we focus on call by value

Call by Value - Example

```
/* 函數的傳值機制 借自 C 語言教學手冊*/

#include <stdio.h>

void add10(int,int);          /* add10()的原型 */

int main()
{
   int a = 3, b = 5;          /* 宣告區域變數a與b */

   printf ( "呼叫函數add10()之前: " );
   printf ( "a = %2d, b = %2d\n", a, b ); /* 印出a、b的值 */

   add10(a,b);

   printf ( "呼叫函數add10()之後: " );
   printf ( "a = %2d, b = %2d\n", a, b ); /* 印出a、b的值 */

   return 0;
}

void add10(int a,int b)
{
   a = a + 10;                /* 將變數a的值加10之後，設回給a */
   b = b + 10;                /* 將變數b的值加10之後，設回給b */

   printf ( "函數 add10 中 :      " );
   printf ( "a = %2d, b = %2d\n", a, b ); /* 印出a、b的值 */
}
```

```
呼叫函數add10()之前: a =  3, b =  5
函數 add10 中 :      a = 13, b = 15
呼叫函數add10()之後: a =  3, b =  5
```

5.9 Random Number Generation

- Lottery, Card Games, Dice, 抽籤
- `rand` function
  - Load `<stdlib.h>`, i.e.,

```
#include <stdlib.h>
. . .
i = rand();
```

  - Returns "random" number between `0` and `RAND_MAX` (at least `32767`), i.e., `0 ≤ rand() ≤ RAND_MAX`
  - Pseudorandom
    - Preset sequence of "random" numbers
    - Same sequence for every function call
- Scaling
  - To get a random number between `1` and `n`

```
1 + ( rand() % n )
```

    - `rand() % n` returns a number between `0` and `n - 1`, where `%` is the remainder operator
    - Add `1` to make random number between `1` and `n`

```
1 + ( rand() % 6)
```

      - Generates a number between `1` and `6`

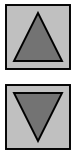
Shifted, Scaled Random Integers Produced by `1 + rand() % 6`

fig05_07.c

```
1  /* Fig. 5.7: fig05_07.c
2     Shifted, scaled integers produced by 1 + rand() % 6 */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int i; /* counter */
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          printf( "\n" );
20       } /* end if */
21
22    } /* end for */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

計算亂數時必須加上此段！

在 1～6 中任取一數時使用
`1 + (rand() % 6)`



Outline

Program Output

第一次執行：

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

再執行一次：

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

再一次：

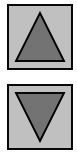
```
6     6     5     5     6
5     1     1     5     3
6     6     2     4     2
6     2     3     4     1
```

5.9 Random Number Generation

- `rand` function
  - rand generates Pseudo-random numbers
    - Preset sequence of "random" numbers
    - Same sequence for every function call
- `srand` function
  - Again, need `#include <stdlib.h>`
  - Takes an integer **`seed`** and jumps to that location in its "random" sequence

    `srand(` *seed* **`);`**
  - **`rand`** 會回傳一個亂數整數，但 **`srand`** 不會回傳任何數值，它的功能只是接收一個種子，然後再呼叫 **`rand`** 時不會重複產生相同的亂數。
  - `srand( time( NULL ) );/*load <time.h> */`
    - `time( NULL )`
      - Returns the time at which the program was executed in seconds
      - “Randomizes" the seed

Randomizing the Die-Rolling



Outline

fig05_09.c (Part 1 of 2)

```
1  /* Fig. 5.9: fig05_09.c
2     Randomizing die-rolling program */
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int i;          /* counter */
10    unsigned seed; /* number used to seed random number generator */
11
12    printf( "Enter seed: " );
13    scanf( "%u", &seed );
14
15    srand( seed ); /* seed 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 outpu
21       printf( "%10d", 1 + ( rand() % 6 ) );
22
```

使用種子 **seed** 可產生不同的亂數系列，**seed** 為 **unsigned integer**, 故先宣告

unsigned seed;

或

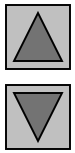
unsigned int seed;

本例是用手動輸入種子，由於 **seed** 是 **unsigned integer** ，需用 **%u** 讀入。

再利用 **srand(seed)** 函數，根據不同的種子產生不同的亂數系列；或用

srand(time(NULL));

利用系統時間當成種子產生不同的亂數系列



Outline

```
23       /* if counter is divisible by 5, begin a new line of output */
24       if ( i % 5 == 0 ) {
25          printf( "\n" );
26       } /* end if */
27
28    } /* end for */
29
30    return 0; /* indicates successful termination */
31
32 } /* end main */
```

fig05_09.c (Part 2 of 2)

Program Output

```
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
```

若要用 time(NULL) 產生亂數種子，要如何修改程式？

利用系統時間產生不同的亂數系列

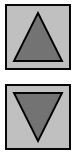
```
/* Fig. 5.9: fig05_09M.c
   Randomizing die-rolling program
   with time(NULL)                    */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>

/* function main begins program execution */
int main()
{
   int i;           /* counter */

   srand( time(NULL) ); /* seed random number generator */

   /* loop 10 times */
   for ( i = 1; i <= 10; i++ ) {

      /* pick a random number from 1
      printf( "%10d", 1 + ( rand() %

      /* if counter is divisible by
      if ( i % 5 == 0 ) {
         printf( "\n" );
      } /* end if */

   } /* end for */

   return 0; /* indicates successful

} /* end main */
```

使用系統時間時記得要加入標頭檔
#include <time.h>

利用 **srand(seed)** 產生不同的亂數系列，此處的種子為系統時間

```
M:\>
M:\>fig05_09M
         3         4         6         3         2
         4         1         4         1         4

M:\>fig05_09M
         6         1         2         4         6
         5         6         5         6         2

M:\>fig05_09M
         3         1         2         1         1
         1         4         6         4         4

M:\>fig05_09M
         4         6         5         4         6
         1         5         1         1         2
```

Rolling a Six-Sided Die 6000 Times



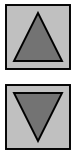
Outline

fig05_08.c (Part 1 of 3)

```c
/* Fig. 5.8: fig05_08.c
   Roll a six-sided die 6000 times */
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main()
{
   int frequency1 = 0; /* rolled 1 counter */
   int frequency2 = 0; /* rolled 2 counter */
   int frequency3 = 0; /* rolled 3 counter */
   int frequency4 = 0; /* rolled 4 counter */
   int frequency5 = 0; /* rolled 5 counter */
   int frequency6 = 0; /* rolled 6 counter */

   int roll; /* roll counter */
   int face; /* represents one roll of the die, value 1 to 6 */

   /* loop 6000 times and summarize results */
   for ( roll = 1; roll <= 6000; roll++ ) {
      face = 1 + rand() % 6; /* random number from 1 to 6 */

```

此段為宣告及定義
分別出現1~6點時之計數器

骰面數字為1~6之間的亂數



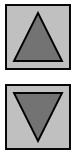
Outline

fig05_08.c (Part 2 of 3)

```
23    /* determine face value and increment appropriate counter */
24    switch ( face ) {
25
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;
49    } /* end switch */
50
```

此處 value 為數值的話，所以用 case 1 其中的 1 為 value 的數值。注意沒有引號 (' ')！

根據出現不同的點數增加個別點數之計數器



Outline

fig05_08.c (Part 3 of 3)

```
50
51    } /* end for */
52
53    /* display results in tabular format */
54    printf( "%s%13s\n", "Face", "Frequency" );
55    printf( "    1%13d\n", frequency1 );
56    printf( "    2%13d\n", frequency2 );
57    printf( "    3%13d\n", frequency3 );
58    printf( "    4%13d\n", frequency4 );
59    printf( "    5%13d\n", frequency5 );
60    printf( "    6%13d\n", frequency6 );
61
62    return 0; /* indicates successful termination */
63
64 } /* end main */
```

本程式每次執行的結果會不會一樣？如果一樣，要如何修改？

Program Output

```
Face      Frequency
   1           1003
   2           1017
   3            983
   4            994
   5           1004
   6            999
```

5.10 虛擬賭博遊戲 Example: A Game of Chance

- Craps simulator (擲雙骰子遊戲)
- Rules
  - Roll two dice
    - 7 or 11 on first throw, player wins
    - 2, 3, or 12 on first throw, player loses
    - 4, 5, 6, 8, 9, 10 - value becomes player's "point"
  - Player must roll his point before rolling 7 to win

5.10 虛擬賭博遊戲 Example: A Game of Chance

雙骰子（Craps）

2 點、3 點、12 點稱為 Craps
7 點跟 11 點稱為 Natural

第一次骰出 Natural 就贏了
第一次骰出 Craps 就輸了
骰出 4, 5, 6, 8, 9, 10，那莊家就會在那個數字放一個記號，成為骰點（Shooter's Point, or Player’s Point）

接著，
骰家在骰出 7 之前，骰出骰點，就贏
骰家在骰出 7 之前，沒骰出骰點，就輸；也就是說，骰到 7 就輸了。

補充 - Enumeration (列舉型態)

- 列舉型態是一種特殊的常數定義方式，利用列舉型態的宣告，即可利用某些有意義的名稱取代不易記憶的整數常數，讓程式的可讀性更高。
- 例如：不同的顏色用 {0, 1, 2, 3, 4, . . . } 或 {red，orange，yellow，green，blue，. . . }，哪種比較明瞭？
- 例子：

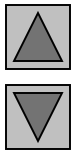
```
enum color {red, blue, green};
enum color shirt, hat;
```

- 第一段是定義列舉型態，其中 **color** 是列舉型態名稱，**red**, **blue** 及 **green** 則為三個列舉常數；第二段是宣告 **shirt** 及 **hat** 這兩個變數是屬於列舉型態 **color** 的變數。
- 在沒有特別指定時，列舉常數 **red** 的值會被設為 0 ， **blue** 的值會被設為 1 ，**green** 的值會被設為 2 ，而且這些值不能夠再更改。
- 列舉常數的值不一定要從 0 開始；下面一些其他例子：

```
enum color {red = 5, green, blue};     //此處 green = 6 及 blue = 7
enum color {red = 10, green = 20, blue = 30};
```

- 所謂常數，就是它的值在程式中不可再改變；也就是說，前例中 **red**, **green**, 及 **blue** 的值是固定的。
- 列舉型態和 C 語言中的 **struct** (結構) 很類似，本課要在講解 **struct** 時 (Chatper 10)才會詳細介紹。

Craps Simulator



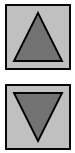
Outline

fig05_10.c (Part 1 of 4)

```
1  /* Fig. 5.10: fig05_10.c
2     Craps */
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 };
9
10 int rollDice( void ); /* function prototype */
11
12 /* function main begins program execution */
13 int main()
14 {
15    int sum;        /* sum of rolled dice */
16    int myPoint;    /* point earned */
17
18    enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */
19
20    /* randomize random number generator using current time */
21    srand( time( NULL ) );
22
23    sum = rollDice(  ); /* first roll of the dice */
24
```

宣告 **Status** 為 enumeration（列舉型態），其成員包括**CONTINUE, WON, LOST** 三個列舉常數（enumerated constant）；通常C語言會自動給列舉常數一個整數，第一個列舉常數的值為 0，第二個列舉常數的值為 1，以此類推。

宣告**gameStatus** 為列舉變數，其值為列舉型態 **Status** 中某一個常數（就是**CONTINUE, WON, LOST** 三個中的一個）



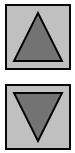
Outline

fig05_10.c (Part 2 of 4)

```
25  /* determine game status based on sum of dice */
26  switch( sum ) {
27
28     /* win on first roll */
29     case 7:
30     case 11:
31        gameStatus = WON;
32        break;
33
34     /* lose on first roll */
35     case 2:
36     case 3:
37     case 12:
38        gameStatus = LOST;
39        break;
40
41     /* remember point */
42     default:
43        gameStatus = CONTINUE;
44        myPoint = sum;
45        printf( "Point is %d\n", myPoint );
46        break; /* optional */
47  } /* end switch */
48
```

Rules:

1. Roll two dice
2. 7 or 11 on first throw, player wins
3. 2, 3, or 12 on first throw, player loses
4. 4, 5, 6, 8, 9, 10 - value becomes player's "point". Player must roll his point before rolling 7 to win

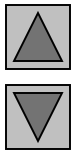


Outline

fig05_10.c (Part 3 of 4)

```
49    /* while game not complete */
50    while ( gameStatus == CONTINUE ) {
51       sum = rollDice(  ); /* roll dice again */
52
53       /* determine game status */
54       if ( sum == myPoint ) { /* win by making point */
55          gameStatus = WON;
56       } /* end if */
57       else {
58
59          if ( sum == 7 ) { /* lose by rolling 7 */
60             gameStatus = LOST;
61          } /* end if */
62
63       } /* end else */
64
65    } /* end while */
66
67    /* display won or lost message */
68    if ( gameStatus == WON ) {
69       printf( "Player wins\n" );
70    } /* end if */
71    else {
72       printf( "Player loses\n" );
73    } /* end else */
74
75    return 0; /* indicates successful termination */
76
77 } /* end main */
```

Rules: (while "CONTIUNE")
4, 5, 6, 8, 9, 10 - value becomes player's "point".
Player must roll his point before rolling 7 to win

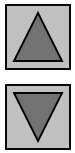


Outline

fig05_10.c (Part 4 of 4)

```
78
79 /* roll dice, calculate sum and display results */
80 int rollDice( void )
81 {
82    int die1;    /* first die */
83    int die2;    /* second die */
84    int workSum; /* sum of dice */
85
86    die1 = 1 + ( rand() % 6 ); /* pick random die1 value */
87    die2 = 1 + ( rand() % 6 ); /* pick random die2 value */
88    workSum = die1 + die2;     /* sum die1 and die2 */
89
90    /* display results of this roll */
91    printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
92
93    return workSum; /* return sum of dice */
94
95 } /* end function rollRice */
```

主要的副程式，擲兩個骰子後
輸出點數總和



Outline

Program Output

```
Player rolled 5 + 6 = 11
Player wins

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

Player rolled 1 + 1 = 2
Player loses

Player rolled 1 + 4 = 5
Point is 5
Player rolled 3 + 4 = 7
Player loses
```

5.11 Storage Classes (儲存類別)

- Storage Class Specifiers
  - Identifier (識別字) – Variables, Function names
  - Attributes (屬性) of Variables: name, type, size and value
  - Other Attributes of Identifiers: storage class (儲存類別), storage duration (儲存時間長短), scope (範圍) and linkage (連結)
  - Storage classes – **auto, register, extern and static**
  - Storage duration – how long an object exists in memory (大多情況下不需考慮)
  - Scope (範圍) – where object can be referenced in program
  - Linkage – specifies the files in which an identifier is known (在多檔案程式情況下才需考慮，more in Chapter 14)
  - The four storage class specifiers can be split into two storage durations: automatic storage duration and static storage duration.

5.11 Storage Classes

- Automatic Storage
  - Object created and destroyed within its block
  - `auto`: default for local variables
    - Example: `auto double x, y;`
    - Local variables have automatic storage duration by default, so keyword `auto` is rarely used.
  - `register`: tries to put variable into high-speed registers
    - Can be used for automatic variables only
    - Often, register declarations are unnecessary
    - Example: `register int counter = 1;`
  - 通常 auto 及 register 這兩個都不常使用；現代的編譯器會自動判斷把變數放到適當的記憶體中，不再需要人為的操控。
- Static Storage
  - Variables exist for entire program execution
  - Default value of zero
  - `static`: local variables defined in functions.
    - Value is kept after function ends（函數執行完後，變數的值會保留下來）
    - Only known in their own function
  - `extern`: default for global variables and functions
    - Known in any function

補充：區域、全域與靜態變數

- 區域變數 (Local Variables)
  - 沒特別指定的變數皆為區域變數，區域變數的活動範圍 (scope) 只在函數範圍內。
- 全域變數 (Global Variables)
  - 如果把變數定義在函數的外面，就成全域變數；當變數為全域變數時，每一個函數及程式區塊皆可使用這個變數。
- 靜態變數 (Static Variables)
  - 靜態變數也在函數內部宣告，但在程式編譯時已配置固定記憶體空間，故靜態變數的值可以保存下來，靜態變數是以 static 宣告。

Call by Value – Example of 區域變數

```
/* 函數的傳值機制 借自 C 語言教學手冊*/

#include <stdio.h>

void add10(int,int);          /* add10()的原型 */

int main()
{
   int a = 3, b = 5;          /* 宣告區域變數a與b */

   printf ( "呼叫函數add10()之前: " );
   printf ( "a = %2d, b = %2d\n", a, b ); /* 印出a、b的值 */

   add10(a,b);

   printf ( "呼叫函數add10()之後: " );
   printf ( "a = %2d, b = %2d\n", a, b ); /* 印出a、b的值 */

   return 0;
}

void add10(int a,int b)
{
   a = a + 10;                /* 將變數a的值加10之後，設回給a */
   b = b + 10;                /* 將變數b的值加10之後，設回給b */

   printf ( "函數 add10 中 :      " );
   printf ( "a = %2d, b = %2d\n", a, b ); /* 印出a、b的值 */
}
```

```
呼叫函數add10()之前: a =  3, b =  5
函數 add10 中 :      a = 13, b = 15
呼叫函數add10()之後: a =  3, b =  5
```

在 add10 函式中a 及 b這兩個變數與主程式中 a, b 所佔的記憶體位置不同。

Example of Global Variables - 1

```
/* 全域變數的範例(一) */

#include <stdio.h>

void func();                          /* 函數func()的原型 */
int a;                                /* 宣告全域變數a */

int main()
{
   a = 100;                           /* 設定全域變數a的值
   printf("呼叫func()之前,a=%d\n",a);
   func();                            /* 呼叫自訂的函數 */
   printf("呼叫func()之後,a=%d\n",a);

   return 0;
}

void func(void)                       /* 自訂的函數func() */
{
   a = 300;                           /* 設定全域變數a的值為300 */
   printf("於func()函數裡,a=%d\n",a);
}
```

由於 a 為全域變數，此三處的 a 在記憶體中位置相同，數值當然也一樣。

```
呼叫func()之前,a=100
於func()函數裡,a=300
呼叫func()之後,a=300
```

Example of Global Variables - 2

```
/* 全域變數的範例(二) */

#include <stdio.h>

void func();
int a = 50;                          /* 定義全域變數a */

int main()
{
   int a = 100;                      /* 定義區域變數a */

   printf("呼叫func()之前,a=%d\n",a);
   func();                           /* 呼叫自訂的函數 */
   printf("呼叫func()之後,a=%d\n",a);

   return 0;
}

void func(void)
{
   a = a + 300;                      /* 這是全域變數a */
   printf("於func()函數裡,a=%d\n",a);
}
```

此處 a 又宣告一次，變成主程式之區域變數

```
呼叫func()之前,a=100
於func()函數裡,a=350
呼叫func()之後,a=100
```

注意在 function 中並未再宣告變數 a ， 所以在此處 a 為全域變數。

Example of Static Variables

```
/* 區域靜態變數使用的範例 */

#include <stdio.h>

void func();              /* 宣告func()函數的原型 */

int main()
{
   func();                /* 呼叫函數func() */
   func();                /* 呼叫函數func() */
   func();                /* 呼叫函數func() */

   return 0;
}

void func(void)
{
   static int a = 100;                    /* 宣告靜態變數a */
   printf("In func(),a = %d\n",a);        /* 印出func()函數中a的值 */
   a = a + 200;
}
```

```
In func(),a = 100
In func(),a = 300
In func(),a = 500
```

此處 a 為靜態變數，一旦宣告後，即使離開 func 函式，在記憶體中 a 所佔的位置仍會保留，所以在函式 func 中 a 的值會一直保存。

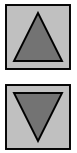
5.12 Scope Rules (範 圍 規 則)

- File scope
  - Identifier defined outside function, known in all functions
  - Used for global variables, function definitions, function prototypes
- Function scope (暫時不討論)
  - Can only be referenced inside a function body
  - Used only for labels (`start:`, `case:`, etc.)

5.12 Scope Rules

- Block scope
  - Identifier declared inside a block { . . . }
    - Block scope begins at definition, ends at right brace
  - Used for variables, function parameters (local variables of function)
  - Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block
- Function prototype scope
  - Used for identifiers in parameter list
  - e.g., `int maximum( int x, int y, int z );  /*function prototype */`，亦可寫為 `int maximum( int, int, int );`

Scoping Example（變數的範圍）



Outline

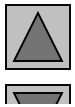
fig05_12.c (Part 1 of 3)

```
1  /* Fig. 5.12: fig05_12.c
2     A scoping example */
3  #include <stdio.h>
4
5  void useLocal( void );       /* function prototype */
6  void useStaticLocal( void ); /* function prototype */
7  void useGlobal( void );      /* function prototype */
8
9  int x = 1; /* global variable */
10
11 /* function main begins program execution */
12 int main()
13 {
14    int x = 5; /* local variable to main */
15
16    printf("local x in outer scope of main is %d\n", x );
17
18    { /* start new scope */
19       int x = 7; /* local variable to new scope */
20
21       printf( "local x in inner scope of main is %d\n", x );
22    } /* end new scope */
23
24    printf( "local x in outer scope of main is %d\n", x );
25
```

在此先定義 x = 1 為全域變數

定義 x = 5 為主程式 main 中之區域變數

定義 x = 7 為此區塊內之區域變數

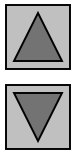


Outline

fig05_12.c (Part 2 of 3)

```
26    useLocal();       /* useLocal has automatic local x */
27    useStaticLocal(); /* useStaticLocal has static local x */
28    useGlobal();      /* useGlobal uses global x */
29    useLocal();       /* useLocal reinitializes automatic local x */
30    useStaticLocal(); /* static local x retains its prior value */
31    useGlobal();      /* global x also retains its value */
32
33    printf( "local x in main is %d\n", x );
34
35    return 0; /* indicates successful termination */
36
37 } /* end main */
38
39 /* useLocal reinitializes local variable x during each call */
40 void useLocal( void )
41 {
42    int x = 25;  /* initialized each time useLocal is called */
43
44    printf( "\nlocal x in a is %d after entering a\n", x );
45    x++;
46    printf( "local x in a is %d before exiting a\n", x );
47 } /* end function useLocal */
48
```

定義 x 為程式 useLocal 中之區域變數



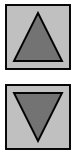
Outline

fig05_12.c (Part 3 of 3)

```
49 /* useStaticLocal initializes static local variable x only the first time
50    the function is called; value of x is saved between calls to this
51    function */
52 void useStaticLocal( void )
53 {
54    /* initialized only first time useStaticLocal is called */
55    static int x = 50;
56
57    printf( "\nlocal static x is %d on entering b\n", x );
58    x++;
59    printf( "local static x is %d on exiting b\n", x );
60 } /* end function useStaticLocal */
61
62 /* function useGlobal modifies global variable x during each call */
63 void useGlobal( void )
64 {
65    printf( "\nglobal x is %d on entering c\n", x );
66    x *= 10;
67    printf( "global x is %d on exiting c\n", x );
68 } /* end function useGlobal */
```

定義 x 為程式 useStaticLocal 中之靜態變數

此處並未定義 x ，故 x 在此處為全域變數，改變 x 則全域變數 x 隨之改變



Program Output

```
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 a is 25 after entering a
local x in a is 26 before exiting a

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

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

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

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

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

5.13 Recursion（遞迴）

- Recursive functions（遞迴函數）
  - Functions that call themselves
  - Can only solve a base case (i.e., the simplest case)
  - Divide a problem into
    - What it can do (the base case)
    - What it cannot do
      - What it cannot do resembles original problem
      - The function launches a new copy of itself (recursion step) to solve what it cannot do
  - Eventually base case gets solved
    - Gets plugged in, works its way up and solves whole problem

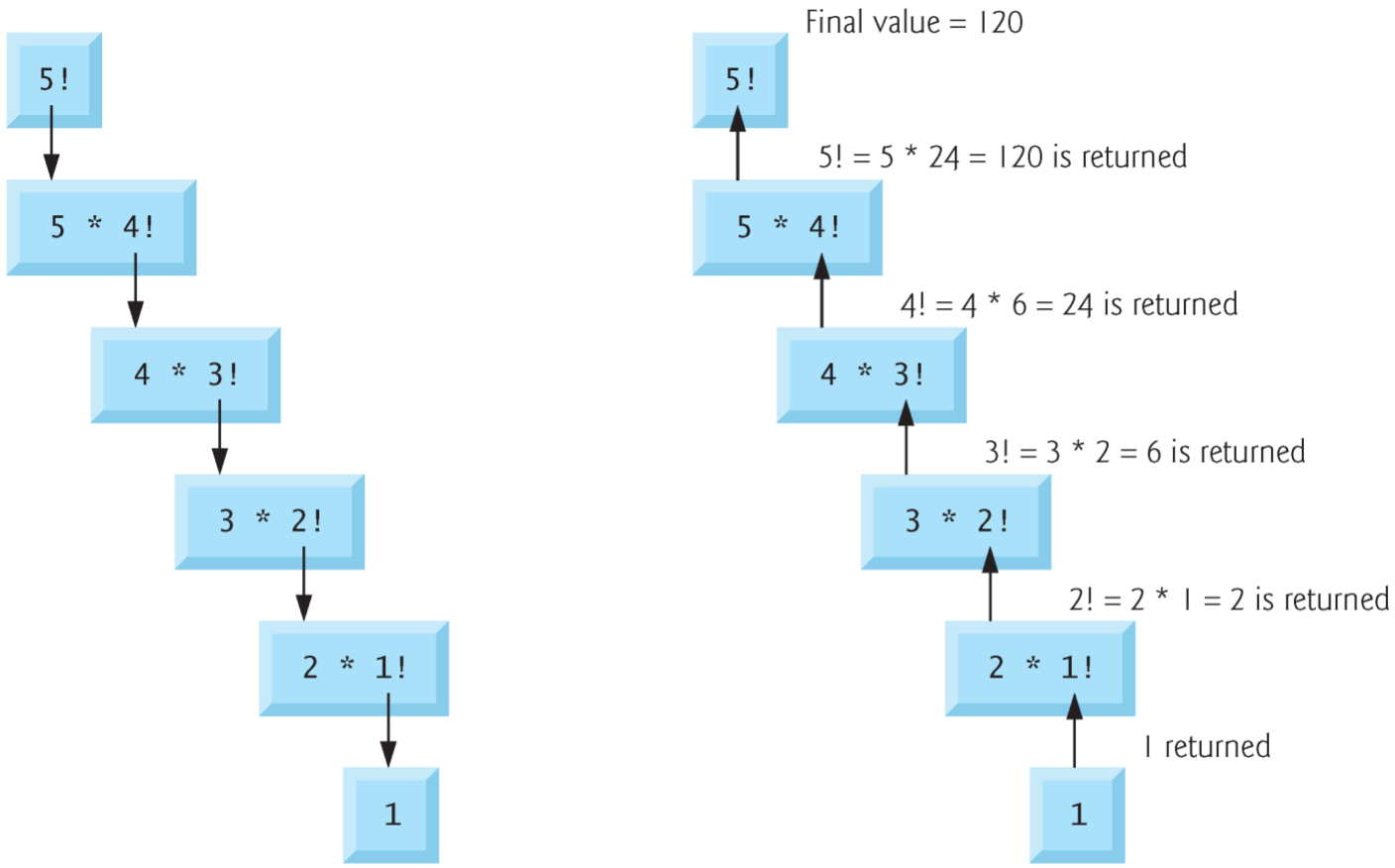
5.13 Recursion

- Example: factorials
  - 5! = 5 * 4 * 3 * 2 * 1
  - n! = n * (n – 1) * (n – 2) * . . . * 1
  - Can be calculated *iteratively* using **for** statement:

```
factorial = 1;
for ( counter = n; counter >= 1; counter-- )
   factorial = factorial * counter;
```

  - Notice that
    - 5! = 5 * 4!
    - 4! = 4 * 3! ...
  - Can compute factorials recursively
  - Solve base case (1! = 0! = 1) then plug in
    - 2! = 2 * 1! = 2 * 1 = 2;
    - 3! = 3 * 2! = 3 * 2 = 6;

5.13 Recursion



(a) Sequence of recursive calls.

(b) Values returned from each recursive call.

Fig. 5.13 | Recursive evaluation of 5!.

5.13 The Factorial, n!

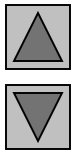
$$fac(n) = \begin{cases} 1 & ; \quad n = 0 \\ n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1 & ; \quad n \geq 1 \end{cases}$$

$$fac(n) = n \times \underbrace{(n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1}_{fac(n-1)} = n \times fac(n-1)$$

$$fac(n) = \begin{cases} 1 & ; \quad n = 0 \\ n \times fac(n-1) & ; \quad n \geq 1 \end{cases}$$

```
int fac( int n )     /* 自訂函數 fac()，計算 n! */
{
   if ( n == 0 )
      return 1;
   else
      return (n*fac(n-1));
}
```

Calculating Factorials with a recursive Function



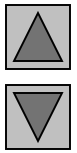
Outline

```
1  /* Fig. 5.14: fig05_14.c
2     Recursive factorial function */
3  #include <stdio.h>
4
5  long factorial( long number ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    int i; /* counter */
11
12    /* loop 10 times.  During each iteration, calculate
13       factorial( i ) and display result */
14    for ( i = 1; i <= 10; i++ ) {
15       printf( "%2d! = %ld\n", i, factorial( i ) );
16    } /* end for */
17
18    return 0; /* indicates successful term
19
20 } /* end main */
21
```

fig05_14.c (Part 1 of 2)

long 也是一種整數型態，在 visual C 中，long 和 int 完全一樣，佔 4 個 bytes ，數值介於 - 2,147,483,648 與 2,147,483,647 之間。

由於 factorial 是 long (長整數) 型態，利用 printf 或 scanf 輸出或輸入時需用 %ld 。



Outline

fig05_14.c (Part 2 of 2)

```
22 /* recursive definition of function factorial */
23 long factorial( long number )
24 {
25    /* base case */
26    if ( number == 0 ) {
27       return 1;
28    } /* end if */
29    else { /* recursive step */
30       return ( number * factorial( number - 1 ) );
31    } /* end else */
32
33 } /* end function factorial */
```

```
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
```

base case, 也可改為
if (number <= 1) {

其他情況時， recursive step

如果利用本程式計算到 15! 會得到什麼結果？

```
 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! = 1932053504
14! = 1278945280
15! = 2004310016
Press any key to continue
```

從 13! 開始計算得到的結果就不對(13! = 6,227,020,800)；Why?

5.14 Example Using Recursion: Fibonacci Series

- Fibonacci series: 0, 1, 1, 2, 3, 5, 8...
  - Each number is the sum of the previous two
  - Can be solved recursively:
    - `fib( n ) = fib( n - 1 ) + fib( n - 2 )`
  - Code for the `fibonacci` function

```
long fibonacci( long n )
{
  if (n == 0 || n == 1)  // base case
    return n;
  else
    return fibonacci( n - 1) + fibonacci( n - 2 );
}
```

5.14 Example Using Recursion: The Fibonacci Series

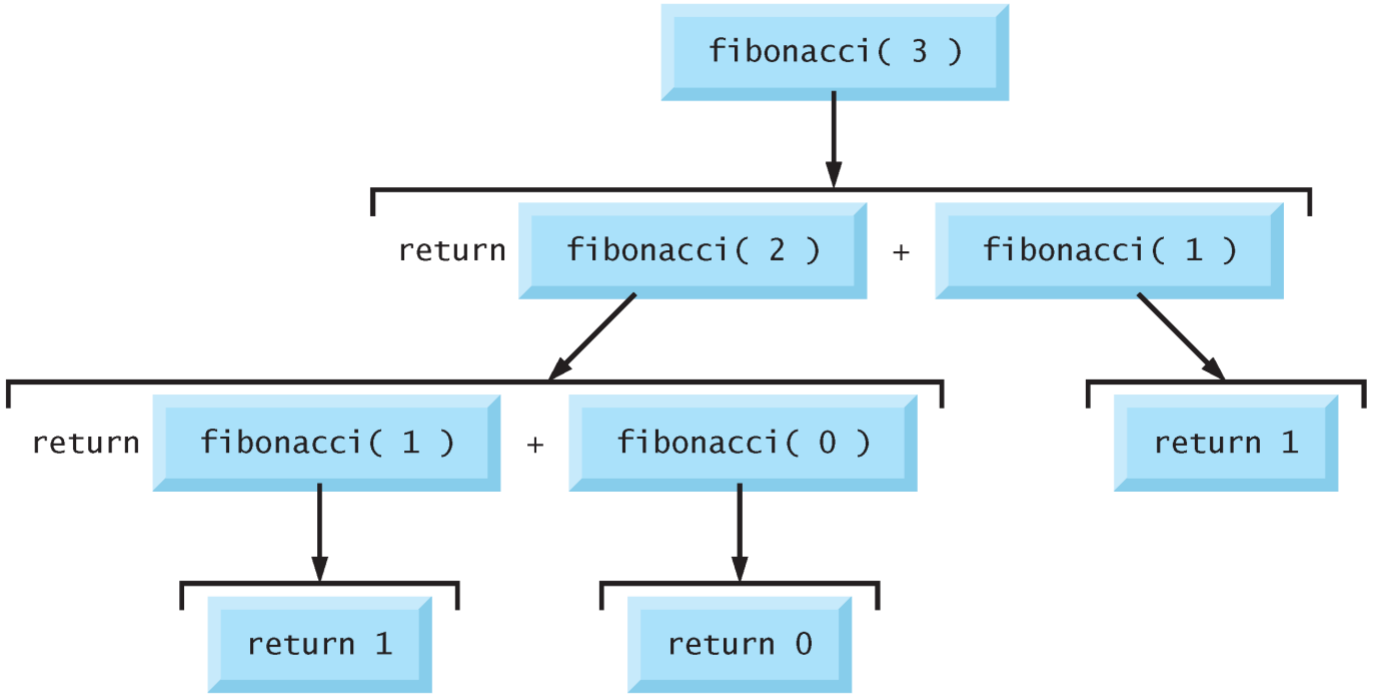
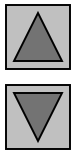


Fig. 5.16 | Set of recursive calls for `fibonacci( 3 )`.

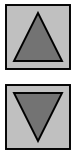
Fibonacci Numbers Generated by Recursive Function



Outline

fig05_15.c (Part 1 of 2)

```
1  /* Fig. 5.15: fig05_15.c
2     Recursive fibonacci function */
3  #include <stdio.h>
4
5  long fibonacci( long n ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10    long result; /* fibonacci value */
11    long number; /* number input by user */
12
13    /* obtain integer from user */
14    printf( "Enter an integer: " );
15    scanf( "%ld", &number );
16
17    /* calculate fibonacci value for number input by user */
18    result = fibonacci( number );
19
20    /* display result */
21    printf( "Fibonacci( %ld ) = %ld\n", number, result );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
26
```



Outline

fig05_15.c (Part 2 of 2)

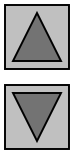
```
27 /* Recursive definition of function fibonacci */
28 long fibonacci( long n )
29 {
30    /* base case */
31    if ( n == 0 || n == 1 ) {
32       return n;
33    } /* end if */
34    else { /* recursive step */
35       return fibonacci( n - 1 ) + fibonacci( n - 2 );
36    } /* end else */
37
38 } /* end function fibonacci */
```

base case

其他情況時， recursive step

Program Output

```
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: 4
Fibonacci( 4 ) = 3
```



Outline

Program Output (continued)

```
Enter an integer: 5
Fibonacci( 5 ) = 5

Enter an integer: 6
Fibonacci( 6 ) = 8

Enter an integer: 10
Fibonacci( 10 ) = 55

Enter an integer: 20
Fibonacci( 20 ) = 6765

Enter an integer: 30
Fibonacci( 30 ) = 832040

Enter an integer: 35
Fibonacci( 35 ) = 9227465
```

5.15 Recursion vs. Iteration

- Repetition
  - Iteration: explicit loop
  - Recursion: repeated function calls
- Termination
  - Iteration: loop condition fails
  - Recursion: base case recognized
- Both can have infinite loops
- Balance
  - Choice between performance (iteration) and good software engineering (recursion)

5.15 Recursion vs. Iteration

Chapter	Recursion examples and exercises
Chapter 5	Factorial function Fibonacci function Greatest common divisor Sum of two integers Multiply two integers Raising an integer to an integer power Towers of Hanoi Recursive `main` Printing keyboard inputs in reverse Visualizing recursion
Chapter 6	Sum the elements of an array Print an array Print an array backward Print a string backward Check if a string is a palindrome Minimum value in an array Linear search Binary search
Chapter 7	Eight Queens Maze traversal

Review

- In this chapter, you have learned:
  - To construct programs modularly from small pieces called functions..
  - The common math functions available in the C standard library.
  - To create new functions.
  - The mechanisms used to pass information between functions.
  - The simulation techniques using random number generation.
    - `#include <stdlib.h>`
    - `#include <time.h>`
    - `1 + ( rand() % 6)` gives number between 1 ~ 6
    - `srand( time( NULL ) );/*load <time.h> */` to randomize the seed
  - The Scope of Variables
    - Global Variables
    - Local Variables
    - Static Variables
  - How to write and use functions that call themselves.

Exercises

5.50 Find the error in each of the following program segments and explain how to correct it:

a)
```
double cube( float ); /* function prototype */
...
cube( float number ) /* function definition */
{
  return number * number * number;
}
```

ANS: Function definition is missing return type.
```
double cube( float ); /* function prototype */
...
double cube( float number ) /* function definition */
{
  return number * number * number;
}
```

b)
```
register auto int x = 7;
```

ANS: Too many storage class definitions. Auto class definition is not necessary.
```
register int x = 7; /* auto removed */
```

c)
```
int randomNumber = srand();
```

ANS: srand() seeds the random number generator, and has a void return type. Function rand() produces random numbers.
```
int randomNumber = rand();
```

Exercises

d)
```
double y = 123.45678;
int x;
x = y;
printf( "%f\n", (double) x );
```

ANS: Decimal value is lost when a double is assigned to an integer. Type-casting the int to double cannot bring back the original decimal value. Only 123.000000 can be printed.

```
double y = 123.45678;
double x;
x = y;
printf( "%f\n", x );
```

e)
```
double square( double number )
{
   double number;
   return number * number;
}
```

ANS: number is defined twice.

```
double square( double number )
{
return number * number;
}
```

Exercises

f)
```
int sum( int n )
{
   if ( n == 0 )
      return 0;
   else
      return n + sum( n );
}
```

ANS: Infinite recursion.

```
int sum( int n )
{
   if ( n == 0 )
      return 0;
   else
      return n + sum( n - 1 );
}
```