5

C Functions

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 and arguments (data)**
    - **Function performs operations or manipulations**
    - **Function returns results**
  - **Function call analogy:**
    - **Boss asks worker to complete task**
      Worker gets information, does task, returns result
      Information hiding: boss does not know details

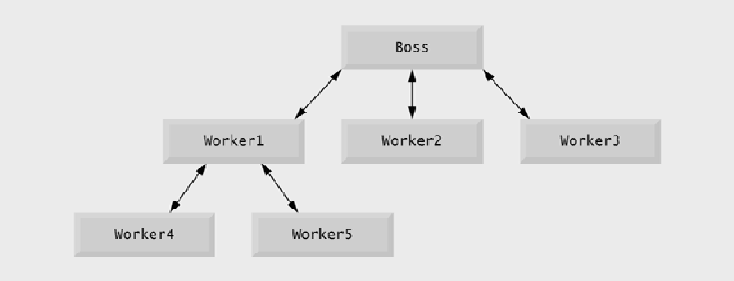

Fig. 5.1 | Hierarchical boss function/worker function relationship.

5.3 Math Library Functions

- **Math library functions**
  - **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**

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( 5.0 )` is `5.0`
		`fabs( 0.0 )` is `0.0`
		`fabs( -5.0 )` is `5.0`

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

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

Function	Description	Example
`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`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**
    - **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**

Software Engineering Observation 5.2

In programs containing many functions, `main` is often implemented as a group of calls to functions that perform the bulk of the program's work.

Software Engineering Observation 5.3

Each function should be limited to performing a single, well-defined task, and the function name should effectively express that task. This facilitates abstraction and promotes software reusability.

5.5 Function Definitions

- **Function definition format**

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

  - **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* `;`

Outline

fig05_03.c

```
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( 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 */
16
17    printf( "\n" );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* square function definition returns square of parameter */
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
```

Function prototype indicates function will be defined later in the program

Call to **square** function

Function definition

Common Programming Errors

5.2: Forgetting to return a value from a function that is supposed to return a value can lead to unexpected errors. The C standard states that the result of this omission is undefined.

5.3: Returning a value from a function with a `void` return type is a syntax error.

Common Programming Error 5.4

Specifying function parameters of the same type as `double x, y` instead of `double x, double y` might cause errors in your programs. The parameter declaration `double x, y` would actually make `y` a parameter of type `int` because `int` is the default.

→ Include the type of each parameter in the parameter list, even if that parameter is of the default type `int`.

Common Programming Error 5.5

Placing a semicolon after the right parenthesis enclosing the parameter list of a function definition is a syntax error.

```
3  #include <stdio.h>
4
5  int square( int y ); /* function prototype */
6
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 */
16
17    printf( "\n" );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* square function definition returns square of parameter */
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 */
```

Common Programming Errors

5.7: Defining a function inside another function is a syntax error.

5.6: Defining a function parameter again as a local variable within the function is a syntax error.

```
3  #include <stdio.h>
4
5  int square( int y ); /* function prototype */
6
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 */
16
17    printf( "\n" );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* square function definition returns square of parameter */
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 */
```

Good Programming Practice 5.6

Choosing meaningful function names and meaningful parameter names makes programs more readable and helps avoid excessive use of comments.

Outline

fig05_04.c

(1 of 2)

```
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( void )
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
```

Function prototype

Function call

```
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 */
```

Function definition

```
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

(2 of 2)

5.6 Function Prototypes

- **Function prototype**
  - **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**
  - **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 to lower types can lead to errors**

Good Programming Practice 5.8

Parameter names are sometimes included in function prototypes (our preference) for documentation purposes. The compiler ignores these names.

5.8 Headers

- **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**

5.9 Calling Functions: Call-by-Value and Call-by-Reference

- **Call by value**
  - **Copy of argument passed to function**
  - **Changes in function do not effect original**
  - **Use when function does not need to modify argument**
    - **Avoids accidental changes**
- **Call by reference**
  - **Passes original argument**
  - **Changes in function effect original**
  - **Only used with trusted functions**
- **For now, we focus on call by value**

5.10 Random Number Generation

- `rand` **function**
  - **Load** `<stdlib.h>`
  - **Returns "random" number between** `0` **and** `RAND_MAX` **(at least** `32767`**)**

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

  - **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`
    - **Add** `1` **to make random number between** `1` **and** `n`

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

number between `1` **and** `6`

Outline

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( void )
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 */
```

Generates a random number between 1 and 6

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

Outline

fig05_08.c

(1 of 3)

```
1  /* Fig. 5.8: fig05_08.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     int frequency1 = 0; /* rolled 1 counter */
10    int frequency2 = 0; /* rolled 2 counter */
11    int frequency3 = 0; /* rolled 3 counter */
12    int frequency4 = 0; /* rolled 4 counter */
13    int frequency5 = 0; /* rolled 5 counter */
14    int frequency6 = 0; /* rolled 6 counter */
15
16    int roll; /* roll counter, value 1 to 6000 */
17    int face; /* represents one roll of the die, value 1 to 6 */
18
19    /* loop 6000 times and summarize results */
20    for ( roll = 1; roll <= 6000; roll++ ) {
21       face = 1 + rand() % 6; /* random number from 1 to 6 */
22
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; /* optional */
49       } /* end switch */
50
51    } /* end for */
52
```

Outline

fig05_08.c

(2 of 3)


```
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 */
```

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

Outline

fig05_08.c

(3 of 3)

5.10 Random Number Generation

- `srand` function
  - `<stdlib.h>`
  - Takes an integer seed and jumps to that location in its "random" sequence

    `srand(` ***seed*** `);`
  - `srand( time( NULL ) );/*load <time.h> */`
    - `time( NULL )`

      Returns the number of seconds that have passed since January 1, 1970

      "Randomizes" the seed

Outline

fig05_09.c

(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( void )
8  {
9     int i;         /* counter */
10    unsigned seed; /* number used to seed random number generator */
11
12    printf( "Enter seed: " );
13    scanf( "%u", &seed ); /* note %u for unsigned */
14
15    srand( seed ); /* seed random number generator */
16
17    /* loop 10 times */
18    for ( i = 1; i <= 10; i++ ) {
19
```

Seeds the **rand** function

```
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          printf( "\n" );
26       } /* end if */
27
28    } /* end for */
29
30    return 0; /* indicates successful termination */
31
32 } /* 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
```

Outline

fig05_09.c

(2 of 2)

5.11 Example: A Game of Chance

- **Craps simulator**
- **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**

Outline

fig05_10.c

(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( void )
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
25    /* determine game status based on sum of dice */
26    switch( sum ) {
27
```

enum (enumeration) assigns numerical values to **CONTINUE**, **WON** and **LOST**

Outline

fig05_10.c

(2 of 4)

```
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
```

Outline

fig05_10.c

(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; /* game over, player won */
56       } /* end if */
57       else {
58
59          if ( sum == 7 ) { /* lose by rolling 7 */
60             gameStatus = LOST; /* game over, player lost */
61          } /* end if */
62
63       } /* end else */
64
65    } /* end while */
66
67    /* display won or lost message */
68    if ( gameStatus == WON ) { /* did player win? */
69       printf( "Player wins\n" );
70    } /* end if */
71    else { /* player lost */
72       printf( "Player loses\n" );
73    } /* end else */
74
75    return 0; /* indicates successful termination */
76
77 } /* end main */
```

Outline

fig05_10.c

(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 */
```

```
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 6 + 4 = 10
Point is 10
Player rolled 3 + 4 = 7
Player loses
```

Outline

fig05_11.c

Good Programming Practice 5.9

Use only uppercase letters in the names of enumeration constants to make these constants stand out in a program and to indicate that enumeration constants are not variables.