Functions and an Introduction to Recursion

6

Objectives

In this chapter you'll:

- Construct programs modularly from functions.
- Use common math library functions.
- Use function prototypes to declare a function.
- Use random-number generation to implement game-playing applications.
- Use C++14 digit separators to make numeric literals more readable
- Understand how the visibility of identifiers is limited to specific regions of programs.
- Understand how the function call/return mechanism is supported by the function-call stack and activation records.
- Understand the mechanisms for passing data to functions and returning results.
- Use inline functions, references and default arguments.
- Define with the same name overloaded functions that perform different tasks based on the number and types of their arguments.
- Define function templates that can generate families of overloaded functions.
- Write and use recursive functions.

Self-Review Exercises

6.1 Answer each of the following:

a) Program components in C++ are called _____ and_____.

ANS: functions, classes.

b) A function is invoked with a(n) _____.

ANS: function call.

c) A variable known only within the function in which it's defined is called a(n) _____.

ANS: local variable.

d) The _____ statement in a called function passes the value of an expression back to the calling function.

ANS: `return`.

e) The keyword _____ is used in a function header to indicate that a function does not return a value or to indicate that a function contains no parameters.

ANS: `void`.

f) An identifier's _____ is the portion of the program in which the identifier can be used.

ANS: scope.

g) The three ways to return control from a called function to a caller are _____, _____ and _____.

ANS: `return;`, `return` *expression*; or encounter the closing right brace of a function.

h) A(n) _____ allows the compiler to check the number, types and order of the arguments passed to a function.

ANS: function prototype.

i) Function _____ is used to produce random numbers.

ANS: `rand`.

j) Function _____ is used to set the random-number seed to randomize the number sequence generated by function `rand`.

ANS: `srand`.

k) A variable declared outside any block or function is a(n) _____ variable.

ANS: global.

l) For a local variable in a function to retain its value between calls to the function, it must be declared _____.

ANS: `static`.

m) A function that calls itself either directly or indirectly (i.e., through another function) is a(n) _____ function.

ANS: recursive.

n) A recursive function typically has two components—one that provides a means for the recursion to terminate by testing for a(n) _____ case and one that expresses the problem as a recursive call for a slightly simpler problem than the original call.

ANS: base.

o) It's possible to have various functions with the same name that operate on different types or numbers of arguments. This is called function _____.

ANS: overloading.

p) The _____ enables access to a global variable with the same name as a variable in the current scope.

ANS: unary scope resolution operator (`::`).

q) The _____ qualifier is used to declare read-only variables.

ANS: `const`.

r) A function _____ enables a single function to be defined to perform a task on many different data types.

ANS:

6.2 For the program in Fig. 6.1, state the scope (global namespace scope or block scope) of each of the following elements:

a) The variable `x` in `main`.
ANS: block scope.
b) The variable `y` in function `cube`'s definition.
ANS: block scope.
c) The function `cube`.
ANS: global namespace scope.
d) The function `main`.
ANS: global namespace scope.
e) The function prototype for `cube`.
ANS: global namespace scope.

```
1  // Exercise 6.2: ex06_02.cpp
2  #include <iostream>
3  using namespace std;
4
5  int cube(int y); // function prototype
6
7  int main() {
8     int x{0};
9
10    for (x = 1; x <= 10; x++) { // loop 10 times
11       cout << cube(x) << endl; // calculate cube of x and output results
12    }
13 }
14
15 // definition of function cube
16 int cube(int y) {
17    return y * y * y;
18 }
```

Fig. 6.1 | Program for Exercise 6.2.

6.3 Write a program that tests whether the examples of the math library function calls shown in Fig. 6.2 actually produce the indicated results.
ANS: See the following program:

```
1  // Exercise 6.3: ex06_03.cpp
2  // Testing the math library functions.
3  #include <iostream>
4  #include <iomanip>
5  #include <cmath>
6  using namespace std;
7
8  int main() {
9     cout << fixed << setprecision(1);
10
11    cout << "sqrt(" << 9.0 << ") = " << sqrt(9.0);
12    cout << "\nexp(" << 1.0 << ") = " << setprecision(6)
13       << exp(1.0) << "\nexp(" << setprecision(1) << 2.0
14       << ") = " << setprecision(6) << exp(2.0);
15    cout << "\nlog(" << 2.718282 << ") = " << setprecision(1)
16       << log(2.718282)
17       << "\nlog(" << setprecision(6) << 7.389056 << ") = "
18       << setprecision(1) << log(7.389056);
```

```
19       cout << "\nlog10(" << 10.0 << ") = " << log10(10.0)
20          << "\nlog10(" << 100.0 << ") = " << log10(100.0) ;
21       cout << "\nfabs(" << 5.1 << ") = " << fabs(5.1)
22          << "\nfabs(" << 0.0 << ") = " << fabs(0.0)
23          << "\nfabs(" << -8.76 << ") = " << fabs(-8.76);
24       cout << "\nceil(" << 9.2 << ") = " << ceil(9.2)
25          << "\nceil(" << -9.8 << ") = " << ceil(-9.8);
26       cout << "\nfloor(" << 9.2 << ") = " << floor(9.2)
27          << "\nfloor(" << -9.8 << ") = " << floor(-9.8);
28       cout << "\npow(" << 2.0 << ", " << 7.0 << ") = "
29          << pow(2.0, 7.0) << "\npow(" << 9.0 << ", "
30          << 0.5 << ") = " << pow(9.0, 0.5);
31       cout << setprecision(3) << "\nfmod("
32          << 2.6 << ", " << 1.2 << ") = "
33          << fmod(2.6, 1.2) << setprecision(1);
34       cout << "\nsin(" << 0.0 << ") = " << sin(0.0);
35       cout << "\ncos(" << 0.0 << ") = " << cos(0.0);
36       cout << "\ntan(" << 0.0 << ") = " << tan(0.0) << endl;
37    }
```

```
sqrt(9.0) = 3.0
exp(1.0) = 2.718282
exp(2.0) = 7.389056
log(2.718282) = 1.0
log(7.389056) = 2.0
log10(10.0) = 1.0
log10(100.0) = 2.0
fabs(5.1) = 5.1
fabs(0.0) = 0.0
fabs(-8.8) = 8.8
ceil(9.2) = 10.0
ceil(-9.8) = -9.0
floor(9.2) = 9.0
floor(-9.8) = -10.0
pow(2.0, 7.0) = 128.0
pow(9.0, 0.5) = 3.0
fmod(2.600, 1.200) = 0.200
sin(0.0) = 0.0
cos(0.0) = 1.0
tan(0.0) = 0.0
```

6.4 Give the function header for each of the following functions:

a) Function `hypotenuse` that takes two double-precision, floating-point arguments, `side1` and `side2`, and returns a double-precision, floating-point result.

ANS: `double hypotenuse(double side1, double side2)`

b) Function `smallest` that takes three integers, `x`, `y` and `z`, and returns an integer.

ANS: `int smallest(int x, int y, int z)`

c) Function `instructions` that does not receive any arguments and does not return a value. [*Note:* Such functions are commonly used to display instructions to a user.]

ANS: `void instructions()`

d) Function `intToDouble` that takes an integer argument, `number`, and returns a double-precision, floating-point result.

ANS: `double intToDouble(int number)`

6.5 Give the function prototype (without parameter names) for each of the following:

a) The function described in Exercise 6.4(a).

ANS: `double hypotenuse(double, double);`

b) The function described in Exercise 6.4(b).

ANS: `int smallest(int, int, int);`

c) The function described in Exercise 6.4(c).

ANS:
```
void instructions();
```

d) The function described in Exercise 6.4(d).

ANS:
```
double intToDouble(int);
```

6.6 Write a declaration for double-precision, floating-point variable `lastVal` that should retain its value between calls to the function in which it's defined.

ANS:
```
static double lastVal;
```

6.7 Find the error(s) in each of the following program segments, and explain how the error(s) can be corrected (see also Exercise 6.46):

a)
```
void g() {
   cout << "Inside function g" << endl;
   void h() {
      cout << "Inside function h" << endl;
   }
}
```

ANS: *Error:* Function `h` is defined in function `g`.
Correction: Move the definition of `h` out of the definition of `g`.

b)
```
int sum(int x, int y) {
   int result{0};

   result = x + y;
}
```

ANS: *Error:* The function is supposed to return an integer, but does not.
Correction: Place a `return result;` statement at the end of the function's body or delete variable `result` and place the following statement in the function:

```
return x + y;
```

c)
```
int sum(int n) { // assume n is nonnegative
   if (0 == n)
      return 0;
   else
      n + sum(n - 1);
}
```

ANS: *Error:* The result of `n + sum(n - 1)` is not returned; `sum` returns an improper result.
Correction: Rewrite the statement in the `else` clause as

```
return n + sum(n - 1);
```

d)
```
void f(double a); {
   float a;
   cout << a << endl;
}
```

ANS: *Errors:* Semicolon after the right parenthesis that encloses the parameter list, and redefining the parameter `a` in the function definition.
Corrections: Delete the semicolon after the right parenthesis of the parameter list, and delete the declaration `float a;`.

e)
```
void product() {
   int a{0};
   int b{0};
   int c{0};
   cout << "Enter three integers: ";
   cin >> a >> b >> c;
   int result{a * b * c};
   cout << "Result is " << result;
   return result;
}
```

ANS: *Error:* The function returns a value when it isn't supposed to.
Correction: Eliminate the `return` statement or change the return type.

6.8 Why would a function prototype contain a parameter type declaration such as `double&`?

ANS:

6.9 ***(True/False)*** All arguments to function calls in C++ are passed by value.

ANS: This creates a reference parameter of type "reference to `double`" that enables the function to modify the original variable in the calling function.

6.10 Write a complete program that prompts the user for the radius of a sphere, and calculates and prints the volume of that sphere. Use an `inline` function `sphereVolume` that returns the result of the following expression: `(4.0 / 3.0 * 3.14159 * pow(radius, 3))`.

ANS: See the following program:

```
// Exercise 6.10 Solution: ex06_10.cpp
// Inline function that calculates the volume of a sphere.
#include <iostream>
#include <cmath>
using namespace std;

const double PI{3.14159}; // define global constant PI

// calculates volume of a sphere
inline double sphereVolume(const double radius) {
   return 4.0 / 3.0 * PI * pow(radius, 3);
}

int main() {
   // prompt user for radius
   cout << "Enter the length of the radius of your sphere: ";
   double radiusValue;
   cin >> radiusValue; // input radius

   // use radiusValue to calculate volume of sphere and display result
   cout << "Volume of sphere with radius " << radiusValue
      << " is " << sphereVolume(radiusValue) << endl;
}
```

Exercises

NOTE: Solutions to the programming exercises are located in the **`ch06solutions`** *folder.*

6.11 Show the value of x after each of the following statements is performed:

a) `x = fabs(7.5);`

ANS: 7.5

b) `x = floor(7.5);`

ANS: 7.0

c) `x = fabs(0.0);`

ANS: 0.0

d) `x = ceil(0.0);`

ANS: 0.0

e) `x = fabs(-6.4);`

ANS: 6.4

f) `x = ceil(-6.4);`

ANS: -6.0

g) `x = ceil(-fabs(-8 + floor(-5.5)));`

ANS: -14.0

6.15 *(Short-Answer Questions)* Answer each of the following questions:

a) What does it mean to choose numbers "at random?"

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the `rand` function useful for simulating games of chance?

ANS: Because it produces a sequence of pseudorandom numbers that appears to be random.

c) Why would you randomize a program by using `srand`? Under what circumstances is it desirable not to randomize?

ANS: The sequence of numbers produced by the random number generator differ each time function `srand` is called. Not randomizing is useful for debugging purposes—the programmer knows the sequence of numbers.

d) Why is it often necessary to scale or shift the values produced by `rand`?

ANS: To produce random values in a specific range.

e) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events such as cars arriving at a toll booth, people arriving in lines, birds arriving at a tree, etc. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specified times.

6.16 *(Random Numbers)* Write statements that assign random integers to the variable n in the following ranges:

a) $1 \leq n \leq 2$

ANS: `n = 1 + rand() % 2;`

b) $1 \leq n \leq 100$

ANS: `n = 1 + rand() % 100;`

c) $0 \leq n \leq 9$

ANS: `n = rand() % 10;`

d) $1000 \leq n \leq 1112$

ANS: `n = 1000 + rand() % 113;`

e) $-1 \leq n \leq 1$

ANS: `n = rand() % 3 - 1;`

f) $-3 \leq n \leq 11$

ANS: `n = rand() % 15 - 3;`

6.17 ***(Random Numbers)*** Write a single statement that prints a number at random from each of the following sets:

a) 2, 4, 6, 8, 10.

ANS: `cout << 2 * (1 + rand() % 5)) << '\n';`

b) 3, 5, 7, 9, 11.

ANS: `cout << 1 + 2 * (1 + rand() % 5)) << '\n';`

c) 6, 10, 14, 18, 22.

ANS: `cout << 6 + 4 * (rand() % 5) << '\n';`

6.43 What does the following program do?

```
1  // Exercise 6.43: ex06_43.cpp
2  // What does this program do?
3  #include <iostream>
4  using namespace std;
5
6  int mystery(int, int); // function prototype
7
8  int main() {
9     cout << "Enter two integers: ";
10    int x{0};
11    int y{0};
12    cin >> x >> y;
13    cout << "The result is " << mystery(x, y) << endl;
14 }
15
16 // Parameter b must be a positive integer to prevent infinite recursion
17 int mystery(int a, int b) {
18    if (1 == b) { // base case
19       return a;
20    }
21    else { // recursion step
22       return a + mystery(a, b - 1);
23    }
24 }
```

ANS: This program multiplies two integers recursively.

```
Enter two integers: 8 2
The result is 16
```

6.46 ***(Find the Error)*** Find the error in each of the following program segments and explain how to correct it:

a)
```
float cube(float); // function prototype

   cube(float number) { // function definition
      return number * number * number;
   }
```

ANS: Error: The function definition defaults to a return type of `int`.
Correction: Specify a return type of `float` for the function definition.

b)
```
int randomNumber{srand()};
```
ANS: Error: Function `srand` takes an `unsigned` argument and does not return a value.
Correction: Use `rand` instead of `srand`.

c)
```
float y{123.45678};
int x;
x = y;
cout << static_cast<float>(x) << endl;
```
ANS: Error: The assignment of `y` to `x` truncates decimal places.
Correction: Declare `x` as type `float` instead of `int` and remove the now-redundant `static_cast`.

d)
```
double square(double number) {
   double number{0};
   return number * number;
}
```
ANS: Error: Variable `number` is declared twice.
Correction: Remove the declaration of variable `number` within the `{}`.

e)
```
int sum(int n) {
   if (0 == n) {
      return 0;
   }
   else {
      return n + sum(n);
   }
}
```
ANS: Error: Infinite recursion.
Correction: Change `sum(n)` to `sum(n - 1)`.

6.50 ***(Unary Scope Resolution Operator)*** What's the purpose of the unary scope resolution operator?

ANS: The unary scope resolution operator is used to access a global variable. In particular, the unary scope resolution operator is useful when a programmer needs to access a global variable when a local variable exists with the same name.

6.53 ***(Find the Error)*** Determine whether the following program segments contain errors. For each error, explain how it can be corrected. [*Note:* For a particular program segment, it's possible that no errors are present.]

a)
```
template <typename A>
int sum(int num1, int num2, int num3) {
   return num1 + num2 + num3;
}
```
ANS: Error: The function return type and parameter types are `int`.
Correction: The function return type and parameter types should be `A` or the function should not be a template.

b)
```
void printResults(int x, int y) {
   cout << "The sum is " << x + y << '\n';
   return x + y;
}
```
ANS: Error: The function specifies a `void` return type and attempts to return a value.
Two possible solutions: (1) change `void` to `int`, or (2) remove the line `return x + y;`.

c)
```
template <A>
A product(A num1, A num2, A num3) {
   return num1 * num2 * num3;
}
```
ANS: Error: Keyword `class` is missing in the template declaration.
Correction: Insert keyword `class` (or keyword `typename`), as in `template <class A>`.

d)
```
double cube(int);
int cube(int);
```
ANS: Error: The signatures are not different. Overloaded functions must have different signatures—the name and/or parameter list must be different. If only the returns types differ, the compiler generates an error message.
Correction: Change either the name or parameter list of one of the functions.

6.55 ***(C++11 Scoped `enum`)*** Create a scoped `enum` named `AccountType` containing constants named `SAVINGS`, `CHECKING` and `INVESTMENT`.

ANS:
```
enum class AccountType {SAVINGS, CHECKING, INVESTMENT};
```

6.56 ***(Function Prototypes and Definitions)*** Explain the difference between a function prototype and a function definition.

ANS: A function prototype tells the compiler the name of a function and the type of data returned by the function. A prototype also describes any additional data required by the function to perform its task (i.e., the function's parameters). A prototype does not contain code to make the function perform the task—it merely "outlines" the function so that the compiler can verify that programs call the function correctly. A function definition contains the actual code that executes to perform the function's specified task when the function is called. Parameter names are optional in the function prototype.