- A recursive function knows how to solve only the simplest case(s), or so-called base case(s). If the function is called with a **base case** (p. 254), the function simply returns a result.
- If the function is called with a more complex problem, the function typically divides the problem into two conceptual pieces—a piece that the function knows how to do and a piece that it does not know how to do. To make recursion feasible, the latter piece must resemble the original problem, but be a slightly simpler or slightly smaller version of it.
- For recursion to terminate, the sequence of **recursive calls** (p. 254) must converge on the base case.
- C++11's `unsigned long long int` type (which can be abbreviated as `unsigned long long`) on some systems enables you to store values in at least 8 bytes (64 bits) which can hold numbers as large as 18,446,744,073,709,551,615.

Section 6.19 Example Using Recursion: Fibonacci Series

- The ratio of successive Fibonacci numbers converges on a constant value of 1.618…. This number frequently occurs in nature and has been called the **golden ratio** or the **golden mean** (p. 257).

Section 6.20 Recursion vs. Iteration

- Iteration and recursion are similar: both are based on a control statement, involve iteration, involve a termination test, gradually approach termination and can occur infinitely.
- Recursion repeatedly invokes the mechanism, and overhead, of function calls. This can be expensive in both processor time and memory space. Each **recursive call** (p. 254) causes another copy of the function's variables to be created; this can consume considerable memory.

Self-Review Exercises

6.1 Answer each of the following:

a) Program components in C++ are called _______ and _______.
b) A function is invoked with a(n) _______.
c) A variable known only within the function in which it's defined is called a(n) _______.
d) The _______ statement in a called function passes the value of an expression back to the calling function.
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.
f) An identifier's _______ is the portion of the program in which the identifier can be used.
g) The three ways to return control from a called function to a caller are _______, _______ and _______.
h) A(n) _______ allows the compiler to check the number, types and order of the arguments passed to a function.
i) Function _______ is used to produce random numbers.
j) Function _______ is used to set the random-number seed to randomize the number sequence generated by function `rand`.
k) A variable declared outside any block or function is a(n) _______ variable.
l) For a local variable in a function to retain its value between calls to the function, it must be declared _______.
m) A function that calls itself either directly or indirectly (i.e., through another function) is a(n) _______ function.
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.
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 _______.

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

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

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

6.2 For the program in Fig. 6.30, state the scope (global namespace scope or block scope) of each of the following elements:

a) The variable `x` in `main`.

b) The variable `y` in function `cube`'s definition.

c) The function `cube`.

d) The function `main`.

e) The function prototype for `cube`.

```
// Exercise 6.2: ex06_02.cpp
#include <iostream>
using namespace std;

int cube(int y); // function prototype

int main() {
   int x{0};

   for (x = 1; x <= 10; x++) { // loop 10 times
      cout << cube(x) << endl; // calculate cube of x and output results
   }
}

// definition of function cube
int cube(int y) {
   return y * y * y;
}
```

Fig. 6.30 | 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.

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.

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

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.]

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

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

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

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

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

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

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.

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

   result = x + y;
}
```
c)
```
int sum(int n) { // assume n is nonnegative
   if (0 == n)
      return 0;
   else
      n + sum(n - 1);
}
```
d)
```
void f(double a); {
   float a;
   cout << a << endl;
}
```
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;
}
```

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

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

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))`.

Answers to Self-Review Exercises

6.1 a) functions, classes. b) function call. c) local variable. d) `return`. e) `void`. f) scope. g) `return;`, `return` *expression*; or encounter the closing right brace of a function. h) function prototype. i) `rand`. j) `srand`. k) global. l) `static`. m) recursive. n) base. o) overloading. p) unary scope resolution operator (`::`). q) `const`. r) `template`.

6.2 a) block scope. b) block scope. c) global namespace scope. d) global namespace scope. e) global namespace scope.

6.3 See the following program:

```cpp
// Exercise 6.3: ex06_03.cpp
// Testing the math library functions.
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main() {
   cout << fixed << setprecision(1);

   cout << "sqrt(" << 9.0 << ") = " << sqrt(9.0);
   cout << "\nexp(" << 1.0 << ") = " << setprecision(6)
      << exp(1.0) << "\nexp(" << setprecision(1) << 2.0
      << ") = " << setprecision(6) << exp(2.0);
   cout << "\nlog(" << 2.718282 << ") = " << setprecision(1)
      << log(2.718282)
      << "\nlog(" << setprecision(6) << 7.389056 << ") = "
      << setprecision(1) << log(7.389056);
   cout << "\nlog10(" << 10.0 << ") = " << log10(10.0)
      << "\nlog10(" << 100.0 << ") = " << log10(100.0) ;
   cout << "\nfabs(" << 5.1 << ") = " << fabs(5.1)
      << "\nfabs(" << 0.0 << ") = " << fabs(0.0)
      << "\nfabs(" << -8.76 << ") = " << fabs(-8.76);
   cout << "\nceil(" << 9.2 << ") = " << ceil(9.2)
      << "\nceil(" << -9.8 << ") = " << ceil(-9.8);
   cout << "\nfloor(" << 9.2 << ") = " << floor(9.2)
      << "\nfloor(" << -9.8 << ") = " << floor(-9.8);
   cout << "\npow(" << 2.0 << ", " << 7.0 << ") = "
      << pow(2.0, 7.0) << "\npow(" << 9.0 << ", "
      << 0.5 << ") = " << pow(9.0, 0.5);
   cout << setprecision(3) << "\nfmod("
      << 2.6 << ", " << 1.2 << ") = "
      << fmod(2.6, 1.2) << setprecision(1);
   cout << "\nsin(" << 0.0 << ") = " << sin(0.0);
   cout << "\ncos(" << 0.0 << ") = " << cos(0.0);
   cout << "\ntan(" << 0.0 << ") = " << tan(0.0) << endl;
}
```

```
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 a) `double hypotenuse(double side1, double side2)`
b) `int smallest(int x, int y, int z)`
c) `void instructions()`
d) `double intToDouble(int number)`

6.5 a) `double hypotenuse(double, double);`
b) `int smallest(int, int, int);`
c) `void instructions();`
d) `double intToDouble(int);`

6.6 `static double lastVal;`

6.7 a) *Error:* Function `h` is defined in function `g`.
Correction: Move the definition of `h` out of the definition of `g`.
b) *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) *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) *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) *Error:* The function returns a value when it isn't supposed to.
Correction: Eliminate the `return` statement or change the return type.

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

6.9 *False.* C++ enables pass-by-reference using reference parameters (and pointers, as we discuss in Chapter 8).

6.10 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

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

a) `x = fabs(7.5);`
b) `x = floor(7.5);`
c) `x = fabs(0.0);`
d) `x = ceil(0.0);`
e) `x = fabs(-6.4);`
f) `x = ceil(-6.4);`
g) `x = ceil(-fabs(-8 + floor(-5.5)));`

6.12 *(Parking Charges)* A parking garage charges a $20.00 minimum fee to park for up to three hours. The garage charges an additional $5.00 per hour for each hour *or part thereof* in excess of three hours. The maximum charge for any given 24-hour period is $50.00. Assume that no car parks for longer than 24 hours at a time. Write a program that calculates and prints the parking charges for each of three customers who parked their cars in this garage yesterday. You should enter the hours parked for each customer. Your program should print the results in a neat tabular format and should calculate and print the total of yesterday's receipts. The program should use the function `calculateCharges` to determine the charge for each customer. Your outputs should appear in the following format:

```
Car       Hours     Charge
1           1.5      20.00
2           4.0      25.00
3          24.0      50.00
TOTAL      29.5      95.50
```

6.13 *(Rounding Numbers)* An application of function `floor` is rounding a value to the nearest integer. The statement

```
y = floor(x + 0.5);
```

rounds the number x to the nearest integer and assigns the result to y. Write a program that reads several numbers and uses the preceding statement to round each of these numbers to the nearest integer. For each number processed, print both the original number and the rounded number.

6.14 *(Rounding Numbers)* Function `floor` can be used to round a number to a specific decimal place. The statement

```
y = floor(x * 10 + 0.5) / 10;
```

rounds x to the tenths position (the first position to the right of the decimal point). The statement

```
y = floor(x * 100 + 0.5) / 100;
```

rounds x to the hundredths position (the second position to the right of the decimal point). Write a program that defines four functions to round a number x in various ways:

a) `roundToInteger(number)`
b) `roundToTenths(number)`
c) `roundToHundredths(number)`
d) `roundToThousandths(number)`

For each value read, your program should print the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth and the number rounded to the nearest thousandth.

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

a) What does it mean to choose numbers "at random?"
b) Why is the `rand` function useful for simulating games of chance?
c) Why would you randomize a program by using `srand`? Under what circumstances is it desirable not to randomize?
d) Why is it often necessary to scale or shift the values produced by `rand`?
e) Why is computerized simulation of real-world situations a useful technique?

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

a) $1 \le n \le 2$
b) $1 \le n \le 100$
c) $0 \le n \le 9$
d) $1000 \le n \le 1112$
e) $-1 \le n \le 1$
f) $-3 \le n \le 11$

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.
b) 3, 5, 7, 9, 11.
c) 6, 10, 14, 18, 22.

6.18 *(Exponentiation)* Write a function `integerPower(`*base*`,` *exponent*`)` that returns the value of

$base^{exponent}$

For example, `integerPower(3, 4) = 3 * 3 * 3 * 3`. Assume that *exponent* is a positive, nonzero integer and that *base* is an integer. Do not use any math library functions.

6.19 *(Hypotenuse Calculations)* Define a function `hypotenuse` that calculates the hypotenuse of a right triangle when the other two sides are given. The function should take two `double` arguments and return the hypotenuse as a `double`. Use this function in a program to determine the hypotenuse for each of the triangles shown below.

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

6.20 *(Multiples)* Write a function `isMultiple` that determines for a pair of integers whether the second is a multiple of the first. The function should take two integer arguments and return `true` if the second is a multiple of the first, `false` otherwise. Use this function in a program that inputs a series of pairs of integers.

6.21 *(Even Numbers)* Write a program that inputs a series of integers and passes them one at a time to function `isEven`, which uses the remainder operator to determine whether an integer is even. The function should take an integer argument and return `true` if the integer is even and `false` otherwise.

6.22 *(Square of Asterisks)* Write a function that displays at the left margin of the screen a solid square of asterisks whose side is specified in integer parameter `side`. For example, if `side` is `4`, the function displays the following:

```
****
****
****
****
```

6.23 *(Square of Any Character)* Modify the function created in Exercise 6.22 to form the square out of whatever character is contained in character parameter `fillCharacter`. Thus, if `side` is `5` and `fillCharacter` is `#`, then this function should print the following:

```
#####
#####
#####
#####
#####
```

6.24 *(Separating Digits)* Write program segments that accomplish each of the following:

a) Calculate the integer part of the quotient when integer `a` is divided by integer `b`.
b) Calculate the integer remainder when integer `a` is divided by integer `b`.
c) Use the program pieces developed in (a) and (b) to write a function that inputs an integer between `1` and `32767` and prints it as a series of digits, each pair of which is separated by two spaces. For example, the integer `4562` should print as follows:

```
4  5  6  2
```

6.25 *(Calculating Number of Seconds)* Write a function that takes the time as three integer arguments (hours, minutes and seconds) and returns the number of seconds since the last time the clock "struck 12." Use this function to calculate the amount of time in seconds between two times, both of which are within one 12-hour cycle of the clock.

6.26 *(Celsius and Fahrenheit Temperatures)* Implement the following integer functions:

a) Function `celsius` returns the Celsius equivalent of a Fahrenheit temperature.
b) Function `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature.
c) Use these functions to write a program that prints charts showing the Fahrenheit equivalents of all Celsius temperatures from 0 to 100 degrees, and the Celsius equivalents of all Fahrenheit temperatures from 32 to 212 degrees. Print the outputs in a neat tabular format that minimizes the number of lines of output while remaining readable.

6.27 *(Find the Minimum)* Write a program that inputs three double-precision, floating-point numbers and passes them to a function that returns the smallest number.

6.28 *(Perfect Numbers)* An integer is said to be a *perfect number* if the sum of its divisors, including 1 (but not the number itself), is equal to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a function `isPerfect` that determines whether parameter `number` is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the divisors of each perfect number to confirm that the number is indeed perfect. Challenge the power of your computer by testing numbers much larger than 1000.

6.29 *(Prime Numbers)* An integer is said to be *prime* if it's divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

a) Write a function that determines whether a number is prime.
b) Use this function in a program that determines and prints all the prime numbers between 2 and 10,000. How many of these numbers do you really have to test before being sure that you've found all the primes?
c) Initially, you might think that *n*/2 is the upper limit for which you must test to see whether a number is prime, but you need only go as high as the square root of *n*. Why? Rewrite the program, and run it both ways. Estimate the performance improvement.

6.30 *(Reverse Digits)* Write a function that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the function should return 1367.

6.31 *(Greatest Common Divisor)* The *greatest common divisor (GCD)* of two integers is the largest integer that evenly divides each of the numbers. Write a function `gcd` that returns the greatest common divisor of two integers.

6.32 *(Quality Points for Numeric Grades)* Write a function `qualityPoints` that inputs a student's average and returns 4 if a student's average is 90–100, 3 if the average is 80–89, 2 if the average is 70–79, 1 if the average is 60–69 and 0 if the average is lower than 60.

6.33 *(Coin Tossing)* Write a program that simulates coin tossing. For each toss of the coin, the program should print `Heads` or `Tails`. Let the program toss the coin 100 times and count the number of times each side of the coin appears. Print the results. The program should call a separate function `flip` that takes no arguments and returns `0` for tails and `1` for heads. [*Note:* If the program realistically simulates the coin tossing, then each side of the coin should appear approximately half the time.]

6.34 *(Guess-the-Number Game)* Write a program that plays the game of "guess the number" as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1 to 1000. The program then displays the following:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
```

The player then types a first guess. The program responds with one of the following:

```
1. Excellent! You guessed the number!
   Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.
```

If the player's guess is incorrect, your program should loop until the player finally gets the number right. Your program should keep telling the player `Too high` or `Too low` to help the player "zero in" on the correct answer.

6.35 *(Guess-the-Number Game Modification)* Modify the program of Exercise 6.34 to count the number of guesses the player makes. If the number is 10 or fewer, print `"Either you know the secret or you got lucky!"` If the player guesses the number in 10 tries, then print `"Ahah! You know the secret!"` If the player makes more than 10 guesses, then print `"You should be able to do better!"` Why should it take no more than 10 guesses? Well, with each "good guess" the player should be able to eliminate half of the numbers. Now show why any number from 1 to 1000 can be guessed in 10 or fewer tries.

6.36 *(Recursive Exponentiation)* Write a recursive function `power(base, exponent)` that, when invoked, returns

$$base^{exponent}$$

For example, `power(3, 4) = 3 * 3 * 3 * 3`. Assume that `exponent` is an integer greater than or equal to 1. *Hint:* The recursion step would use the relationship

$$base^{exponent} = base \cdot base^{exponent - 1}$$

and the terminating condition occurs when `exponent` is equal to `1`, because

$$base^{1} = base$$

6.37 *(Fibonacci Series: Iterative Solution)* Write a *nonrecursive* version of the function `fibonacci` from Fig. 6.26.

6.38 *(Towers of Hanoi)* In this chapter, you studied functions that can be easily implemented both recursively and iteratively. In this exercise, we present a problem whose recursive solution demonstrates the elegance of recursion, and whose iterative solution may not be as apparent.

The Towers of Hanoi is one of the most famous classic problems every budding computer scientist must grapple with. Legend has it that in a temple in the Far East, priests are attempting to move a stack of golden disks from one diamond peg to another (Fig. 6.31). The initial stack has 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from one peg to another under the constraints that exactly one disk is moved at a time and at no time may a larger disk be placed above a smaller disk. Three pegs are provided, one being used for temporarily holding disks. Supposedly, the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

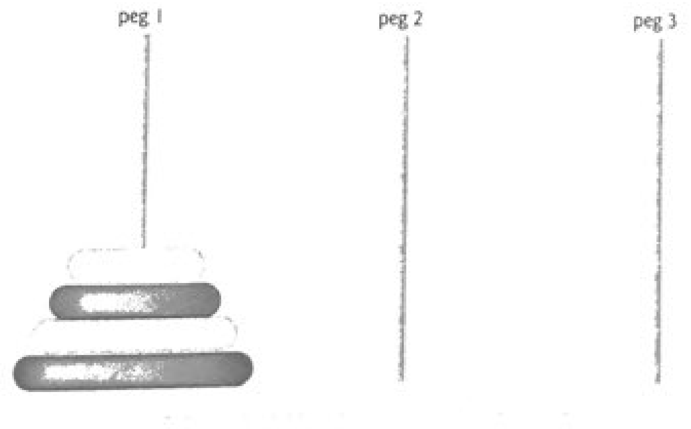

Fig. 6.31 | Towers of Hanoi for the case with four disks.

Let's assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that prints the precise sequence of peg-to-peg disk transfers.

If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, attacking this problem with recursion in mind allows the steps to be simple. Moving *n* disks can be viewed in terms of moving only *n* – 1 disks (hence, the recursion), as follows:

a) Move $n - 1$ disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
b) Move the last disk (the largest) from peg 1 to peg 3.
c) Move the $n - 1$ disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving $n = 1$ disk (i.e., the base case). This task is accomplished by simply moving the disk, without the need for a temporary holding area. Write a program to solve the Towers of Hanoi problem. Use a recursive function with four parameters:

a) The number of disks to be moved
b) The peg on which these disks are initially threaded
c) The peg to which this stack of disks is to be moved
d) The peg to be used as a temporary holding area

Display the precise instructions for moving the disks from the starting peg to the destination peg. To move a stack of three disks from peg 1 to peg 3, the program displays the following moves:

1 → 3 (This means move one disk from peg 1 to peg 3.)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3

6.39 *(Towers of Hanoi: Iterative Version)* Any program that can be implemented recursively can be implemented iteratively, although sometimes with more difficulty and less clarity. Try writing an iterative version of the Towers of Hanoi. If you succeed, compare your iterative version with the recursive version developed in Exercise 6.38. Investigate issues of performance, clarity and your ability to demonstrate the correctness of the programs.

6.40 *(Visualizing Recursion)* It's interesting to watch recursion "in action." Modify the factorial function of Fig. 6.25 to print its local variable and recursive call parameter. For each recursive call, display the outputs on a separate line and add a level of indentation. Do your utmost to make the outputs clear, interesting and meaningful. Your goal here is to design and implement an output format that helps a person understand recursion better. You may want to add such display capabilities to the many other recursion examples and exercises throughout the text.

6.41 *(Recursive Greatest Common Divisor)* The greatest common divisor of integers `x` and `y` is the largest integer that evenly divides both `x` and `y`. Write a recursive function `gcd` that returns the greatest common divisor of `x` and `y`, defined recursively as follows: If `y` is equal to `0`, then `gcd(x, y)` is `x`; otherwise, `gcd(x, y)` is `gcd(y, x % y)`, where `%` is the remainder operator. [*Note:* For this algorithm, `x` must be larger than `y`.]

6.42 *(Distance Between Points)* Write function `distance` that calculates the distance between two points *(x1, y1)* and *(x2, y2)*. All numbers and return values should be of type `double`.

6.43 What does the following program do?

```cpp
// Exercise 6.44: ex06_44.cpp
// What does this program do?
#include <iostream>
using namespace std;

int mystery(int, int); // function prototype

int main() {
   cout << "Enter two integers: ";
   int x{ };
   int y{ };
   cin >> x >> y;
   cout << "The result is " << mystery(x, y) << endl;
}

// Parameter b must be a positive integer to prevent infinite recursion
int mystery(int a, int b) {
   if (1 == b) { // base case
      return a;
   }
   else { // recursion step
      return a + mystery(a, b - 1);
   }
}
```

6.44 After you determine what the program of Exercise 6.43 does, modify the program to function properly after removing the restriction that the second argument be nonnegative.

6.45 *(Math Library Functions)* Write a program that tests as many of the math library functions in Fig. 6.2 as you can. Exercise each of these functions by having your program print out tables of return values for a diversity of argument values.

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;
}
```
b)
```
int randomNumber{srand()};
```
c)
```
float y{123.45678};
int x;
x = y;
cout << static_cast<float>(x) << endl;
```
d)
```
double square(double number) {
   double number{0};
   return number * number;
}
```
e)
```
int sum(int n) {
   if (0 == n) {
      return 0;
   }
   else {
      return n + sum(n);
   }
}
```

6.47 *(Craps Game Modification)* Modify the craps program of Fig. 6.9 to allow wagering. Package as a function the portion of the program that runs one game of craps. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a `wager`. Use a `while` loop to check that `wager` is less than or equal to `bankBalance` and, if not, prompt the user to reenter `wager` until a valid `wager` is entered. After a correct `wager` is entered, run one game of craps. If the player wins, increase `bankBalance` by `wager` and print the new `bankBalance`. If the player loses, decrease `bankBalance` by `wager`, print the new `bankBalance`, check on whether `bankBalance` has become zero and, if so, print the message `"Sorry. You busted!"` As the game progresses, print various messages to create some "chatter" such as "Oh, you're going for broke, huh?", "Aw cmon, take a chance!" or "You're up big. Now's the time to cash in your chips!".

6.48 *(Circle Area)* Write a C++ program that prompts the user for the radius of a circle, then calls inline function circleArea to calculate the area of that circle.

6.49 *(Pass-by-Value vs. Pass-by-Reference)* Write a complete C++ program with the two alternate functions specified below, each of which simply triples the variable count defined in main. Then compare and contrast the two approaches. These two functions are

a) function tripleByValue that passes a copy of count by value, triples the copy and returns the new value and

b) function tripleByReference that passes count by reference via a reference parameter and triples the original value of count through its alias (i.e., the reference parameter).

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

6.51 *(Function Template minimum)* Write a program that uses a function template called minimum to determine the smaller of two arguments. Test the program using integer, character and floating-point number arguments.

6.52 *(Function Template maximum)* Write a program that uses a function template called maximum to determine the larger of two arguments. Test the program using integer, character and floating-point number arguments.

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;
}
```

b)
```
void printResults(int x, int y) {
   cout << "The sum is " << x + y << '\n';
   return x + y;
}
```

c)
```
template <A>
A product(A num1, A num2, A num3) {
   return num1 * num2 * num3;
}
```

d)
```
double cube(int);
int cube(int);
```

6.54 *(C++11 Random Numbers: Modified Craps Game)* Modify the program of Fig. 6.9 to use the new C++11 random-number generation features shown in Section 6.9.

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

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

Making a Difference

As computer costs decline, it becomes feasible for every student, regardless of economic circumstance, to have a computer and use it in school. This creates exciting possibilities for improving the