Sample Run: In this sample run, the user input is shaded.

```
Enter a number in decimal: 57

Decimal 57 = 111001 binary
```

QUICK REVIEW

1. The process of solving a problem by reducing it to smaller versions of itself is called recursion.
2. A recursive definition defines a problem in terms of smaller versions of itself.
3. Every recursive definition has one or more base cases.
4. A recursive algorithm solves a problem by reducing it to smaller versions of itself.
5. Every recursive algorithm has one or more base cases.
6. The solution to the problem in a base case is obtained directly.
7. A function is called recursive if it calls itself.
8. Recursive algorithms are implemented using recursive functions.
9. Every recursive function must have one or more base cases.
10. The general solution breaks the problem into smaller versions of itself.
11. The general case must eventually be reduced to a base case.
12. The base case stops the recursion.
13. While tracing a recursive function:
    - Logically, you can think of a recursive function as having an unlimited number of copies of itself.
    - Every call to a recursive function—that is, every recursive call—has its own code and its own set of parameters and local variables.
    - After completing a particular recursive call, control goes back to the calling environment, which is the previous call. The current (recursive) call must execute completely before control goes back to the previous call. The execution in the previous call begins from the point immediately following the recursive call.
14. A function is called directly recursive if it calls itself.
15. A function that calls another function and eventually results in the original function call is said to be indirectly recursive.
16. A recursive function in which the last statement executed is the recursive call is called a tail recursive function.
17. To design a recursive function, you must do the following:
    a. Understand the problem requirements.
    b. Determine the limiting conditions. For example, for a list, the limiting condition is the number of elements in the list.

c. Identify the base cases and provide a direct solution to each base case.
d. Identify the general cases and provide a solution to each general case in terms of smaller versions of itself.

EXERCISES

The number in parentheses at the end of an exercise refers to the learning objective listed at the beginning of the chapter.

1. Mark the following statements as true or false.
   a. A recursive solution of a problem reduces the problem into smaller versions of itself. (1)
   b. Every recursive definition must have one or more base cases. (2)
   c. The general case stops the recursion. (2, 3)
   d. In the general case, the solution to the problem is obtained directly. (2, 3)
   e. It is not necessary for a recursive function to have a base case because the general case provides the solution. (2, 4)
   f. A recursive function always returns a value. (4)
   g. Every call to a recursive function has its own code and its own set of parameters and local variables. (4)
   h. A function that calls itself is called directly recursive. (5)
2. What is a base case? (2)
3. What is a recursive case? (2)
4. What is direct recursion? (5)
5. What is indirect recursion? (5)
6. What is tail recursion? (1, 5)
7. Consider the following recursive function: (2, 3, 4, 6)

```
int mystery(int number)                          //Line 1
{
    if (number == 0)                             //Line 2
        return number;                           //Line 3
    else                                         //Line 4
        return(mystery(number + 1) - number);    //Line 5
}
```

   a. Identify the base case.
   b. Identify the general case.
   c. What valid values can be passed as parameters to the function `mystery`?
   d. If `mystery(0)` is a valid call, what is its value? If not, explain why.
   e. If `mystery(10)` is a valid call, what is its value? If not, explain why.
   f. If `mystery(-3)` is a valid call, what is its value? If not, explain why.

8. Consider the following recursive function: (2, 3, 4, 6)

```
void funcRec(int u, char v)                          //Line 1
{
    if (u == 0)                                      //Line 2
        cout << v;                                   //Line 3
    else                                             //Line 4
    {                                                //Line 5
        char w;                                      //Line 6
        w = static_cast<char>
                (static_cast<int>(v) + 1);           //Line 7
        funcRec(u - 1, w);                           //Line 8
    }                                                //Line 9
}                                                    //Line 10
```

Answer the following questions:

a. Identify the base case.

b. Identify the general case.

c. What is the output of the following statement?

```
funcRec(5, 'A');
```

9. Consider the following recursive function:

```
void recFun(int x)
{
    if (x > 0)
    {
        cout << x % 10 << " ";
        recFun(x / 10);
    }
    else if (x != 0)
        cout << x << endl;
}
```

What is the output of the following statements? (4, 6)

a. `recFun(258);` b. `recFun(7);` c. `recFun(36);` d. `recFun(-85);`

10. Consider the following recursive function:

```
void recFun(int u)
{
    if (u == 0)
        cout << "Zero! ";
    else
    {
        cout << "Negative ";
        recFun(u + 1);
    }
}
```

What is the output, if any, of the following statements? (4, 6)

a. `recFun(8);` b. `recFun(0);` c. `recFun(-2);`

11. Consider the following recursive function:

```
void exercise(int x)
{
    if (x > 0 && x < 10)
    {
        cout << x << " ";
        exercise(x + 1);
    }
}
```

What is the output of the following statements? (4, 6)

a. `exercise(0);` b. `exercise(5);` c. `exercise(10);` d. `exercise(-5);`

12. Consider the following function:

```
int test(int x, int y)
{
    if (x <= y)
        return y - x;
    else
        return test(x - 1, y + 1);
}
```

What is the output of the following statements? (4, 6)

a. `cout << test(3, 100) << endl;`

b. `cout << test(15, 7) << endl;`

13. Consider the following function:

```
int func(int x)
{
    if (x == 0)
        return 2;
    else if (x == 1)
        return 3;
    else
        return (func(x - 1) + func(x - 2));
}
```

What is the output of the following statements? (4, 6)

a. `cout << func(0) << endl;`

b. `cout << func(1) << endl;`

c. `cout << func(2) << endl;`

d. `cout << func(5) << endl;`

14. Consider the following recursive function:

```
void recFun(int x, int y)
{
    if (x > 0 && y > 0)
    {
```

```
        if (x >= y && y != 0)
        {
            cout << x % y << " ";
            recFun(x - y, y);
        }
        else if (y > x && x != 0)
        {
            cout << y % x << " ";
            recFun(y - x, x);
        }
    }
    else
        cout << x + y << endl;
}
```

What is the output of the following statements? (4, 6)

a. `recFun(180, 38);` b. `recFun(75, 26);` c. `recFun(13, 86);`

d. `recFun(56, 148);`

15. Consider the following function:

```
int test(int x, int y)
{
    if (abs(x - y) <= 1)
        return x + y;
    else if (x > y)
        return test(x - 1, y + 1);
    else if (y > x)
        return test(x + 1, y - 1);

}
```

What is the output of the following statements? (4, 6)

a. `cout << test(8, 2) << endl;`

b. `cout << test(5, 16) << endl;`

c. `cout << test(-25, 2) << endl;`

d. `cout << test(8, -6) << endl;`

e. `cout << test(-20, -36) << endl;`

16. Suppose that `intArray` is an array of integers, and `length` specifies the number of elements in `intArray`. Also, suppose that `low` and `high` are two integers such that `0 <= low < length`, `0 <= high < length`, and `low < high`. That is, `low` and `high` are two indices in `intArray`. Write a recursive definition that reverses the elements in `intArray` between `low` and `high`. (2, 3, 4, 6)

17. Write a recursive algorithm to multiply two positive integers *m* and *n* using repeated addition. Specify the base case and the general case. (2, 3, 4, 6)

18. Consider the following problem: How many ways can a committee of four people be selected from a group of 10 people? There are many other similar problems in which you are asked to find the number of ways to select a set of items from a given set of items. The general problem can be stated as follows: Find the number of ways r different things can be chosen from a set of n items, in which r and n are nonnegative integers and $r \leq n$. Suppose $C(n, r)$ denotes the number of ways r different things can be chosen from a set of n items. Then, $C(n, r)$ is given by the following formula:

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

in which the exclamation point denotes the factorial function. Moreover, $C(n, 0) = C(n, n) = 1$. It is also known that $C(n, r) = C(n-1, r-1) + C(n-1, r)$. (2, 3, 4, 6)

   a. Write a recursive algorithm to determine $C(n, r)$. Identify the base case(s) and the general case(s).

   b. Using your recursive algorithm, determine $C(5, 3)$ and $C(9, 4)$.

PROGRAMMING EXERCISES

1. Write a recursive function that takes as a parameter a nonnegative integer and generates the following pattern of stars. If the nonnegative integer is 4, then the pattern generated is:

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

   Also, write a program that prompts the user to enter the number of lines in the pattern and uses the recursive function to generate the pattern. For example, specifying 4 as the number of lines generates the above pattern.

2. Write a recursive function to generate the following pattern of stars:

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

   Also, write a program that prompts the user to enter the number of lines in the pattern and uses the recursive function to generate the pattern. For example, specifying 4 as the number of lines generates the above pattern.

3. Write a recursive function, `vowels`, that returns the number of vowels in a string. Also, write a program to test your function.
4. Write a recursive function named `sumSquares` that returns the sum of the squares of the numbers from `0` to `num`, in which `num` is a nonnegative `int` variable. Do not use global variables; use the appropriate parameters. Also write a program to test your function.
5. Write a recursive function that finds and returns the sum of the elements of an `int` array. Also, write a program to test your function.
6. A palindrome is a string that reads the same both forward and backward. For example, the string `"madam"` is a palindrome. Write a program that uses a recursive function to check whether a string is a palindrome. Your program must contain a value-returning recursive function that returns `true` if the string is a palindrome and `false` otherwise. Do not use any global variables; use the appropriate parameters.
7. Write a recursive function that returns both the smallest and the largest element in an `int` array. Also, write a program to test your function.
8. Write a recursive function that returns `true` if the digits of a positive integer are in increasing order; otherwise, the function returns `false`. Also, write a program to test your function.
9. Write a recursive function, `reverseDigits`, that takes an integer as a parameter and returns the number with the digits reversed. Also, write a program to test your function.
10. Write a recursive function, `sumDigits`, that takes an integer as a parameter and returns the sum of the digits of the integer. Also, write a program to test your function.
11. Write a recursive function, `power`, that takes as parameters two integers x and y such that x is nonzero and returns x^y. You can use the following recursive definition to calculate x^y. If $y \geq 0$:

$$power(x, y) = \begin{cases} 1 & \text{if } y = 0 \\ x & \text{if } y = 1 \\ x \times power(x, y - 1) & \text{if } y > 1. \end{cases}$$

If $y < 0$:

$$power(x, y) = \frac{1}{power(x, -y)}.$$

Also, write a program to test your function.

12. (**Greatest Common Divisor**) Given two integers x and y, the following recursive definition determines the greatest common divisor of x and y, written $\gcd(x,y)$:

$$\gcd(x, y) = \begin{cases} x & \text{if } y = 0 \\ \gcd(y, x\%y) & \text{if } y \neq 0 \end{cases}$$

Note: In this definition, % is the mod operator.

Write a recursive function, `gcd`, that takes as parameters two integers and returns the greatest common divisor of the numbers. Also, write a program to test your function.

13. **(Ackermann's Function)** The Ackermann's function is defined as follows:

$$A(m,n) = \begin{cases} n+1, & \text{if } m = 0 \\ A(m-1,1), & \text{if } n = 0 \\ A(m-1, A(m,n-1)), & \text{otherwise,} \end{cases}$$

in which m and n are nonnegative integers. Write a recursive function to implement Ackermann's function. Also write a program to test your function. What happens when you call the function with $m = 4$ and $n = 3$?

14. Write a recursive function to implement the recursive algorithm of Exercise 16 (reversing the elements of an array between two indices). Also, write a program to test your function.

15. Write a recursive function to implement the recursive algorithm of Exercise 17 (multiplying two positive integers using repeated addition). Also, write a program to test your function.

16. Write a recursive function to implement the recursive algorithm of Exercise 18 (determining the number of ways to select a set of things from a given set of things). Also, write a program to test your function.

17. **(Recursive Sequential Search)** The sequential search algorithm given in Chapter 8 is nonrecursive. Write and implement a recursive version of the sequential search algorithm.

18. In the Programming Example, Converting a Number from Decimal to Binary, given in this chapter, you learned how to convert a decimal number into the equivalent binary number. Two more number systems, octal (base 8) and hexadecimal (base 16), are of interest to computer scientists. In fact, in C++, you can instruct the computer to store a number in octal or hexadecimal. (Appendix C describes these number systems.)

    The digits in the octal number system are 0, 1, 2, 3, 4, 5, 6, and 7. The digits in the hexadecimal number system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. So A in hexadecimal is 10 in decimal, B in hexadecimal is 11 in decimal, and so on.

    The algorithm to convert a positive decimal number into an equivalent number in octal (or hexadecimal) is the same as discussed for binary numbers. Here, we divide the decimal number by 8 (for octal) and by 16 (for hexadecimal). Suppose a_b represents the number a to the base b. For example, 75_{10} means 75 to the base 10 (that is decimal), and 83_{16} means 83 to the base 16 (that is, hexadecimal). Then $753_{10} = 1361_8$ and $753_{10} = 2F1_{16}$.

    Write a program that uses a recursive function to convert a number in decimal to base 8 or base 16.

19. The function `sqrt` from the header file `cmath` can be used to find the square root of a nonnegative real number. Using Newton's method, you can also write an algorithm to find the square root of a nonnegative real number within a given tolerance as follows: Suppose x is a nonnegative real number, a is the approximate square root of x, and *epsilon* is the tolerance. Start with $a = x$.

    a. If $|a^2 - x| \leq epsilon$, then a is the square root of x within the tolerance; otherwise:

    b. Replace a with $(a^2 + x) / (2a)$ and repeat Step a in which $|a^2 - x|$ denotes the absolute value of $a^2 - x$.

    Write a recursive function to implement this algorithm to find the square root of a nonnegative real number. Also, write a program to test your function.