✔ Quick Quiz 10.1

1. _____ is the phenomenon of a function calling itself.
2. Name and describe the two parts of a recursive definition of a function.
3. A nonrecursive function for computing some value may execute more rapidly than a recursive function that computes the same value. (True or false)
4. For the following recursive function, find `f(5)`:

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

5. For the function in Question 4, find `f(0)`.
6. For the function in Question 4, suppose `+` is changed to `*` in the inductive case. Find `f(5)`.
7. For the function in Question 4, what happens with the function call `f(-1)`?

Exercises 10.1

Exercises 1–6 assume ASCII representation of characters and the following function `f()`:

```
void f(char ch)
{
  if (('A' <= ch) && (ch <= 'H'))
  {
    f(ch - 1);
    cout << ch;
  }
  else
    cout << endl;
}
```

Tell what output will be produced by the function call.

1. `f('C')`
2. `f('G')`
3. `f('3')`
4. `f('C')` if `ch - 1` is replaced by `ch + 1` in the function
5. `f('C')` if the output statement and the recursive call to `f()` are interchanged
6. `f('C')` if a copy of the output statement is inserted before the recursive call to `f()`

7. Given the following function f() and assuming ASCII representation of characters, use the method illustrated in this section to trace the sequence of function calls and returns in evaluating f('a', 'e') and f('h', 'c'):

```
int f(char ch1, char ch2)
{
  if (ch1 > ch2)
    return 0;
  if (ch1 + 1 == ch2)
    return 1;
  // else
  return F(ch1 + 1, ch2 - 1) + 2;
}
```

Exercises 8–10 assume ASCII representation of characters and the following function g():

```
void g(char ch, int n)
{
  if (n <= 0)
    cout << endl;
  else
  {
    g(ch - 1, n - 1);
    cout << ch;
    g(ch + 1, n - 1);
  }
}
```

8. What output will be produced by the function call g('M', 4)? (*Hint:* First try g('M', 2), then g('M', 3).)
9. How many letters are output by the call g('M', 10)?
10. If the output statement is moved before the first recursive call to g(), what output will be produced by g('M', 4)?

Determine what is calculated by the recursive functions in Exercises 11–15:

11.
```
unsigned f(unsigned n)
{
  if (n == 0)
    return 0;
  // else
    return n * f(n - 1);
}
```

12.
```
unsigned f(double x, unsigned n)
{
  if (n == 0)
    return 0;
```

```
  // else
    return n + f(x, n - 1);
}
```

13.
```
unsigned f(unsigned n)
{
  if (n < 2)
    return 0;
  // else
    return 1 + f(n / 2);
}
```

14.
```
unsigned f(unsigned n)
{
  if (n == 0)
    return 0;
  // else
    return f(n / 10) + n % 10;
}
```

15.
```
unsigned f(int n)
{
  if (n < 0)
    return f(-n);
  if (n < 10)
    return n;
  // else
  return f(n / 10);
}
```

16–20. Write nonrecursive versions of the functions in Exercises 11–15.

The exercises that follow ask you to write functions. You should test these functions with driver programs as instructed in Programming Problems 1–16 at the end of this chapter.

21. Write a recursive function that returns the number of digits in a nonnegative integer.

22. Write a nonrecursive version of the function in Exercise 21.

23. Write a recursive function `printReverse()` that displays an integer's digits in reverse order.

24. Write a nonrecursive version of the function `printReverse()` in Exercise 23.

25. Modify the recursive exponentiation function in the text so that it also works for negative exponents. One approach is to modify the recursive definition of x^n so that, for negative values of n, division is used instead of multiplication and n

is incremented rather than decremented:

$$x^n = \begin{cases} 1 & \text{if } n \text{ is } 0 \\ x^{n-1} \times x & \text{if } n \text{ is greater than } 0 \\ x^{n+1}/x & \text{otherwise} \end{cases}$$

26. Write a nonrecursive version of the function in Exercise 25 (i.e., a nonrecursive exponential function in which exponents may be 0, positive, or negative).

Assuming declarations of the form

```
const int MAX_CAPACITY = . . .;  // user defined
typedef ElementType = . . . ;    // user defined
typedef ElementType ArrayType[MAX_CAPACITY];
```

write recursive definitions of the functions whose prototypes are given in Exercises 27–29.

27. `void reverseArray(ArrayType a, int first, int last);`
`/* Reverse the contents of a[first], ..., a[last] */`

28. `int sumArray(ArrayType a, int n);`
`/* Return the sum of a[0], ..., a[n - 1] */`

29. `int location(ArrayType a, int first, int last, Element elm);`
`/* Return the location of elm in a[first],`
`...., a[last]. If not found, return 0. */`

30. Using the basic string operations length, concatenate, copy, and find (see Section 5.2), develop a recursive algorithm for reversing a string.

31. Proceed as in Exercise 30, but develop a nonrecursive algorithm.

32. Write a recursive function that implements the algorithm in this section for determining if a number is a palindrome.

33. The *greatest common divisor* of two integers a and b, GCD(a, b), not both of which are zero, is the largest positive integer that divides both a and b. The *Euclidean algorithm* for finding this greatest common divisor of a and b is as follows: Divide a by b to obtain the integer quotient q and the remainder r, so that $a = bq + r$ (if $b = 0$, GCD(a, b) = a). Then GCD(a, b) = GCD(b, r). Replace a with b and b with r and repeat this procedure. Because the remainders are decreasing, eventually a remainder of 0 will result. The last nonzero remainder is GCD(a, b). For example,

$$\begin{aligned} 1260 &= 198 \cdot 6 + 72 & \text{GCD}(1260, 198) &= \text{GCD}(198, 72) \\ 198 &= 72 \cdot 2 + 54 & &= \text{GCD}(72, 54) \\ 72 &= 54 \cdot 1 + 18 & &= \text{GCD}(54, 18) \\ 54 &= 18 \cdot 3 + 0 & &= 18 \end{aligned}$$

(*Note*: If either a or b is negative, replace them with their absolute values in this algorithm.) Write a recursive greatest common divisor function.

34. Proceed as in Exercise 33, but write a nonrecursive function.