Exercises

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

   a. $x = \text{fabs}(7.5)$
   
   b. $x = \text{floor}(7.5)$
   
   c. $x = \text{fabs}(0.0)$
   
   d. $x = \text{ceil}(0.0)$
   
   e. $x = \text{fabs}(-6.4)$
   
   f. $x = \text{ceil}(-6.4)$
   
   g. $x = \text{ceil}(\text{fabs}(-8 + \text{floor}(-5.5)))$

6.12  A parking garage charges a $2.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour or part thereof in excess of three hours. The maximum charge for any given 24-hour period is $10.00. 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:

<table>
<thead>
<tr>
<th>Car</th>
<th>Hours</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5</td>
<td>2.00</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>2.50</td>
</tr>
<tr>
<td>3</td>
<td>24.0</td>
<td>10.00</td>
</tr>
<tr>
<td>TOTAL</td>
<td>29.5</td>
<td>14.50</td>
</tr>
</tbody>
</table>

6.13  An application of function floor is rounding a value to the nearest integer. The statement
\[ y = \text{floor}( x + .5 ); \]

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 Function \text{floor} can be used to round a number to a specific decimal place. The statement

\[ y = \text{floor}( x * 10 + .5 ) / 10; \]

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

\[ y = \text{floor}( x * 100 + .5 ) / 100; \]

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. \text{roundToInteger}( \text{number} )

b. \text{roundToTenths}( \text{number} )

c. \text{roundToHundredths}( \text{number} )

d. \text{roundToThousandths}( \text{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 Answer each of the following questions:

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

b. Why is the \text{rand} function useful for simulating games of chance?

c. Why would you randomize a program by using \text{srand}? Under what circumstances is it desirable not to randomize?

d. Why is it often necessary to scale or shift the values produced by \text{rand}?

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

6.16 Write statements that assign random integers to the variable \( n \) in the following ranges:
For each of the following sets of integers, write a single statement that prints a number at random from the set:

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

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

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

d. 1000 ≤ n ≤ 1112

e. 1 ≤ n ≤ 1

f. 3 ≤ n ≤ 11

6.17 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. The function integerPower should use for or while to control the calculation. Do not use any math library functions.

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

<table>
<thead>
<tr>
<th>Triangle</th>
<th>Side 1</th>
<th>Side 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.0</td>
<td>4.0</td>
</tr>
<tr>
<td>2</td>
<td>5.0</td>
<td>12.0</td>
</tr>
<tr>
<td>3</td>
<td>8.0</td>
<td>15.0</td>
</tr>
</tbody>
</table>

6.20 Write a function multiple 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 Write a program that inputs a series of integers and passes them one at a time to function `even`, which uses the modulus 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 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 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 Use techniques similar to those developed in Exercise 6.22 and Exercise 6.23 to produce a program that graphs a wide range of shapes.

6.25 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:

```
Pa
g
ge 4 of 14
```
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.

**6.27 (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.

Write a program that inputs three double-precision, floating-point numbers and passes them to a function that returns the smallest number.

**6.29 (Perfect Numbers)** An integer is said to be a **perfect number** if the sum of its factors, 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 `perfect` 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 factors 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.

**6.30 (PrimeNumbers)** An integer is said to be **prime** if it is 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 have 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.

**6.31 (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.
6.32  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.

6.33  Write a function qualityPoints that inputs a student's average and returns 4 if a student's average is 90 or 100, 3 if the average is 80 or 89, 2 if the average is 70 or 79, 1 if the average is 60 or 69 and 0 if the average is lower than 60.

6.34  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.]

6.35  (Computers in Education) Computers are playing an increasing role in education. Write a program that helps an elementary school student learn multiplication. Use rand to produce two positive one-digit integers. It should then type a question such as

How much is 6 times 7?

The student then types the answer. Your program checks the student's answer. If it is correct, print "Very good!", then ask another multiplication question. If the answer is wrong, print "No. Please try again.", then let the student try the same question repeatedly until the student finally gets it right.

6.36  (Computer Assisted Instruction) The use of computers in education is referred to as computer-assisted instruction (CAI). One problem that develops in CAI environments is student fatigue. This can be eliminated by varying the computer's dialogue to hold the student's attention. Modify the program of Exercise 6.35 so the various comments are printed for each correct answer and each incorrect answer as follows:

Responses to a correct answer

Very good!
Excellent!
Nice work!
Keep up the good work!

Responses to an incorrect answer

No. Please try again.
Wrong. Try once more.
Don't give up!
No. Keep trying.

Use the random number generator to choose a number from 1 to 4 to select an
appropriate response to each answer. Use a switch statement to issue the responses.

**6.37** More sophisticated computer-aided instruction systems monitor the student's performance over a period of time. The decision to begin a new topic often is based on the student's success with previous topics. Modify the program of Exercise 6.36 to count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage of correct responses. If the percentage is lower than 75 percent, your program should print "Please ask your instructor for extra help" and terminate.

**6.38** *(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.

**6.39** Modify the program of Exercise 6.38 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.

**6.40** Write a recursive function `power(base, exponent)` that, when invoked, returns `base^exponent`
For example, \( \text{power}(3, 4) = 3 \times 3 \times 3 \times 3 \). Assume that \( \text{exponent} \) is an integer greater than or equal to 1. \( \textbf{Hint:} \) The recursion step would use the relationship

\[
\text{base}^{\text{exponent}} = \text{base} \times \text{base}^{\text{exponent} - 1}
\]

and the terminating condition occurs when \( \text{exponent} \) is equal to 1, because

\[
\text{base}^1 = \text{base}
\]

\textbf{6.41 (Fibonacci Series)} The Fibonacci series

\[0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots\]

begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms. (a) Write a \textit{nonrecursive} function \( \text{fibonacci}(n) \) that calculates the \( n \)th Fibonacci number. (b) Determine the largest \textit{int} Fibonacci number that can be printed on your system. Modify the program of part (a) to use \textit{double} instead of \textit{int} to calculate and return Fibonacci numbers, and use this modified program to repeat part (b).

\textbf{6.42 (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 \textbf{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. 6.41). 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.

\textbf{Figure 6.41. Towers of Hanoi for the case with four disks.}

(This item is displayed on page 322 in the print version)
Let us 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

Your program should print the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from
peg 1 to peg 3, your program should print the following series of 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

6.43 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 6.42. Investigate issues of performance, clarity and your ability to demonstrate the correctness of the programs.

6.44 (Visualizing Recursion) It is interesting to watch recursion "in action." Modify the factorial function of Fig. 6.29 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.

6.45 (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 \(\text{gcd}\) that returns the greatest common divisor of \(x\) and \(y\), defined recursively as follows: If \(y\) is equal to 0, then \(\text{gcd}(x, y) = x\); otherwise, \(\text{gcd}(x, y) = \text{gcd}(y, x \mod y)\), where \(\mod\) is the modulus operator. [Note: For this algorithm, \(x\) must be larger than \(y\).]

6.46 Can \textit{main} be called recursively on your system? Write a program containing a function \textit{main}. Include static local variable \textit{count} and initialize it to 1. Postincrement and print the value of \textit{count} each time \textit{main} is called. Compile your program. What happens?

6.47 Exercises 6.356.37 developed a computer-assisted instruction program to teach an elementary school student multiplication. This exercise suggests enhancements to that program.

a. Modify the program to allow the user to enter a grade-level capability. A grade level of 1 means to use only single-digit numbers in the problems, a grade level of 2 means to use numbers as large as two digits, etc.

b. Modify the program to allow the user to pick the type of arithmetic problems he or
she wishes to study. An option of 1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only and 5 means a random mix of problems of all these types.

**6.48** Write function `distance` that calculates the distance between two points \((x_1, y_1)\) and \((x_2, y_2)\). All numbers and return values should be of type `double`.

**6.49** What is wrong with the following program?

```cpp
// Exercise 6.49: ex06_49.cpp
// What is wrong with this program?
#include <iostream>
using std::cin;
using std::cout;

int main()
{

    int c;

    if ( ( c = cin.get() ) != EOF )
    {
        main();
        cout << c;
    } // end if

    return 0; // indicates successful termination
} // end main
```

**6.50** What does the following program do?

```cpp
// Exercise 6.50: ex06_50.cpp
// What does this program do?
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

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

int main()
{

    int x, y;

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

    return 0; // indicates successful termination
} // end main

// Parameter b must be a positive integer to prevent infinite recursion
int mystery( int a, int b )
```

[Page 324]
23  {
24      if ( b == 1 ) // base case
25          return a;
26      else // recursion step
27          return a + mystery(a, b - 1);
28  } // end function mystery

6.51 After you determine what the program of Exercise 6.50 does, modify the program to function properly after removing the restriction that the second argument be nonnegative.

6.52 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.

6.53 Find the error in each of the following program segments and explain how to correct it:

   a. float cube(float); // function prototype
      double cube(float number) // function definition
      {
          return number * number * number;
      }

   b. register auto int x = 7;

   c. int randomNumber = srand();

   d. float y = 123.45678;
      int x;
      x = y;
      cout << static_cast<float>(x) << endl;

   e. double square(double number) {
      double number;
      return number * number;
   }

   f. int sum(int n) {
      if (n == 0) 
          return 0;
      else 
          return n + sum(n);
   }

6.54 Modify the craps program of Fig. 6.11 to allow wagering. Package as a function the
portions 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`, and 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!".

6.55 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.

6.56 Write a complete C++ program with the two alternate functions specified below, of which each 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).

6.57 What is the purpose of the unary scope resolution operator?

6.58 Write a program that uses a function template called `min` to determine the smaller of two arguments. Test the program using integer, character and floating-point number arguments.

6.59 Write a program that uses a function template called `max` to determine the largest of three arguments. Test the program using integer, character and floating-point number arguments.

6.60 Determine whether the following program segments contain errors. For each error, explain how it can be corrected. [Note: For a particular program segment, it is possible that no errors are present in the segment.]

```cpp
a. template < class 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 );