6

Functions and an Introduction to Recursion

Form ever follows function.
—Louis Henri Sullivan

E pluribus unum.
(One composed of many.)
—Virgil

O! call back yesterday, bid time return.
—William Shakespeare

Call me Ishmael.
—Herman Melville

When you call me that, smile!
—Owen Wister

Answer me in one word.
—William Shakespeare

There is a point at which methods devour themselves.
—Frantz Fanon

Life can only be understood backwards; but it must be lived forwards.
—Soren Kierkegaard

OBJECTIVES

In this chapter you'll learn:

- To construct programs modularly from functions.
- To use common math functions available in the C++ Standard Library.
- To create functions with multiple parameters.
- The mechanisms for passing information between functions and returning results.
- How the function call/return mechanism is supported by the function call stack and activation records.
- To use random number generation to implement game-playing applications.
- How the visibility of identifiers is limited to specific regions of programs.
- To write and use recursive functions, i.e., functions that call themselves.

Assignment Checklist

Name: _____ Date: _____

Section: _____

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24	
Short Answer	25, 26, 27, 28, 29	
Programming Output	30, 31, 32, 33, 34, 35, 36	
Correct the Code	37, 38, 39, 40, 41, 42, 43	
Lab Exercises		
Lab Exercise 1 — Prime Numbers	YES NO	
Follow-Up Question and Activity	1	
Lab Exercise 2 — Reversing Digits	YES NO	
Follow-Up Questions and Activities	1, 2, 3	
Lab Exercise 3 — Greatest Common Divisor	YES NO	
Follow-Up Questions and Activities	1, 2	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3, 4, 5, 6, 7, 8	
Programming Challenges	1, 2, 3	

Prelab Activities

Matching

Name: _____ **Date:** _____

Section: _____

After reading Chapter 6 of *C++ How to Program, Seventh Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

Term	Description
___ 1. Divide-and-conquer	a) Using existing functions as building blocks to create new programs.
___ 2. Function call	b) Technique for reducing the range of values produced by function `rand`.
___ 3. Local variable	c) Contains function prototypes and definitions of various data types.
___ 4. Pass-by-reference	d) Seeds the random-number generator.
___ 5. `static`	e) Invokes a function.
___ 6. Pass-by-value	f) Known only in the function in which it is defined.
___ 7. Scaling	g) Passes a copy of an argument's value to a function.
___ 8. Scope	h) Technique for constructing a program from smaller, more manageable pieces.
___ 9. `rand`	i) Generates and returns a pseudo-random number.
___ 10. Header file	j) The portion of a program in which an identifier may be referenced.
___ 11. Software reuse	k) Provides a called function with the ability to access the caller's data directly.
___ 12. `srand`	l) Signifies that a local variable retains its value after the function in which it was defined is exited.

Prelab Activities

Name:

Fill in the Blank

Name: _____ **Date:** _____

Section: _____

Fill in the blanks in each of the following statements:

13. A reference acts as a(n) _____ for another variable.

14. Most functions have a list of _____ that provide the means of communicating data between functions.

15. Each part of the C++ standard library has a corresponding _____ that contains the function prototypes for all functions in that part of that library and definitions of various data types and constants needed by those functions.

16. To specify that a function does not return a value, use the return type _____.

17. The compiler ignores variable names in function _____.

18. To use function `sqrt`, the _____ header file must be included.

19. C++ automatically generates a function template specialization based on the _____ of the arguments passed to the function template.

20. When a called function returns, its _____ is popped from the function-call stack.

21. Variables of the _____ storage class are destroyed when the program exits the block in which they are declared.

22. For a recursion to terminate, it must reduce the original problem to the simplest form, called the _____.

23. _____ are placeholders for fundamental types or user-defined types in function templates.

24. The element of chance can be introduced into computer applications by using the C++ Standard Library function _____.

Prelab Activities

Name:

Short Answer

Name: _____ **Date:** _____

Section: _____

In the space provided, answer each of the given questions. Your answers should be as concise as possible; aim for two or three sentences.

25. Compare and contrast pass-by-reference and pass-by-value.

26. Give an example of a function prototype. Identify the return type, the function name and the argument list. What is the purpose of the return type? What is the purpose of the argument list?

27. Provide an example of each of the four scopes for an identifier: function scope, file scope, block scope and function-prototype scope. Briefly discuss each scope.

Prelab Activities

Name:

Short Answer

28. Describe the differences between `rand` and `srand` and the role that each plays in generating random numbers.

29. What is an enumeration?

Prelab Activities

Name:

Programming Output

Name: _____ **Date:** _____

Section: _____

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

30. What is output by the given program segment? Assume that the `<cmath>` header file has been included.

```
1  double x = -5.49;
2
3  cout << ceil( x ) << endl << fabs( x ) << endl << floor( x );
```

Your answer:

31. What is output by the following program segment when function `f1` is invoked?

```
1  void f1()
2  {
3     int x = 5;
4
5     f2( x );
6     cout << x << endl;
7  } // end function f1
8
9  void f2( int x )
10 {
11    x += 5;
12    cout << x << endl;
13 } // end function f2
```

Your answer:

Prelab Activities

Name:

Programming Output

32. What is the output of

```
cout << mystery( 6, 2, 5 ) << endl;
```

assuming the following definition of `mystery`?

```
1  int mystery( int x, int y, int z )
2  {
3     int value = x;
4
5     if ( y > value )
6        value = y;
7
8     if ( z > value )
9        value = z;
10
11    return value;
12 } // end function mystery
```

Your answer:

33. What is output by the following program segment when function `f3` is called twice?

```
1  void f3()
2  {
3     static int x = 0;
4
5     ++x;
6     cout << x << endl;
7  } // end function f3
```

Your answer:

Prelab Activities

Name:

Programming Output

34. What is output by the following program?

```
1  #include <iostream>
2  using namespace std;
3
4  void f1();
5
6  int main()
7  {
8     int x = 0;
9
10    cout << "Initially, x = " << x << endl;
11    f1();
12    cout << "At the end, x = " << x << endl;
13 } // end main
14
15 // definition for f1
16 void f1()
17 {
18    int x = 3;
19
20    cout << "During call to f1, x = " << x << endl;
21
22 } // end function f1
```

Your answer:

35. What is the output of

```
cout << mystery2( 5, 4 ) << endl;
```

Prelab Activities

Name:

Programming Output

assuming the following definition of `mystery2`? [*Note:* This problem is intended for those students who have studied recursion in Sections 6.19–6.21 of *C++ How to Program, Seventh Edition*.]

```
1  int mystery2( int x, int y )
2  {
3     if ( y == 0 )
4        return x;
5
6     else if ( y < 0 )
7        return mystery2( x - 1, y + 1 );
8
9     else
10       return mystery2( x + 1, y - 1 );
11
12 } // end function mystery2
```

Your answer:

36. What is the output of

```
cout << mystery2( 5, -4 ) << endl;
```

assuming the same definition of `mystery2` used in *Programming Output Exercise 35*?

Your answer:

Prelab Activities

Name:

Correct the Code

Name: _____ **Date:** _____

Section: _____

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note:* It is possible that a program segment may contain multiple errors.]

37. The following program segment defines function `maximum`, which returns the largest of three integers:

```
1  int maximum( int x, int y, int z );
2  {
3     int max = x;
4
5     if ( y > max )
6        max = y;
7
8     if ( z > max )
9        max = x;
10
11    return max;
12 } // end function maximum
```

Your answer:

38. The following program segment should output five random numbers in the range from 1 to 6, inclusive:

```
1  for ( int i = 1; i <= 5; i++ ) {
2     cout << setw( 10 ) << 1 + srand() % 6;
```

Your answer:

Prelab Activities

Name:

Correct the Code

39. The following program segment creates an enumeration (`Status`), creates a variable of the enumeration's type and sets it to `WON`:

```
1  enum Status = { CONTINUE; WON; LOST };
2  Status myStatus = 1;
```

Your answer:

40. The following program should display three lines of text:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6     cout << "Before call to f1.\n";
7     f1();
8     cout << "After call to f1.\n";
9  } // end main
10
11 // f1 definition
12 void f1()
13 {
14    cout << "During call to f1.\n";
15 } // end function f1
```

Your answer:

Prelab Activities

Name:

Correct the Code

41. The following segment should define two functions:

```
1  void f2()
2  {
3     cout << "During call to f2.\n";
4
5     void f3()
6     {
7        cout << "During call to f3.\n";
8     }
9  } // end function f2
```

Your answer:

42. The given program segment should recursively calculate factorials. [*Note*: This problem is intended for those students who have studied recursion in Sections 6.19–6.21 of *C++ How to Program, Seventh Edition*.]

```
1  void factorial( unsigned long number )
2  {
3     if ( number <= 1 )
4        return 1;
5     else
6        return number * factorial( number + 1 );
7  } // end function factorial
```

Your answer:

Prelab Activities

Name:

Correct the Code

43. The following program should display a character input by the user:

```
1  #include <iostream>
2
3  void f4( int c );
4
5  int main()
6  {
7     char myChar;
8
9     cout << "Enter a character: ";
10    cin >> myChar;
11
12    f4( myChar )
13 } // end main
14
15 // f4 definition
16 void f4( char c )
17 {
18    cout << "You just entered the character: " << myChar << endl;
19    return myChar;
20 } // end function f4
```

Your answer:

Lab Exercises

Lab Exercise 1 — Prime Numbers

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 6.1)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up question. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 6 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Writing function prototypes to specify the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters and the order in which these parameters are expected.
- Making function calls and passing arguments to a function.
- Returning values from a function.

The follow-up question and activity will also give you practice:

- Writing a program that performs a similar task.

Description of the Problem

Write a program that inputs a series of integers and passes them one at a time to funciton `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.

Sample Output

```
Enter an integer: 4
4 is an even integer

Enter an integer: 3
3 is an odd integer

Enter an integer: 2
2 is an even integer
```

Lab Exercises

Name:

Lab Exercise 1 — Prime Numbers

Template

```
1  // Lab 1: even.cpp
2  // Determine whether inputs are odd or even.
3  #include <iostream>
4  using namespace std;
5
6  /* Write a function prototype for function even, which takes an int parameter
7     and returns a bool */
8
9  int main()
10 {
11    int x; // current input
12
13    // loop for 3 inputs
14    for ( int i = 1; i <= 3; i++ )
15    {
16       cout << "Enter an integer: ";
17       cin >> x;
18
19       // determine if input is even
20       if ( /* Write a call to function even to determine if x is even */ )
21          cout << x << " is an even integer\n\n";
22       else
23          cout << x << " is an odd integer\n\n";
24    } // end for
25
26    cout << endl;
27 } // end main
28
29 /* Define function even. Use the modulus operator to determine if an integer
30    is evenly divisible by two. */
```

Fig. L 6.1 | even.cpp.

Problem-Solving Tips

1. Function even contains an algorithm for determining if a number is even. Function even should return true if the number is even and false otherwise. If a number is divisible by two without a remainder, it is even. Use the modulus operator, %, to test for a remainder.
2. Be sure to follow the spacing and indentation conventions mentioned in the text.
3. If you have any questions as you proceed, ask your lab instructor for assistance.

Follow-Up Question and Activity

1. Write another function odd that returns true if an integer is odd and false otherwise. Modify main to use function odd instead of function even. The program's output should be identical to the one shown in the sample output for *Lab Exercise 1*.

Lab Exercises

Name:

Lab Exercise 2 — Reversing Digits

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 6.2)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 6 of *C++ How To Program, Seventh Edition*. In this lab, you will practice

- Writing function prototypes to specify the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters and the order in which the parameters are expected.
- Using multiple functions to perform specific tasks.

The follow-up questions and activities also will give you practice:

- Nesting function calls.
- Tracing through a program.
- Comparing call-by-reference to call-by-value.

Description of the Problem

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`.

Sample Output

```
Enter a number between 1 and 9999: 7631
The number with its digits reversed is: 1367
```

Lab Exercises

Name:

Lab Exercise 2 — Reversing Digits

Template

```
1  // Lab 2: reverse.cpp
2  // Reverse the digits of a number.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  /* Write prototype for reverseDigits */
8
9  int main()
10 {
11    int number; // input number
12
13    cout << "Enter a number between 1 and 9999: ";
14    cin >> number;
15
16    cout << "The number with its digits reversed is: ";
17
18    // find number with digits reversed
19    cout << /* Write call to reverseDigits */ << endl;
20 } // end main
21
22 // reverseDigits returns number obtained by reversing digits of n
23 /* Write function header for reverseDigits */
24 {
25    int reverse = 0; // reversed number
26    int divisor = 1000; // current divisor
27    int multiplier = 1; // current multiplier
28
29    // loop until single-digit number
30    while ( n > 9 )
31    {
32       // if n >= current divisor, determine digit
33       if ( n >= divisor )
34       {
35          // update reversed number with current digit
36          reverse += n / divisor * multiplier;
37          n %= divisor; // update n
38          /* Write a line of code that reduces divisor by a factor of 10 */
39          /* Write a line of code that increases multiplier by a factor of 10 */
40       } // end if
41       else // else, no digit
42          divisor /= 10; // update divisor
43    } // end while
44
45    reverse += n * multiplier;
46    return reverse; // return reversed number
47 } // end function reverseDigits
```

Fig. L 6.2 | reversedigits.cpp.

Lab Exercises

Name:

Lab Exercise 2 — Reversing Digits

Problem-Solving Tips

1. This problem requires that one integer be input by the user.
2. You should write a function called `reverseDigits` that takes the four-digit number as an argument.
3. The algorithm for `reverseDigits` is as follows: Using a `while` loop, isolate each individual digit and store it in its reverse position. This task is accomplished by dividing the number by `divisor` to get one digit and multiplying the number by its new position (e.g., one's position, ten's position, etc.). For example, the first digit is multiplied by one, the second digit is multiplied by ten, etc.
4. In the body of the loop, during each iteration, the number must be reduced by a factor of 10, the `divisor` must be reduced by a factor of 10 and the multiplier must be increased by a factor of 10.
5. Be sure to follow the spacing and indentation conventions mentioned in the text.
6. If you have any questions as you proceed, ask your lab instructor for assistance.

Follow-Up Questions and Activities

1. Add code to `main` so that it reverses `number` twice (i.e., reverse it once, then reverse the result of the first function call). Do not store the value of `number` after it has been reversed once. Is the number that has been reversed twice equal to the original number?

2. What happens if `divisor` is initialized to `100` instead of `1000`? Try to solve this problem on paper first; then compare your answer with the one given by the computer. Are they the same?

Lab Exercises

Name:

Lab Exercise 2 — Reversing Digits

3. Rewrite `reverseDigits` to take its arguments pass-by-reference. It should no longer return a value; it should just modify the number it is passed. [*Note:* This problem is intended for students who have studied pass-by-reference in Section 6.14 in *C++ How to Program, Seventh Edition*.]

Lab Exercises

Name:

Lab Exercise 3 — Greatest Common Divisor

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 6.3)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 6 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Writing function prototypes to specify the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters and the order in which the parameters are expected.
- Passing multiple arguments to a function.

The follow-up questions and activities also will give you practice:

- Using the conditional operator (`?:`).
- Passing arguments by reference.

Description of the Problem

The greatest common divisor (GCD) of two integers is the largest integer that evenly divides into each of the two integers. Write a function `gcd` that returns the greatest common divisor of two integers.

Lab Exercises

Name:

Lab Exercise 3 — Greatest Common Divisor

Sample Output

```
Enter two integers: 6 8
The greatest common divisor of 6 and 8 is 2

Enter two integers: 789 4
The greatest common divisor of 789 and 4 is 1

Enter two integers: 9999 27
The greatest common divisor of 9999 and 27 is 9

Enter two integers: 73652 8
The greatest common divisor of 73652 and 8 is 4

Enter two integers: 99 11
The greatest common divisor of 99 and 11 is 11
```

Template

```
1  // Lab 3: gcd.cpp
2  // Finds greatest common divisor (GCD) of 2 inputs.
3  #include <iostream>
4  using namespace std;
5
6  /* Write prototype for gcd */
7
8  int main()
9  {
10    int a; // first number
11    int b; // second number
12
13    // loop for 5 pairs of inputs
14    for ( int j = 1; j <= 5; j++ )
15    {
16       cout << "Enter two integers: ";
17       cin >> a >> b;
18
19       cout << "The greatest common divisor of "
20          << a << " and " << b << " is ";
21
22       // find greatest common divisor of a and b
23       cout <<  /* Write call for gdc */ << endl;
24    } // end for
25 } // end main
26
27 // gcd finds greatest common divisor of x and y
28 /* Write header for gcd */
29 {
30    int greatest = 1; // current greatest common divisor, 1 is minimum
31
32    // loop from 2 to smaller of x and y
33    for ( int i = 2; i <= ( ( x < y ) ? x : y ); i++ )
34    {
```

Fig. L 6.3 | gcd.cpp (Part 1 of 2).

Lab Exercises

Name:

Lab Exercise 3 — Greatest Common Divisor

```
35         // if current i divides both x and y
36         if ( /* Write condition to determine if both x and y are
37         divisible by i */ )
38            greatest = i; // update greatest common divisor
39      } // end for
40
41      /* Write a statement to return greatest */
42   } // end function gcd
```

Fig. L 6.3 | `gcd.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. The program requires that two integers be input. Write a loop that allows several pairs of integers to be input during execution. This will allow you to test the program more thoroughly.
2. The program should contain one function `gcd` that implements the greatest-common-divisor algorithm. Every value up to and including the smallest of the two numbers must be divided into both the numbers. Use the modulus operator to check for a remainder. If the remainder is zero for both numbers, a common divisor has been found. Return the greatest common divisor from the function.
3. Be sure to follow the spacing and indentation conventions mentioned in the text.
4. If you have any questions as you proceed, ask your lab instructor for assistance.

Follow-Up Questions and Activities

1. Rewrite the `for` loop in your solution that corresponds to line 37 in the program template so that it does not use the conditional operator (`?:`). [*Hint:* You will need to use the logical AND operator (`&&`).]
2. Create another function called `input`, that receives references to two integer variables and replaces their values with two values input by the user. Use this function to input the values for `a` and `b`.

Lab Exercises

Name:

Debugging

Name: _____ **Date:** _____

Section: _____

The program (Fig. L 6.4) in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code has been corrected.

Sample Output

[*Note:* This program uses random-number generation. The output will vary each time the program executes. Run the program multiple times to confirm that it is working properly.]

```
13:20:20
01:20:20 PM
```

```
00:04:15
00:04:15 AM
```

```
04:38:58
04:38:58 AM
```

Broken Code

```
1  // Debugging: time.cpp
2  #include <iostream>
3  #include <cmath>
4  #include <ctime>
5  using namespace std;
6
7  void generateTime( int, int, int );
8  void printUniversal( int, int, int );
9  void printStandard( int, int, int );
10 bool morning( int, int, int );
11 bool afterNoon( int, int, int );
12
13 int main()
14 {
15    int hour; // hour variable
16    int minute; // minute variable
17    int second; // second variable
18
19    srand( time() );
20
21    generateTime( hour, minute, second );
```

Fig. L 6.4 | Time1.cpp: (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
22
23     printUniversal( hour, minute, second );
24     printStandard( hour, minute, second );
25  } // end main
26
27  void generateTime( int &h, int &m, int &s )
28  {
29     h = rand() % 24 + 0;
30     m = rand() % 48 + 0;
31     s = rand() % 60 + 0;
32  }
33
34  void printUniversal( int h, int m, int s )
35  {
36     cout << ( h < 10 ? "0" : "" ) << h << "?";
37     cout << ( m < 10 ? "0" : "" ) << m << ":";
38     cout << ( s < 10 ? "0" : "" ) << s << endl;
39  }
40
41  void printStandard( int h, int m, int s )
42  {
43     cout << ( h % 12 < 10 ? "0" : "" ) << h % 12 << ":";
44     cout << ( m < 10 ? "0" : "" ) << m << ":";
45     cout << ( s < 10 ? "0" : "" ) << s << " ";
46     if ( morning( h, m, s )
47        cout << "AM" << endl;
48     if ( afternoon( h, m, s )
49        cout << "PM" << endl;
50  }
51
52  bool morning( int h, int m, int s )
53  {
54     return h < 12;
55  }
56
57  bool afternoon( int h, int m, int s )
58  {
59     return s >= 12;
60  }
```

Fig. L 6.4 | `time.cpp`. (Part 2 of 2.)

Postlab Activities

Coding Exercises

Name: _____ **Date:** _____

Section: _____

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For each of the following problems, write a program or a program segment that performs the specified action.

1. Write a function that takes an integer parameter and returns the value of that integer modulus 7 (e.g., `n % 7`).

2. Write a short program that simulates 10 flips of a coin. Produce random results.

Postlab Activities

Name:

Coding Exercises

3. Change your coin-flipping program into a game: Every time heads comes up, the player wins $1; every time tails comes up, the player loses $1. Write a function `calculateTotal` that adds up the player's winnings after 10 flips of the coin.

4. Write a function `divides` that takes two integer parameters and returns `true` if the first integer divides evenly into the second one (i.e., the second divided by the first yields the remainder `0`).

Postlab Activities

Name:

Coding Exercises

5. Write a function `incrementByTen` that uses a `static` variable `count`. Each call to the function should display the value of `count` and then increment `count` by `10`. Initialize `count` to be `0` at the start of the program.

6. Write a function `halve` that takes an integer parameter and returns the value of that integer divided by `2`.

Postlab Activities

Name:

Coding Exercises

7. Modify `halve` to receive its parameter by reference. This version of `halve` should not return a value and should instead alter the original value in its parameter. [*Note:* This problem is intended for those students who have studied pass-by-reference in Section 6.14 in *C++ How to Program, Seventh Edition*.]

8. Write a function `minimumValue` that prints and returns the minimum of its five integer parameters.

Postlab Activities

Name:

Programming Challenges

Name: _____ **Date:** _____

Section: _____

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a C++ program for each of the problems in this section. The answers to these problems are available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. 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 number of 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 output should appear in the following format:

```
Enter the hours parked for three cars: 1.5 4.0 24.0
 Car      Hours       Charge
 1          1.5         2.00
 2          4.0         2.50
 3         24.0        10.00
 TOTAL     29.5        14.50
```

Hints:

- Use a `for` loop to prompt the user for the number of hours parked for each of the three customers.
- Declare variables to store the total number of hours and the total charges for each customer.
- The variables for all charges and numbers of hours should be of type `double`.
- Function `calculateCharges` should use a nested `if...else` statement to determine the customer charge.

2. 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` loop to control the calculation. Do not use any math library functions.

Hints:

- Use a `for` loop to loop *X* times where *X* is the value of `exponent`.
- During each iteration of the loop, multiply variable `product` by the value of the `base` parameter.
- Return the value of `product` after *X* iterations of the `for` loop.

Postlab Activities

Name:

Programming Challenges

- Sample output:

```
Enter base and exponent: 5 2
5 to the power 2 is: 25
```

3. Computers are playing an increasing role in education. Write a program that will help an elementary school student learn multiplication. Use `rand` to produce two positive one-digit integers. The program should then output a question using the numbers, such as:

   ```
   How much is 6 times 7?
   ```

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

Hints:

- Use sentinel-controlled repetition to limit the number of questions that the program asks the user. At the same time, check that the user has entered the correct answer.
- Seed the `rand` function with `srand` to randomize the program
- Sample output:

```
Enter -1 to End.
How much is 4 times 9 (-1 to End)? 36
Very good!

How much is 7 times 0 (-1 to End)? 0
Very good!

How much is 7 times 8 (-1 to End)? 55
No. Please try again.
? 56
Very good!

How much is 5 times 0 (-1 to End)? -1
That's all for now. Bye.
```