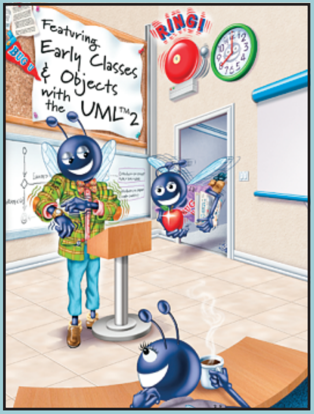
5



Control Statements: Part 2

OBJECTIVES

In this chapter you will learn:

- The essentials of counter-controlled repetition.
- To use the `for` and `do`...`while` repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the `switch` selection statement.
- To use the `break` and `continue` program control statements to alter the flow of control.
- To use the logical operators to form complex conditional expressions in control statements.
- To avoid the consequences of confusing the equality and assignment operators.

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	
Short Answer	22, 23, 24, 25, 26	
Programming Output	27, 28, 29, 30, 31, 32	
Correct the Code	33, 34, 35, 36, 37, 38, 39	
Lab Exercises		
Lab Exercise 1 — Integer Average	YES NO	
Follow-Up Question and Activity	1	
Lab Exercise 2 — Asterisk Triangles	YES NO	
Follow-Up Question and Activity	1	
Lab Exercise 3 — Pythagorean Triples	YES NO	
Follow-Up Questions and Activities	1, 2, 3, 4	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3, 4, 5, 6, 7, 8, 9	
Programming Challenges	10, 11, 12	

Prelab Activities

Matching

Name: ____________________ **Date:** ____________________

Section: ____________________

After reading Chapter 5 of *C++ How to Program: Fifth 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. Controlling expression	a)	The default display values for `bool` values.
___	2. `for`	b)	Format settings that stay in effect until they are changed.
___	3. `default` case	c)	A convenient control statement for performing counter-controlled repetition.
___	4. `pow`	d)	Causes immediate exit from a repetition statement.
___	5. Sticky settings	e)	A standard math library function.
___	6. `break`	f)	Logical AND.
___	7. `continue`	g)	Logical OR.
___	8. `&&`	h)	Where a variable can be used in a program.
___	9. `||`	i)	Specifies the field width in which the next value output should appear.
___	10. Scope	j)	The `switch` statement compares the value of this with each `case` label.
___	11. Stream manipulator `setw`	k)	Skips to the next iteration in a repetition statement.
___	12. `1` and `0`	l)	An optional part of a `switch` statement.

Prelab Activities

Name:

Fill in the Blank

Name: ____________________ **Date:** ____________________

Section: ____________________

Fill in the blanks in each of the following statements:

13. Typically, `for` statements are used for __________ repetition and `while` statements are used for __________ repetition.

14. The __________ repetition statement tests the loop-continuation condition at the end of the loop.

15. Listing `case`s consecutively with __________ between them enables the `case`s to perform the same set of statements.

16. The `switch` statement can be used only for testing constant __________ expressions.

17. C++ provides several data types to represent __________—`int`, `char`, `short` and `long`.

18. The `switch` selection statement differs from other control statements in that it does not require __________ around multiple statements in each case.

19. When used as a condition, any __________ value implicitly converts to `true`; __________ implicitly converts to `false`.

20. An expression containing `&&` or `||` operators evaluates only until the truth or falsehood of the expression is known. This performance feature for the evaluation of logical AND and logical OR expressions is called __________ evaluation.

21. In addition to selection and repetition statements, the __________ and __________ statements are used with control statements to alter the flow of control.

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.

22. Explain the difference between a `while` statement and a `do...while` statement. Draw an activity diagram for each.

23. Under what circumstances would the `default` case in a `switch` statement execute? What would happen under those circumstances if there is not a `default` case?

Prelab Activities

Name:

Short Answer

24. Fill in the third column in the following tables:

expression1	expression2	expression1 || expression2
`false`	`false`	____________
`false`	`true`	____________
`true`	`false`	____________
`true`	`true`	____________

expression1	expression2	expression1 && expression2
`false`	`false`	____________
`false`	`true`	____________
`true`	`false`	____________
`true`	`true`	____________

25. What happens when a `break` or `continue` statement is encountered inside a repetition statement. Are their effects different for `while` and `do...while` loops versus `for` loops?

26. When must you use braces (`{}`) in conjunction with a selection or repetition statement?

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.]

27. What is output by the following `switch` statement?

```
int x = 1;

switch ( x )
{
   default:
      cout << "none ";
   case 1:
      cout << "one ";
   case 2:
      cout << "two ";
   case 3:
      cout << "three ";
   case 4:
      cout << "four ";
}
```

Your answer:

28. What is output by the following `for` loop?

```
for ( int i = 0; i < 5; i++ )
   cout << i << " ";
```

Your answer:

Prelab Activities

Name:

Programming Output

29. What is output by the following program segment?

```cpp
int x = 1;
int y = 2;
int z = 3;

if ( x == 2 || y + 1 == z )
   cout << "A";

if ( z / x < y && x + y > z )
   cout << "B";

if ( y - z <= 0 || x - z <= 0 )
   cout << "C";
```

Your answer:

30. What is output by the following program segment?

```cpp
for ( int i = 1; i <= 10; i++ )
{
   switch ( i )
   {
      case 1:
         cout << "The value of x is 1\n";
         break;
      case 4:
         cout << "The value of x is 4\n";
      case 6:
         cout << "The value of x is 6\n";
         break;
      default:
         cout << "The value of x is neither 1, 4 nor 6\n";
   } // end switch
} // end for
```

Prelab Activities

Name:

Programming Output

Your answer:

31. What is output by the following program segment?

```
int x;

for ( x = 1; x <= 10; x++ )
{
   if ( x == 7 )
      break;

   if ( x == 3 )
      continue;

   cout << x << " ";
}

cout << endl << "The final value of x is: " << x << endl;
```

Your answer:

32. What is the output of the following program segment?

```
int x = 1;

for ( ; x <= 10; x++ );
   cout << "The value of x is: " << x << endl;
```

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, syntax 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.]

33. The following program segment should calculate the product of the integers between `1` and `5`, inclusive.

```
for ( int i = 1; i < 5; i++ )
{
   int product = 1;

   product *= i;
}
```

Your answer:

34. The following `for` loop should divide `i` by `i - 1`, using integer division, and print the result.

```
while ( int i = 1; i <= 5; i++ )
   cout << i / ( i - 1 ) << " ";
```

Your answer:

Prelab Activities

Name:

Correct the Code

35. The following `for` loop should print all the integers between `5` and `1000`, inclusive, that are evenly divisible by `5`.

```
int i = 5;

for ( ; ; i += 5 )
{
   cout << i << " ";

   if ( i = 1000 )
      break;
}
```

Your answer:

36. The following `switch` statement should print either `x is 5`, `x is 10` or `x is neither 5 nor 10`.

```
switch ( x )
{
   case: 5
      cout << "x is 5\n";

   case: 10
      cout << "x is 10\n";

   case default:
      cout << "x is neither 5 nor 10\n";
}
```

Your answer:

Prelab Activities

Name:

Correct the Code

37. The following program segment should print the sum of consecutive odd and even integers between 1 and 10, inclusive. The expected output is shown below the code segment.

```
for ( int i = 1, j = 2; i <= 10 && j <= 10; i++, j++ )
   cout << i << " + " << j << " = " << i + j << endl;
```

```
1 + 2 = 3
3 + 4 = 7
5 + 6 = 11
7 + 8 = 15
9 + 10 = 19
```

Your answer:

38. The following for loop should compute the product of i times 2, plus 1. For example, if the counter is 4, the program should print 4 * 2 + 1 = 9. It should loop from 1 to 10.

```
for ( int i = 1, i = 10, i++ )
   cout << i << " * 2 + 1 = " << ++( i * 2 ) << endl;
```

Your answer:

Prelab Activities

Name:

Correct the Code

39. The following program segment should print the value of x * y until either x reaches 5 or y reaches 5:

```
int x = 1;

for ( int y = 2; x == 5 && y == 5; y++ )
{
   cout << x * y << endl;
   x++;
}
```

Your answer:

Lab Exercises

Lab Exercise 1—Integer Average

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 5.1)
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 at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 5 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Using sentinel-controlled repetition with a `for` loop.

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

- Using counter-controlled repetition with a `for` loop.

Description of the Problem

Write a program that uses a `for` statement to calculate and print the average of several integers. Assume the last value read is the sentinel `9999`. A typical input sequence might be

```
10 8 11 7 9 9999
```

indicating that the program should calculate the average of all the values preceding `9999`.

Sample Output

```
Enter integers (9999 to end):
10 8 11 7 9 9999

The average is: 9
```

Lab Exercises

Name:

Lab Exercise 1—Integer Average

Template

```cpp
// Lab 1: IntegerAverage.cpp
// Calculate the average of several integers.

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
   int value; // current value
   int count = 0; // number of inputs
   int total; // sum of inputs

   // prompt for input
   cout << "Enter integers (9999 to end):" << endl;
   cin >> value;

   // loop until sentinel value read from user
   /* Write a for header to initialize total to 0
      and loop until value equals 9999 */
   {
      /* Write a statement to add value to total */
      /* Write a statement to increment count */

      cin >> value; // read in next value
   } // end for

   // if user entered at least one value
   if ( count != 0 )
      cout << "\nThe average is: "
         << /* Convert total to a double and divide it by count */ << endl;
   else
      cout << "\nNo values were entered." << endl;

   return 0; // indicate program ended successfully
} // end main
```

Fig. L 5.1 | `IntegerAverage.cpp`.

Problem-Solving Tips

1. When used for sentinel-controlled repetition, a `for` loop can be written much like a `while` loop, using the same loop-continuation condition as a `while` loop.
2. When performing sentinel-controlled repetition, a `for` loop does not need to increment any counter variable. But it can still initialize a variable if so desired.
3. If you have any questions as you proceed, ask your lab instructor for help.

Follow-Up Question and Activity

1. Modify the program to perform counter-controlled repetition. Assume that the first integer entered by the user represents the number of subsequent integers that the user will input to be averaged.

Lab Exercises

Name:

Lab Exercise 2 — Asterisk Triangles

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 5.2)
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 at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 5 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Using counter-controlled repetition with `for` loops.
- Using nested `for` loops.

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

- Using nested `for` loops.

Description of the Problem

Write a program that uses `for` statements to print the following patterns separately, one below the other. Use `for` loops to generate the patterns. All asterisks (`*`) should be printed by a single statement of the form `cout << '*';` (this causes the asterisks to print side by side). [*Hint:* The last two patterns require that each line begin with an appropriate number of blanks.]

```
(a)             (b)             (c)             (d)
*               **********      **********               *
**              *********        *********              **
***             ********          ********             ***
****            *******            *******            ****
*****           ******              ******           *****
******          *****                *****          ******
*******         ****                  ****         *******
********        ***                    ***        ********
*********       **                      **       *********
**********      *                        *      **********
```

Lab Exercises

Name:

Lab Exercise 2 — Asterisk Triangles

Sample Output

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

**********
*********
********
*******
******
*****
****
***
**
*

**********
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *

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

Template

```cpp
// Lab 2: AsteriskTriangles.cpp
// Draw four triangles composed of asterisks.

#include <iostream>
using std::cout;
using std::endl;

```

Fig. L 5.2 | `AsteriskTriangles.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Asterisk Triangles

```
int main()
{
   int row; // the row position
   int column; // the column position
   int space; // number of spaces to print

   // first triangle
   /* Write a for header to iterate row from 1 to 10 */
   {
      /* Write a for header to iterate column from 1 to row */
         cout << "*";

      cout << endl;
   } // end for

   cout << endl;

   // second triangle
   /* Write a for header to iterate row from 10 down to 1 */
   {
      /* Write a for header to iterate column from 1 to row */
         cout << "*";

      cout << endl;
   } // end for

   cout << endl;

   // third triangle
   /* Write a for header to iterate row from 10 down to 1 */
   {
      /* Write a for header to iterate space from 10 down to one more than row */
         cout << " ";

      /* Write a for header to iterate column from 1 to row */
         cout << "*";

      cout << endl;
   } // end for

   cout << endl;

   // fourth triangle
   /* Write a for header to iterate row from 10 down to 1 */
   {
      /* Write a for header to iterate space from 1 to one less than row */
         cout << " ";

      /* Write a for header to iterate column from 10 down to row */
         cout << "*";

      cout << endl;
   } // end for

   return 0; // indicate program ended successfully
} // end main
```

Fig. L 5.2 | `AsteriskTriangles.cpp`. (Part 2 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Asterisk Triangles

Problem-Solving Tips

1. Use nested `for` loops—the outer loop will iterate over the rows and the inner loop will iterate over the columns.
2. For pattern (a), simply output as many asterisks for each row as that row number.
3. For pattern (b), have the `row` counter count backwards from `10` to `1` and output as many asterisks for each row as that row number.
4. For pattern (c), have the `row` counter count backwards from `10` to `1` and output `10 - row` spaces followed by `row` asterisks.
5. For pattern (d), have the `row` counter count backwards from `10` to `1` and output `row - 1` spaces followed by `10 - row + 1` asterisks.
6. If you have any questions as you proceed, ask your lab instructor for help.

Follow-Up Question and Activity

1. Combine your code from the four separate problems into a single program that prints all four patterns side by side by making clever use of nested `for` loops.

Lab Exercises

Name:

Lab Exercise 3 — Pythagorean Triples

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 5.3)
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 at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 5 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Using counter-controlled repetition.
- Using “brute force” to solve a problem.
- Nesting `for` loops.

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

- Using `break` statements.
- Using `continue` statements.
- Using `long` integers.

Description of the Problem

A right triangle can have sides that are all integers. A set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Find all Pythagorean triples for `side1`, `side2` and `hypotenuse` all no larger than 500. Use a triple-nested `for` loop that tries all possibilities. This is an example of brute force computing. You will learn in more advanced computer-science courses that there are many interesting problems for which there is no known algorithmic approach other than using sheer brute force.

Lab Exercises

Name:

Lab Exercise 3 — Pythagorean Triples

Sample Output

```
Side 1  Side 2  Side3
3       4       5
5       12      13
6       8       10
7       24      25
8       15      17

...

300     400     500
319     360     481
320     336     464
325     360     485
340     357     493
A total of 386 triples were found.
```

Template

```cpp
// Lab 3: pythagorean.cpp
// Find Pythagorean triples using brute force computing.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int count = 0; // number of triples found
   long int hypotenuseSquared; // hypotenuse squared
   long int sidesSquared; // sum of squares of sides

   cout << "Side 1\tSide 2\tSide3" << endl;

   // side1 values range from 1 to 500
   /* Write a for header for side1 */
   {
      // side2 values range from current side1 to 500
      /* Write a for header for side2 */
      {
         // hypotenuse values range from current side2 to 500
         /* Write a for header for hypotenuse */
         {
            // calculate square of hypotenuse value
            /* Write a statement to calculate hypotenuseSquared */

            // calculate sum of squares of sides
            /* Write a statement to calculate the sum of the sides Squared */

            // if (hypotenuse)^2 = (side1)^2 + (side2)^2,
            // Pythagorean triple
            if ( hypotenuseSquared == sidesSquared )
            {
```

Fig. L 5.3 | `pythagorean.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 3 — Pythagorean Triples

```
                  // display triple
                  cout << side1 << '\t' << side2 << '\t'
                     << hypotenuse << '\n';
                  count++; // update count
               } // end if
            } // end for
         } // end for
      } // end for

      // display total number of triples found
      cout << "A total of " << count << " triples were found." << endl;
      return 0; // indicate successful termination
   } // end main
```

Fig. L 5.3 | `pythagorean.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. This program does not require any input from the user.
2. This program can take several minutes to run, depending on your computer's processor speed. If you have a CPU monitor available on your system, it is worth taking a look at it when this program executes.
3. Do not be concerned that you are trying values that do not seem to make sense, such as a 1–1–500 triangle. Remember that brute-force techniques try all possible values.
4. The formula for the Pythagorean Theorem is $hypotenuse^2 = (side\ 1)^2 + (side\ 2)^2$.
5. To avoid producing duplicate Pythagorean triples, start the second `for` loop at `side2 = side1` and the third `for` loop at `hypotenuse = side2`. This way, when a Pythagorean triple is found, `side1` will be the shortest side of the triangle and `hypotenuse` will be the longest side.
6. Be sure to follow the spacing and indentation conventions mentioned in the text. Before and after each control statement, place a line of vertical space to make the control statement stand out. Indent all the body statements of `main`, and indent all of the body statements of each control statement.
7. If you have any questions as you proceed, ask your lab instructor for help.

Follow-Up Questions and Activities

1. How many times did this program execute the innermost `for` loop? Add another counter to the program that counts the number of times this loop iterates. Declare a new variable of type `long`, named `loopCounter` and initialize it to `0`. Then add a statement in the innermost `for` statement that increments `loopCounter` by `1`. Before exiting the program, print the value of `loopCounter`. Do the numbers match?

Lab Exercises

Name:

Lab Exercise 3 — Pythagorean Triples

2. Add a `break` statement to the program inside the innermost `for` loop. This `break` statement should be called after the 20th Pythagorean triple is found. Explain what happens to the program after the 20th triple is found. Are all three `for` loops exited, or just the innermost one? What happens when the `break` statement is placed inside the middle loop? The outermost loop?

3. Add a `continue` statement to the program that prevents a Pythagorean triple from being found when `side1` is equal to `8`. Using your solution to *Follow-Up Question 1*, calculate how many times this new program executes the innermost `for` loop. Explain why the `continue` statement affected the output.

4. Explain why a `long` variable is used for `hypotenuseSquared` and `sideSquared`. Modify the program so that they are both of type `short` instead of type `long`. Rerun the program. What happens?

Lab Exercises

Name:

Debugging

Name: ______________________ **Date:** ______________________

Section: ______________________

The program 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

```
i is now equal to 1
        j is now equal to 0
                i + j = 1       i - j = 1
                i * j = 0       i ^ j = 1
        j is now equal to 1
                i + j = 2       i - j = 0
                i * j = 1       i ^ j = 1
                i / j = 1       i % j = 0
        j is now equal to 2
                i + j = 3       i - j = -1
                i * j = 2       i ^ j = 1
                i / j = 0.5     i % j = 1
        j is now equal to 3
                i + j = 4       i - j = -2
                i * j = 3       i ^ j = 1
                i / j = 0.33    i % j = 1
i is now equal to 2
        j is now equal to 0
                i + j = 2       i - j = 2
                i * j = 0       i ^ j = 1
        j is now equal to 1
                i + j = 3       i - j = 1
                i * j = 2       i ^ j = 2
                i / j = 2       i % j = 0
        j is now equal to 2
                i + j = 4       i - j = 0
                i * j = 4       i ^ j = 4
                i / j = 1       i % j = 0
        j is now equal to 3
                i + j = 5       i - j = -1
                i * j = 6       i ^ j = 8
                i / j = 0.67    i % j = 2

The final values of i and j are: 3 and 4
```

Lab Exercises

Name:

Debugging

Broken Code

```
// Debugging: debugging.cpp

#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setprecision;

int main()
{
   int i = 1;
   double a;
   double b;

   cout << setprecision( 2 );

   for ( int i; i <= 2; i++ )
      cout << "i is now equal to " << i << endl;

      for ( int j; j <= 3; j++ )
      {
         cout << "\tj is now equal to " << j << endl;

         cout << "\t\ti + j = " << i + j << "\ti - j = "
              << i - j << endl;
         cout << "\t\ti * j = " << i * j << "\ti ^ j = "
              << pow( i, j ) << endl;

         if ( j = 0 )
            continue;

         else
         {
            a = i;
            b = j;
            cout << "\t\ti / j = " << a / b
                    "\ti % j = " << a % b << endl;
         } // end else

      } // end for

   cout << "\nThe final values of i and j are: " << i
        << " and " << j << endl;

   return 0;

} // end main
```

Fig. L 5.4 | `debugging.cpp`.

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 successfully 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 `for` loop that displays all the odd integers from `1` to `100`, inclusive.

2. Write a `do...while` loop that counts from `10` to `0` and displays each value.

3. Write a program that inputs an integer between `1` and `5` and uses a `switch` statement to display the number's corresponding letter in the alphabet (i.e., A, B, C, D or E).

Postlab Activities

Name:

Coding Exercises

4. Write a `while` loop that sums all the integers between `1` and `10`, inclusive, except for `3` and `6`. Display the sum.

5. Write a sentinel-controlled loop (use a sentinel value of `-1`) that contains statements which input and output integers. Do not print the number if it is either `7` or `63`.

Postlab Activities

Name:

Coding Exercises

6. Write a loop that reads in a maximum of 10 numbers and sums them. If the user enters the sentinel value `-1`, terminate the loop. Display the sum.

7. Write a program that computes and prints the average of the integers between `1` and `10` inclusive. Display the number as a fixed decimal with three digits of precision.

Postlab Activities

Name:

Coding Exercises

8. Write a counter-controlled `for` loop that iterates from `1` to `10` and displays the value of its counter. Terminate the loop when the counter has a value of `6`.

9. Modify your solution to *Coding Exercise 8* to use a `continue` statement such that every value except `6` is displayed.

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 at `www.deitel.com` and `www.prenhall.com/deitel`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. One interesting application of computers is the drawing of graphs and bar charts. Write a program that reads five numbers (each between 1 and 30). Assume that the user enters only valid values. For each number that is read, your program should print a line containing that number of adjacent asterisks. For example, if your program reads the number 7, it should print *******.

```
Enter 5 numbers between 1 and 30: 16 12 8 27 9
****************
************
********
***************************
*********
```

2. A company pays its employees as managers (who receive a fixed weekly salary), hourly workers (who receive a fixed hourly wage for up to the first 40 hours they work and "time and a half"—1.5 times their hourly wage—for overtime hours worked), commission workers (who receive $250 plus 5.7 percent of their gross weekly sales), or pieceworkers (who receive a fixed amount of money per item for each of the items they produce—each pieceworker in this company works on only one type of item). Write a program to compute the weekly pay for each employee. You do not know the number of employees in advance. Each type of employee has its own pay code: Managers have code 1, hourly workers have code 2, commission workers have code 3 and pieceworkers have code 4. Use a `switch` to compute each employee's pay according to that employee's paycode. Within the `switch`, prompt the user (i.e., the payroll clerk) to enter the appropriate facts your program needs to calculate each employee's pay according to that employee's paycode. Sample output is provided next. Model your code to produce these results.

Postlab Activities

Name:

Programming Challenges

```
Enter paycode (-1 to end): 3
Commission worker selected.
Enter gross weekly sales: 4000
Commission Worker's pay is $478.00

Enter paycode (-1 to end): 2
Hourly worker selected.
Enter the hourly salary: 4.50
Enter the total hours worked: 20
Worker's pay is $90.00

Enter paycode (-1 to end): 4
Pieceworker selected.
Enter number of pieces: 50
Enter wage per piece: 3
Pieceworker's pay is $150.00

Enter paycode (-1 to end): -1
```

3. Write a program that prints the following diamond shape. You may use output statements that print either a single asterisk (*) or a single blank. Maximize your use of repetition (with nested `for` statements) and minimize the number of output statements.

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