Control Statements: Part 2; Logical Operators

5

Objectives

In this chapter you'll:

- Learn the essentials of counter-controlled iteration.
- Use the `for` and `do...while` iteration statements to execute statements in a program repeatedly.
- Understand multiple selection using the `switch` selection statement.
- Use the `break` and `continue` program-control statements to alter the flow of control.
- Use the logical operators to form compound conditions in control statements.
- Understand the representational errors associated with using floating-point data types to hold monetary values.
- Understand some of the challenges of processing monetary amounts as we begin building a `DollarAmount` class, which uses integers and integer arithmetic to represent and manipulate monetary amounts.

Self-Review Exercises

5.1 Fill in the blanks in each of the following statements:

a) Typically, _____ statements are used for counter-controlled iteration and _____ statements for sentinel-controlled iteration.

ANS: `for`, `while`.

b) The `do...while` statement tests the loop-continuation condition _____ executing the loop's body; therefore, the body always executes at least once.

ANS: after.

c) The _____ statement selects among multiple actions based on the possible values of an integer variable or expression.

ANS: `switch`.

d) The _____ statement, when executed in an iteration statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

ANS: `continue`.

e) The _____ operator can be used to ensure that two conditions are *both* true before choosing a certain path of execution.

ANS: `&&` (conditional AND).

f) If the loop-continuation condition in a `for` header is initially _____, the program does not execute the `for` statement's body.

ANS: `false`.

5.2 State whether each of the following is *true* or *false*. If the answer is *false*, explain why.

a) The `default` case is required in the `switch` selection statement.

ANS: False. The `default` case is optional. Nevertheless, it's considered good software engineering to always provide a `default` case.

b) The `break` statement is required in the `default` case of a `switch` selection statement to exit the `switch` properly.

ANS: False. The `break` statement is used to exit the `switch` statement. The `break` statement is not required when the `default` case is the last case. Nor will the `break` statement be required if having control proceed with the next case makes sense.

c) The expression `(x > y && a < b)` is `true` if either the expression `x > y` is `true` or the expression `a < b` is `true`.

ANS: False. When using the `&&` operator, both of the relational expressions must be `true` for the entire expression to be `true`.

d) An expression containing the `||` operator is `true` if either or both of its operands are `true`.

ANS: True.

5.3 Write a C++ statement or a set of C++ statements to accomplish each of the following:

a) Sum the odd integers between 1 and 99 using a `for` statement. Use the `unsigned int` variables `sum` and `count`.

ANS:

```
unsigned int sum{0};
for (unsigned int count{1}; count <= 99; count += 2) {
      sum += count;
}
```

b) Print the value `333.546372` in a 15-character field with precisions of `1`, `2` and `3`. Print each number on the same line. Left-justify each number in its field. What three values print?

ANS:
```
cout << fixed << left
   << setprecision(1) << setw(15) << 333.546372
   << setprecision(2) << setw(15) << 333.546372
   << setprecision(3) << setw(15) << 333.546372 << endl;
```
Output is:
```
333.5          333.55         333.546
```

c) Calculate the value of `2.5` raised to the power `3` using function `pow`. Print the result with a precision of `2` in a field width of `10` positions. What prints?

ANS:
```
cout << fixed << setprecision(2) << setw(10) << pow(2.5, 3) << endl;
```
Output is:
```
     15.63
```

d) Print the integers from 1 to 20 using a `while` loop and the `unsigned int` counter variable `x`. Print only 5 integers per line. [*Hint:* When `x % 5` is 0, print a newline character; otherwise, print a tab character.]

ANS:
```
unsigned int x{1};
while (x <= 20) {
   if (x % 5 == 0) {
      cout << x << endl;
   }
   else {
      cout << x << '\t';
   }

   ++x;
}
```

e) Repeat Exercise 5.3(d) using a `for` statement.

ANS:
```
for (unsigned int x = 1; x <= 20; ++x) {
   if (x % 5 == 0) {
      cout << x << endl;
   }
   else {
      cout << x << '\t';
   }
}
```

5.4 Find the errors in each of the following code segments and explain how to correct them.

a)
```
unsigned int x{1};
while (x <= 10); {
   ++x;
}
```

ANS: *Error:* The semicolon after the `while` header causes an infinite loop.
Correction: Delete the semicolon after the `while` header.

b)
```
for (double y{0.1}; y != 1.0; y += .1) {
   cout << y << endl;
}
```
ANS: *Error:* Using a floating-point number to control a `for` iteration statement.
Correction: Use an `unsigned int` and perform the proper calculation to get the values.
```
for (unsigned int y = 1; y != 10; ++y) {
   cout << (static_cast<double>(y) / 10) << endl;
}
```

c)
```
switch (n) {
   case 1:
      cout << "The number is 1" << endl;
   case 2:
      cout << "The number is 2" << endl;
      break;
   default:
      cout << "The number is not 1 or 2" << endl;
}
```
ANS: *Error:* Missing `break` statement in the first `case`.
Correction: Add a `break` statement at the end of the first `case`. This is not an error if you want the statement of `case 2:` to execute every time the `case 1:` statement executes.

d) The following code should print the values 1 to 10.
```
unsigned int n{1};
while (n < 10) {
   cout << n++ << endl;
}
```
ANS: *Error:* Improper relational operator used in the loop-continuation condition.
Correction: Use `<=` rather than `<`, or change `10` to `11`.

Exercises

NOTE: Solutions to the programming exercises are located in the **`ch05solutions`** *folder.*

5.5 Describe the four basic elements of counter-controlled iteration.

ANS: The four elements of counter-controlled repetition are: a control variable (or loop counter); the control variable's initial value; the control variable's increment that's applied during each iteration of the loop and the loop-continuation condition that determines if looping should continue.

5.6 Compare and contrast the `while` and `for` iteration statements.

ANS: Both iteration statements execute their bodies zero or more times. The `for` statement can declare all the elements of counter-controlled repetition in its header, whereas the `while` statement header contains only a required loop-continuation condition. In the `for` statement, eliminating the loop-continuation condition results in an infinite loop, whereas in a `while` statement this is a syntax error.

5.7 Discuss a situation in which it would be more appropriate to use a `do...while` statement than a `while` statement. Explain why.

ANS: A `do...while` statement executes its body zero or more times, so if you want to guarantee that the loop body executes at least once, you'd choose a `do...while` statement over a `while` statement.

5.8 Compare and contrast the `break` and `continue` statements.

ANS: The `break` statement terminates a switch or a loop immediately. The `continue` statement terminates the current iteration of a loop and moves to the next iteration.

5.9 *(Find the Code Errors)* Find the error(s), if any, in each of the following:

a)
```
For (unsigned int x{100}, x >= 1, ++x) {
   cout << x << endl;
}
```

ANS: `For` should be `for`. The commas should be semicolons. The `++` should be a decrement such as `--`.

b) The following code should print whether integer `value` is odd or even:

```
switch (value % 2) {
   case 0:
      cout << "Even integer" << endl;
   case 1:
      cout << "Odd integer" << endl;
}
```

ANS: `case 0` needs a `break` statement.

c) The following code should output the odd integers from 19 to 1:

```
for (unsigned int x{19}; x >= 1; x += 2) {
   cout << x << endl;
}
```

ANS: `+=` should be `-=`.

d) The following code should output the even integers from 2 to 100:

```
unsigned int counter{2};

do {
   cout << counter << endl;
   counter += 2;
} While (counter < 100);
```

ANS: `While` should be `while`. Operator `<` should be `<=`.

5.10 What does the following program do?

```
1  // Exercise 5.10: Printing.cpp
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6     for (int i{1}; i <= 10; i++) {
7        for (int j{1}; j <= 5; j++) {
8           cout << '@';
9        }
10
11       cout << endl;
12    }
13 }
```

ANS: This program outputs 10 rows of five @ symbols. *Ouput:*

```
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
```

5.18 Assume that `i = 1`, `j = 2`, `k = 3` and `m = 2`. What does each of the following statements print?

a) `cout << (i == 1) << endl;`

ANS: 1.

b) `cout << (j == 3) << endl;`

ANS: 0.

c) `cout << (i >= 1 && j < 4) << endl;`

ANS: 1.

d) `cout << (m <= 99 && k < m) << endl;`

ANS: 0.

e) `cout << (j >= i || k == m) << endl;`

ANS: 1.

f) `cout << (k + m < j || 3 - j >= k) << endl;`

ANS: 0.

g) `cout << (!m ) << endl;`

ANS: 0.

h) `cout << (!(j - m)) << endl;`

ANS: 1.

i) `cout << (!(k > m)) << endl;`

ANS: 0.

5.26 What does the following program segment do?

```
for (unsigned int i{1}; i <= 5; i++) {
   for (unsigned int j{1}; j <= 3; j++) {
      for (unsigned int k{1}; k <= 4; k++) {
         cout << '*';
      }

      cout << endl;
   }

   cout << endl;
}
```

ANS:

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

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

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

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

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

5.27 ***(Replacing `continue` with a Structured Equivalent)*** Describe in general how you'd remove any `continue` statement from a loop in a program and replace it with some structured equivalent. Use the technique you develop here to remove the `continue` statement from the program in Fig. 5.14.

ANS: A loop can be rewritten without a `continue` statement by moving all the code that appears in the body of the loop after the `continue` statement to an `if` statement that tests for the opposite of the `continue` condition. Thus, the code that was originally after the `continue` statement executes only when the `if` statement's conditional expression is true (i.e., the "continue" condition is false). When the "continue" condition is true, the body of the `if` does not execute and the program "continues" to the next iteration of the loop by not executing the remaining code in the loop's body.