```
1  // Fig. 2.7: WhileCounter.cs
2  // Counter-controlled repetition with the while repetition statement.
3  using System;
4
5  public class WhileCounter
6  {
7     public static void Main( string[] args )
8     {
9        int counter = 1; // declare and initialize control variable
10
11       while (counter <= 10 )
12       {
13          Console.Write( "{0}  ", counter );
14          ++counter; // increment control variable
15       } // end while
16
17       Console.WriteLine(); // output a newline
18    } // end Main
19 } // end class WhileCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

**Fig. 2.7 |** Counter-controlled repetition with the `while` repetition statement.

*Line 13 in the while statement displays control variable counter's value during each iteration of the loop. Line 14 increments the control variable by 1 for each iteration of the loop. The loop-continuation condition in the while (line 11) tests whether the value of the control variable is less than or equal to 10 (the final value for which the condition is true). The application performs the body of this while even when the control variable is 10. The loop terminates when the control variable exceeds 10 (i.e., counter becomes 11).*

*The application in Fig. 2.7 can be made more concise by initializing counter to 0 in line 9 and incrementing counter in the while condition with the prefix increment operator as follows:*

```
while ( ++counter <= 10 ) // loop-continuation condition
   Console.Write( "{0}  ", counter );
```

*This code saves a statement (and eliminates the need for braces around the loop's body), because the while condition performs the increment before testing the condition. Code written in such a condensed fashion might be more difficult to read, debug, modify and maintain.*

## 2.9 for Repetition Statement

*Section 5.11 presented the essentials of counter-controlled repetition. The while statement can be used to implement any counter-controlled loop. C# also provides the* ***for*** *repetition statement, which specifies the elements of counter-controlled-repetition in a single line of code. In general, counter-controlled repetition should be implemented with a for statement. Figure 2.8 reimplements the application in Fig. 2.7 using the for statement.*

```
1  // Fig. 2.8: ForCounter.cs
2  // Counter-controlled  repetition with the for repetition statement.
3  using System;
4
5  public class ForCounter
6  {
7     public static  void Main( string[] args )
8     {
9        // for statement header includes initialization,
10       // loop-continuation  condition and increment
11       for ( int counter = 1; counter <= 10; counter++ )
12             Console.Write( "{0}   ", counter );
13
14       Console.WriteLine();  // output a newline
15    } // end Main
16 } // end class ForCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

**Fig. 2.8 |** Counter-controlled repetition with the `for` repetition statement.

*When the lines 11–12 begin executing, control variable counter is declared and initialized to 1. Next, the loop-continuation condition, counter <= 10 (which is between the two required semicolons) is evaluated. The initial value of counter is 1, so the condition initially is true. Therefore, the body statement (line 12) displays control variable counter's value, which is 1. After executing the loop's body, the application increments counter in the expression counter++, which appears to the right of the second semicolon. Then the loop-continuation test is performed again to determine whether the application should continue with the next iteration of the loop. At this point, the control-variable value is 2, so the condition is still true—and the application*

*performs the body statement again (i.e., the next iteration of the loop). This process continues until the numbers 1 through 10 have been displayed and the counter's value becomes 11, causing the loop-continuation test to fail and repetition to terminate (after 10 repetitions of the loop body at line 12). Then the application performs the first statement after the for—in this case, line 14.*

*Fig. 2.8 uses (in line 11) the loop-continuation condition counter <= 10. If you incorrectly specified counter < 10 as the condition, the loop would iterate only nine times—a common logic error called an off-by-one error. Figure 2.9 takes a closer look at the for statement in Fig. 2.8. The for's first line (including the keyword for and everything in parentheses after for)—line 11 in Fig. 2.8—is sometimes called the for statement header, or simply the for header. The for header "does it all"—it specifies each of the items needed for counter-controlled repetition with a control variable. If there's more than one statement in the body of the for, braces are required to define the body of the loop.*

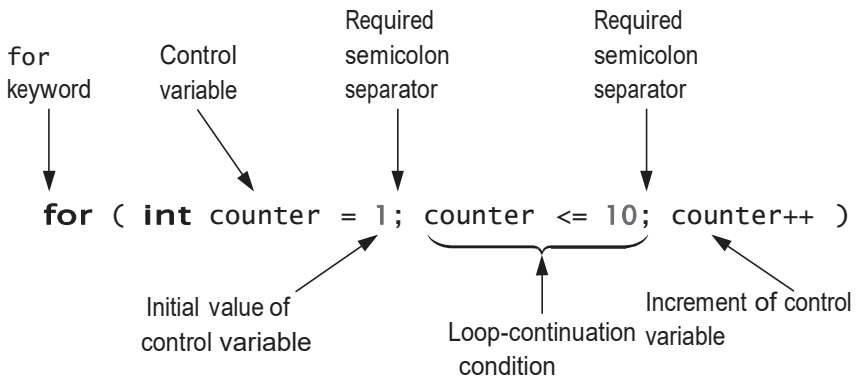


**Fig. 2.9 |** `for` statement header components.

## 2.10 Examples Using the for Statement

*The following examples show techniques for varying the control variable in a for statement. Note the change in the relational operator for loops that decrement the control variable.*

***a) Vary the control variable from 1 to 100 in increments of 1.***

```
for ( int i = 1; i <= 100; i++ )
```

***b) Vary the control variable from 100 to 1 in decrements of 1.***

```
for ( int i = 100; i >= 1; i-- )
```

***c) Vary the control variable from 7 to 77 in increments of 7.***

```
for ( int i = 7; i <= 77; i += 7 )
```

***d) Vary the control variable from 20 to 2 in decrements of 2.***

```
for ( int i = 20; i >= 2; i -= 2 )
```

***e) Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.***

```
for ( int i = 2; i <= 17; i += 3 )
```

***f) Vary the control variable over the following sequence of values: 99, 88, 77, 66, 55,44, 33, 22, 11, 0.***

```
for ( int i = 99; i >= 0; i -= 11 )
```

### Application: Summing the Even Integers from 2 to 20

```
1  // Fig. 2.10: Sum.cs
2  // Summing integers with the for statement.
3  using System;
4
5  public class Sum
6  {
7     public static void Main( string[] args )
8     {
9        int total = 0; // initialize total
10
11       // total even integers from 2 through 20
12       for ( int number = 2; number <= 20; number += 2 )
13          total += number;
14
15       Console.WriteLine( "Sum is {0}", total ); // display results
16    } // end Main
17 } // end class Sum
```

```
Sum is 110
```

**Fig. 2.10 |** Summing integers with the `for` statement.

*The initialization and increment expressions can be comma-separated lists that enable you to use multiple initialization expressions or multiple increment expressions. For example, you could merge the body of the for statement in lines 12–13 of Fig. 2.10 into the increment portion of the for header by using a comma as follows:*

```
for ( int number = 2; number <= 20; total += number, number += 2 )
   ; // empty statement
```

## 2.11 do...while Repetition Statement

*The do…while repetition statement is similar to the while statement. In the while, the application tests the loop-continuation condition at the beginning of the loop, before ex- ecuting the loop's body. If the condition is false, the body never executes. The do…while statement tests the loop-continuation condition after executing the loop's body; therefore, the body always executes at least once. When a do…while statement terminates, execution continues with the next statement in sequence. Figure 2.11 uses a do…while (lines 11–15) to output the numbers 1–10.*

```
1  // Fig. 2.11: DoWhileTest.cs
2  // do...while repetition statement.
3  using System;
4
5  public class DoWhileTest
6  {
7     public static void Main( string[] args )
8     {
9        int counter = 1; // initialize counter
10
11       do
12       {
13          Console.Write( "{O}  ", counter );
14          ++counter;
15       } while ( counter <= 10 ); // end do...while
16
17       Console.WriteLine();  // outputs a newline
18    } // end Main
19 } // end class DoWhileTest
```

```
1  2  3  4  5  6  7  8  9  10
```

**Fig. 2.11 |** `do...while` repetition statement.

*Line 9 declares and initializes control variable counter. Upon entering the do…while statement, line 13 outputs counter's value, and line 14 increments counter. Then the application evaluates the loop-continuation test at the bottom of the loop (line 15). If the condition is true, the loop continues from the first body statement in the do…while (line 13). If the condition is false, the loop terminates, and the application continues with the next statement after the loop. Figure 2.12 contains the diagram for the do…while statement. It's not necessary to use braces in the do…while repetition statement if there's only one statement in the body. For example,*

```
while ( condition )
```

*is normally the first line of a while statement. A do…while statement with no braces around a single-statement body appears as:*

```
do
   statement
while ( condition );
```

*To avoid confusion, a do…while statement with one body statement can be written as follows:*

```
do
{
   statement
} while ( condition );
```

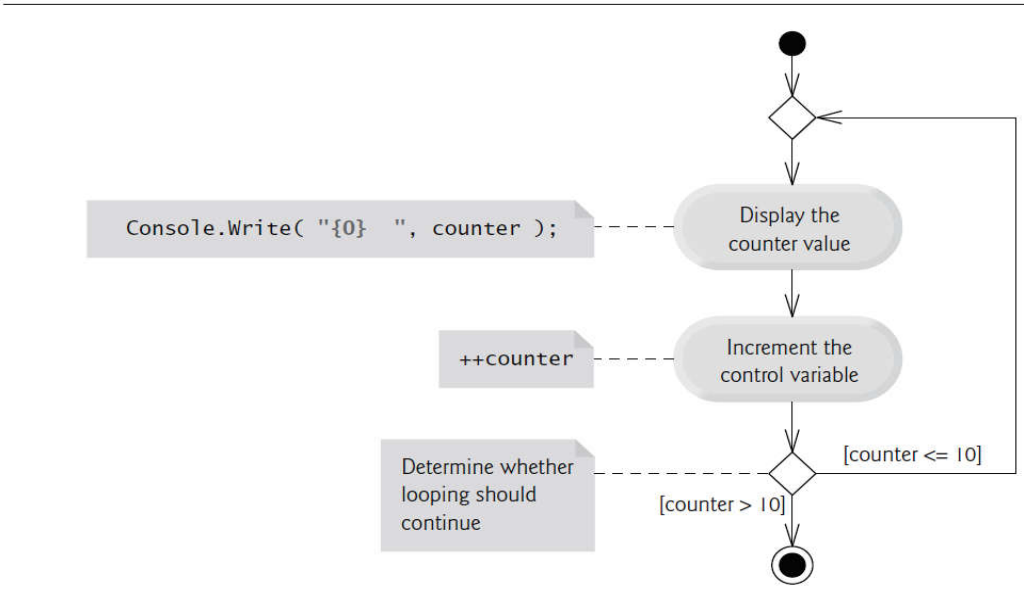


**Fig. 2.12 |** do…while repetition statement.

## 2.12 switch Multiple-Selection Statement

*We discussed the if single-selection statement and the if...else double-selection statement. C# provides the switch multiple-selection statement to perform different actions based on the possible values of an expression. Each action is associated with the value of a constant integral expression or a constant string expression that the variable or expression on which the switch is based may assume. A constant integral expression is any expression involving character and integer constants that evaluates to an integer value—i.e., values of type sbyte, byte, short, ushort, int, uint, long, ulong and char. A constant string expression is any expression composed of string literals that always results in the same string.*

```
1  // Fig. 2.13: GradeBook.cs
2  // GradeBook class uses switch statement to count letter grades.
3  using System;
4
5  public static void Main( string[] args )
6  {
7        int grade; // grade entered by user
8        string input; // text entered by the user
9
10       Console.WriteLine( "{0}\n{1}",
11          "Enter the integer grades in the range 0-100.",
12          "Type <Ctrl> z and press Enter to terminate input:" );
13
14       input = Console.ReadLine(); // read user input
15
16       // loop until user enters the end-of-file indicator (<Ctrl> z)
17       while ( input != null )
18       {
19          grade = Convert.ToInt32( input ); // read grade off user input
20          total += grade; // add grade to total
21          ++gradeCounter; // increment number of grades
22
23          // call method to increment appropriate counter
24          // determine which grade was entered
25          switch ( grade / 10 )
26          {
27            case 9:     // grade was in the 90s
28            case 10:    // grade was 100
29               ++aCount;  // increment aCount
30               break; // necessary to exit switch
31
32            case 8:    // grade was between 80 and 89
33               ++bCount; // increment bCount
34               break;    // exit switch
```

```
35              case   7: // grade was between 70 and 79
36                 ++cCount; // increment cCount
37                 break; // exit switch
38              case   6: // grade was between 60 and 69
39                 ++dCount; // increment dCount
40                 break; // exit switch
41              default: // grade was less than 60
42                 ++fCount; // increment fCount
43                 break; // exit switch
44            } // end switch
45          input = Console.ReadLine();  // read user input
46        } // end While
47       Console.WriteLine( "{0}A: {1}\nB: {2}\nC: {3}\nD: {4}\nF: {5}",
48                   "Number of students who received each grade:\n",
48                    aCount, bCount,cCount, dCount, fCount );
47  } // end Main
```

**Fig. 2.13 |** `switch` statement to count A, B, C, D and F grades.

*Lines 7–8 declare variables grade and input, which will first store the user's input as a string (in the variable input), then convert it to an int to store in the variable grade. Lines 10–12 prompt the user to enter integer grades and to type Ctrl + z, then press Enter to terminate the input. The notation Ctrl + z means to simultaneously press both the Ctrl key and the z key when typing in a Command Prompt. Ctrl + z is the Windows key sequence for typing the end-of-file indicator. This is one way to inform an application that there's no more data to input. If Ctrl + z is entered while the application is awaiting input with a ReadLine method, null is returned.*

*Line 14 uses the ReadLine method to get the first line that the user entered and store it in variable input. The while statement (lines 17–46) processes this user input. The condition at line 17 checks whether the value of input is a null reference. Line 19 converts the string in input to an int type. Line 20 adds grade to total. Line 21 increments gradeCounter. a switch statement (lines 25–44) that determines which counter to increment. In this example, we assume that the user enters a valid grade in the range 0–100. A grade in the range 90–100 represents A, 80–89 represents B, 70–79 represents C, 60–69 represents D and 0–59 represents F. The switch statement consists of a block that contains a sequence of case labels and an optional default label.*

*When the flow of control reaches the switch statement, the*

*application evaluates the expression in the parentheses (grade / 10) following keyword switch—this is called the switch expression. The application attempts to match the value of the switch expression with one of the case labels. The switch expression in line 25 performs integer division, which truncates the fractional part of the result. Thus, when we divide any value in the range 0–100 by 10, the result is always a value from 0 to 10. We use several of these values in our case labels. For example, if the user enters the integer 85, the switch expression evaluates to int value 8.*

## 2.13 break and continue Statements

*In addition to selection and repetition statements, C# provides statements break and continue to alter the flow of control. The preceding section showed how break can be used to terminate a switch statement's execution. This section discusses how to use break to terminate any repetition statement.*

### break Statement

*The break statement, when executed in a* ***while****,* ***for****,* ***do…while****,* ***switch****, or* ***foreach****, causes immediate exit from that statement. Execution typically continues with the first statement after the control statement—you'll see that there are other possibilities as you learn about additional statement types in C#. Common uses of the break statement are to escape early from a repetition statement or to skip the remainder of a switch (as in Fig. 2.13). Figure 2.14 demonstrates a break statement exiting a for.*

*When the if nested at line 13 in the for statement (lines 11–17) determines that count is 5, the break statement at line 14 executes. This terminates the for statement, and the application proceeds to line 19 (immediately after the for statement), which displays a message indicating the value of the control variable when the loop terminated. The loop fully executes its body only four times instead of 10 because of the break.*

```
1  // Fig. 2.14 : BreakTest.cs
2  // break statement exiting a for statement.
3  using System;
4
5  public class BreakTest
6  {
7     public static void Main( string[] args )
8     {
9        int count; // control variable also used after loop terminates
10
11       for ( count = 1; count <= 10; count++ ) // loop 10 times
12       {
13          if ( count == 5 ) // if count is 5,
14             break; // terminate loop
15
16          Console.Write( "{0} ", count );
17       } // end for
18
19       Console.WriteLine( "\nBroke out of loop at count = {0}", count);
20    } // end Main
21 } // end class BreakTest
```

```
1 2 3 4
Broke out of loop at count = 5
```

**Fig. 2.14 |** `break` statement exiting a `for` statement.

### *continue Statement*

*The continue statement, when executed in a* ***while****,* ***for****,* ***do…while****, or* ***foreach****, skips the remaining statements in the loop body and proceeds with the next iteration of the loop. In* ***while*** *and* ***do…while*** *statements, the application evaluates the loop-continuation test immediately after the continue statement executes. In a for statement, the increment expression normally executes next, then the application evaluates the loop-continuation test.*

```
1  // Fig. 2.15: ContinueTest.cs
2  // continue statement terminating an iteration of a for statement.
3  using System;
4
5  public class ContinueTest
6  {
7     public static void Main( string[] args )
8     {
9        for ( int count = 1; count <= 10; count++ ) // loop 10 times
10       {
```

```
11            if ( count == 5 ) // if count is 5,
12               continue; // skip remaining code in loop
13
14            Console.Write( "{0} ", count );
15         } // end for
16
17         Console.WriteLine( "\nUsed continue to skip displaying 5" );
18      } // end Main
19   } // end class ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip displaying 5
```

**Fig. 2.15 |** `continue` statement terminating an iteration of a `for` statement.

*Figure 2.15 uses the continue statement in a for to skip the statement at line 14 when the nested if (line 11) determines that the value of count is 5. When the continue statement executes, program control continues with the increment of the control variable in the for statement (line 9).*

*In Section 3.13, we stated that a while can be used in most cases in place of for. One exception occurs when the increment expression in the while follows a continue statement. In this case, the increment doesn't execute before the repetition-continuation condition evaluates, so the while does not execute in the same manner as the for.*