

6

Control Statements: Part 2

© 2006 Pearson Education, Inc. All rights reserved.

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 an application 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.

© 2006 Pearson Education, Inc. All rights reserved.

6.2 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires:
 - Control variable (loop counter)
 - Initial value of the control variable
 - Increment/decrement of control variable through each loop
 - Loop-continuation condition that tests for the final value of the control variable

© 2006 Pearson Education, Inc. All rights reserved.

```
1 // Fig. 6.1: 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 ) // loop-continuation condition
12         {
13             Console.WriteLine( "{0} ", counter );
14             counter++; // increment control variable
15         } // end while
16
17         Console.WriteLine(); // output a new line
18     } // end Main
19 } // end class WhileCounter
```

1 2 3 4 5 6 7 8 9 10

Control-variable name is counter
Control-variable initial value is 1

Condition tests for counter's final value

Increment for counter

© 2006 Pearson Education, Inc. All rights reserved.

6.3 for Repetition Statement

- Handles counter-controlled-repetition details
- Format:
for (initialization; loopContinuationCondition; increment)
statement;
- If the counter is initialize in the For...Next header, then that variable has a scope of that loop.
- Can usually be rewritten as:
initialization;
while (loopContinuationCondition)
{
statement;
increment;
}

© 2006 Pearson Education, Inc. All rights reserved.

```
1 // Fig. 6.2: 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.WriteLine( "{0} ", counter );
13
14        Console.WriteLine(); // output a new line
15    } // end Main
16 } // end class ForCounter
```

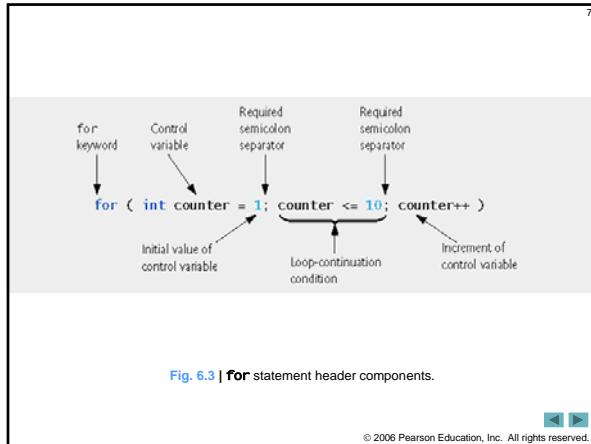
1 2 3 4 5 6 7 8 9 10

Control-variable name is counter
Control-variable initial value is 1

Condition tests for counter's final value

Increment for counter

© 2006 Pearson Education, Inc. All rights reserved.



8

6.4 Examples Using the for Statement

- Varying control variable in for headers
 - Vary control variable from 1 to 100 in increments of 1
 - for (int i = 1; i <= 100; i++)
 - Vary control variable from 100 to 1 in increments of -1
 - for (int i = 100; i >= 1; i--)
 - Vary control variable from 7 to 77 in increments of 7
 - for (int i = 7; i <= 77; i += 7)
 - Vary control variable from 20 to 2 in decrements of -2
 - for (int i = 20; i >= 2; i -= 2)
 - Vary control variable over the sequence: 2, 5, 8, 11, 14, 17, 20
 - for (int i = 2; i <= 20; i += 3)
 - Vary control variable over the sequence: 99, 87, 76, 66, 55, 44, 33, 22, 11, 0
 - for (int i = 99; i >= 0; i -= 11)

© 2006 Pearson Education, Inc. All rights reserved.

9

```

1 // Fig. 6.5: 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

© 2006 Pearson Education, Inc. All rights reserved.

10

6.5 do...while Repetition Statement

- do...while structure
 - Similar to while structure
 - Tests loop-continuation after performing body of loop
 - i.e., loop body always executes at least once
- Format:


```

do
    statement;
while ( condition );
  
```

© 2006 Pearson Education, Inc. All rights reserved.

11

```

1 // Fig. 6.7: 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.WriteLine( "{0}", 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

Outline

- Declares and initializes control variable counter
- Variable counter's value is displayed before testing counter's final value

DoWhileTest.cs

© 2006 Pearson Education, Inc. All rights reserved.

12

6.6 switch Multiple-Selection Statement

- switch statement
 - Used for multiple selections
- Each case must have a break
 - Exception: case body is empty and it falls through to another case
- Goes to default if none of the cases matches
 - Optional

© 2006 Pearson Education, Inc. All rights reserved.

```

69 // add 1 to appropriate counter for specified grade
70 private void IncrementLetterGradeCounter( int grade )
71 {
72     // determine which grade was entered
73     switch ( grade / 10 )
74     {
75         case 9: // grade was in the 90s
76             aCount++; // increment aCount
77             break; // necessary to exit switch
78         case 8: // grade was between 80 and 89
79             bCount++; // increment bCount
80             break; // exit switch
81         case 7: // grade was between 70 and 79
82             cCount++; // increment cCount
83             break; // exit switch
84         case 6: // grade was between 60 and 69
85             dCount++; // increment dCount
86             break; // exit switch
87         default: // grade was less than 60
88             fCount++; // increment fCount
89             break; // exit switch
90     } // end switch
91 } // end method IncrementLetterGradeCounter

```

Outline

GradeBook.cs
(4 of 5)

(grade / 10) is the controlling expression

switch statement determines which case label to execute, depending on controlling expression

default case for grade less than 60

© 2006 Pearson Education, Inc. All rights reserved.

Software Engineering Observation 6.2

Provide a default label in switch statements. Cases not explicitly tested in a switch that lacks a default label are ignored. Including a default label focuses you on the need to process exceptional conditions.

© 2006 Pearson Education, Inc. All rights reserved.

Good Programming Practice 6.7

Although each case and the default label in a switch can occur in any order, place the default label last for clarity.

© 2006 Pearson Education, Inc. All rights reserved.

6.6 switch Multiple-Selection Statement (Cont.)

- Expression in each case
 - Constant integral expression
 - Combination of integer constants that evaluates to a constant integer value
 - Character constant
 - E.g., 'A', '7' or '\$'
 - Constant variable

© 2006 Pearson Education, Inc. All rights reserved.

Fig. 6.11 | switch multiple-selection statement UML activity diagram with break statements.

© 2006 Pearson Education, Inc. All rights reserved.

6.7 break and continue Statements

- break/continue
 - Alter flow of control
- break statement
 - Causes immediate exit from control structure
 - Used in while, for, do...while, switch, or foreach statements
- continue statement
 - Skips remaining statements in loop body
 - Proceeds to next iteration
 - Used in while, for, do...while or foreach statements

© 2006 Pearson Education, Inc. All rights reserved.

19

```

1 // Fig. 6.12: 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.WriteLine( "{0}", count );
17        } // end for
18
19        Console.WriteLine( "\nBroke out of loop at count = {0}", count );
20    } // end Main
21 } // end class BreakTest

```

Outline

BreakTest.cs

Loop 10 times

Exit for statement (break) when count equals 5

```

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

```

© 2006 Pearson Education, Inc. All rights reserved.

20

```

1 // Fig. 6.13: 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.WriteLine( "{0}", count );
15        } // end for
16
17        Console.WriteLine( "\nUsed continue to skip printing 5" );
18    } // end Main
19 } // end class ContinueTest

```

Outline

ContinueTest.cs

Loop 10 times

Skip line 14 and proceed to line 9 when count equals 5

```

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

```

© 2006 Pearson Education, Inc. All rights reserved.

21

Software Engineering Observation 6.4

There is a tension between achieving quality software engineering and achieving the best-performing software. Often, one of these goals is achieved at the expense of the other. For all but the most performance-intensive situations, apply the following rule of thumb: First, make your code simple and correct; then make it fast and small, but only if necessary.

© 2006 Pearson Education, Inc. All rights reserved.

22

6.8 Logical Operators

- Logical operators
 - Allows for forming more complex conditions
 - Combines simple conditions
- C# logical operators
 - && (conditional AND)
 - || (conditional OR)
 - & (boolean logical AND)
 - | (boolean logical inclusive OR)
 - ^ (boolean logical exclusive OR)
 - ! (logical NOT)

© 2006 Pearson Education, Inc. All rights reserved.

23

6.8 Logical Operators (Cont.)

- Conditional AND (&&) Operator
 - Consider the following `if` statement


```
if ( gender == FEMALE && age >= 65 )
    seniorFemales++;
```
 - Combined condition is **true**
 - if and only if both simple conditions are **true**
 - Combined condition is **false**
 - if either or both of the simple conditions are **false**

© 2006 Pearson Education, Inc. All rights reserved.

24

| expression1 | expression2 | expression1 && expression2 |
|-------------|-------------|----------------------------|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

Fig. 6.14 | && (conditional AND) operator truth table.

© 2006 Pearson Education, Inc. All rights reserved.

6.8 Logical Operators (Cont.)

• Conditional OR (||) Operator

- Consider the following `if` statement

```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 )
    Console.WriteLine( "Student grade is A" );
```
- Combined condition is **true**
 - if either or both of the simple conditions are **true**
- Combined condition is **false**
 - if both of the simple conditions are **false**

© 2006 Pearson Education, Inc. All rights reserved.

| expression1 | expression2 | expression1 expression2 |
|-------------|-------------|----------------------------|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

Fig. 6.15 || (conditional OR) operator truth table.

© 2006 Pearson Education, Inc. All rights reserved.

6.8 Logical Operators (Cont.)

• Short-Circuit Evaluation of Complex Conditions

- Parts of an expression containing `&&` or `||` operators are evaluated only until it is known whether the condition is true or false
- E.g., `(gender == FEMALE) && (age >= 65)`
 - Stops immediately if gender is not equal to **FEMALE**

© 2006 Pearson Education, Inc. All rights reserved.

6.8 Logical Operators (Cont.)

• Boolean Logical AND (&) Operator

- Works identically to `&&`
- Except `&` always evaluate both operands

• Boolean Logical OR (|) Operator

- Works identically to `||`
- Except `|` always evaluate both operands

© 2006 Pearson Education, Inc. All rights reserved.

6.8 Logical Operators (Cont.)

• Boolean Logical Exclusive OR (^) Operator

- One of its operands is **true** and the other is **false**
 - Evaluates to **true**
- Both operands are **true** or both are **false**
 - Evaluates to **false**

• Logical Negation (!) Operator

- Unary operator
- Opposite of the boolean value

© 2006 Pearson Education, Inc. All rights reserved.

| expression1 | expression2 | expression1 ^ expression2 |
|-------------|-------------|---------------------------|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | false |

Fig. 6.16 ^ (boolean logical exclusive OR) operator truth table.

© 2006 Pearson Education, Inc. All rights reserved.

31

| expression | !expression |
|------------|-------------|
| false | true |
| true | false |

Fig. 6.17 | ! (logical negation) operator truth table.

© 2006 Pearson Education, Inc. All rights reserved.

32

```

1 // Fig. 6.18: Logical Operators.cs
2 // Logical operators.
3 using System;
4
5 public class LogicalOperators
6 {
7     public static void Main( string[] args )
8     {
9         // create truth table for && (conditional AND) operator
10        Console.WriteLine( "(0)\n(1): (2)\n(3): (4)\n(5): (6)\n(7): (8)\n",
11            "Conditional AND (&&)", "false && false", ( false && false ),
12            "false && true", ( false && true ),
13            "true && false", ( true && false ),
14            "true && true", ( true && true ) );
15
16        // create truth table for || (conditional OR) operator
17        Console.WriteLine( "(0)\n(1): (2)\n(3): (4)\n(5): (6)\n(7): (8)\n",
18            "Conditional OR (||)", "false || false", ( false || false ),
19            "false || true", ( false || true ),
20            "true || false", ( true || false ),
21            "true || true", ( true || true ) );
22
23        // create truth table for & (boolean logical AND) operator
24        Console.WriteLine( "(0)\n(1): (2)\n(3): (4)\n(5): (6)\n(7): (8)\n",
25            "Boolean logical AND (&)", "false & false", ( false & false ),
26            "false & true", ( false & true ),
27            "true & false", ( true & false ),
28            "true & true", ( true & true ) );
29    }
30 }

```

Outline

Logical Operators.cs (1 of 3)

Conditional AND truth operator

Conditional OR truth operator

Boolean logical AND operator

© 2006 Pearson Education, Inc. All rights reserved.

33

```

29 // create truth table for | (boolean logical inclusive OR) operator
30 Console.WriteLine( "(0)\n(1): (2)\n(3): (4)\n(5): (6)\n(7): (8)\n",
31     "Boolean logical inclusive OR (|)",
32     "false | false", ( false | false ),
33     "false | true", ( false | true ),
34     "true | false", ( true | false ),
35     "true | true", ( true | true ) );
36
37 // create truth table for ^ (boolean logical exclusive OR) operator
38 Console.WriteLine( "(0)\n(1): (2)\n(3): (4)\n(5): (6)\n(7): (8)\n",
39     "Boolean logical exclusive OR (^)",
40     "false ^ false", ( false ^ false ),
41     "false ^ true", ( false ^ true ),
42     "true ^ false", ( true ^ false ),
43     "true ^ true", ( true ^ true ) );
44
45 // create truth table for ! (logical negation) operator
46 Console.WriteLine( "(0)\n(1): (2)\n(3): (4)",
47     "Logical negation (!)", "false", ( !false ),
48     "!true", ( !true ) );
49
50 } // end Main
51 } // end class LogicalOperators

```

Outline

Logical Operators.cs (2 of 3)

Boolean logical inclusive OR operator

Boolean logical exclusive OR operator

Logical negation operator

© 2006 Pearson Education, Inc. All rights reserved.

34

```

Conditional AND (&&)
false && false: False
false && true: False
true && false: False
true && true: True

Conditional OR (||)
false || false: False
false || true: True
true || false: True
true || true: True

Boolean logical AND (&)
false & false: False
false & true: False
true & false: False
true & true: True

Boolean logical inclusive OR (|)
false | false: False
false | true: True
true | false: True
true | true: True

Boolean logical exclusive OR (^)
false ^ false: False
false ^ true: True
true ^ false: True
true ^ true: False

Logical negation (!)
!false: True
!true: False

```

Outline

Logical Operators.cs (3 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

35

| Operators | Associativity | Type |
|-----------------------------|---------------|------------------------------|
| . | | |
| new ++(postfix) --(postfix) | left to right | highest precedence |
| ++ -- + - (type) | right to left | unary prefix |
| * / % | left to right | multiplicative |
| + - | left to right | additive |
| < <= > >= | left to right | relational |
| == != | left to right | equality |
| & | left to right | boolean logical AND |
| ^ | left to right | boolean logical exclusive OR |
| | left to right | boolean logical inclusive OR |
| && | left to right | conditional AND |
| | left to right | conditional OR |
| ?: | right to left | conditional |
| = += -= *= /= %= | right to left | assignment |

Fig. 6.19 | Precedence/associativity of the operators discussed so far.

© 2006 Pearson Education, Inc. All rights reserved.

36

6.9 Structured Programming Summary

- Sequence structure
 - “built-in” to C#
- Selection structure
 - if, if...else and switch
- Repetition structure
 - while, do...while, for, and foreach

© 2006 Pearson Education, Inc. All rights reserved.