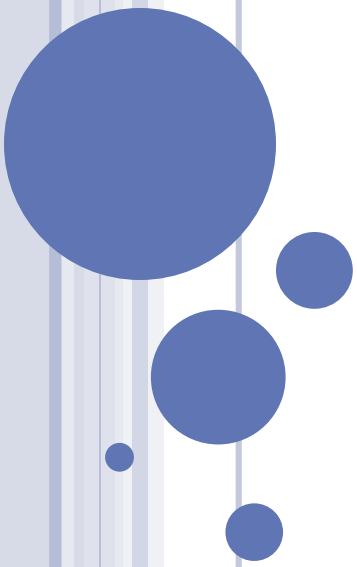


# Advanced Programming

## Chapters 5 & 6: Control Structures

Dr Shahriar Bijani

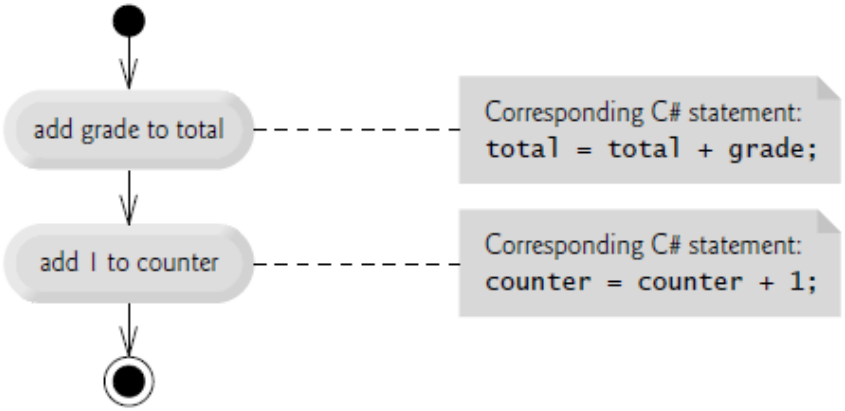
# Reference

- **Visual C# 2012 How to Program**, Paul Deitel & Harvey Deitel, 5th Edition, Prentice Hall.

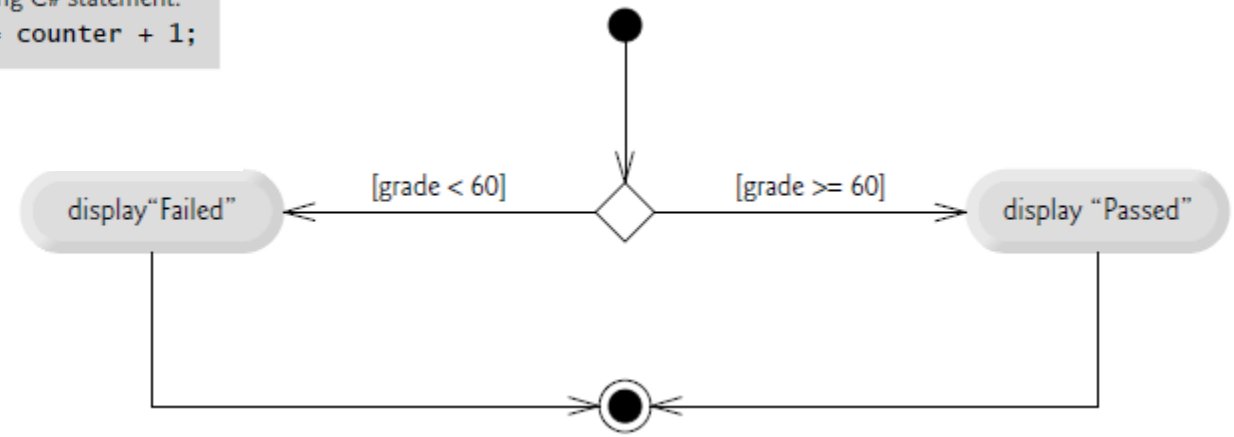
# 5.4 Control Structures

Bohm and Jacopini's demonstrated that all apps could be written using only 3 control structures:

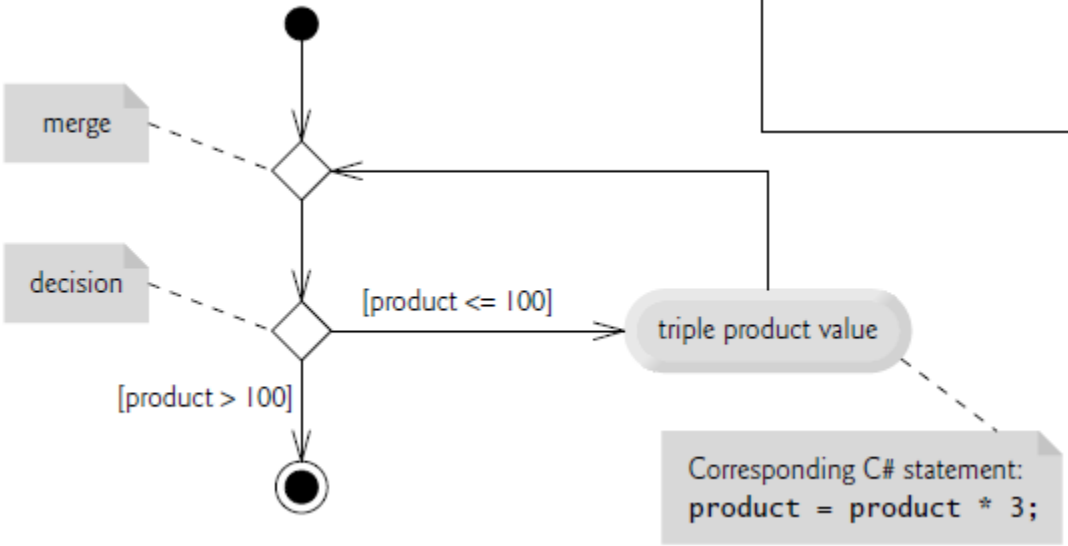
### *1) Sequence Structure*




### *2) Selection Structure*




### *3)Repetition Structure*



# IF STATEMENT

- Nested if…else

```
if ( grade >= 90 )
   Console.WriteLine( "A" );
   else if ( grade >= 80 )
      Console.WriteLine( "B" );
   else if ( grade >= 70 )
      Console.WriteLine( "C" );
   else if ( grade >= 60 )
      Console.WriteLine( "D" );
   else
      Console.WriteLine( "F" );
```

# Dangling-else Problem

- The C# compiler always associates an `else` with the *immediately preceding* `if`

```
if ( x > 5 )
   if ( y > 5 )
      Console.WriteLine( "x and y are > 5" );
else
    Console.WriteLine( "x is <= 5" );
```

is not equal to:

```
if ( x > 5 )
{
   if ( y > 5 )
      Console.WriteLine( "x and y are > 5" );
}
else
   Console.WriteLine( "x is <= 5" );
```

# 5.8 Formulating Algorithms: Counter-Controlled Repetition

- A counter is used to determine when the loop should stop

```
1    // Fig. 5.6: GradeBook.cs
2    // Class average with counter-controlled repetition.
3
4  using System;
5
6 public class GradeBook
7 {
8    // auto-implemented property CourseName
9    public string CourseName { get; set; }
10
11   // constructor initializes CourseName property
12   public GradeBook( string name )
13   {
14        CourseName = name; // set CourseName to name
15   } // end constructor
16
17   // display a welcome message to the GradeBook user
18   public void DisplayMessage()
19   {
20      // property CourseName gets the name of the course
21      Console.WriteLine( "Welcome to the grade book for\n{0}!\n",
22         CourseName );
23   } // end method DisplayMessage
24
25 public void DetermineClassAverage()
26     {
27        int total,          // sum of grades
28            gradeCounter,   // number of grades entered
29            gradeValue,     // grade value
30            average;        // average of all grades
31
32        // initialization phase
33        total = 0;          // clear total
34        gradeCounter = 1;   // prepare to loop
35
```

```
37          // processing phase
38          while ( gradeCounter <= 10 )  // loop 10 times
39          {
40             // prompt for input and read grade from user
41             Console.Write( "Enter integer grade: " );
42
43             // read input and convert to integer
44             gradeValue = Int32.Parse( Console.ReadLine() );
45             // add gradeValue to total
46             total = total + gradeValue;
47             // add 1 to gradeCounter
48             gradeCounter = gradeCounter + 1;
44          } // end while
45
46       // termination phase
47       average = total / 10; // integer division yields integer result
48
49       // display total and average of grades
50       Console.WriteLine( "\nTotal of all 10 grades is {0}", total );
51       Console.WriteLine( "Class average is {0}", average );
52    } // end method DetermineClassAverage
53 } // end class GradeBook
```

# 5.9 Formulating Algorithms: Sentinel-Controlled Repetition

- Continues an arbitrary amount of times

```
1     // Fig. 5.9: GradeBook.cs
2     // Class with sentinel-controlled repetition.
...
8        public void ClassAverage()
9        {
10          int total,            // sum of grades
11              gradeCounter,     // number of grades entered
12              gradeValue;       // grade value
13
14          double average;       // average of all grades
15
16          // initialization phase
17          total = 0;            // clear total
18          gradeCounter = 0;     // prepare to loop
19
20          // processing phase
21          // prompt for input and convert to integer
22          Console.Write( "Enter Integer Grade, -1 to Quit: " );
23          gradeValue = Int32.Parse( Console.ReadLine() );
24
```

```
25          // loop until a -1 is entered by user
26          while ( gradeValue != -1 )
27          {
28             // add gradeValue to total
29             total = total + gradeValue;
30
31             // add 1 to gradeCounter
32             gradeCounter = gradeCounter + 1;
33
34             // prompt for input and read grade from user
35             // convert grade from string to integer
36             Console.Write( "Enter Integer Grade, -1 to Quit: " );
37             gradeValue = Int32.Parse( Console.ReadLine() );
38
39          } // end while
40
41          // termination phase
42          if ( gradeCounter != 0 )
43          {
44             average = ( double ) total / gradeCounter;
45
46             // display average of exam grades
47             Console.WriteLine( "\nClass average is {0}", average );
48          }
49          else
50          {
51             Console.WriteLine( "\nNo grades were entered" );
52          }
53
54       } // end of method
55    ...
```

# 5.11 Compound Assignment Operators

- ++, -=, *=, /=, and %=

| Assignment operator | Sample expression | Explanation | Assigns |
|---|---|---|---|
| *Assume:* `int c = 3, d = 5, e = 4, f = 6, g = 12;` | | | |
| `+=` | `c += 7` | `c = c + 7` | `10` to `c` |
| `-=` | `d -= 4` | `d = d - 4` | `1` to `d` |
| `*=` | `e *= 5` | `e = e * 5` | `20` to `e` |
| `/=` | `f /= 3` | `f = f / 3` | `2` to `f` |
| `%=` | `g %= 9` | `g = g % 9` | `3` to `g` |

**Fig. 5.13** Arithmetic compound assignment operators.

| Operator | Called | Sample expression | Explanation |
|---|---|---|---|
| `++` | preincrement | `++a` | Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides. |
| `++` | postincrement | `a++` | Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1. |
| `--` | predecrement | `--b` | Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides. |
| `--` | postdecrement | `b--` | Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1. |

**Fig. 5.14** The increment and decrement operators.

## 6.2 Essentials of Counter-Controlled Repetition

- Control variable
  - controls the loop continuation
- Initial value of the control variable
- Incrementing/decrementing of the variable
- The condition
  - to continue looping

**WhileCounter.cs**

```
1   // Fig. 6.1: WhileCounter.cs
2   // Counter-controlled repetition.
3
4   using System;
5
6   class WhileCounter
7   {
8      static void Main( string[] args )
9      {
10        int counter = 1;          // initialization
11
12        while ( counter <= 5 )    // repetition condition
13        {
14           Console.Write("{0} ", counter );
15           counter++;             // increment
16
17        } // end while
18
19     } // end method Main
20
21  } // end class WhileCounter
```

**Program Output**

```
1
2
3
4
5
```

- Similar code:

```
int counter = 0;
while ( ++counter <= 5 ) // loop-continuation condition
   Console.Write( "{0} ", counter );
```

# 6.3 for Repetition Structure

```
1    // Fig. 6.2: ForCounter.cs
2    // Counter-controlled repetition with the for structure.
3
4    using System;
5
6    class ForCounter
7    {
8       static void Main( string[] args )
9       {
10         // initialization, repetition condition and incrementing
11         for ( int counter = 1; counter <= 5; counter++ )
12            Console.Write("{0} ", counter );
13      }
14   }
```

**ForCounter.cs**

```
1 2 3 4 5
```

**Program Output**

# FOR VS. WHILE

- General form of the `for` statement

```
for ( initialization; loop_Continuation_Condition; increment )
    statement;
```

- Can usually be rewritten as:

```
initialization;
while ( loop_Continuation_Condition )
 {
       statement;
       increment;
 }
```

- **`for`** statement:
  - If the control variable is declared in the *initialization* expression, It will be unknown outside the `for` statement

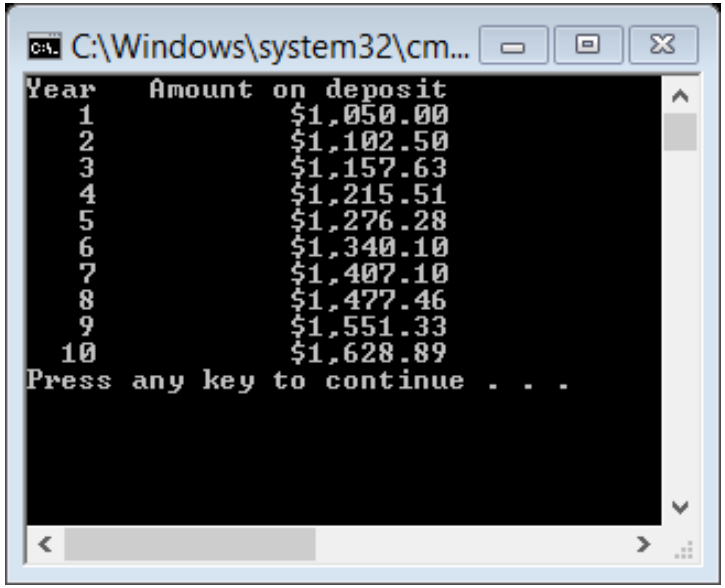
# An Example of for Statement

```
// Fig. 6.6: Interest.cs
// Compound-interest calculations with for.
using System;
public class Interest
{
   public static void Main( string[] args )
   {
      decimal amount; // amount on deposit at end of each year
      decimal principal = 1000; // initial amount before interest
      double rate = 0.05; // interest rate

      // display headers
      Console.WriteLine( "Year{0,20}", "Amount on deposit" );

      // calculate amount on deposit for each of ten years
      for ( int year = 1; year <= 10; ++year )
      {
         // calculate new amount for specified year
         amount = principal *
            ( ( decimal ) Math.Pow( 1.0 + rate, year ) );

         // display the year and the amount
         Console.WriteLine( "{0,4}{1,20:C}", year, amount );
      } // end for
   } // end Main
} // end class Interest
```



# DO/WHILE

- Using a **`do/while`** loop
  - Action is performed, then the loop condition is tested
  - Loop **must be run once**
  - Always uses brackets ({) to prevent confusion

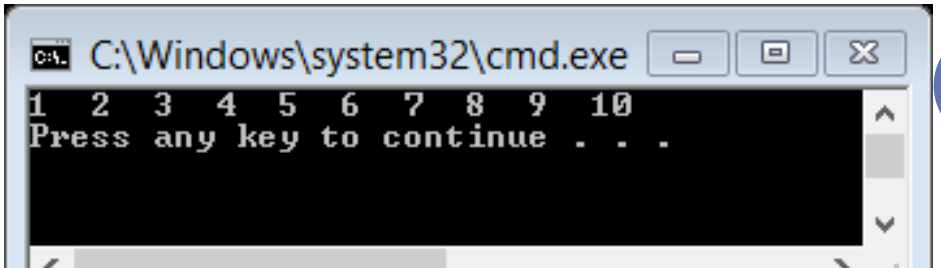
# DO/WHILE EXAMPLE

```
// Fig. 6.7: DoWhileTest.cs
// do...while repetition statement.
using System;

public class DoWhileTest
{
   public static void Main( string[] args )
   {
      int counter = 1; // initialize counter

      do
      {
         Console.Write( "{0}  ", counter );
         ++counter;
      } while ( counter <= 10 ); // end do...while

      Console.WriteLine(); // outputs a newline
   } // end Main
} // end class DoWhileTest
```



# 6.6 switch Multiple-Selection Structure

- The **switch** statement
  - Constant expressions
    - String
    - Integral (characes and integers):
      sbyte, byte, short, ushort, int, uint, long, ulong , char and *enum*
  - Cases
    - Case '*x*' :
      - Use of constant variable cases
    - Empty cases
  - The **break** statement
    - Exit the **switch** statement

# switch Example

```
private void IncrementLetterGradeCounter( int grade )
   {
      // determine which grade was entered
      switch ( grade / 10 )
      {
         case 10: // grade was 100
            ++aCount; // increment aCount
            break; // necessary to exit switch
         case 8: // grade was between 80 and 89
            ++bCount; // increment bCount
            break; // exit switch
         case 7: // grade was between 70 and 79
            ++cCount; // increment cCount
            break; // exit switch
         case 6: // grade was between 60 and 69
            ++dCount; // increment dCount
            break; // exit switch
         default: // grade was less than 60
            ++fCount; // increment fCount
            break; // exit switch
      } // end switch
   } // end method IncrementLetterGradeCounter
// … More on fig 6.9 of the Book
```

# 6.7 BREAK AND CONTINUE STATEMENTS

- Use to change the flow of control.

- The **`break`** statement
  - Used to exit a loop early
- The **`continue`** statement
  - Used to skip the rest of the statements and begin the loop at the first statement in the loop

- You can program without them!

# Example 1

```
// Fig. 6.12: BreakTest.cs
// break statement exiting a for statement.
using System;

public class BreakTest
{
   public static void Main( string[] args )
   {
      int count; // control variable also used after loop terminates

      for ( count = 1; count <= 10; ++count ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            break; // terminate loop

         Console.Write( "{0} ", count );
      } // end for

      Console.WriteLine( "\nBroke out of loop at count = {0}", count );
   } // end Main
} // end class BreakTest
```

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

# Example 2

```
public class ContinueTest
{
   public static void Main( string[] args )
   {
      for ( int count = 1; count <= 10; ++count ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            continue; // skip remaining code in loop

         Console.Write( "{0} ", count );
      } // end for

      Console.WriteLine( "\nUsed continue to skip displaying 5" );
   } // end Main
} // end class ContinueTest
```

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

- **Software Engineering Observation 6.2**

*Some* programmers *feel that break and continue* statements violate *structured programming*.

They prefer not to use *break or continue* statements

# 6.8 Logical Operators

- Operators
  - Boolean Logical AND (&)
  - Boolean Logical OR (|)

  } *always* evaluate both of their operands

  - Conditional AND (&&)
  - Conditional OR (||)

  } short-circuit evaluation

  - Logical exclusive OR or XOR (^)
  - Logical NOT (!)
    - You can avoid it!

# Side Effects of Logical Operators

- Consider the side effects on using boolean and conditional operators:

```
( mid + final == 20 ) || ( ++final >= 10 )
```

no evaluation

```
( mid + final == 20 ) | ( ++final >= 10 )
```