CS 112 Introduction to Programming

Lecture #9:

More Control Structures

http://flint.cs.yale.edu/cs112/

Outline

- Admin. and review
- Loop statements
  - `while` statement
  - Nested control
  - `do/while` statement
  - `for` statement
- `break` and `continue` statements

Recap

- The `while` statement

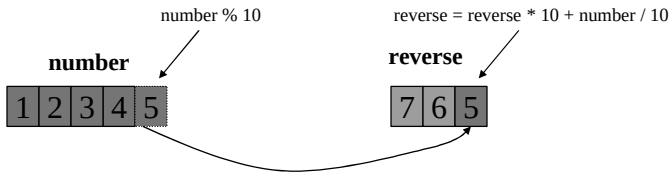
```
while ( condition )
   statement;
```

- Some typical ways to write `while` loops
  - Counter-based
  - Sentinel-based
  - Others: e.g., ReverseNumber.cs

ReverseNumber

Assume initial input is “1234567”. Above is the current state.

ReverseNumber




```
{
   lastDigit = number % 10;
   reverse = reverse * 10 + lastDigit;

}
```

5

ReverseNumber


```
while (number > 0)
{
   lastDigit = number % 10;
   reverse = reverse * 10 + lastDigit;
   number = number / 10;
}
```

6

Outline

- Admin. and review
- Loop statements
  - `while` statement
  - Nested control
  - `do/while` statement
  - `for` statement
- `break` and `continue` statements

Nested Control

- The insertion of one control structure inside another
  - Loops with **`if`** statements

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
        Input the next exam result

        If the student passed
                Add one to passes
        Else
                Add one to failures

        Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
        Print "Raise tuition"
```

Example: Analysis.cs

Outline

Analysis.cs

```
// Analysis.cs
// Analysis of Examination Results.

using System;

class Analysis
{
   static void Main( string[] args )
   {
      int passes = 0,          // number of
          failures = 0,        // number of
          student = 1,         // student counter
          grade;               // one exam grade

      // process 10 students; counter-controlled loop
      while ( student <= 10 )
      {
         Console.Write( "Enter grade (0-100): " );
         grade = Int32.Parse( Console.ReadLine() );

         if ( result >= 60 )
            passes = passes + 1;

         else
            failures = failures + 1;

         student = student + 1;
      }

```

Initialize both passes and failures to 0
Set the student count to 1

A while loop that will loop 10 times

A nested if statement that determines which counter should be added to

If the grade >= 60 add one to passes

Else add one to failures

Keep track of the total number of students

Outline

Analysis.cs

```
      // termination phase
      Console.WriteLine();
      Console.WriteLine( "Passed: " + passes );
      Console.WriteLine( "Failed: " + failures );

      if ( passes > 8 )
         Console.WriteLine( "Raise Tuition\n" );

   } // end of method Main

} // end of class Analysis
```

Display the results to the user

If the total number of passes was greater than 8 then also tell the user to raise the tuition

Outline

- Admin. and review
- Loop statements
  - `while` statement
  - Nested control
  - `do/while` statement
  - `for` statement
- `break` and `continue` statements

The do Statement

- The *do statement* has the following syntax:

Uses both the `do` and `while` reserved words

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

The statement is executed once initially, then the condition is evaluated

The statement is repetitively executed until the condition becomes false

do/while Flowchart

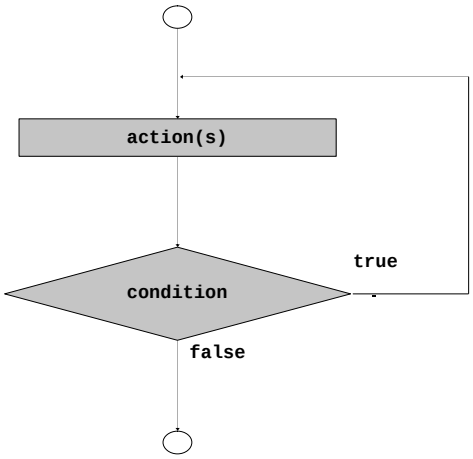


Fig. 5.13 Flowcharting the **do**/**while** repetition structure.

Outline

DoWhileLoop.cs

```
// DoWhileLoopCounter.cs
// The do/while repetition structure.

using System;

class DoWhileLoopCounter
{
   static void Main( string[] args )
   {
      int counter = 1;

      do
      {
         Console.WriteLine( counter );
         counter++;
      } while ( counter <= 5 );

   } // end method Main

} // end class DoWhileLoop
```

The counter is initialized to one

These actions are performed at least one

The incrementing task

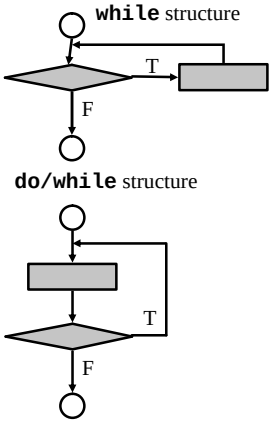
Continue looping as long as counter is less than 6

Program Output

```
1
2
3
4
5
```

Comparing the `while` and do Loops

- The **`while`** loops vs. the **`do/while`** loops
  - Using a **`while`** loop
    - Condition is tested
    - The action is performed
    - Loop could be skipped altogether
  - Using a **`do/while`** loop
    - Action is performed
    - Then the loop condition is tested
    - Loop will be run at least once



Question: write a program to get max from user and then print the numbers from 1 to max

Outline

- Admin. and review
- Loop statements
  - `while` statement
  - Nested control
  - `do/while` statement
  - `for` statement
- `break` and `continue` statements

The `for` Statement

- The *`for` statement* has the following syntax:

Reserved word

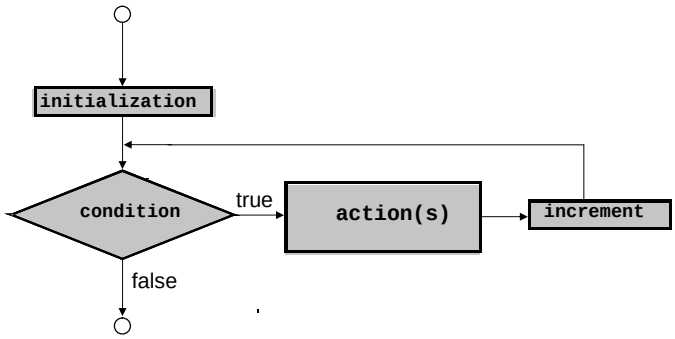
The *initialization* portion is executed once before the loop begins

The statement is executed until the *condition* becomes false

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

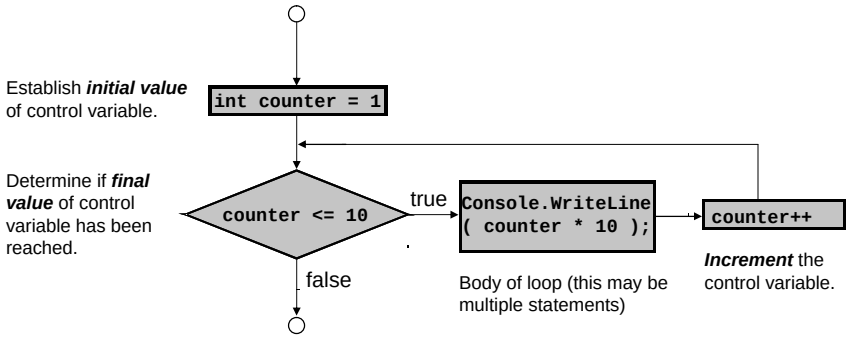
The *increment* portion is executed at the end of each iteration

Flowchart of a `for` loop




```
for ( initialization ; condition ; increment )
   action(s);
```

The `for` Statement: Example




```
for (int counter = 1; counter <= 10; counter++)
  Console.WriteLine (counter * 10);


// beginning of the next statement
```


The for Statement

- A `for` loop is equivalent to the following `while` loop:

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

The `for` Statement

- It is well suited for executing a specific number of times that can be determined in advance
  - Increment/Decrement
    - When incrementing
      - In most cases < or <= is used
    - When decrementing
      - In most cases > or >= is used

- Example: ForCounter.cs

Outline

ForCounter.cs

```
1    // ForCounter.cs
2    // Counter                                    for structure.
3
4    using Syst
5
6    class ForCounter
7    {
8       static void Main(
9       {
10         // initializat                                  nting
11         // are all included in the for structure
12         for ( int counter = 1; counter <= 5; counter++ )
13            Console.WriteLine( counter );
14      }
15   }
```

This is where the counter variable is initialized. It is set to 1

The counter is incremented (1 is added to it)

The loop w greater than five (it will stop once it gets to six)

Program Output

```
1
2
3
4
5
```

Note: **counter** can only be used in the body of the **for** loop!

When the loop ends the variable expires! We will discuss this issue later!

The flexibility of the `for` Statement

- Each expression in the header of a `for` loop is optional
  - If the initialization is left out, no initialization is performed
  - If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
  - If the increment is left out, no increment operation is performed
- Both semi-colons are always required in the `for` loop header

```
for ( ; ; )
{
   // do something
}
```

23

A Problem to Think About

How to print this?

```
xxxxxxxx
xxxxxxx
xxxxxx
xxxxx
xxxx
xxx
xx
x
```

What about this?

```
xxxxxxxxxxxxxxx
 xxxxxxxxxxxxx
  xxxxxxxxxxx
   xxxxxxxxx
    xxxxxxx
     xxxxx
      xxx
       x
```

24

Outline

- Admin. and review
- Loop statements
  - `while` statement
  - Nested control
  - `do/while` statement
  - `for` statement
- `break` and `continue` statements

25

Statements `break` and `continue`

- Used to alter the flow of control
  - The **`break`** statement
    - Used to exit a loop early
  - The **`continue`** statement
    - Used to skip the rest of the statements in a loop and restart at the first statement in the loop
- Programs can be completed without their usage; use with caution.

26

Using `Break`: Loop-and-a-Half Idiom

Initialize total to zero *Initialize counter to zero* ***Input the first grade (possibly the sentinel)*** *While (grade != sentinel)* *Add this grade into the running total* *Add one to the grade counter* ***Input next grad (possibly the sentinel)*** *If the counter is not equal to zero* *Set the average to the total divided by the counter* *Print the average* *Else* *Print "No grades were entered"*	*Initialize total to zero* *Initialize counter to zero* *While (true)* ***Input next grade (possibly the sentinel)*** *If (the user has entered the sentinel)* ***break;*** *Add this grade into the running total* *Add one to the grade counter* *If the counter is not equal to zero* *Set the average to the total divided by the counter* *Print the average* *Else* *Print "No grades were entered"*

Outline

BreakTester.cs

```
1    // BreakTester.cs
2    // Using the break statement in a for structure.
3
4    using System;
5
6
7    class BreakTester
8    {
9       static void
10      {
11         string o
12         int count;
13
14         for ( count = 1; count <= 10; count++ )
15         {
16            if ( count == 3 )
17               break;
18                                  // if count == 3
19
20                                  " ";
21
22         } // end for loop
23
24         output += "\nBroke out of loop at count = " + count;
25
26         Console.WriteLine( output );
28
29      } // end method Main
30
31   } // end class BreakTester
```

A loop that starts at one, goes to ten, and increments by one

If count = 3 then break out of the loop

Display the last value that the counter was at before it broke

Prints the message!

Outline

ContinueTester.cs

```
// ContinueTester.cs
// Using the continue statement in a for structure.

using System;


class ContinueTester
{
   static void Main( string[] )
   {
      string output = "";

      for ( int count = 1; count <= 10; count++ )
      {
         if ( count == 5 )
            continue;       // skip remaining code in loop
                            // only if count == 5

         output += count + " ";
      }

      output += "\nUsed continue to skip printing 5";

      Console.WriteLine( output );

   } // end method Main

} // end class ContinueTester
```

A loop that starts at 1, goes to 10, and

If count = 5 then continue looping causing the program to skip the rest of the loop

Prints the message.

The `switch` Statement

- The *`switch` statement* provides another means to decide which statement to execute next
- The switch statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement list associated with the first value that matches

The `switch` Statement: Syntax

The general syntax of a switch statement is:

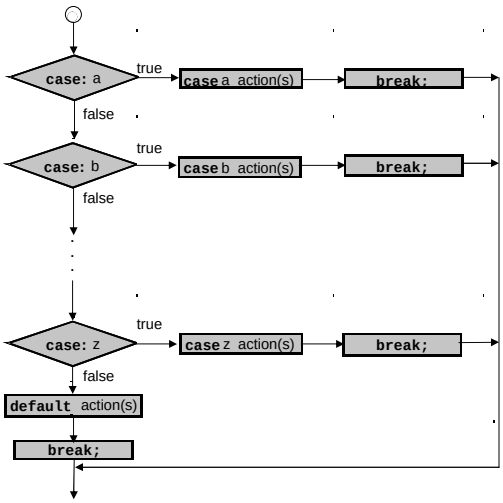
`switch` and `case` and `default` are reserved words

```
switch ( expression )
{
   case value1 :
      statement-list1
   case value2 :
      statement-list2
   case  ...

   default :
      statement-list
}
```

If *expression* matches *value2*, control jumps to here

The `switch` Statement



The `switch` Statement

- The expression of a switch statement must result in an *integral data type*, like an integer or character or a *string*
- Note that the implicit boolean condition in a switch statement is equality - it tries to match the expression with a value
- SwitchTester.cs

33

The `switch` Statement

- A switch statement can have an optional *default case* as the last case in the statement
  - The default case has no associated value and simply uses the reserved word `default`
  - If the default case is present, control will transfer to it if no other case value matches
  - If there is no default case, and no other value matches the expression, control falls through to the statement after the switch
- A *break statement* is used as the last statement in each case's statement list
  - A `break` statement causes control to transfer to the end of the switch statement

34