

Copyright © 2015 by The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House  
427 S. 4<sup>th</sup> Street #300  
Louisville KY 40202

# Flow of Control Statements in C#

.NET Cohort

Coding Bootcamp

# Lesson Goals

- Examine the flow of control statements common to C#
- Start writing code!

# Types of Flow of Control Statements

- Conditional Execution
  - if, if ... else, switch
- Looping
  - while, do, for, foreach
- Jump
  - break, continue, return, goto, throw

## The if Statement

The if statement executes the following statement(s) only if the test expression is true.

A common beginner mistake is to try to use the single equals sign (=) in an if statement. A single equals sign always sets a value to a variable. Setting x to the value 10 does not return true or false, and it will cause an error.

```
// one statement, no curly braces needed
if (x <= 10)
    x = x - 1;
```

```
// two statements, code block { }
if (x == 1)
{
    x = x + 5;
    y = x - 3;
}
```

```
if (x = 10)
{
    /* error, = is an assignment,
     * we must use == to check for
     * values */
}
```

## The if...else Statement

This is a two-way branch. If the test expression is true, statement 1 is executed; otherwise statement 2 is executed.

You can use multiple if statements for complex logic so if...else, if...else is acceptable syntax.

```
if (x < 10)
{
    Console.WriteLine("x is less than 10");
}
else if (x == 10)
{
    Console.WriteLine("x is equal to 10");
}
else
{
    Console.WriteLine("x is greater than 10");
}
```

# Switch Statements

- The switch statement implements *multi-way branching*. It is intended to be easier to read than the equivalent if/else logic.
- Switch cases (matches) are evaluated in order. Once a match is found, the statement executes and then execution moves to the bottom.
- Switch cases must break after statements are executed.

# A Switch Example

```
int x = 10;

switch (x % 2) // find remainder of x / 2
{
    case 0:
        Console.WriteLine("Even!");
        break; // go to end of switch
    default:
        Console.WriteLine("Odd!");
        break;
}
```



# Falling Through Cases

You can list multiple cases vertically if you want multiple cases to execute the same code.

```
switch (x)
{
    case 1:
    case 2:
    case 3:
        Console.WriteLine("1, 2, or 3");
        break;
    case 5:
        Console.WriteLine("It is 5");
        x *= 10;
        break;
}
```

## The while Loop

The test expression is evaluated before the loop starts. The loop executes **only if the expression evaluates to true** and will continue to execute until the test expression becomes false.

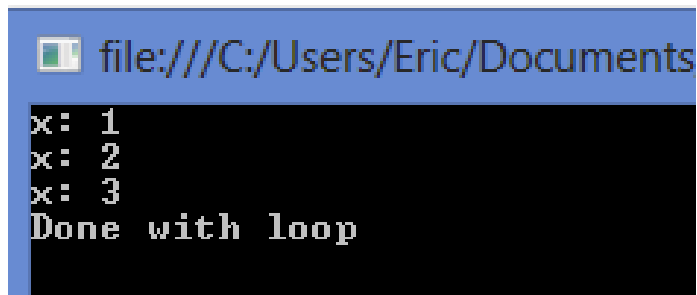
We must always be careful in loops like this that change the value of the test expression (x) at some point. If we forget to add one to x, the loop will continue running forever and eventually crash the program!

This situation is called an *infinite loop* and it makes you look silly.

```
int x = 1;

while (x < 4)
{
    Console.WriteLine("x: {0}", x);
    x++; // add one
}

Console.WriteLine("Done with loop");
```

A screenshot of a Windows command prompt window. The title bar is blue and shows the file path "file:///C:/Users/Eric/Documents". The command prompt has a black background with white text. It shows the output of the C# code: "x: 1", "x: 2", "x: 3", and "Done with loop".

```
file:///C:/Users/Eric/Documents
x: 1
x: 2
x: 3
Done with loop
```

## The do Loop

Unlike the while loop, the do loop evaluates at the bottom. Thus, a do loop will always execute at least one time.

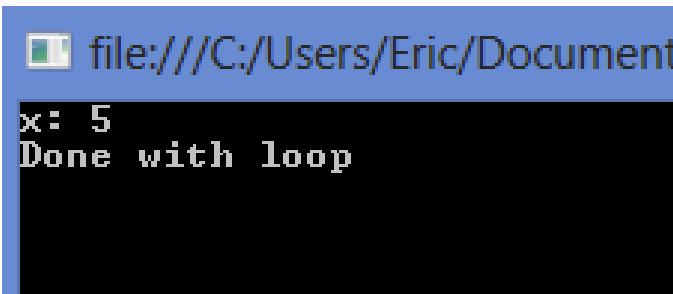
We use a while loop whenever we wouldn't want the code to execute if the test condition fails right away. We use the do loop when we want the code to execute at least once, regardless of the test condition.

A good example is a menu. Typically we want to show the menu at least once and only exit if the user inputs the quit option.

```
int x = 5;

do
{
    Console.WriteLine("x: {0}", x);
    x++; // add one
} while (x < 4);

Console.WriteLine("Done with loop");
```

A screenshot of a Windows command prompt window. The title bar is blue and shows the file path "file:///C:/Users/Eric/Document". The console output is as follows:  
x: 5  
Done with loop

## The for Loop

The for loop executes so long as the test expression at the top returns true. The for definition contains an initializer, a condition, and an iteration expression. In other words, it counts up or down a pre-set number of times.

The most common use of a for loop is going through items in a list where the size of the list is known.

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine(i);  
}  
  
Console.WriteLine("Done with loop");
```

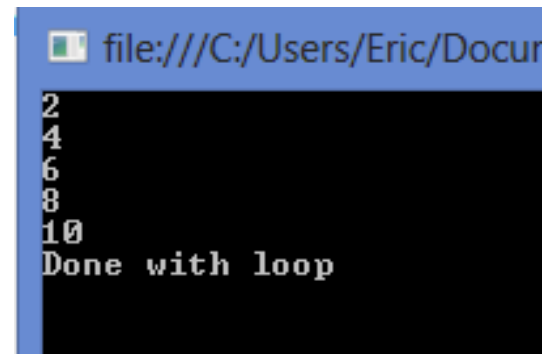


```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
Done with loop
```

# Counting by twos

```
for (int i = 2; i <= 10; i+=2)
{
    Console.WriteLine(i);
}

Console.WriteLine("Done with loop");
```

A screenshot of a Windows console window. The title bar is blue and shows the file path 'file:///C:/Users/Eric/Docur'. The console output is as follows:  
2  
4  
6  
8  
10  
Done with loop

# A Word On Loop Variables

- Though we always strive to make our variable names descriptive, traditionally loop variables are given identifiers of i, j, k, l, m, and n.
- If you have nested loops (loops within loops) it is typical for the outer loop to use i, the first inner loop to use j, then k, and so on. This can give developers a sense of “depth” in the nested code.

## Jump Statements

The break statement will immediately exit the current (innermost) loop.

The continue statement will immediately jump to the top of the current (innermost) loop

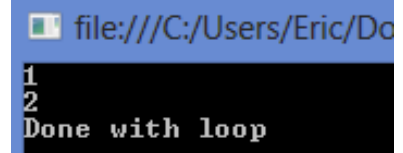
The goto statement can jump to any block it is nested in that is labeled. We don't use this except under extreme circumstances; it is considered poorly structured and hard to debug and maintain. It's so hated that I'm not even going to show it to you.

You can find more information [here](#) though.

```
for (int i = 1; i <= 10; i++)
{
    if (i == 3)
        break;

    Console.WriteLine(i);
}

Console.WriteLine("Done with loop");
```

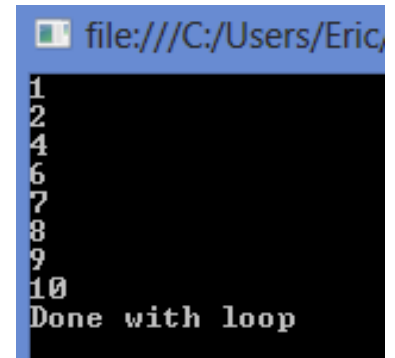


```
file:///C:/Users/Eric/Do
1
2
Done with loop
```

```
for (int i = 1; i <= 10; i++)
{
    if (i == 3 || i == 5)
        continue;

    Console.WriteLine(i);
}

Console.WriteLine("Done with loop");
```



```
file:///C:/Users/Eric/Do
1
2
4
6
7
8
9
10
Done with loop
```

# Casting Basics

- Explicit casts are when the developer calls methods to convert one type to another.
- Implicit casts occur when the compiler can figure out how to cast. It only occurs if there wouldn't be data loss (ex: byte to int).
- For now, we are going to examine the Parse and TryParse methods, which convert strings to other simple types.



# Parse

- *Parsing* takes a string that represents a value and converts it into the actual typed value.
- All of the predefined simple types have a static Parse method.
- If the string can not be parsed, the system throws an exception (error).

```
Console.Write("Enter a number: ");  
string input = Console.ReadLine();  
  
int number = int.Parse(input); // convert input string to int  
Console.WriteLine("Your number doubled is: {0}", number * 2);  
  
Console.ReadLine();
```

# TryParse

- TryParse takes two parameters and returns a bool indicating success.
  - The first parameter is the string to Parse.
  - The second parameter is the output variable to populate, if successful.
- TryParse is useful when working with users, because users are not to be trusted — they often put in bad data or typos. Crashing the program is bad form so we should always check user data and gracefully handle errors.

# TryParse in Action

```
bool validInput = false;
int number;

do
{
    Console.Write("Enter a number: ");
    string input = Console.ReadLine();

    validInput = int.TryParse(input, out number);

    if (!validInput)
        Console.WriteLine("That was not a number... try again");
} while (!validInput);

Console.WriteLine("Your number doubled is: {0}", number * 2);
```

# The Guessing Game

## DEMO

# Lab Exercise: Let's make some improvements

- Start the game by asking the player for their name. Embed the name in game messages where appropriate.
- If the player doesn't enter a number within the range (1-20), give them a proper message.
- Allow the player to enter Q at any time to quit the game (hint: break;).
- Add variables and logic to count the number of guesses it took the player.
- If the player guesses the answer on the first try, give them a special victory message.

# Lab Exercise: FizzBuzz

- Write a loop that outputs the numbers from 1 to 100.
- If the number is a multiple of 3, print the word “Fizz” next to the number.
- If the number is a multiple of 5, print the word “Buzz” next to the number.
- If it is both, print “FizzBuzz” next to the number.

# Lab Exercise: Factorizor

See Factorizor PDF in lecture directory.