



CSI 120 Week 4

LECTURE NOTES

Debugging

[di:b^g-ing]

1. *Being the detective in a crime movie where you are also the murderer.*

Logic and Decision Making

Logic and decision making in programming is similar to how we make decisions in life.

If I am getting ready to leave my house I might ask, is it cold outside?

I am asking a question with a Yes/No or True/False value.

IF it is cold outside I will get a jacket.

IF it is NOT cold outside, I will not get a jacket

What VARIABLE do I need to check to see if it is cold or not?

Revisiting the Boolean Data type

The `bool` or **Boolean** data type can store one of only 2 values: `True` or `False`

A **Boolean** variable is created using the `bool` keyword

```
bool isHot = true;  
bool isCold = false;
```

All **Boolean** statements must evaluate to either **true** or **false**. There is no other possible value for a **bool**

Relational Operators

Relational operators are used to make a comparison between two values and will evaluate to a **Boolean** value of **true** or **false**

Less than < and Greater than > are examples of **relational operators**

- $5 > 10$; This is a **bool** value and will be **false**
- $7 < 8$; This is a **bool** value and will be **true**

Variables can be compared using **relational operators**.

- For the following example assume $\text{int } x = 22$; and $\text{int } y = 15$;
- $x > y$; this is a **bool** value and will be **true**
- $y > x$; this is a **bool** value and will be **false**
- $y < x$; this is a **bool** value and will be **true**

Intro to IF Statements

An IF Statement is a way of asking a Yes/No or True/False Question. We can then take different actions based on whether the answer to our question was true or false.

The syntax for an if statement is as follows:

```
if (Boolean value)
{
    //The code inside this code block only runs if the Boolean value in parentheses is true
}
```

Assume `int x = 10;` and `int y = 5;` for the following examples.

```
if (y < x)
{
    Console.WriteLine($"{y} is less than {x}");
}
```

Using IF on a bool variable

- ❑ if checks can also be performed on bool variables as well as bool values

```
int m = 42;
```

```
int n = 7;
```

```
bool myBool = m < n;
```

```
if(myBool)
```

```
{
```

Would this code block run?

```
}
```

== and != Operator

- ❑ == Evaluates to true if both values on either side are the same
- ❑ != Evaluates to true if the values on either side are not the same. The ! can be read as “not”

```
int m = 8
```

```
int n = 7;
```

```
bool myBool = m == n; - This is false 8 does not equal 7
```

```
myBool = n == 7; -This is true the value of n is 7 and 7 equals 7
```

```
myBool = m != 100; This is true m is not equal to 100
```

```
myBool = m != 8; This is false because m is equal to 8
```


LET'S CODE IT! – IntroToIF



All Relational Operators

An overview of all relational operators in C#

The table below lists all relational operators (Liberty & MacDonald, 2009; Dorman, 2010):

Name and meaning	Operator	Expression	Evaluates to (when <code>x</code> is 10)
Equals	<code>==</code>	<code>x == 12</code>	False
		<code>x == 10</code>	True; <code>x</code> is only equal to 10 and unequal to everything else
Not equals	<code>!=</code>	<code>x != 10</code>	False
		<code>x != 9</code>	True; <code>x</code> is unequal to everything except 10
Greater than	<code>></code>	<code>x > 3</code>	True
		<code>x > 10</code>	False; <code>x</code> is 10 and not greater than 10
Greater than <i>or</i> equal to	<code>>=</code>	<code>x >= 0</code>	True; <code>x</code> is greater than 0
		<code>x >= 10</code>	True; <code>x</code> is equal to 10
		<code>x >= 99</code>	False
Less than	<code><</code>	<code>x < 9</code>	False
		<code>x < 50</code>	True; <code>x</code> is less than 50.
Less than <i>or</i> equal to	<code><=</code>	<code>x <= 9</code>	False
		<code>x <= 10</code>	True; <code>x</code> is equal to 10
		<code>x <= 25</code>	True; <code>x</code> is less than 25.

Tip: The equals operator has two equals signs (`==`) while the [assignment operator](#) (`=`) has one. A mistake that programmers make from time to time is confusing these two (Liberty & MacDonald, 2009).

if else structure

- ❑ else allows you to write a code block that will only run should an if statement fail to run. Consider the following examples

```
if (x > 100)
```

```
{
```

```
    Console.WriteLine($"{x} is greater than 100");
```

```
}
```

The else only runs when the statement inside of the if is false

```
else
```

```
{
```

```
    Console.WriteLine($"{x} is not greater than 100");
```

```
}
```

else if

- ❑ else if allows you to make another comparison after the initial IF. The else if will only run if all previous if statements have failed

```
if (grade > 90)
```

```
{
```

```
    Console.WriteLine("The grade is an A");
```

```
}
```

```
else if (grade > 80)
```

```
{
```

```
    Console.WriteLine("The grade is a B");
```

```
}
```

```
else if (grade > 70)
```

```
{
```

```
    Console.WriteLine("The grade is a C");
```

```
}
```


LET'S CODE IT! – IfElse



Logical Operators

- ❑ Logical operators are a way of combining Boolean statements
- ❑ `&&` can be read as AND. For a Boolean statement using `&&` to be true both terms must be true
 - ❑ `bool myBool = true && false;` - myBool will be false
 - ❑ `mybool = true && true;` - myBool will be true
 - ❑ `myBool = false && false;` myBool will be false
- ❑ `||` can be read as OR. In order for a Boolean statement using `||` to be true at least one of the terms must be true
 - ❑ `myBool = false || true;` - myBool will be true
 - ❑ `myBool = false || false;` - myBool will be false
 - ❑ `myBool = true || true;` - myBool will be true

Logical Operators Continued

- ❑ ! can be read as NOT and will reverse the value of any Boolean statement or variable

- ❑ myBool = !(true); - myBool will be false

- ❑ myBool = !(5>2); myBool will be false

- ❑ myBool = !(5>10); myBool will be true

- ❑ Consider the following examples:

```
int x = 10;
```

```
int y = 25;
```

```
bool boolA = (x > 5) && (y == 30);
```

```
bool boolB = ((x != 3) || (y < 10)) && (y >= 25);
```

LET'S CODE IT! – LogicalOperators



Labels and Goto

- ❑ A label in C# can be used to create a location in code that can be automatically skipped to. This can be used along with both if/else as well as try and catch to validate input.
- ❑ A **label** can be created by Typing the name of the label followed by a colon
 - ❑promptUser:
- ❑ That **label** can now be skipped to by using the **goto** keyword
 - ❑goto promptUser;

Labels/Goto Try/Catch

- ❑ Combining a Label and Goto with a try catch can allow you to make sure that a Convert or Parse is successful

```
promptUser: //This is the label
```

```
Console.WriteLine("Enter a number between 0 and 10"); //Prompt the user
```

```
int number;
```

```
try
```

```
{
```

```
    number = int.Parse(Console.ReadLine());
```

```
}
```

```
catch
```

```
{
```

```
    goto promptUser;
```

```
}
```


LET'S CODE IT! – TryCatchLabelGoto



Labels/Goto and If

- ❑ Combining a Label and Goto with an If/Else can give you a way to check user input.
- ❑ Consider the following example

promptUser:

```
Console.WriteLine("Enter a number between 0 and 10");
```

```
int number = int.Parse(Console.ReadLine());
```

```
if(number < 0 || number > 10)
```

```
{
```

```
    goto promptUser;
```

```
}
```


LET'S CODE IT! – IfLabelGoto

