Conditional Logic

Conditional Logic is all about the IF word. In fact, it is practically impossible to program effectively without using IF. You can write simple programs like our calculator. However, for anything more complicated, you need to get the hang of Conditional Logic. As an example, take the calculator program you have just written. It only has a Plus button. We will be adding a second button soon, a Subtract button. Now, you cannot say beforehand which of the two buttons your users will click. Do they want to add, or subtract? You need to be able to write code that does the following:

IF the Plus button was clicked, add up
IF the Minus button was clicked, subtract

You can rearrange the two statements above.

Was the Plus button clicked? Yes, or No?
Was the Minus button clicked? Yes, or No?

Therefore, the answer for each is either going to be (Yes, or No) the button is either clicked, or not clicked.

IF Statements

To test for YES or NO values, you can use an IF statement. You set them up like this:

```
if ( )
{
 }
```

So you start with the word “if” (in lowercase), and type a pair of round brackets. In between the round brackets, you type what you want to check for (Was the button clicked?). After the round brackets, it is convenient (but not strictly necessary) to add a pair of curly brackets. In between your curly brackets, you type your code. Your code is what you want to happen IF the answer to your question was YES, or IF the answer was NO. Here is a coding example:

```
bool buttonClicked = true;
if (buttonClicked = = true)
{
     MessageBox.Show(“The button was clicked”);
 }
```

Notice the first line of code:

```
bool buttonClicked = true;
```

This is a variable type "**bool**". The bool is short for Boolean. You use a Boolean variable type when you want to check for **true** or **false** values (YES, or NO, if you prefer). This type of variable can only ever be true or false. The name of the bool variable above is **buttonClicked**. We have set the value to true. The next few lines are our IF Statement:

```
if (buttonClicked = = true)
{
    MessageBox.Show("The button was clicked");
}
```

The double equals sign (= =) is something else you need to get used to when using IF Statements. It means, "Has a value of". The double equals sign known as a Conditional Operator. However, the whole of the line reads:

"IF buttonClicked has a value of true"

If you miss out one of the equals signs, you would have this:

```
if (buttonClicked = true)
```

What you are doing here is assigning a value of true to the variable **buttonClicked**. It is not checking if **buttonClicked** "Has a value of" **true**. The difference is important, and will cause you lots of problems if you get it wrong! In between the curly brackets of the IF statement, we have a simple MessageBox line. However, this line will be executed. IF **buttonClicked** has a value of **true**. Let us try it out. Start a new project for this (File > New project). Add a button to your new form, and set the Text property to "IF Statement". Double click the button, and add the code from above. Therefore, your coding window will look like this:

```
private void button1_Click(object sender, EventArgs e)
{

    bool buttonClicked = true;

    if (buttonClicked == true)
    {
        MessageBox.Show("The button was clicked");
    }

}
```

Run your program and click the button. You should see the message box. Now halt the program and change this line:

```
bool buttonClicked = true;
```

To this:

```
bool buttonClicked = false;
```

Therefore, the only change is from **true** to **false**. Run your program again, and click the button. What happens? Nothing! The reason that nothing happens is that our IF Statement is checking for a value of true:

```
if (buttonClicked = = true)
```

C# will execute the code between the curly brackets IF, and only IF, **buttonClicked** has a value of **true**. Since you changed the value to **false**, it does not bother with the MessageBox in between the curly brackets, but moves on instead.

Else

You can also say, what should happen if the answer was false. All you need to do is make use of the **else** word. You do it like this:

```
if (buttonClicked = = true)
{
}
else
{
}
```

So you just type the word **else**, after the curly brackets of the IF Statement. Then add another pair of curly brackets. You then write your code for what should happen if the IF Statement was **false**. Change your code to this:

```
if (buttonClicked = = true)
{
    MessageBox.Show("buttonClicked has a value of true");
}
else
{
    MessageBox.Show("buttonClicked has a value of false");
}
```

So the whole thing reads:

"IF it's true that **buttonClicked** has a value of **true**, do one thing. If it's not true, do another thing."

Run your program, and click the button. You should see the second MessageBox display. Halt the program and change the first line back to **true**. So this:

```
bool buttonClicked = true;
```

Instead of this:

```
bool buttonClicked = false;
```

Run the program again, and click the button. This time, the first message box will display. The whole point of using **IF … Else** Statements, though, is to execute one piece of code instead of some other piece of code.

Else … IF

Instead of using just the **else** word, you can use **else if**, instead. If we use our calculator as an example, we would want to do this:

```
bool plusButtonClicked = true;
bool minusButtonClicked = false;
if (plusButtonClicked = = true)
{
//WRITE CODE TO ADD UP HERE
}
else if (minusButtonClicked = = true)
{
//WRITE CODE TO SUBTRACT HERE
}
```

So the code checks to see which button was clicked. If it is the Plus Button, then the first IF Statement is executed. If it is the Minus Button, then the second IF Statement is executed. But **else if** is just the same as **if**, but with the word **else** at the start. In fact, we can now add a minus button to our calculator. We will use **else if**.

Now you need to create new project, open it, and add a new button into **Form1**. Set the following properties for it in the Properties Window:

Name: btnMinus
Font: Microsoft Sans Serif, 16, Bold
Location: Move it to the right of your Plus button
Size: 49, 40
Text: –

Now double click your Minus button to get at its code. Add the following two Boolean variables outside of the Minus button code, just above it:

```
bool plusButtonClicked = true;
bool minusButtonClicked = false;
```

You coding window will then look like this:

```
bool plusButtonClicked = false;
bool minusButtonClicked = false;

private void btnMinus_Click(object sender, EventArgs e)
{

}
```

Now add the following code inside of the Minus button:

```
total1 = total1 + double.Parse(txtDisplay.Text);
txtDisplay.Clear( );

plusButtonClicked = false;
minusButtonClicked = true;
```

Your coding window will then look like the one below:

```
bool plusButtonClicked = false;
bool minusButtonClicked = false;

private void btnMinus_Click(object sender, EventArgs e)
{
    totall = totall + double.Parse(txtDisplay.Text);
    txtDisplay.Clear();

    plusButtonClicked = false;
    minusButtonClicked = true;

}
```

All we have done here is to set up two Boolean variables. We have set them both to **false** outside of the code. (They have been set up outside of the code because other buttons need to be able to use them; they have been set to false because no button has been clicked yet.) When the Minus button is clicked, we will set the Boolean variable **minusButtonClicked** to true and the **plusButtonClicked** to false. However, the first two lines are exactly the same as for the Plus button:

```
total1 = total1 + double.Parse(txtDisplay.Text);
txtDisplay.Clear( );
```

The first line just moves the numbers from the text box into the **total1** variable. The second line clears the text box. Now access the code for your Plus button. Add two lines of code to the end:

```
private void btnPlus_Click(object sender, EventArgs e)
{
    totall = totall + double.Parse(txtDisplay.Text);
    txtDisplay.Clear();

    plusButtonClicked = true;
    minusButtonClicked = false;
}
```

So the only thing you are adding is this:

```
plusButtonClicked = true;
minusButtonClicked = false;
```

The Plus button resets the Boolean variables. This time, **plusButtonClicked** gets set to true, and **minusButtonClicked** gets set to false. It was the other way round for the Minus button. The reason we are resetting these Booleans variables is that we can use them in an (**if … else if)** statement. We can add up if the **plusButtonClicked** variable is true, and subtract if **minusButtonClicked** is true. We will still do the calculating in the Equals button. So change your equal's button to this:

```
private void btnEquals_Click(object sender, EventArgs e)
{

    if (plusButtonClicked == true)
    {
        total2 = totall + double.Parse(txtDisplay.Text);
    }
    else if (minusButtonClicked == true)
    {
        total2 = totall - double.Parse(txtDisplay.Text);
    }

        txtDisplay.Text = total2.ToString();
        totall = 0;
}
```

We are using Conditional Logic to decide which of the two buttons was clicked. The first if statement checks if the **plusButtonClicked** variable is true. If it is, then the addition is done (this is exactly the same as before). If the first IF Statement is false, then C# moves down to the else if statement. If **minusButtonClicked** is true, then the subtraction is done instead. The only difference between the addition and subtraction lines is the Operator symbols: a plus (+) instead of a minus (-). The final two lines of code are the same as before – convert the number to text and display it in the text box, and then reset the total1 variable to zero.

Run your calculator and try it out. You should be able to add and subtract.

Conditional Operators

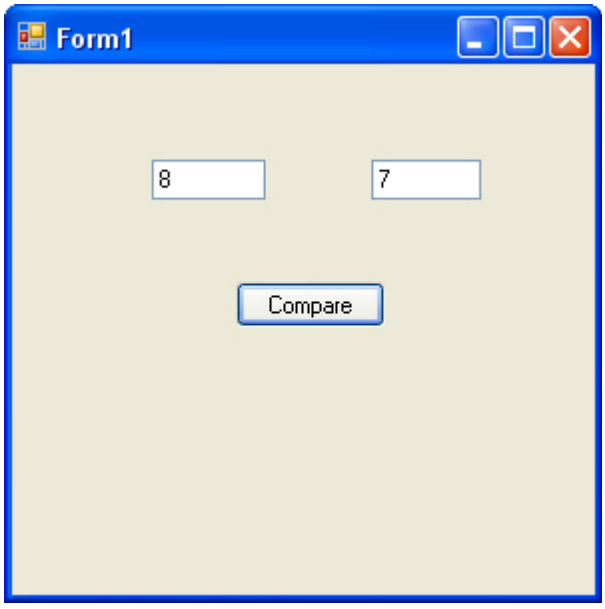
You've already met one Conditional Operator, the double equals sign (= =). You use this in IF Statements when you want to check if a variable "has a value of" something:

```
if ( myVariable = = 10)
{
//EXECUTE SOME CODE HERE
}
```

Therefore, the above line reads, "IF whatever is inside of **myVariable** has a value of 10, execute some code". Other Conditional Operators you will use when you are coding are these:

>	**Greater Than**
<	**Less Than**
>=	**Greater Than or Equal to**
<=	**Less Than or Equal to**
!=	**Not Equal to**
!	**Not**
&&	**And**
||	**Or**

Because you need to learn these Operators, let us get some practice with them. In between the round brackets after the word "switch", we have typed the name of our variable (**theOperator**). We want to check what is inside of this variable. It will be one of four options: +, -, *, /. Therefore, after the first case, we type a plus symbol. It is in between double quotes because it is text. You end a **case** line with a colon:



Double click the button to get at the coding window. What we will do is to get the numbers from the text boxes, test, and compare them. Therefore, the first thing to do is to set up some variables:

```
int firstNumber;
int secondNumber;
```

Then get the text from the text boxes and store them in the variables (after converting them to integers first).

```
firstNumber = int.Parse(textBox1.Text);
secondNumber = int.Parse(textBox2.Text);
```

What we want to do now is to compare the two numbers. Is the first number bigger than the second number? To answer this, we can use an IF Statement, along with one of our new Conditional Operators. So add this to your code:

```
if (firstNumber > secondNumber)
{
MessageBox.Show("The first number was greater than the second number");
}
```

Your coding window will then look like this (our message box above is only on two lines because it cannot all fit on this page):

```
private void button1_Click(object sender, EventArgs e)
{
  int firstNumber;
  int secondNumber;

  firstNumber = int.Parse(textBox1.Text);
  secondNumber = int.Parse(textBox2.Text);

  if (firstNumber > secondNumber)
  {
    MessageBox.Show("The first number was greater than the second number");
  }

}
```

Therefore, in between the round brackets after **if**, we have our two variables. We're then comparing the two and checking to see if one is **Greater Than** (>) the other. If **firstNumber** is Greater Than **secondNumber** then the message box will display. Run your program and click your button. You should see the message box display. Type a (6) in the first text box, and click the button again. The message box will not display. It will not display because (6) is not greater than (7). The message box code is inside of the curly brackets of the IF Statement. Moreover, the IF Statement only is executed, if **firstNumber** is Greater Than **secondNumber**. If it is not, C# will just move on to the next line. You do not have any more lines, so C# is finished. Stop your program and go back to your code. Add a new if statement below your first one:

```
if (firstNumber < secondNumber)
{
MessageBox.Show("The first number was less than the second number");
}
```

Again, our message box above is spread over two lines because there is not enough room for it on this page. Your message box should go on one line. However, the code is just about the same! The thing we've changed is to use the Less Than symbol (<) instead of the Greater Than symbol (>). We have also changed the text that the message box displays. Run your program, and type a (6) in the first text box. You should see your new message box display. Now type an (8) in the first text box, and click your button. The first message box will display. Can you see why? If your program does not work at all, make sure it is like ours in the image below:

```
int firstNumber;
int secondNumber;

firstNumber = int.Parse(textBox1.Text);
secondNumber = int.Parse(textBox2.Text);

if (firstNumber > secondNumber)
{
    MessageBox.Show("The first number was greater than the second number");

}

if (firstNumber < secondNumber)
{
    MessageBox.Show("The first number was less than the second number");

}
```

With your program still running, type a (7) in the first box. You will then have a (7) in both text boxes. Before you click your button, can you guess what will happen? The reason that nothing happens at all is that you have not written any code to say what should happen if both numbers are equal. For that, try these new symbols:

>=	**Greater Than or Equal to**

And these ones:

<=	**Less Than or Equal to**

Try these new Conditional Operators in place of the ones you already have. Change the text for your message boxes to suit. Run your code again. When you click the button, both message boxes will display, one after the other. Can you see why this happens? Another Conditional Operator to try is "Not Equal To" (!=). This is an exclamation mark followed by an equals sign. It is used like this:

```
if (firstNumber != secondNumber )
{
    //SOME CODE HERE
}
```

So, "IF firstNumber is not equal to secondNumber execute some code". You can even use the exclamation mark by itself. You do this when you want to test for a false value between the round brackets after IF statement. It is mostly used with Boolean values. Here is an example:

```
bool testValue = false;
if (!testValue)
{
MessageBox.Show("Value was false");
}
```

Therefore, the exclamation mark goes before the Boolean value you want to test. It is a shorthand way of saying "If the Boolean value is false". You can write the line like this instead:

```
if (testValue = = false)
```

However, experienced programmers just use the exclamation mark instead. It has called the NOT Operator. On the other hand, the "IF NOT true" Operator. The final two Operators we'll have a look at are these:

&&	means **AND**
||	means **OR**

These two are known as Logical Operators, rather than Conditional Operators (so is the NOT operator). The two ampersand together (&&) mean AND. You use them like this:

```
bool isTrue = false;
bool isFalse = false;
if ( isTrue = = false && isFalse = = false )
{
}
```

You use the AND operator when you want to check more than one value at once. Therefore, in the line above, you are checking if both values are false. If and ONLY if both of your conditions are met, the code between curly brackets will be executed. In the code above, we are saying this:

"If **isTrue** has a value of false AND if **isFalse** has a value of false then and only then executed the code between curly brackets."

If **isTrue** is indeed true, for example, then any code between curly brackets will not be executed, they both have to be false, in our code. You can test for only one condition of two being met. In which, use the OR (| |) operator. The OR operators is two straight lines. These can be found above

the back slash character on a UK keyboard, which is just to the left of the letter "Z". (The | character is known as the pipe character.) You use them like this:

```
bool isTrue = false;
bool isFalse = false;
if ( isTrue = = false || isFalse = = false )
{
}
```

We are now saying this:

"If **isTrue** has a value of false OR if **isFalse** has a value of false then and only then executed the code between curly brackets."

If just one of our variables is false, then the code in between curly brackets will get executed.

Try not to worry if you do not have a thorough grasp of all the Conditional Operators yet – you will get the hang of them as you go along. However, try the next exercise.