C# Series: Basic Calculator

Text Edition
## Contents

### Section I Introduction and Set up
- **Part I** Introduction
- **II** New Project
- **III** Adding Controls

### Section II Editing
- **Part I** Numerical Buttons
- **II** Negate and Dot
- **III** Operations and Equal
Section I
Section Overview

This section will introduce you to the Complete Programs tutorial series, the tutorial, creating the project, and adding controls. This project is meant for beginners, so the program will not be as complicated as the others. The program will be similar to the basic calculator in windows.

Part I Introduction

The C# Complete Program tutorials, similar to the XNA Complete Games, cover programs from start to finish. We start off with a very simple program, and get more advanced as we go. This program is a very simple calculator. We will cover a very abstract way to handle button presses and operations.
Part II  New Project

It is time to get started! Open Visual Studio 2008 (or the express editions) and create a new C# Windows Forms Application called “BasicCalculatorTutorial”. The first thing you need to do is change the dimensions of the window to something a bit wider. Change the Size property of the form to a new dimension. I used the following: 350, 277. Change the Text property of the form to identify the program, “BasicCalculator”. You can rename Form1.cs if you wish; I will not do that in the tutorial. This will be our base window, and we will add new controls in part 3.
### TextBox
Add a large TextBox at the top of the form, and call it txtOutput. This will be used to communicate to the user what is entered, and what is a result.

### GroupBoxes
There will be two types of buttons, Numerical and Operations. We will be putting those buttons in their own group. So create a new GroupBox control with an object name of “gboxNumerical” and the text property set to “Numerical”. Fill the empty vertical space at the bottom. As for the horizontal space, make the control fill slightly over 50% of the form's width.
Now for the other group, fill the entire bottom right. Call this object “gBoxOperations” and set the text property to “Operations”.

**Numerical Buttons**

Now that we have groups, we need to fill them up. Add all the numerical buttons to the numerical group (0 through 9). We also need to include negation (+/-) and dot (.) buttons here since they modify the value. I organized my buttons 3x3. Set all the numerical buttons to “btn” plus that number (the 0 button should be called “btnZero”, 1 button is “btnOne”, and so on). Call the negate button “btnNegate”, and the dot button “btnDot”. As for the Text properties, use the numbers those buttons represent.
Operation Buttons

The last set of controls we need in our program are the operation buttons. Buttons are needed for addition (btnPlus), subtraction (btnMinus), multiplication (btnMultiply), division (btnDivide), equals (btnEqual), and clear (btnClear). For the text, use the symbols those represent (for clear use C).
Now is the time to edit our code to accept button presses and compute results. We need a few fields in our project (Form1.cs code):

- string buffer = ""; - Holds the current value of the entered number.
- char op; - Holds the operation needed to compute the result.
- double[] operand = new double[2]; - Holds the two operands (operand[0] OPERATION operand[1])
- double result; - Holds the result of the two operands.
- int step = 1; - Holds the current step of the compute process (step 1 means we need to fill operand[0], while step 2 means we need to fill operand[1]).

### Part I Numerical Buttons

For all the number buttons (0 to 9), set the click event to a new method called “NumberButtons”.
NumberButtons Method

This method should have been automatically generated for you since you set it as a click event.

```csharp
private void NumberButtons(object sender, EventArgs e)
{
}
```

Convert sender to a button in a new button object by either use casting, or the as keyword. I named the new button object b. If b is null, simply return to quit the method. We also do not want a hundred zeros before another digit is entered (0000001 is the same as 1). So if b.Text == “0” && buffer.Length == 0, return to quit the program.

```csharp
private void NumberButtons(object sender, EventArgs e)
{
    Button b = sender as Button;
    if ((b == null) || (b.Text == "0" && buffer.Length == 0))
        return;
}
```

All that is left to do is add the button’s text to the buffer and set txtOutput.Text to buffer.

```csharp
private void NumberButtons(object sender, EventArgs e)
{
    Button b = sender as Button;
    if ((b == null) || (b.Text == "0" && buffer.Length == 0))
        return;

    buffer += b.Text;
    txtOutput.Text = buffer;
}
```
Negate Method
Set the negate button’s click property to a new method called Negate. This method will add a minus sign at the beginning of the buffer. If the value is already negated and the negate button is pressed, remove the negation. Otherwise, add it. Finally, set the txtOutput’s Text property to the buffer.

```csharp
private void Negate(object sender, EventArgs e)
{
    if (buffer.Contains('-'))
    {
        buffer = buffer.Replace("-", "");
    }
    else
    {
        buffer = "-" + buffer;
    }
    txtOutput.Text = buffer;
}
```

Dot Method
Set the dot button’s click property to a new method Dot. We want to limit the number of dots in the buffer. No number has more than one dot, so if a dot is already in the buffer, do not add it. If there is no dot in the buffer, add a dot to it. Finally, set the txtOutput’s Text property to the buffer.

```csharp
private void Dot(object sender, EventArgs e)
{
    if (!buffer.Contains('.'))
    {
        buffer = buffer + ".";
        txtOutput.Text = buffer;
    }
}
```
Operations Method
For all the operations (+, -, *, /), set the click event to a new method called “Operations”.

```
private void Operations(object sender, EventArgs e)
{
}
```

Just like the NumberButtons method, make a new Button object and convert sender to a button. Set the op char equal to b.Text[0] (remember, a string is just an array of chars. So the char at 0 will be the correct operation we want). If we pressed an operation and the buffer is 0, set the current step’s operand to the result; otherwise, parse the buffer. Increase the step by one and set the buffer to an empty string.

```
private void Operations(object sender, EventArgs e)
{
    Button b = sender as Button;
    op = b.Text[0];

    if (buffer.Length == 0)
        operand[step - 1] = result;
    else
        operand[step - 1] = Double.Parse(buffer);

    step++;
    buffer = "";
}
```
Equal Method

Set the equal button’s click property to a new method Equal. If the buffer has some data in it, parse it to operand[1]. Now we need a switch statement to handle the op variable. For all the operations, result = operand[0] OPERATION operand[1];. Finally, set the txtOutput’s Text property to the buffer, set the step to 1, and set the buffer to an empty string.

```
private void Equal(object sender, EventArgs e)
{
    if (buffer.Length != 0)
    operand[1] = Double.Parse(buffer);
    switch (op)
    {
        case '+': result = operand[0] + operand[1]; break;
        case '-': result = operand[0] - operand[1]; break;
        case '*': result = operand[0] * operand[1]; break;
        case '/': result = operand[0] / operand[1]; break;
    }
    txtOutput.Text = result.ToString();
    step = 1;
    buffer = "";
}
```

Operations Method Changes

What would happen if we entered 4 * 3 * 2 * 5 * 8 ....? With the current code, our program will return some messy results. When we enter 4 * 3 and press * again, we want it to print the result (12) and store that in the first operand since we are already set on the operation. Before we empty the buffer, remove the step++ and add an if statement. If step is 2 (we entered our second operand and pressed an operation), call the Equal method (pass it two nulls). After the call is done, set the first operand to result, and set the step to 2. If the value of step is anything else, set txtOutput.Text to operand[0].ToString(); and increase step.

```
private void Operations(object sender, EventArgs e)
{
    Button b = sender as Button;
    op = b.Text[0];
    
    if (buffer.Length == 0)
    operand[step - 1] = result;
    else
    operand[step - 1] = Double.Parse(buffer);
    
    if (step == 2)
    {
        Equal(null, null);
    }
```
operand[0] = result;
step = 2;
}
else
{
txtOutput.Text = operand[0].ToString();
step++;
}
buffer = "";
}

Clear Method
Set the Clear button’s click event to a new method Clear. When we press C one
time, it should empty the buffer but keep all the other data. If we press C again, it will
clear the operands, set the step to 1, set result to 0, and reset the op variable. An
easy way to monitor if we pressed this button twice is if the buffer has already been
emptied. At the end of the method, set txtOutput.Text to buffer.

Clear Method
private void Clear(object sender, EventArgs e)
{
    if (buffer.Length == 0)
    {
        step = 1;
        operand[0] = operand[1] = 0.0;
        op = ' ';
        result = 0.0;
    }
    else
    {
        buffer = "";

        txtOutput.Text = buffer;
    }
}
Congratulations!

You just made a basic calculator