

# Visual Basic

## Programming Lab

### Calculator Program

#### Example of Calculator Program

This Visual Basic (VB) Calculator Program is an exercise designed to have students experience the development of an *object-oriented program*. The exercise takes students through the design and development of a user interface and program instructions. VB is a programming language used to develop Windows and web applications. VB evolved from BASIC (Beginner's All-purpose Symbolic Instruction Code), which is a language first used in a command line interfaces like MS-DOS. VB programs are created using Visual Studio, which is an Integrated Development Environment (IDE). The IDE includes an interface designer and code editor, debugger, and compiler. These various components will be introduced and explained in this exercise as each component is utilized.

All computer programs begin with planning an automated solution to a problem. Remember that the information processing cycle includes:

- Input - get data
- Process – computer instructions to convert data into information
- Output – display results or information
- Store – some programs can store the results on a storage device for later use or reference

For this exercise, you are asked to create a computer program that will allow the user to enter two numbers on the screen, and then choose either to ADD or SUBTRACT the numbers and then display the results. The results area can be CLEARED from the screen so the user can enter other numbers to add or subtract. The user may also choose to EXIT the program. In other words, you are creating a simple calculator that will add or subtract two numbers entered by the user.

When developing a computer program, you begin by designing the user interface and computer instructions on paper. After designing the program, you create the program using the IDE. The program is then tested and debugged using test data for each possible process. Finalizing a computer program includes completing thorough documentation of the design and implementation processes, and writing instructions for deploying and using the program.

Synopsis of how project will be completed:

### Part 1 – Plan and Build User Interface

- I. Design the user interface (form layout).
- II. Decide which control objects (buttons) will be used to process the input.
- III. Design the required instructions to generate the desired results.
- IV. Create a Visual Basic project.
- V. Construct the interface by adding control objects and setting their properties.

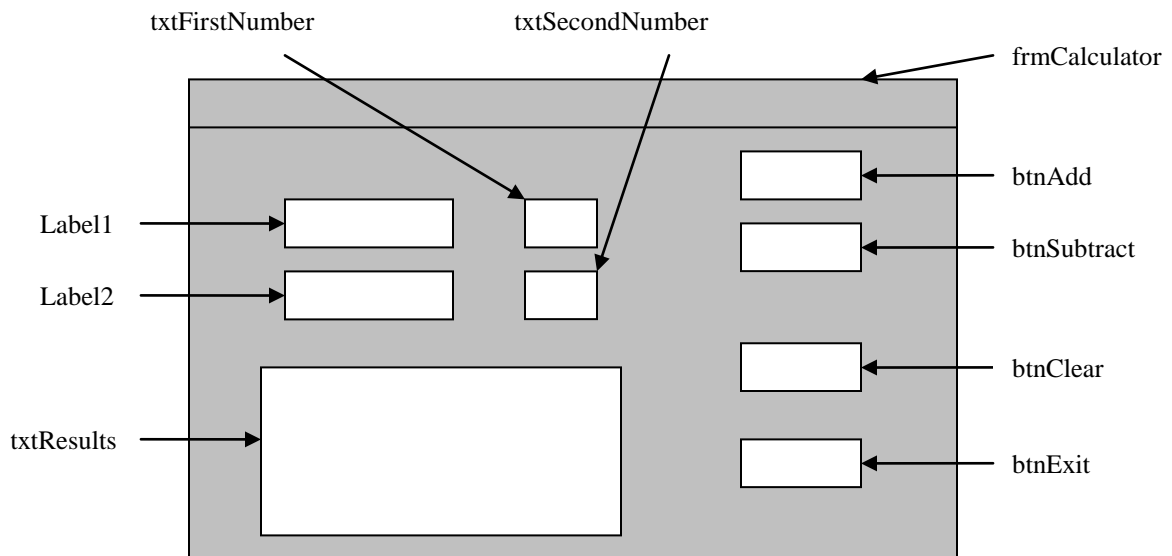
### Part 2 – Develop Program Code

- VI. Enter the computer instructions to process the input and generate the desired output.
- VII. Test and debug the program.
- VIII. Execute program and capture sample output for assignment submission.
- IX. Close the project and exit Visual Basic.
- X. Submit assignment.

## Part 1: Plan and Build User Interface

### I. Design the user interface (form layout).

To create the calculator program as shown on the cover page, two labels, three textboxes, and four buttons will need to be utilized. A programmer begins designing the graphical user interface (GUI) by sketching a form layout.



The depicted form layout design also contains names for each of the control objects. The object names' follow a **naming convention** commonly used by programmers in industry. An object name is in lowercase with each significant word capitalized to improve code readability. In addition, each name begins with a three letter prefix to identify the type object (i.e. btnExit, txtResults). **Spaces are invalid in object names so make sure you don't enter any spaces when building the project.**

What follows is list of the control objects used in the project. The control object name and a short description of the object are provided.

<b>CONTROL OBJECTS USED IN THIS PROJECT</b>	
<b>Control Object</b>	<b>Description</b>
<b>Labels:</b>  <b>Project examples:</b> Label1 Label2	Used next to other controls to identify information on the form. Labels can also be used to display (output) results.  NOTE: The default object names Label1 and Label2 are acceptable for these two controls since they will not be referenced in code. These labels are used to inform the user what should be entered in the two textboxes.
<b>TextBoxes</b>  <b>Project examples:</b> txtFirstNumber txtSecondNumber  txtResults	Used to get information from the user (input) or display results (output).  Two separate textboxes will be used to enter each of the two numbers.  The textbox that will be used to display the results.
<b>Buttons:</b>  <b>Project examples:</b> btnAdd  btnSubtract  btnClear  btnExit	Used to initiate an action (Process) by reacting to a user click event.  The button that initiates the code to add the two numbers.  The button that initiates the code to subtract the two numbers.  The button that initiates the code to clear the three textboxes.  The button that initiates the code to end the program.
<b>Form:</b> frmCalculator	The name of the Form on which the GUI will be created.

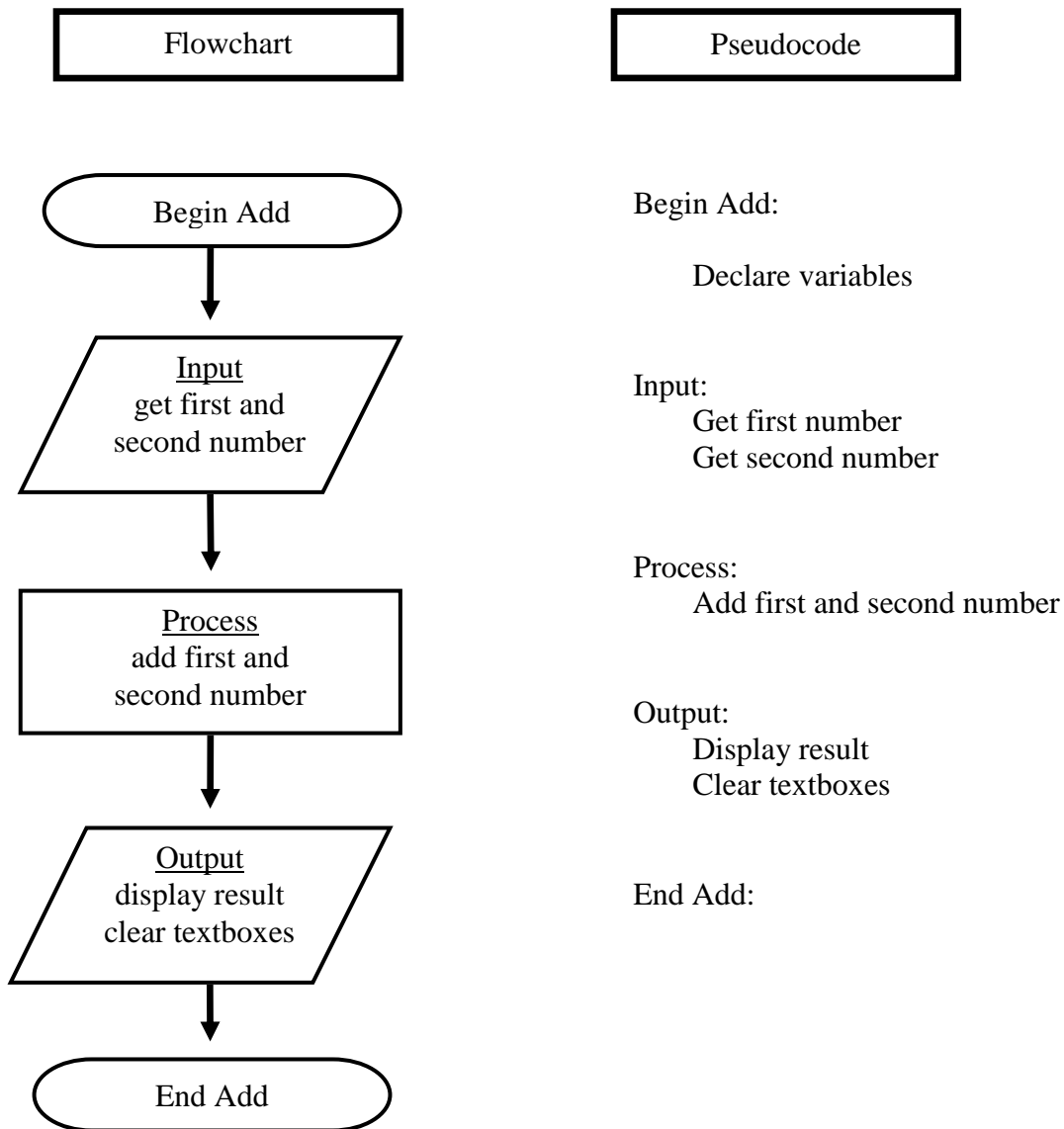
## **II. Decide which control objects (buttons) will be used to process the input.**

User interfaces are created on a form by adding control objects to it. After the application is completed and executed, the user will see labels asking that two numbers be entered in separate textboxes. After entering two numbers, the user selects whether to add or subtract the two numbers, clear the form, or exit the program by clicking one of the four buttons. The sub procedure which will handle each separate button's click event is set as the application is being built.

## **III. Design the required instructions to generate the desired results**

When the user clicks on a button, the button click event is triggered. Sub procedures with computer instructions will be written for each button, because the computer instructions to add, subtract, clear, or exit are not created automatically as the user interface is developed.

A programmer must design the logic flow using such tools as a flowchart or pseudocode<sup>1</sup>. The design should then be desk checked to make sure the design will work as desired. When a programmer desk checks their design, they play computer and walk through the logic using test data to see if the expected results would be generated in all cases.



See page 32 for a description of the flowcharting symbols used above.

---

<sup>1</sup> Pseudocode is used to write out the intended sequence of programming statements to describe what the computer will do (process) *without strictly following programming language syntax* (rules and commands). It is used in the planning process to note all the steps that will be needed, and may be clarified with natural language when necessary to emphasize the detailed handling of the process(es).

#### IV. Create a Visual Basic project.

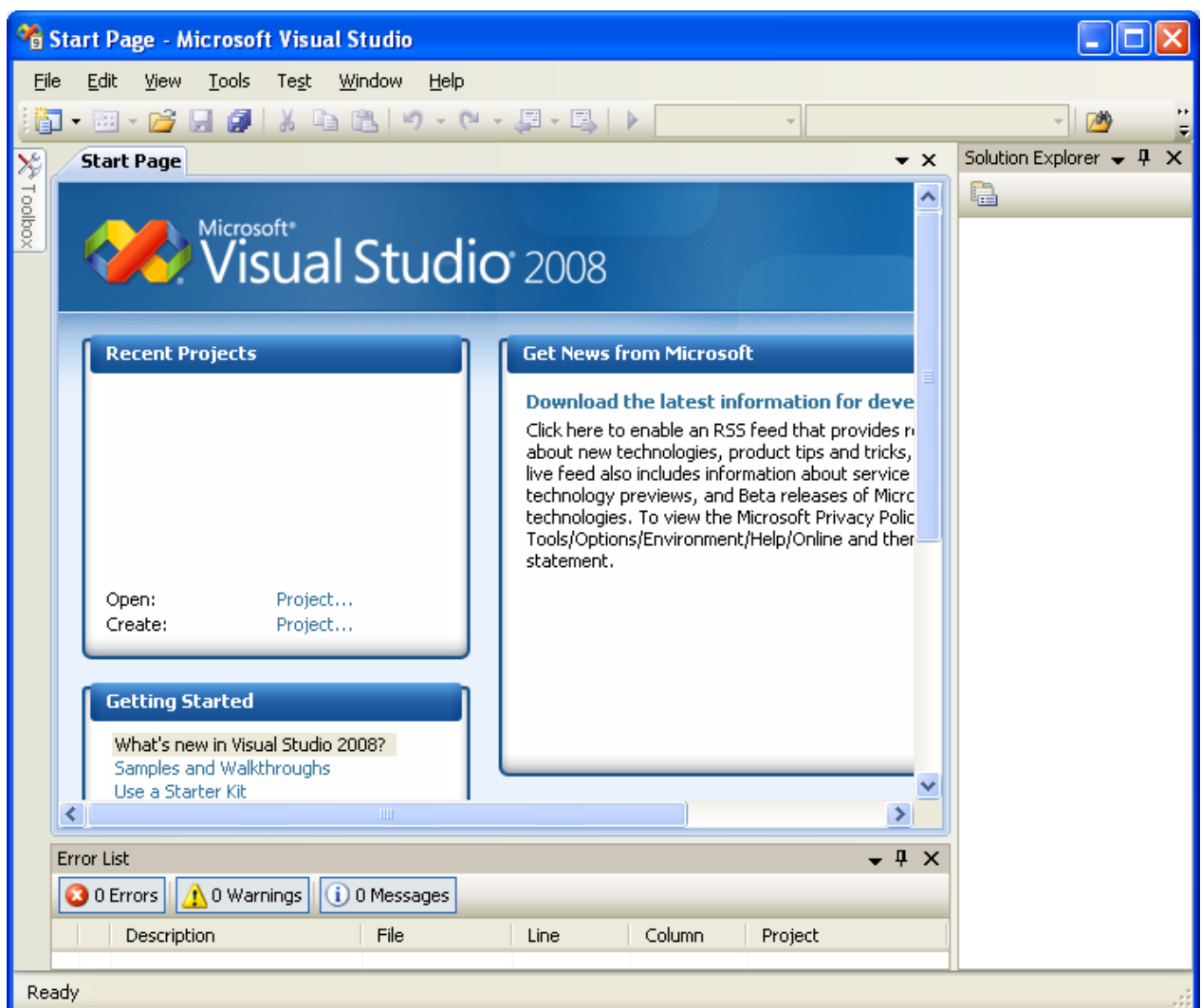
After the interface and logic have been designed, the programmer can begin building the application. Visual Basic applications are stored in projects. **The process of creating a project is different in Visual Studio Professional and Visual Basic Express Edition, so make sure you follow the instructions for the version you are using.** In the computer lab you will be using the Professional version, and at home students are mostly likely using the Express Edition.

##### 1. Instructions for **Visual Studio Professional**:

Click on

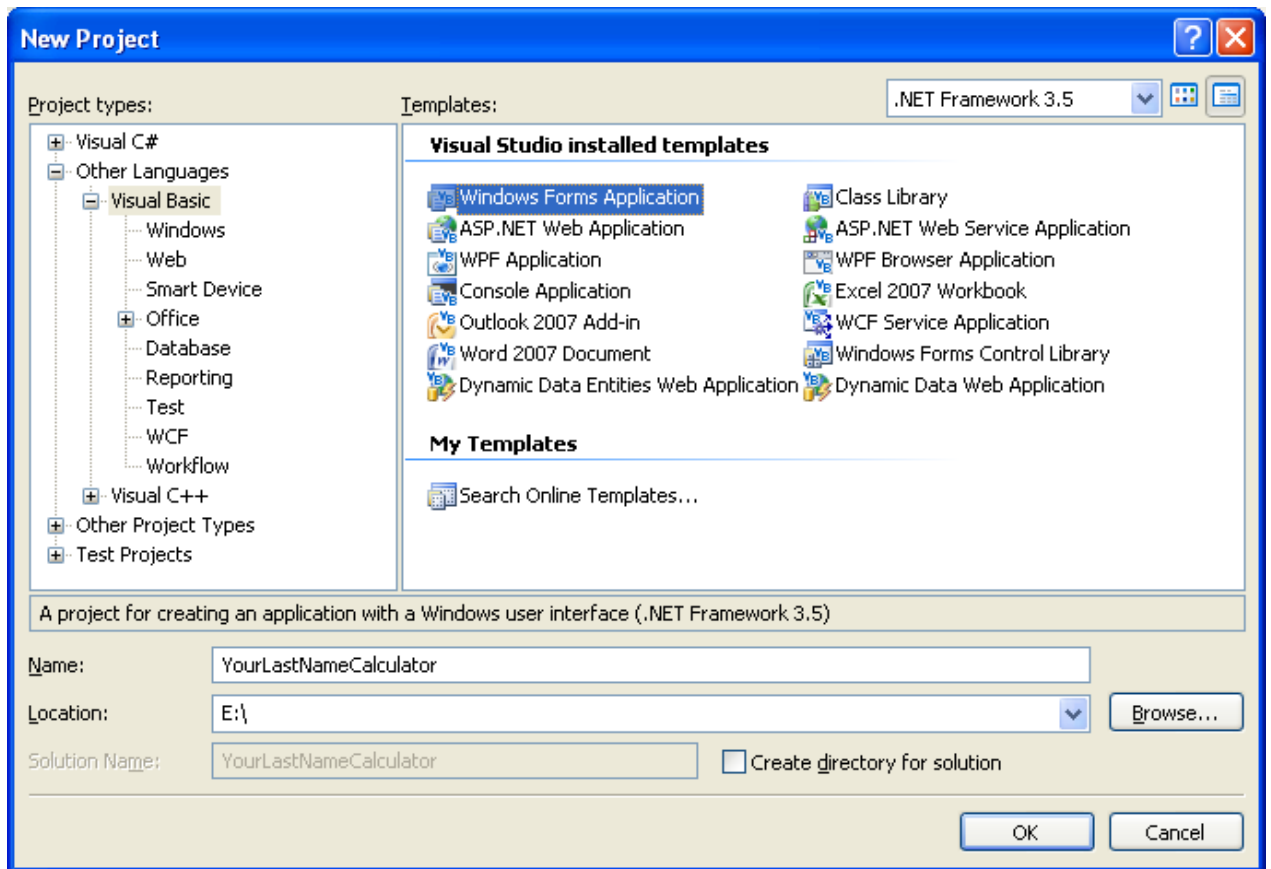
**Start → Programs → Microsoft Visual Studio 2008 → Microsoft Visual Studio 2008**

The Microsoft Visual Studio program appears. It may take a few seconds to open the first time it is used.



If necessary, maximize the application window to fill the entire screen by clicking the **maximize button** in the upper right-hand corner of the window. Do NOT be alarmed if your Start Page does not look like the one pictured. Start Pages will vary depending on who used it last and on the selected options.

2. To create a project, click on **File** on the menu bar, scroll down and mouse over **New**, and then slide over and select **Project...**



In the New Project dialog box, select or enter the following values:

- Project type: **Visual Basic**
- Template: **Windows Forms Application**
- Framework: **.Net Framework 3.5** (upper right of dialog window)
- Name: **YourLastNameCalculator** (be sure to enter your actual last name)
- Location: Click on **Browse** to find the letter assigned to your flash drive. The drive letter assigned to your flash drive will vary by computer, so it best to use the **Browse** button to navigate your flash drive.
- Remove the check mark in front of **Create directory for solution** if it has one.
- Click **OK**.

After a few seconds a new project with an empty form should be displayed.

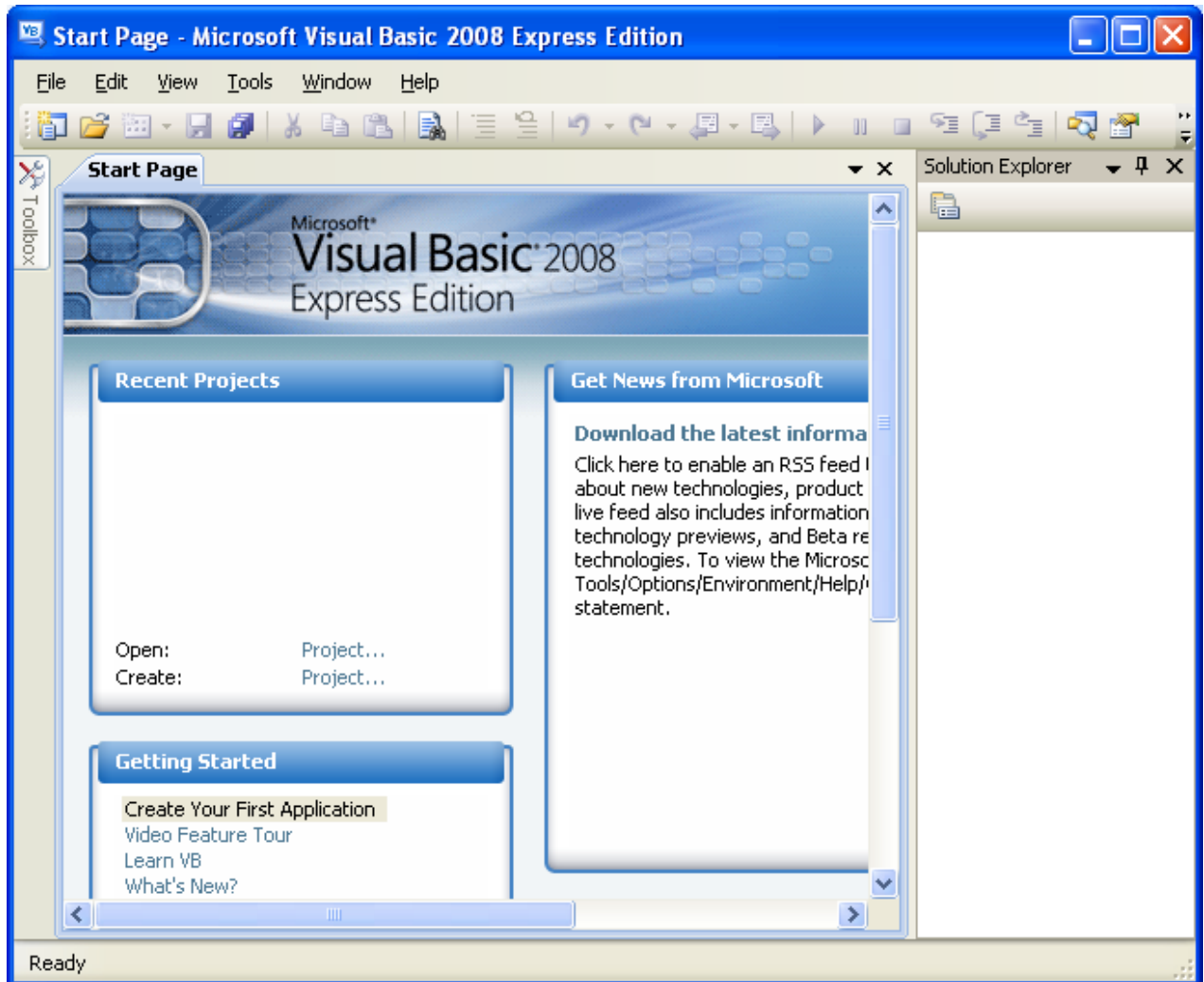
- Click on **File** on the menu bar, and select **Save All**.
- **Skip to step V on page 9**, for instructions on how to continue constructing the user interface.

1. Instructions for **Visual Basic Express Edition**:

Click on

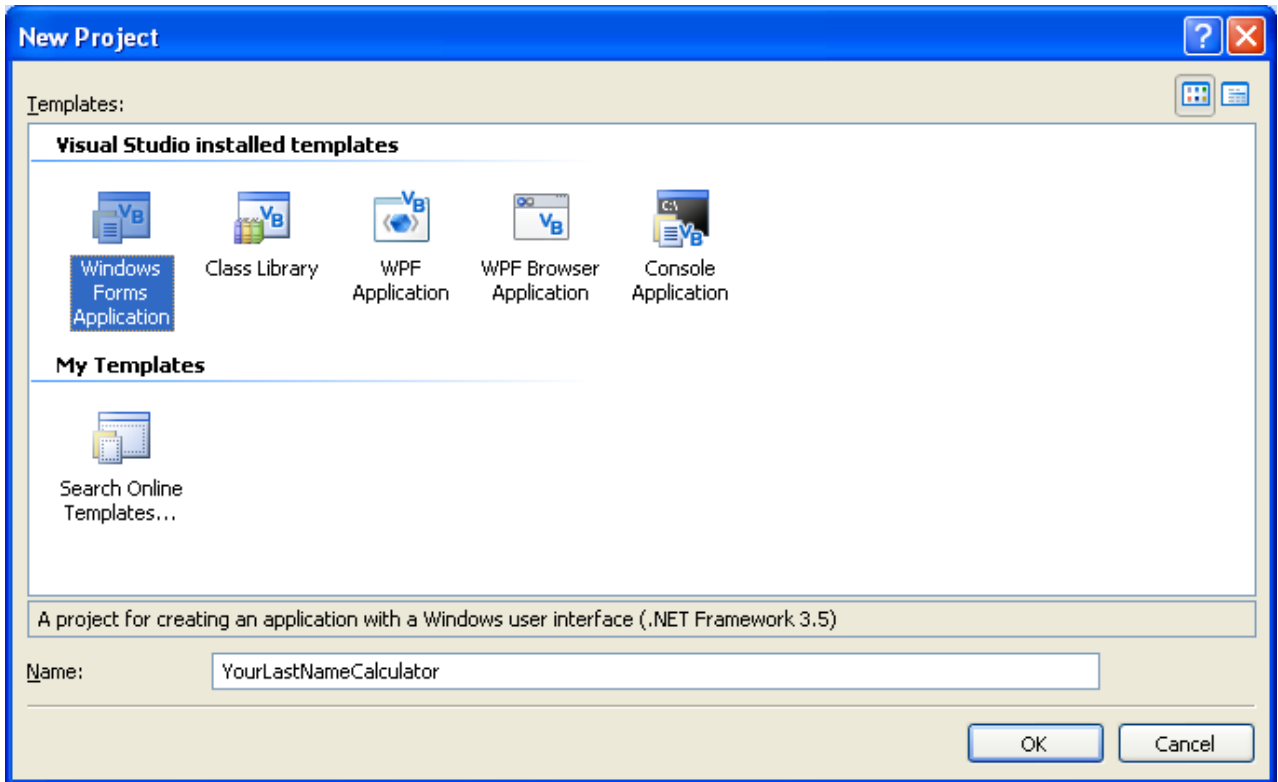
**Start → Programs → Microsoft Visual Basic 2008 Express Edition**

The Microsoft Visual Basic Express Edition program appears. It may take a few seconds to open the first time it is used.



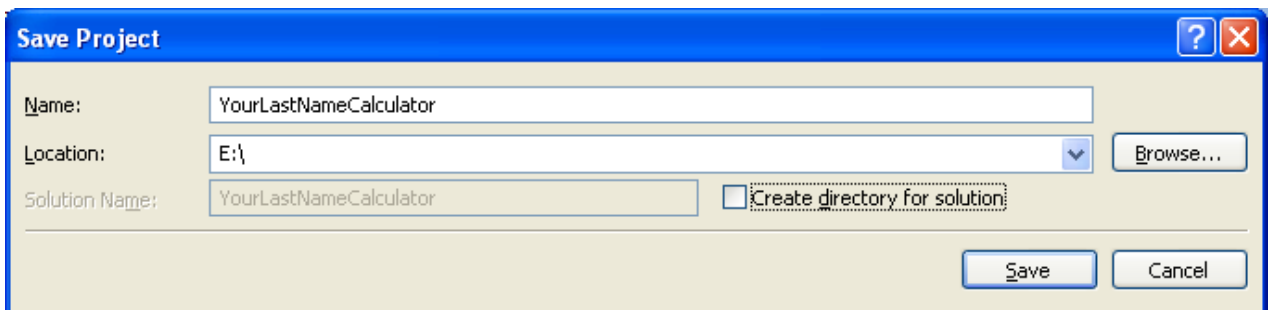
If necessary, maximize the application window to fill the entire screen by clicking the **maximize button** in the upper right-hand corner of the window. Do NOT be alarmed if your Start Page does not look like the one pictured. Start Pages will vary depending on who used it last and on the selected options.

2. To create a project, click on **File** on the menu bar, select **New Project...**



In the New Project dialog box, select or enter the following values:

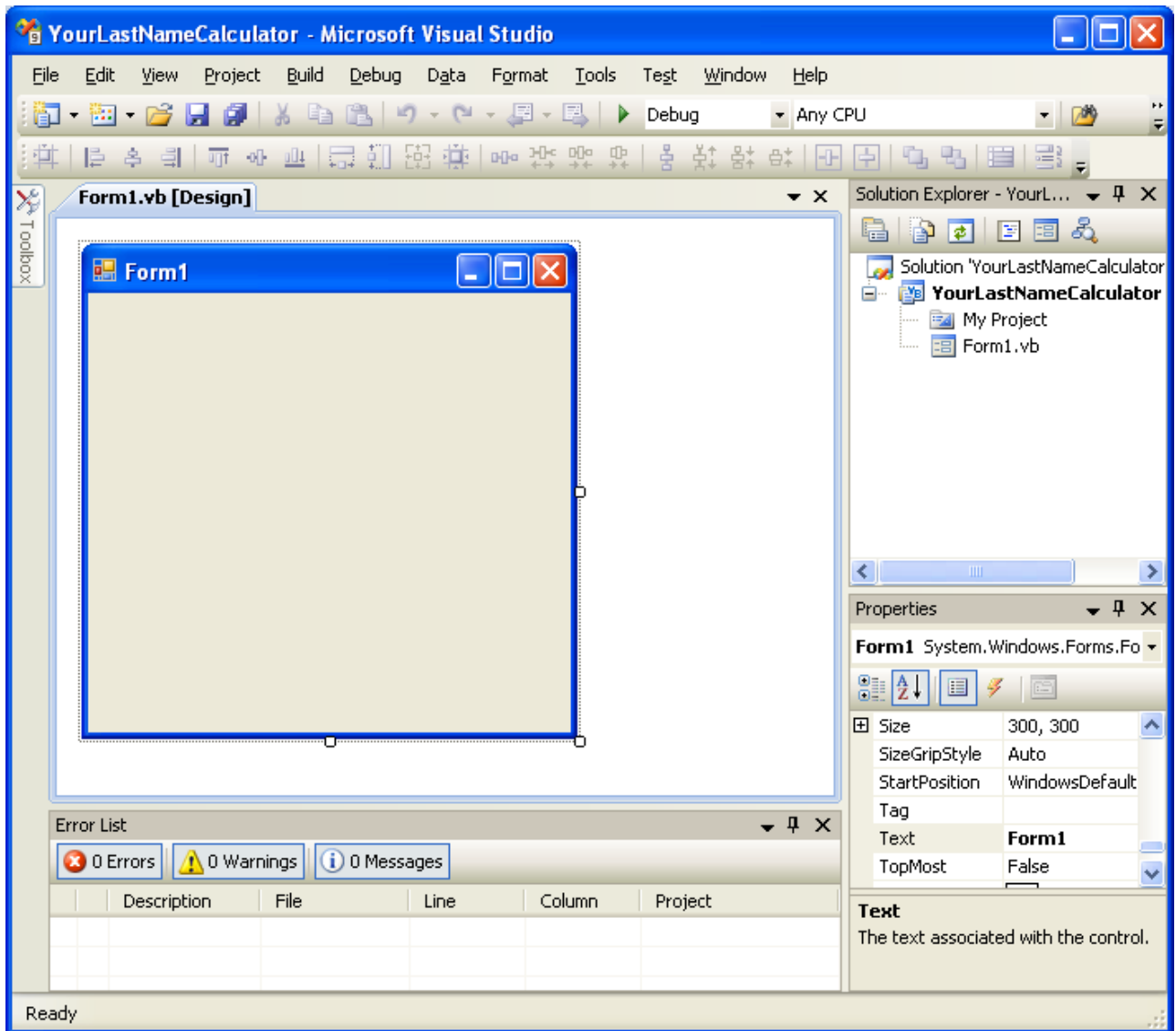
- Template: **Windows Forms Application**
- Name: **YourLastNameCalculator** (be sure to enter your actual last name)
- Click **OK**.  
After a few seconds a new project with an empty form should be displayed.
- Click on **File** on the menu bar, and select **Save All**, and the Save Project dialog box will open.






- Name: should already have YourLastNameCalculator
- Location: Click on **Browse** to find the letter assigned to your flash drive. The drive letter assigned to your flash drive will vary by computer, so it best to use the **Browse** button to navigate your flash drive.
- Remove the check mark in front of **Create directory for solution** if it has one.
- Click **OK**.
- The rest of the process is the same for the Professional and Express versions, so **images for Professional version will be used in the remainder of this document**.

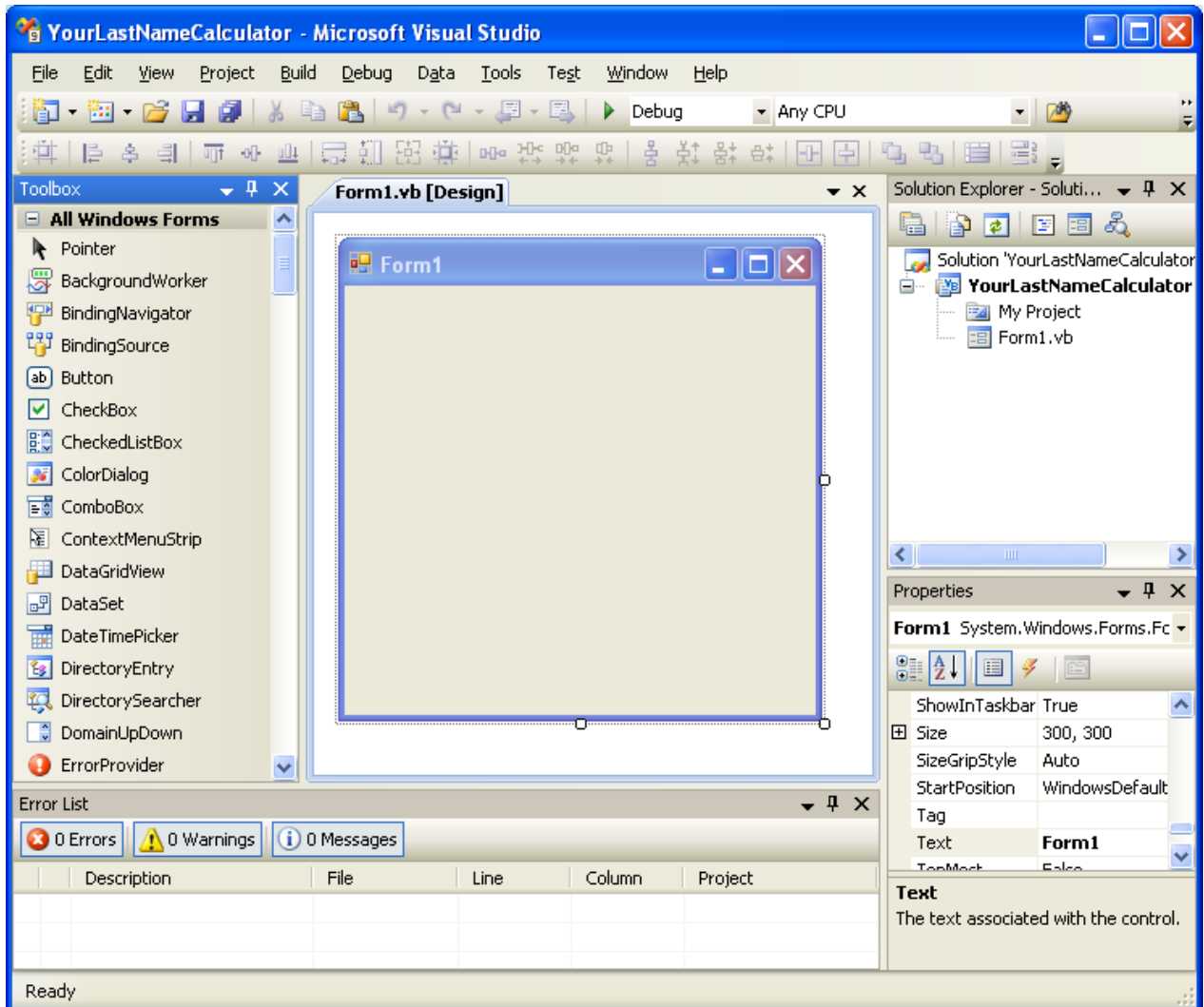
## V. Construct the interface by adding control objects and setting their properties.

1. To make sure your Visual Basic screen closely resembles the images shown in this document do the following:
  - Click **Window** on the menu bar.
  - Click on **Reset Window Layout**.
  - Click **YES**, when asked if you want to restore the default window environment.

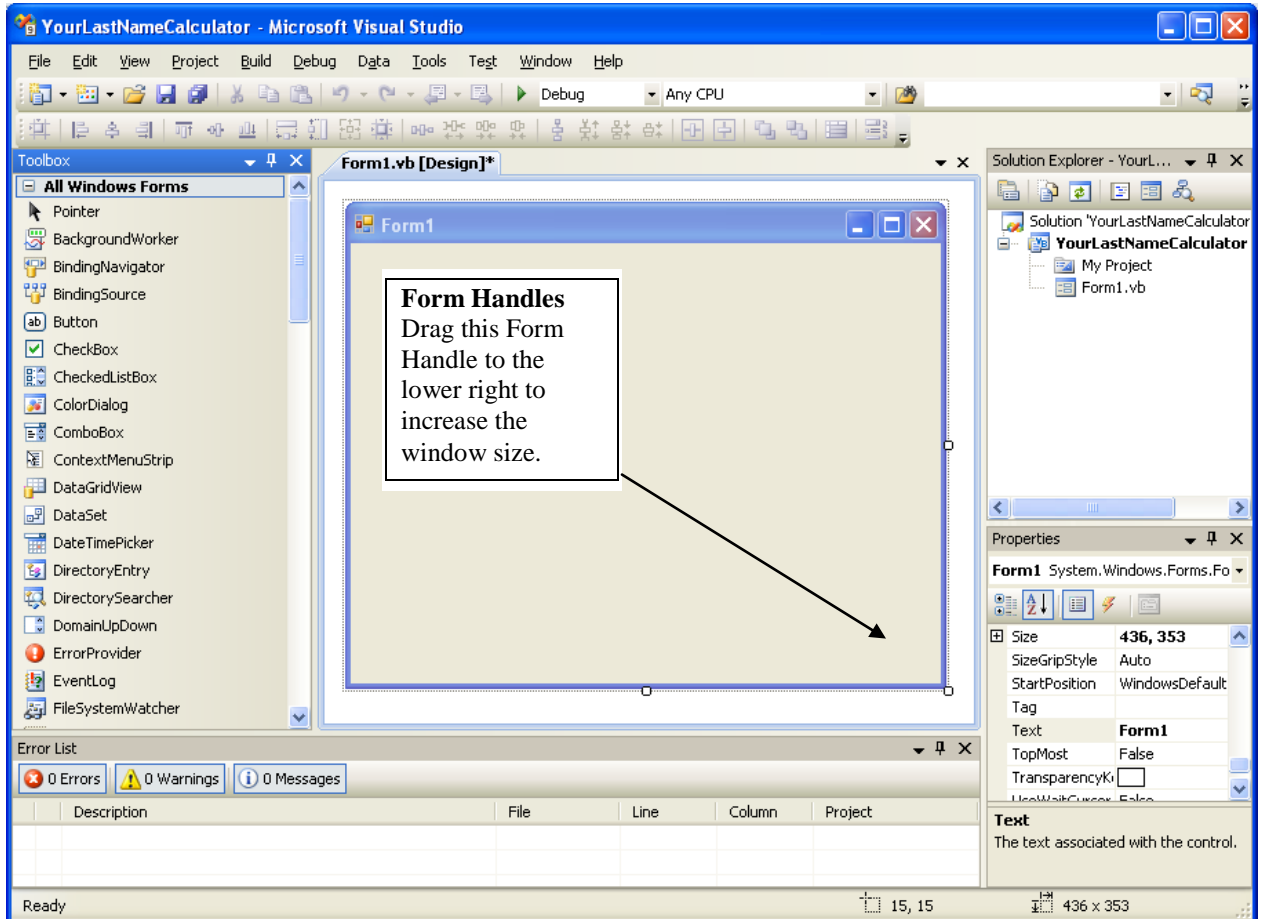


## 2. Display the **Toolbox**:

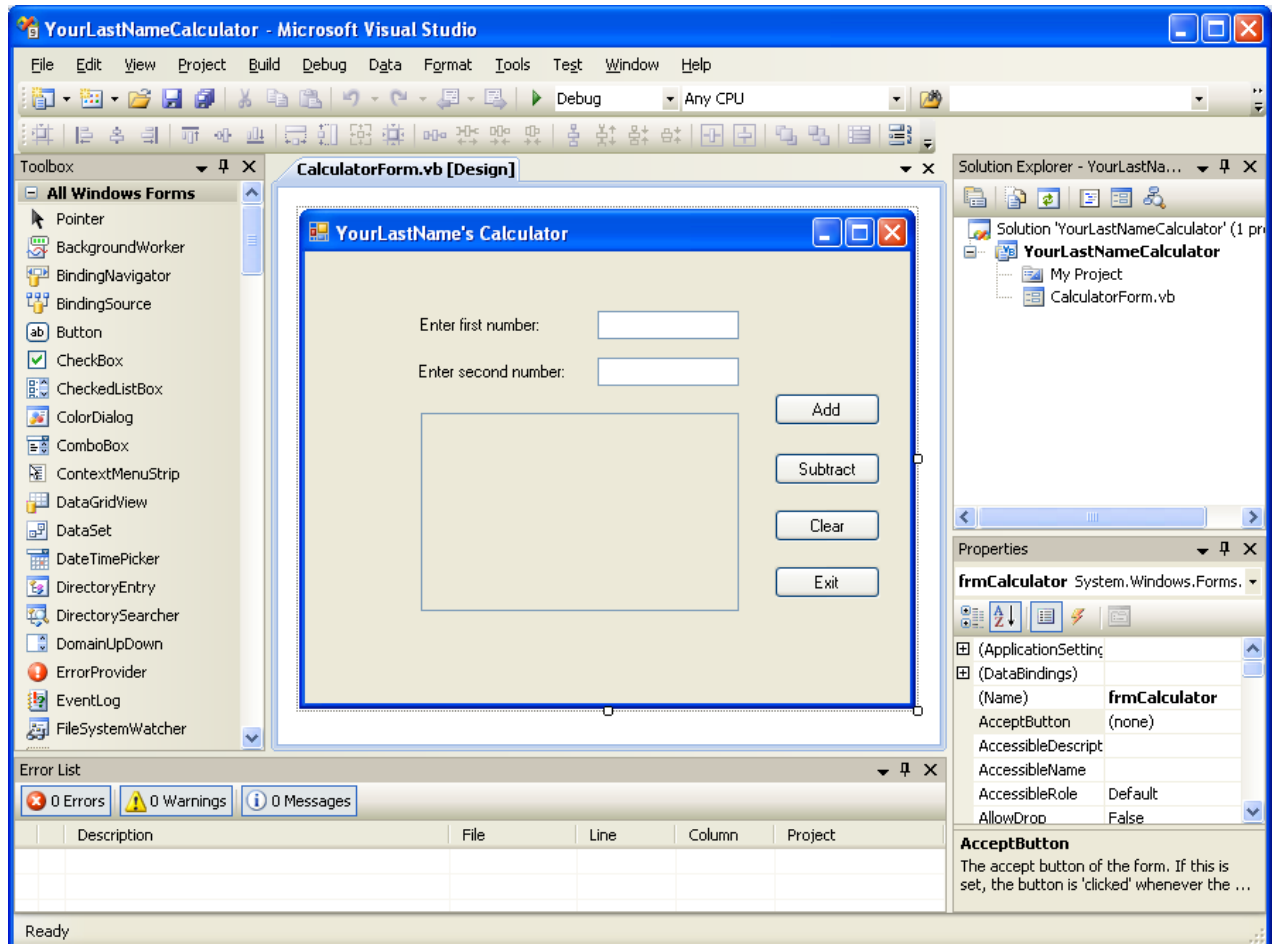
- Click **View** on the menu bar select **Toolbox**.
- It can take a few seconds for the Toolbox to expand the first time this task is performed.
- If the **Auto Hide** thumbtack icon in the **Toolbox** header faces this way , click it so it changes to  (  ) which “tacks” the toolbox open.



2. **Resize** the form to give yourself more room by using the form resizing handles.
  - **Click** on the **form** object **one time** to select it.
  - **Mouse over** the **form handle** on the **bottom right** corner, and the pointer should turn into a two way arrow. Note, the Maximize control on the title bar of the form does not work in design.
  - **Click** on the **bottom right handle** and **drag it down** to make the form a little bigger.
  - In the **Toolbox** panel click on the **All Windows Forms** tab if it is not displayed.



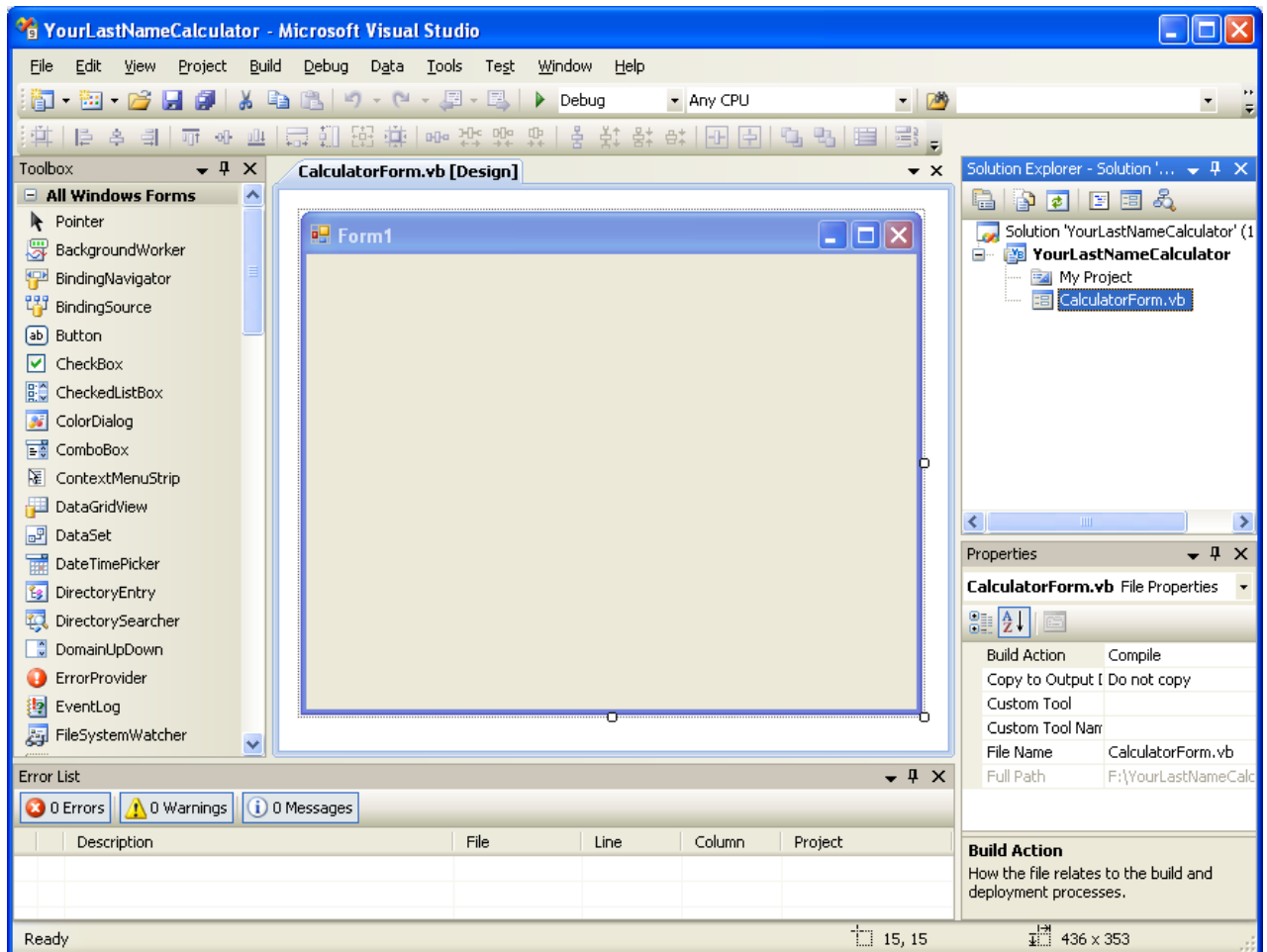
The empty form object will be used to create the GUI shown below. Detailed instructions on how to create the interface are provided on the following pages.




4. Rename the form file from Form1.vb to CalculatorForm.vb.
  - In the **Solution Explorer** panel, **click on Form1.vb** one time to select it.
  - **Right-click** on the icon of **Form1.vb** and **select Rename** from the context menu.
  - Enter **CalculatorForm.vb** and press the **Enter** key.

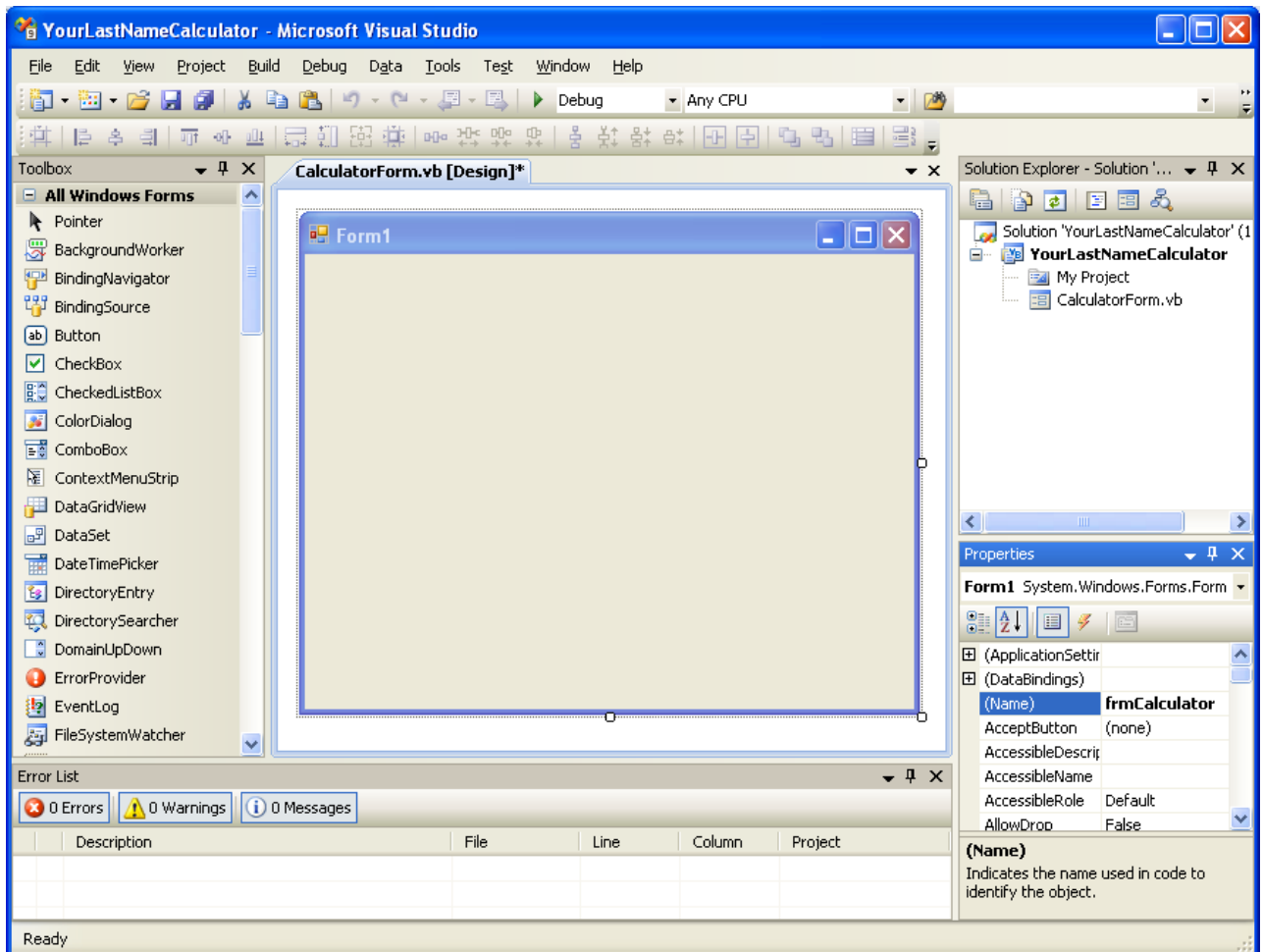
**Note: It is very important that the .vb extension be included in the name.**

  - Visual Studio may ask if you want to rename all references in the project. **Click on No.**

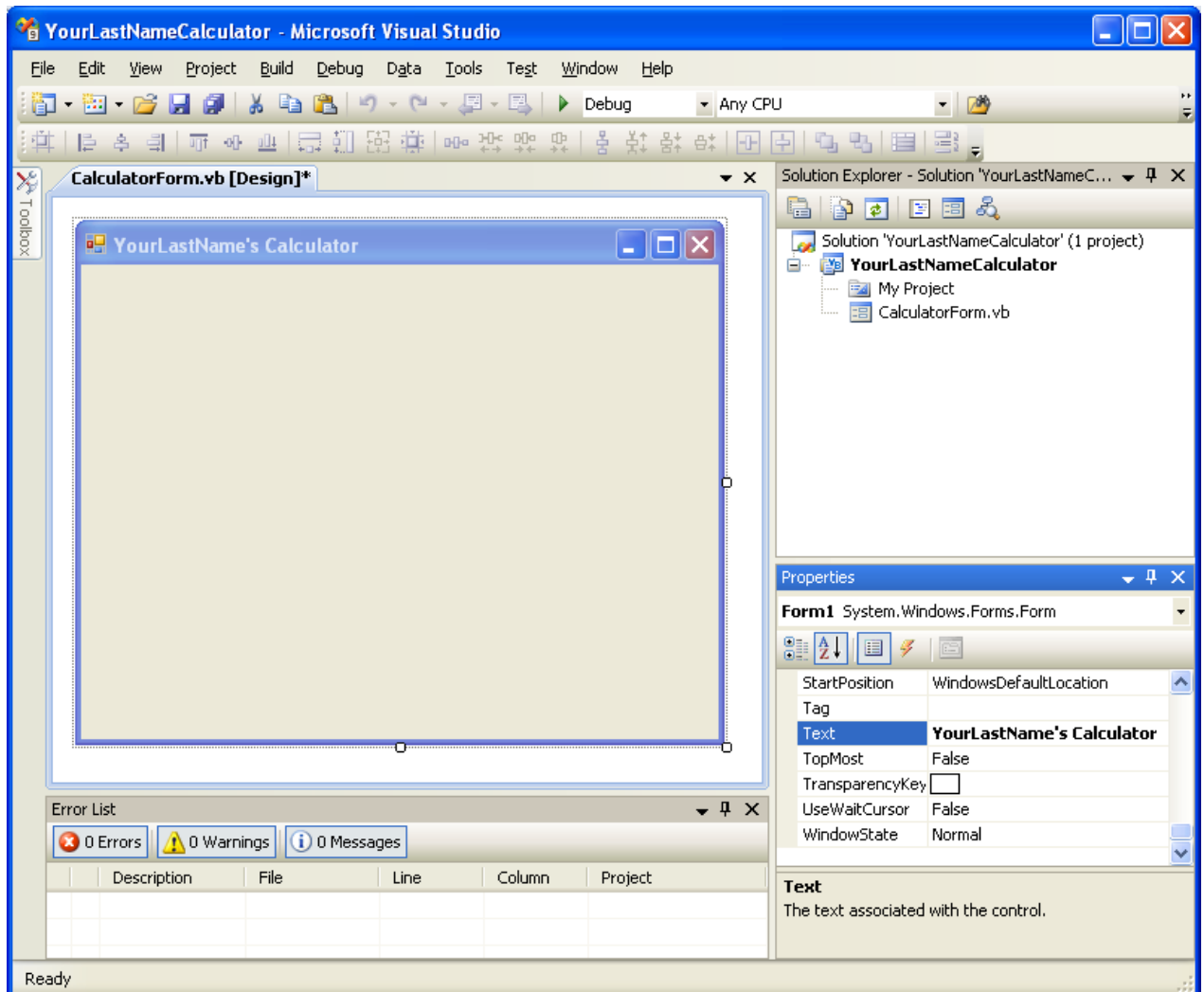



5. Rename the form object to frmCalculator.


- In the **Design** panel, click on Form1 one time to select it.
- In the **Properties** panel, click on the  icon to list the properties in ascending order.
- In the **(Name)** property, **double click on form name** to select it.  
The form name maybe Form1 or CalculatorFrom.  
Scroll to the top of the property list if the (Name) property is not visible.
- Enter **frmCalculator** and press the **Enter** key.  
Note: The form object name should not include the .vb filename extension.



6. Set the title bar property to your last name by modifying the Text property.
  - In the **Design** panel, **click on Form1** one time to make sure it is still selected.
  - In the **Properties** panel, scroll down until the **Text** property is visible.
  - In the **Text** property, **double click on Form1** to select it.
  - Enter **YourLastName's Calculator** and press the **Enter** key.  
Be sure to change YourLastName to your actual last name.
  - The text displayed on title bar of the form should now be changed.

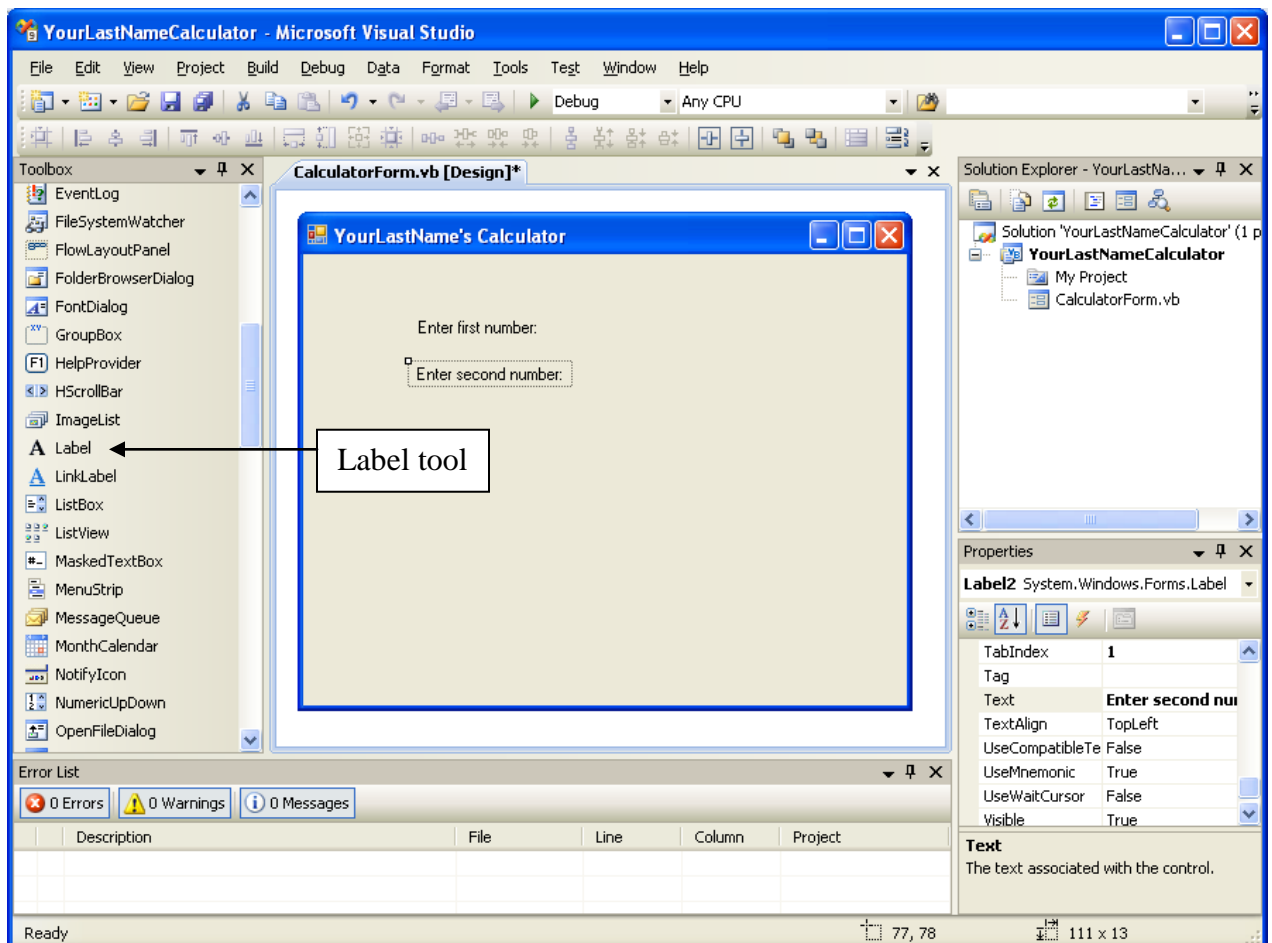


7. Save your work by clicking on the **Save All**  button on the toolbar **or** click on **File** on the menu bar, and select **Save All**.


Save All saves all files that may be opened, and Save  only saves the file that is currently being worked on. Be sure to save you work at various points throughout this project.

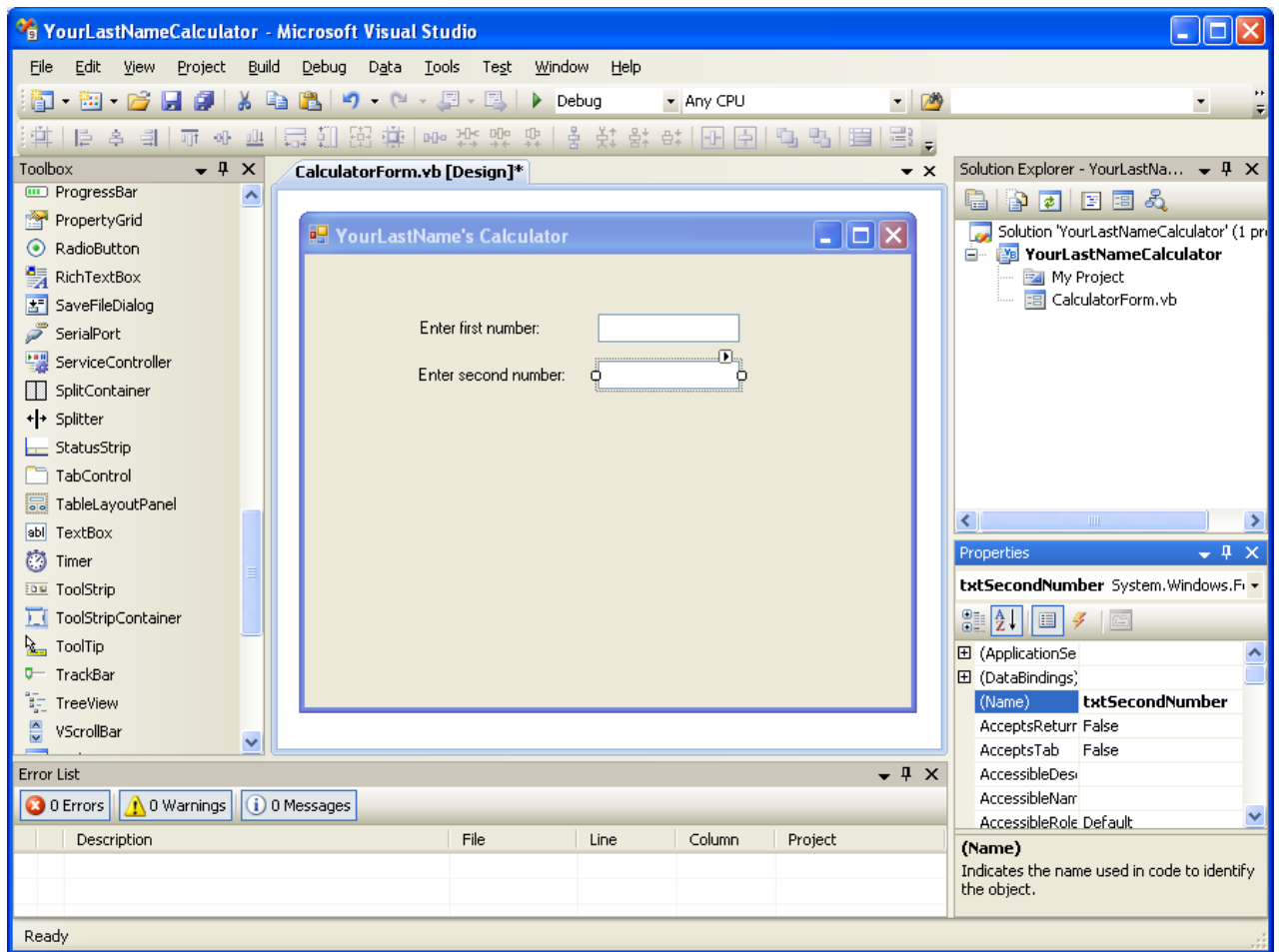
8. You are now ready to build the user interface by adding control objects from the toolbox onto the form (frmCalculator). You'll be adding two Labels, three Textboxes, and four Buttons. The order in which you add the control objects to a form determines the default tab order, which is the order the user moves from object to object when pressing the tab key. To correctly set the tab order, it is best to add the controls in the preferred order. We'll begin with the Labels, followed by the Textboxes, and end with the Buttons. Labels are used to display information to the user. In this project the labels identify what should be entered into each of the textboxes.

- In the **toolbox**, click on the **A** **Label tool one time** to select it, and the pointer should become the Label tool icon. You may need to scroll down in the toolbox to see it.
- **Move** the mouse pointer to the form.
- **Click** about 1 inch from the top of the form and 1/2 inch from the left side of the form and click once to insert the label.
- The **Properties** window now shows properties for Label1 that you just created. You do not need to change the default name of the Label (Label1) because it will not be referenced in the code.
- Scroll down to the **Text** property, and **double click on Label1** to select it.
- Type in **Enter first number:** and press the **Enter** key.  
Notice that the label's size will adjust to fit the text value that you entered.
- **Repeat** the procedure to **create the second label**, but set the **Text** property to **Enter second number:**



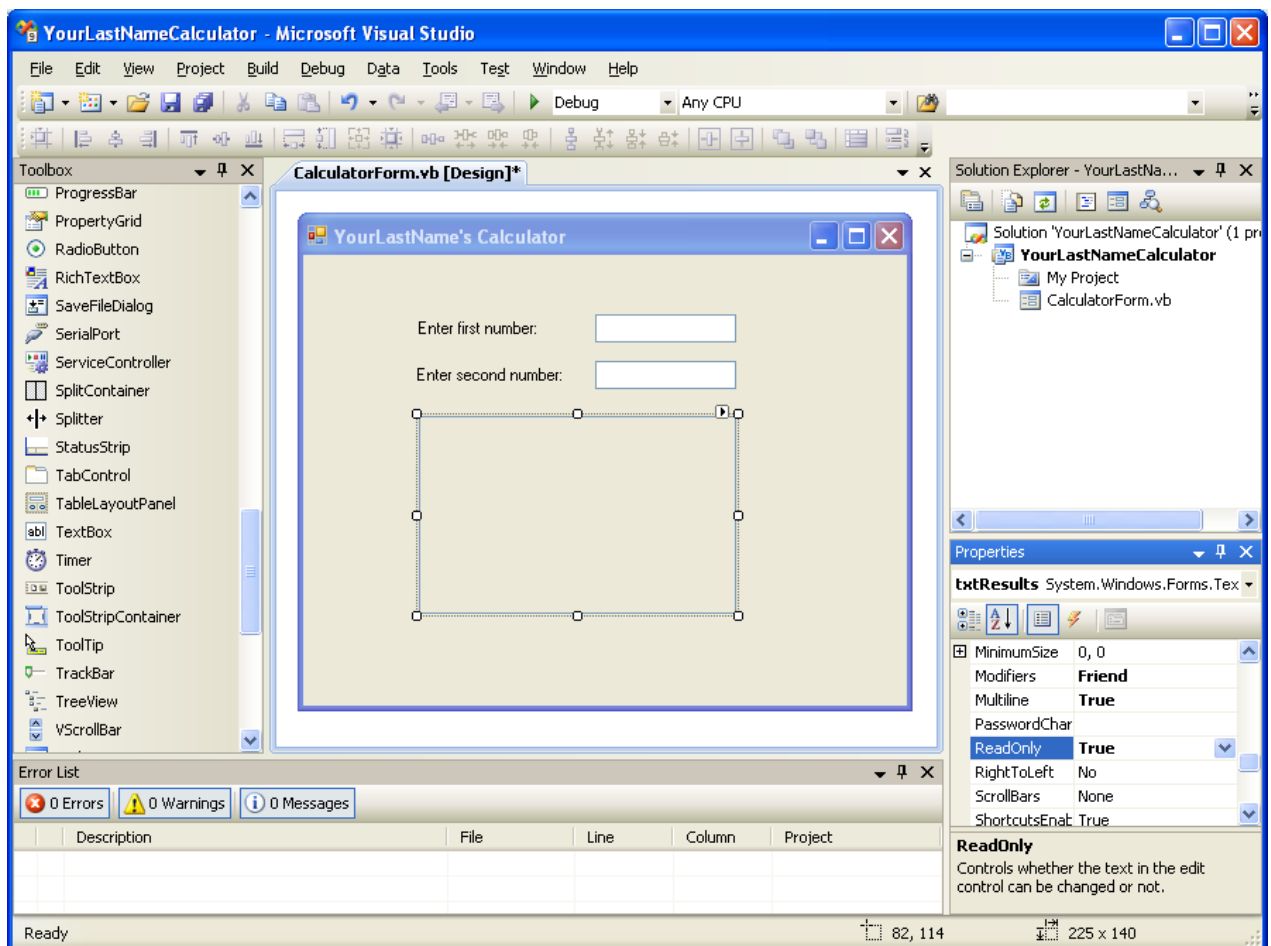
9. Create the textboxes using a similar procedure with the TextBox tool.

- **Click** the  **TextBox** tool in the Toolbox to select it.  
You may need to scroll down in the Toolbox to see the TextBox tool.
- **Move** the pointer over the form.
- **Click** just to the right of the first Label on the form.
- A textbox appears on the form. Notice that if you move the textbox by dragging it on the form, blue or pink lines may appear to assist you in aligning your controls on the form. When the textbox is selected it has resizing so the size can be adjusted to accommodate smaller or larger values.
- **Click** on the **textbox control one time** to make sure it is selected.
- In the **(Name)** property, **double click on TextBox1** to select the text.
- **Type in txtFirstNumber** to replace the default value.  
The spelling of the property name must be entered as shown so that the code entered later in the project will work as presented.
- Do not set the Text property field. The user will use the empty textbox to enter a number.
- **Repeat** the procedure to create the **second textbox**,  
but set the **(Name)** property to **txtSecondNumber**.




10 Create a TextBox for the results and set additional properties.

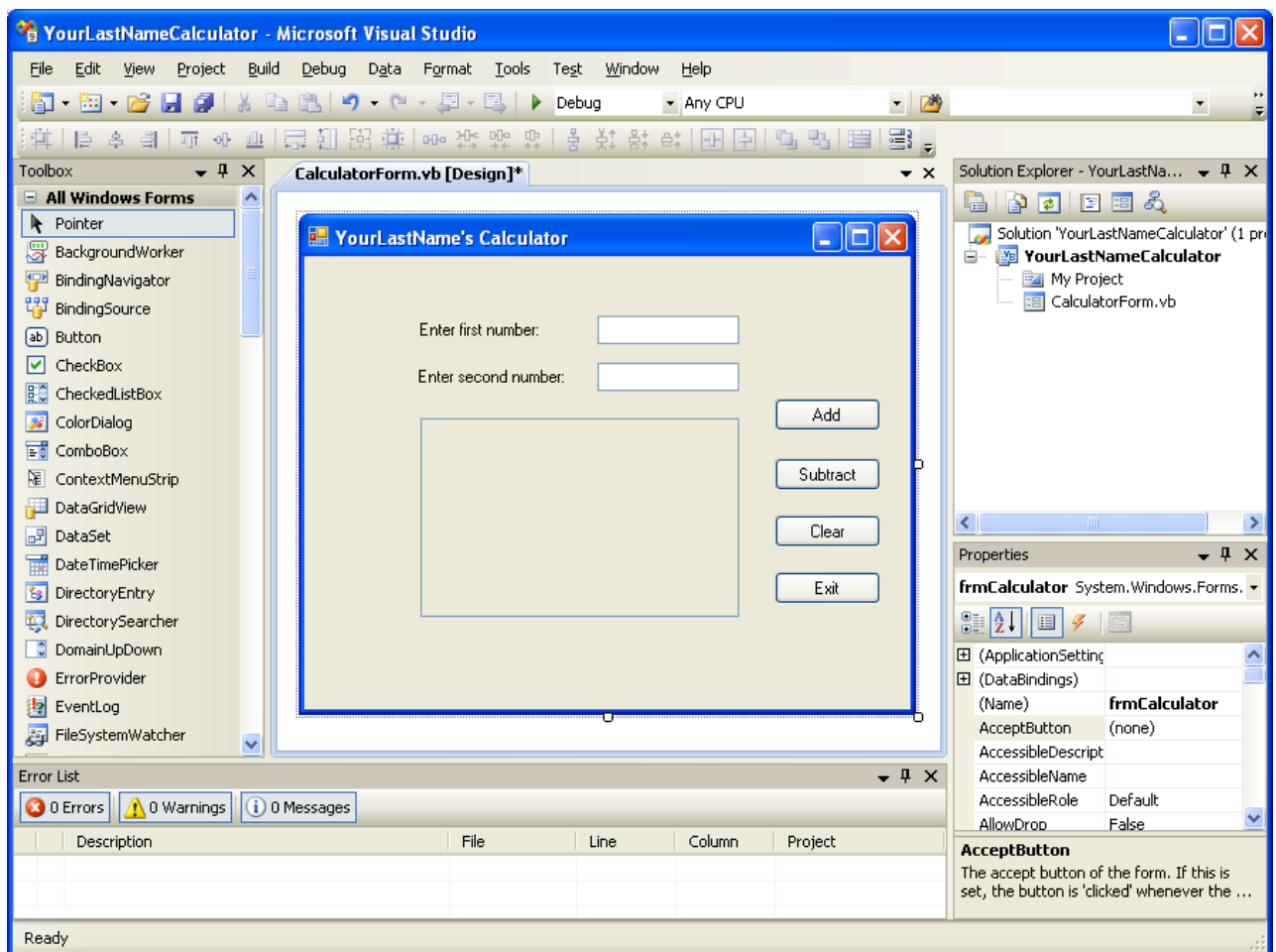
- Click the **abl** **TextBox** tool once to select it.
- Click on the form right under the second label, but aligned to the left.
- Scroll to the top of the properties list, and set the **(Name)** property to **txtResults**.
- Scroll down in the properties list and set the **MultiLine** property to **True**.  
This allows for multiple results to be displayed in the single TextBox.
- Scroll down in the properties list and set the **ReadOnly** property to **True**.  
This blocks the user from being able to type data into the textbox. The user will only be able READ the contents. The background color of the textbox will now match the form's background color so the user will know that they cannot enter text there.
- **Adjust the size** of the results textbox by selecting and dragging the sizing handle in the lower right.



11. The last controls to add are the four buttons. We'll begin with the add button.

- **Click** the  **Button tool** once to select it.  
May need to scroll to the top of Toolbox to see the Button tool.
- **Click** on the form at the desired location.
- **Click and drag the resizing handle** on the lower right to make it the desired size.
- In the Properties window, change the **(Name)** property from Button1 to **btnAdd**.
- In the **Text** property enter the word **Add** to change the caption that is displayed on the button.
- **Repeat** this process for the remaining buttons using the following settings:

<u>Object Name</u>	<u>Text Property</u>
btnSubtract	Subtract
btnClear	Clear
btnExit	Exit

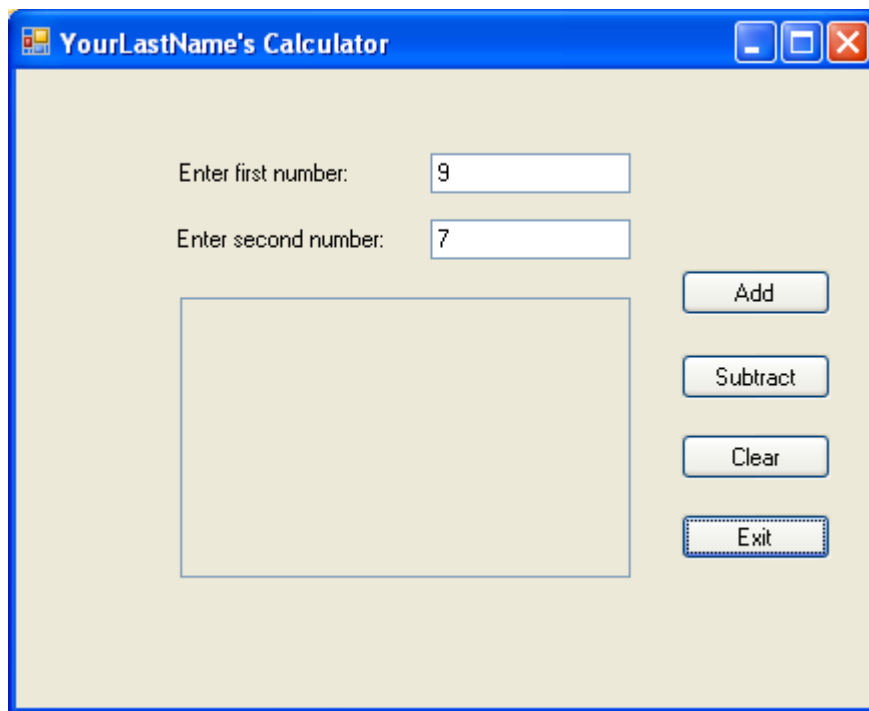


12. Check your control objects for **position** and **alignment**, using the blue and pink lines that appear when you move an object by dragging it on the form. Check again to be sure that you have all the controls the user will need to calculate the sum or difference of two numbers.

13. **Save** your work.

14. Test the user interface by compiling and executing the program that has been developed to this point.

- Click **Debug** on the menu and select **Start Debugging**.  
You can also press the F5 function key, or click on the green arrow on the toolbar.
- The program will be compiled and executed.
- You should be able to enter a number in each of the textboxes, but when you click on any of the buttons nothing will happen. That is because we haven't written the computer instructions that should be executed when a particular button is clicked. We will be entering the computer instructions in the next section. So far only the code used to generate the interface has been developed.



- To end the program **click** on the **close button** on the title bar.

15. Close the project and exit Visual Studio.

- Click on **File** on the menu bar and select **Close Solution**.
- Click on **File** on the menu bar and select **Exit**.

## Part 2: Develop Program Code

### VI. Enter the computer instructions to process the input and generate the desired output.

After creating the user interface, the sub procedures are created in the Code window. In this section we will be entering the code that has been provided.


1. Start Visual Studio and open the project if it was closed as instructed at the end of the previous section.

Visual Studio Professional:

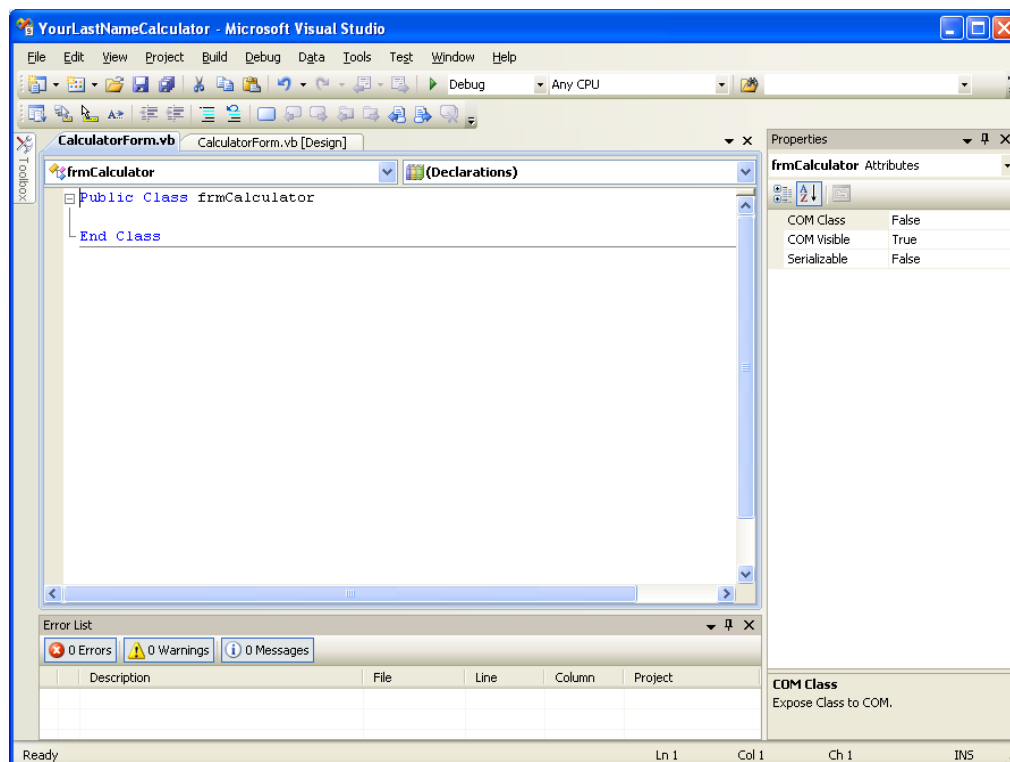
- Click on **Start → Programs → Microsoft Visual Studio 2008 → Microsoft Visual Studio 2008**
- Click on **File** on the menu bar, mouse over **Open** and from the submenu select **Project/Solution**.
- In the **Open Project dialog box**, navigate **your storage device**, and into the **YourLastNameCalculator folder**.
- **Double click** on the solution file (**YourLastNameCalculator.sln**).

Visual Basic Express Edition:

- Click on **Start → Programs → Microsoft Visual Basic 2008 Express Edition**
- Click on **File** on the menu bar, and select **Open Project**.
- In the **Open Project dialog box**, navigate **your storage device**, and into the **YourLastNameCalculator folder**.
- **Double click** on the solution file (**YourLastNameCalculator.sln**).

2. Before entering code view, **click the Auto Hide (tack) icon**  to hide the toolbox so that you have a larger code window area. You will still see the Toolbox tab in case you need to use it again.

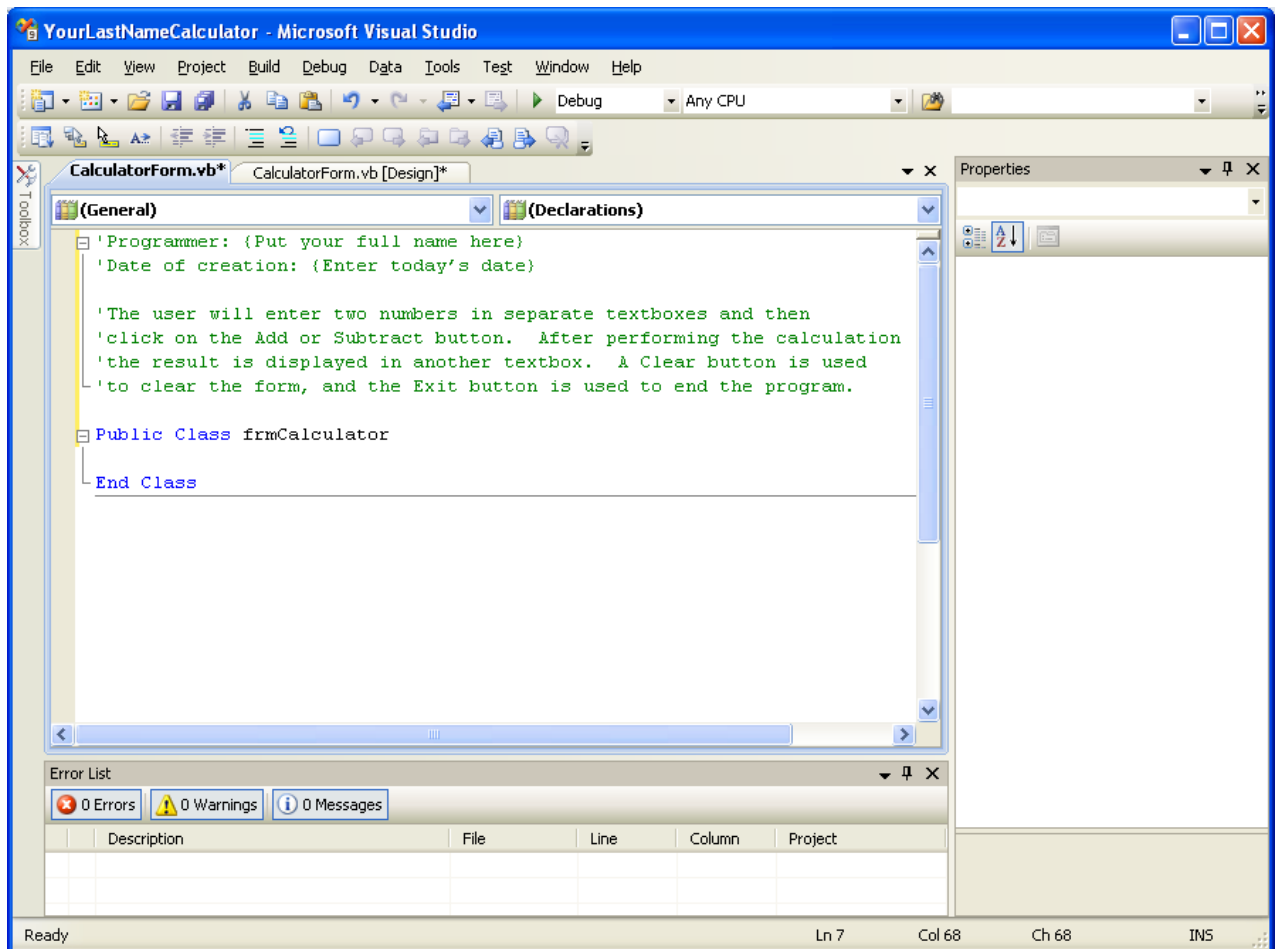
3. Click **View** on the menu bar, and select **Code** to switch to code view.



4. Programs should be documented for future reference. Documentation is inserted into the program as comments. Comments are NOT converted to executable code. To differentiate comments from lines of program code, **add an apostrophe before EVERY comment**. After you press the enter key, the line will turn green indicating that the line is a comment.

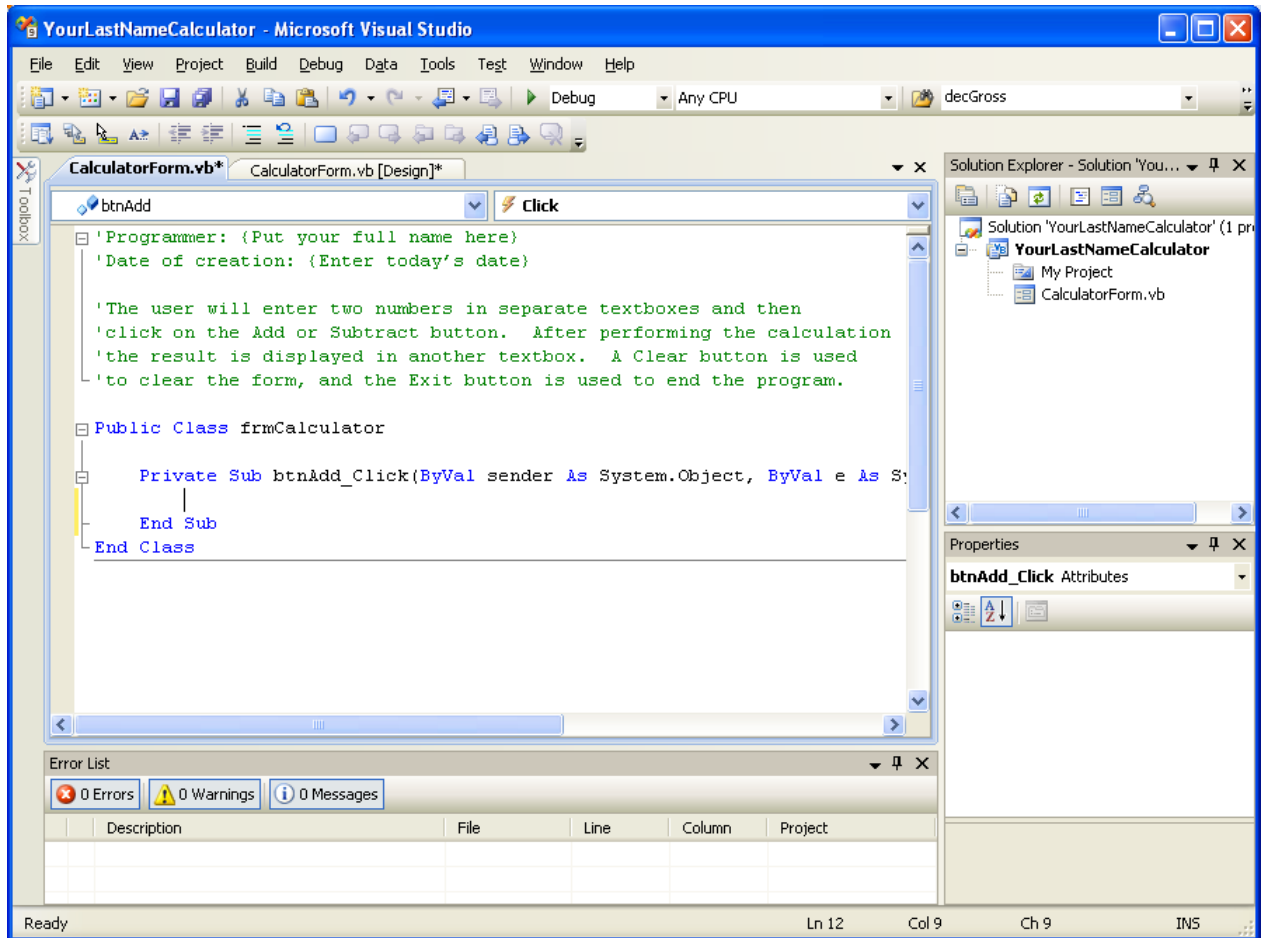
The documentation at the top of the program should include the programmer's name, date the program was created or modified, and a short description of what the program does. Comments should also be inserted throughout the code to document what each major step does.

- Place the insertion point in front of **Public Class frmCalculator**.
- Press the **Enter key twice** to insert two blank lines at the top of the code page.
- **Move the insertion pointer** back up to the first blank line.  
*Be careful NOT to change any system supplied lines in the code.*  
*The program will not work if system supplied lines are altered or deleted.*
- **Enter** your name, date, and the short description preceded with an n apostrophe as shown below.



4. To create the default shell for each sub procedure we need to switch back to Designer view and double click on each of the buttons. We will need to switch back and forth for each button.

- Click **View** on the menu and select **Designer, OR** click the **CalculatorForm.vb [Design]** tab in the editing panel.
- **Double click** on the **Add button** while in design view to create the shell for the add procedure and to automatically switch to code view.



- Create the shells for the Subtract, Clear, and Exit buttons by repeating the same process.
  - Click **View** on the menu and select **Designer**.
  - **Double click** on the **Subtract button**, and the next time on **Clear**, and finally on **Exit**.

5. Enter the program code for each procedure as shown below.

- **Place the insertion point** in the correct procedure before beginning to type.
- **Enter the code** as shown on the following pages so the application will execute as designed.
- After typing in Try as presented below and pressing Enter, the **Try-Catch block** will be created. Enter the code in the correct section of the Try-Catch block.
- The **naming convention** used for the object names will also be used for the variables. Variable names are in lowercase with each significant word capitalized to improve code readability. In addition, each name begins with a three letter prefix to identify the data type (i.e. `decAddResult`, `strAddResult`). **Spaces are invalid in variable names so make sure you don't enter any spaces when declaring and using a variable.**

```
'Programmer: {Put your full name here}
'Date of creation: {Enter today's date}
```

```
'The user will enter two numbers in separate textboxes and then
'click on the Add or Subtract button. After performing the calculation
'the result is displayed in another textbox. A Clear button is used
'to clear the form, and the Exit button is used to end the program.
```

```
Public Class frmCalculator
```

```
Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAdd.Click
    'This procedure takes two number values from textboxes and adds them together.
```

```
    'Declare variables with data types
```

```
    Dim decAddResult As Decimal
```

```
    Dim strAddResult As String
```

```
    'Use Try-Catch block to capture invalid values entered by users
```

```
    Try
```

```
        'Convert string values from the textboxes and add the two numbers
```

```
        decAddResult = Decimal.Parse(txtFirstNumber.Text) + Decimal.Parse(txtSecondNumber.Text)
```

```
        'Build string for results using string concatenation (include a space before underscore)
```

```
        strAddResult = "Add: " & txtFirstNumber.Text & " + " & _
            txtSecondNumber.Text & " = " & _
            decAddResult & ControlChars.CrLf
```

```
        'Display the results
```

```
        txtResults.AppendText(strAddResult)
```

```
        'Clear the textboxes
```

```
        txtFirstNumber.Clear()
```

```
        txtSecondNumber.Clear()
```

```
        'Place the insertion point in the first textbox
```

```
        txtFirstNumber.Focus()
```

```
    Catch ex As Exception
```

```
        MessageBox.Show("You must enter two numbers", _
            "Correct Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
```

```
    End Try
```

```
End Sub
```

---

```

Private Sub btnSubtract_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSubtract.Click
    'This procedure takes two number values from textboxes and subtracts the second from the first.

    'Declare variables
    Dim decSubtractResult As Decimal
    Dim strSubtractResult As String

    'Use Try-Catch block to capture invalid values entered by users
    Try
        'Convert string values from two textboxes and subtract the second from the first
        decSubtractResult = Decimal.Parse(txtFirstNumber.Text) - Decimal.Parse(txtSecondNumber.Text)

        'Build string for results using string concatenation (insert a space before underscores)
        strSubtractResult = "Subtract: " & txtFirstNumber.Text & " - " & _
            txtSecondNumber.Text & " = " & _
            decSubtractResult & ControlChars.CrLf

        'Display the result
        txtResults.AppendText(strSubtractResult)

        'Clear the textboxes
        txtFirstNumber.Clear()
        txtSecondNumber.Clear()

        'Place the insertion point in the first textbox
        txtFirstNumber.Focus()

    Catch ex As Exception
        MessageBox.Show("You must enter two numbers", _
            "Correct Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

    End Try
End Sub

```

---

```

Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnClear.Click
    'Clear all results in the txtResults
    txtResults.Clear()

    'Place the insertion point in the first textbox
    txtFirstNumber.Focus()
End Sub

```

---

```

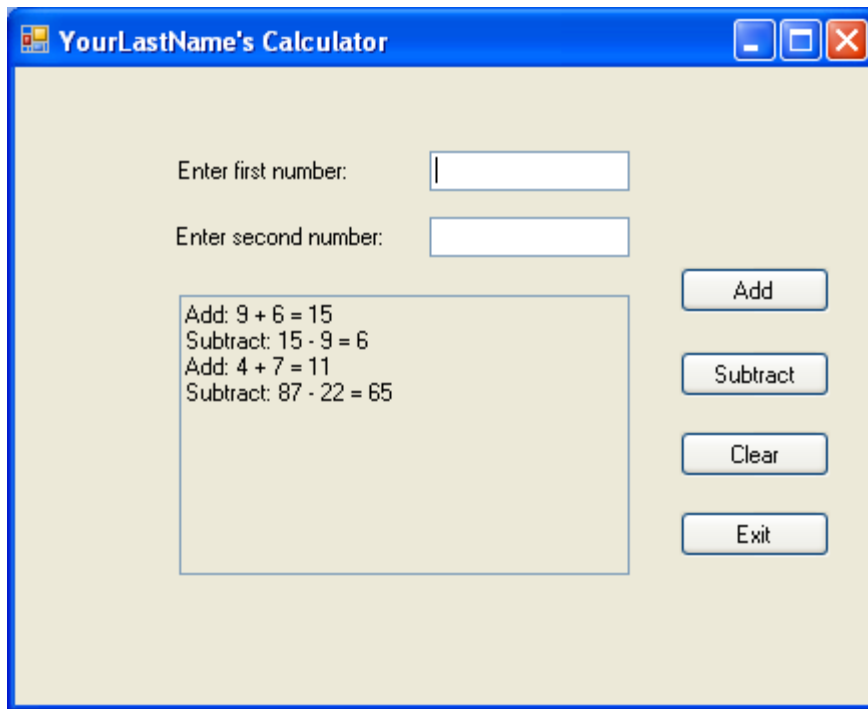
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click
    'End the program by closing the form
    Close()
End Sub
End Class

```

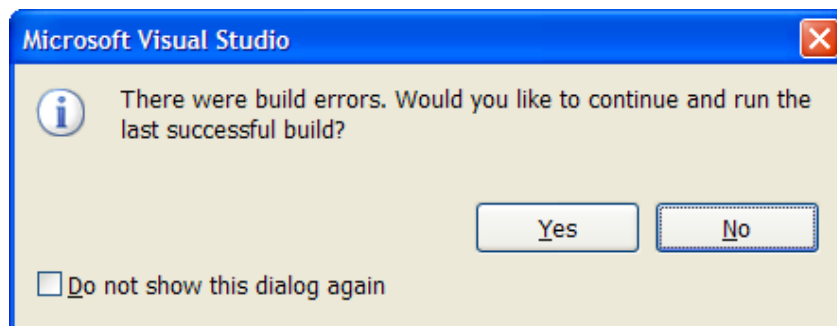
## VII. Test and debug the program.

After code is entered or modified, it must be tested. Execute the program to check if the program produces the desired results. Usually test data would be developed and there would be a set of data to ensure that each line code is tested. For example, we would want to make sure that the error message in Catch section displays correctly and is displayed when invalid (alphabetic or blank) data is entered.

- Click on **D**ebug on the menu bar, and select **S**tart **D**ebugging to run the project.
- If the project compiles and executes successfully, the program will should be displayed.
  - To test the program, the programmer pretends to be a user.
  - **Enter two** different numbers and **click on Add**.
  - **Enter two** different numbers and **click on Subtract**.
  - **Click on Clear** to clear the results textbox.
  - **Click on Exit** to end the program.



- If the compiler finds errors you will be asked if you would like to continue.
  - **Click on No**.



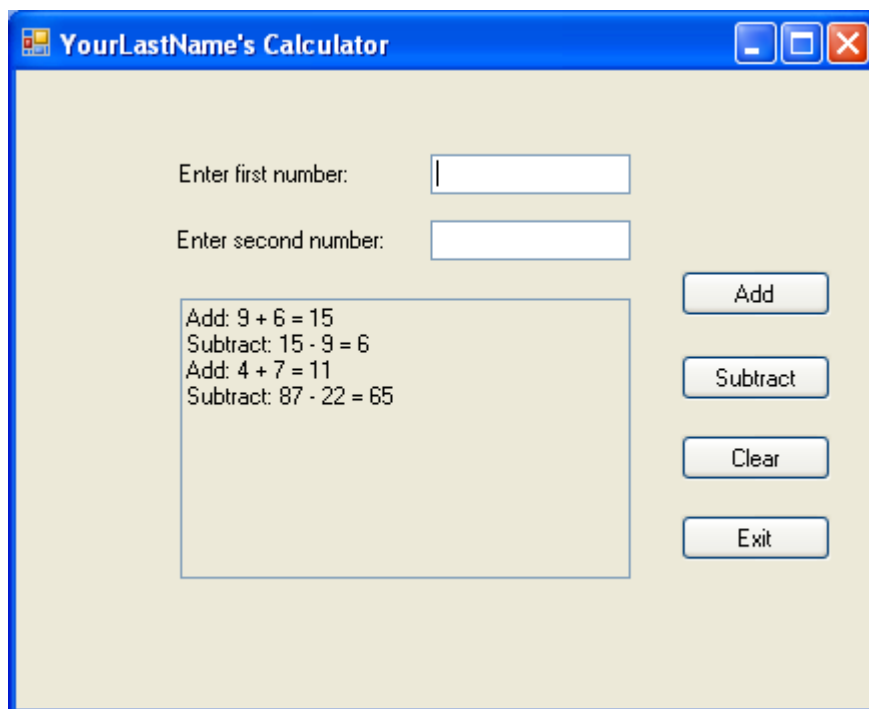
- The error(s) will appear in the **Error List** window at the bottom of the screen.

**Double-click** on the first error message in the **Error List** and focus will be set in the code on the exact line where the error occurred. You probably will see a wavy blue line under words that are incorrect, or a line of code highlighted yellow. Mouse over the incorrect code to see the screen tip describing the error. Correct the errors. Most errors will be typographical errors, misspellings, or unnamed objects or undeclared variables. You may need to switch back to Design view to correct the names of the control objects such as txtFirstNumber.

- If you are NOT able to make changes to the code, it may be because the program is still executing. To stop execution click on **Debug** on the menu bar and select **Stop Debugging**.
- After correcting the errors, try executing the program again.

### VIII. Execute program and capture sample output for assignment submission.

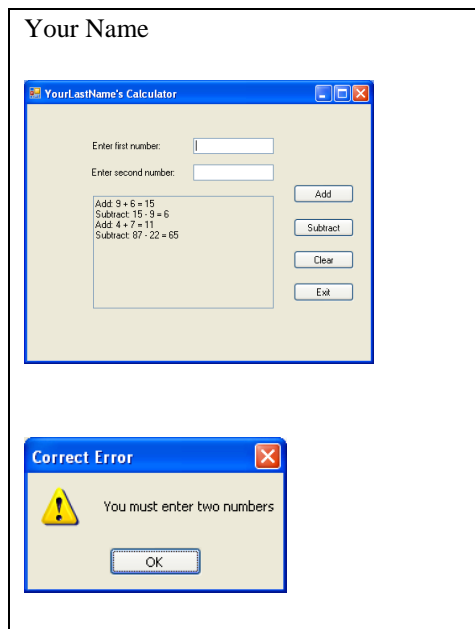
- 1) Click on **Debug** on the menu bar, and select **Start Debugging** to run the project.
- 2) **Enter numbers** in both textboxes and **click Add and Subtract** for several sets of numbers. **Do NOT** clear the results until after have captured the output as described in the next few steps.
- 3) Capture a picture of the output on the active form by pressing (**Alt+Prt Scn**). The **Alt** key is located on the bottom left side of the keyboard, and the **Print Screen** key is on top to the right of the **F12** key. This places a copy of screen in the system's Clipboard so that it can be pasted into another application.



- 4) Open a new **Word** document. In your opened Word document, **enter your name** at the top of the document, **press Enter two times**, **paste** the sample output into the document, and **press Enter two times**. **Return to the Calculator Program** window by clicking on the Calculator application on the task bar.
- 5) Generate the error message by entering invalid data. Enter **25** for the first number and **leave the second textbox empty**. **Click the Add button** and it should generate the error message stating that “You must enter two numbers”. After the error message is displayed, **DO NOT** click OK, and continue with the next step.



- 6) Capture a picture of the Message Box by pressing (**Alt+Prt Scn**). The message box is captured into the system's Clipboard, so you may **close** the Message Box by **clicking the OK** button.
- 7) Return to your **Word** document and **paste** the captured error message below the picture of the sample output. Sample Word Document:



- 8) **Save** the Word document in the solution folder on your flash drive as **YourLastNameOutput.docx**.
- 9) **Close** Word after saving the document.

## IX. Close the project and exit Visual Studio:

- Save All work
- Exit **Visual Studio** by selecting **Exit** from the **File** menu.  
Click **Yes** if you are prompted to save changes to files.

## X. Submit assignment:

### ZIP Option:

Your instructor may require you to ZIP your project before submitting it, however zipping software must be installed on your computer in order to do this. If you have the software, try the following:

1. Using **Windows Explore**, navigate to **your folder** (YourLastNameCalculator) with the project.
2. **Click** on the folder one time to select it and **verify** that the following files are included.

<b>bin</b>	<b>folder</b>
<b>My Project</b>	<b>folder</b>
<b>obj</b>	<b>folder</b>
CalculatorForm.Designer.vb	
CalculatorForm.resx	
CalculatorForm.vb	
YourLastNameCalculator.sln	
YourLastNameCalculator.suo	
YourLastNameCalculator.vbproj	
YourLastNameOutput.docx	

3. **Right click** on YourLastNameCalculator folder and select **Add to YourLastNameCalculator.zip**.  
This option will vary depending on the software installed on your computer.
4. Submit **YourLastNameCalculator.zip**
5. You have completed the programming module.

### No ZIP Option:

1. If you are not able to zip the folder due to not having the proper software installed, just submit the following files.

CalculatorForm.Designer.vb
CalculatorForm.vb
YourLastNameOutput.docx

2. You have completed the programming module.

**Reference: Programming terms and concepts used in the Calculator program.**

<b>form</b>	Term used for the object that will be defined as a program's graphical user interface (GUI). Also known as Window. A GUI provides an intuitive interface for the program user to control the actions that can be taken in the program code.
<b>object</b>	In this project, an occurrence of a type of control from the Toolbox that has its own methods (actions it can do) and properties (adjectives that describe it).
<b>control</b>	Used interchangeably with the term "object" to refer to an occurrence of an object. The term "control" is used to imply that the object is used on the Form to allow user control of the actions taken in the program code.
<b>code</b>	Instructions written in the Visual Basic syntax that carry out the intended actions of the program.
<b>method</b>	One or more predefined actions associated with an object. A method's name may be used in the program code to invoke an action for an object. An example of a method name is <b>Close</b> , which is the action of closing (or ending) an occurrence of a form.
<b>property</b>	One or more characteristics (or attributes) of an object. The property value is set by the programmer to define the object. An example of a property is <b>.Text</b> , which contains the characters that the object will display on the Form. <code>Label1.Text = "Hello."</code>
<b>procedure</b>	Visual Basic is a modular language that separates actions into their own code units, called procedures. Procedures usually define one specific action or calculation to be taken by the program. In this project, the procedures are "event procedures" that are invoked by some event that happens in the form or its objects.
<b>event</b>	An event involves an object and some action that (the user) takes using that object. An example would be mouse-clicking a Button, where the Button is the object and the click is the event. ( <code>btnAdd.Click</code> )
<b>IDE</b>	Integrated Development Environment. The Visual Basic user interface is divided into separate windows within the main window that contain file and folder names (Solution Explorer), properties for each selected object (Properties), form design and code windows for editing (Document Window with tabs for each document or form that is open), and predefined tools (objects) that can be added to the forms' design (Toolbox). If any of these windows are missing from the IDE, they can be reopened using <b>View</b> on the menu bar, or the <b>Window</b> menu item, and select <b>Reset Window Layout</b> .
<b>debug</b>	When the program is started in the IDE, the process is known as "debugging" the program. The code is interpreted line by line to find the precise line of code that is causing an error. The code can be stopped and changes made to correct the error. Debug is restarted until all actions of the program can be tested with test data to eliminate errors in language syntax and/or program logic. It is good practice to Save All before running the program. (Instructions on page 13)
<b>documentation</b>	Good programming requires that everything regarding the design, creation, and implementation of a computer program be written in a clear and organized manner so that programs may be modified or updated in the future by any programmer. Documentation includes the planning sketches and flowcharts, pseudocode, final program code, as well as internal documentation that is included within the program code to identify the purpose and intended results of each section of code. Documentation within the code is written in "comment lines."
<b>comment lines</b>	All lines of text within program code that are not in Visual Basic syntax and not intended to be interpreted as program code are set off from the actual code by beginning the text with an apostrophe ' . Known as "comments," the text color after the apostrophe changes to green in the code page so that it is easily recognized as a comment and not part of the code. Type an apostrophe and then the text comment.

<b>system supplied lines</b>	Some code is automatically added to the code document as you create the form. This code is known as system-supplied or system-generated code. In this project, none of the system-supplied code lines should be modified or deleted.
<b>Try-Catch Block</b>	Try-Catch is used to enclose a block of code that could potentially cause an exception (or error) in the program. The block is started with the keyword TRY. After the block of code, if any problems were found with the user input, like missing values, division by zero, or invalid data types (text vs. number), then the block of code is skipped and program control goes directly to the statement "Catch ex as Exception". The programmer defines the action to be taken if an error is found. In this project, a message box is constructed in code to inform the user that a correction must be made. The program will not continue until the user responds to the message box.
<b>Concatenation (symbol &amp;)</b>	Used to join two strings together to form a new string. <b>"good" &amp; "bye"</b> creating a new string <b>"goodbye"</b>
<b>Declare (Dim) a variable</b>	Variables are words that the programmer selects to describe the value that is to be held in an area of memory (RAM). The naming convention of variables follows the same naming convention as controls. The first word is all lowercase, and all additional words begin with an uppercase letter. The last word of the variable name is the data type for that variable. <code>Dim decAddResult As Decimal</code> declares a variable named decAddResult where addResult says what is held in memory and Decimal is the data type of the variable meaning it is a number with decimal values.
<b>Decimal.Parse</b>	This method is used in this project to convert the String type data value of the TextBox.Text property to a number data type (Decimal) that can then be used in a calculation. Only number data types like decimal and integer may be used in calculations.
<b>Continue a statement with an underscore</b>	<b>(symbol _)</b> When a complete line of code (or a statement) is too long to fit the viewable screen, the line can be continued on the next line of the code document by typing a [spacebar] and an underscore character ( _) at the end of the broken line.  <pre>strAddResult = "Add: " &amp; txtFirstNumber.Text &amp; " + " &amp; _                 txtSecondNumber.Text &amp; " = " &amp; _                 decAddResult &amp; ControlChars.CrLf</pre> <p>Otherwise, without the space and underscore character, VB will think this is meant to be two separate lines of code, and will most likely cause a syntax error.</p>
<b>Blue curvy lines under code</b>	When the text editor in VB puts blue curvy lines under words, it is an indication that something is wrong with the spelling, use, or existence of the keyword or object. Hover the mouse pointer over the underlined words, and a screen tip will appear to tell you the problem with the words.
<b>.Focus() .Clear()</b>	Methods of a textbox. Focus places the insertion point in the textbox indicating that the program's attention is in that object. Clear sets the .Text property of the textbox to "null string", or no text. An example is txtFirstNumber.Focus() Also txtFirstNumber.Clear()

**Reference: Flowchart symbols used in this document.**



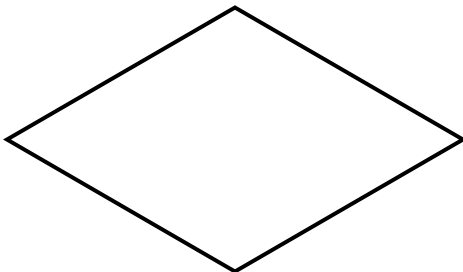
**Terminal** – Symbol used to show the Start or End of the program.



**Input/Output** – Symbol used to show the input/output operations of such things as getting, displaying or printing data.



**Processing** – Symbol used to show calculations and/or data-manipulation operations.



**Decision** – Symbol used to show logic or comparison operations. Decision symbol shows one entry and two exit paths. Answer to a “yes” or “no” question determines the path taken.



**Connector** – Symbol used to join or connect different flow lines.

Mesa Community College – Business and Information Systems – Computer Information System (CIS)  
2003 – Developed by Angeline Surber and Linda Cronquist with input from other CIS faculty members.  
2004 – Revised by John Corker, Linda Cronquist, Juan Marquez, and Angeline Surber.  
2006 – Revised by Mary Fee  
2008 – Revised by Juan Marquez