

Lab 6 – String Variables & Loops in VB

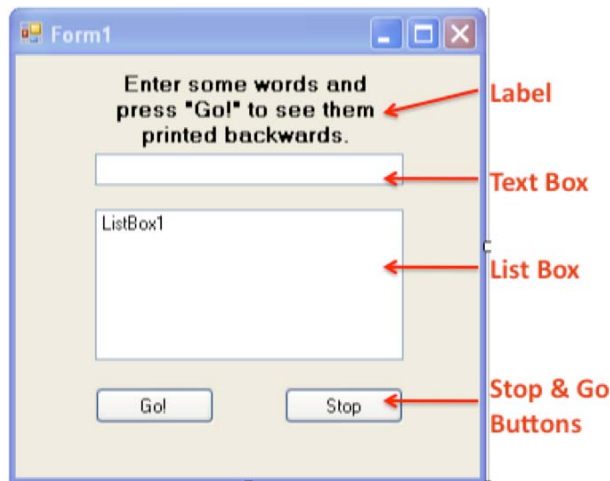
Access Visual Basic

1. Open VB from the “start” menu, that is:
Start\All Programs\Microsoft Visual Basic 2008 Express Edition
2. Select “**File → New Project**” to create a new project
3. Select “**Windows Form Application**”. You can give your “application” a name, or simply accept the default offered “**WindowsApplication1**”. Click “**OK**”.

Prepare the Graphic User Interface

As you have seen in previous labs and examples, you can use a VB "textbox" for users to enter data to your programs. Develop the "form" shown in the figure below that we will use in the project. The form should contain a label asking the user to enter some words, a text-box (named "TextBox1") for entering the words, a list box (named "ListBox1") for displaying the results of running the program and a button named "Button1" (but with caption changed to "Go!") to initiate the processing of the users entered words, and a button named "Button2" (but with caption changed to "Stop") to terminate the program.

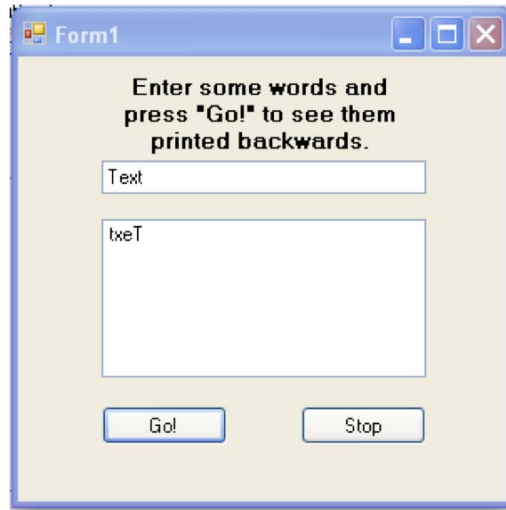
From the toolbar on the left-hand side of the VB interface, you can double click on the label control and it will place a label named "Label1" on the form. Similarly, if you double_click on the "textbox" control you will get "TextBox1" on the form. Repeat for the list box "ListBox1" and the two button controls "Button1" and "Button2". Your form should look something like this.



Add VB Code to the Project

We can capture the words that the user types into the text box by using the `TextBox1.TEXT` property, as shown in the following VB code which is associated with the "Go!" button. To add the code to the "Go!" button, double click on the button (which opens up the "code" window), then type the necessary code.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim x As String, y As Integer
    x = TextBox1.Text
    y = Len(x)
    ListBox1.Items.Add(Mid(x, y, 1) & _
                      Mid(x, y - 1, 1) & _
                      Mid(x, y - 2, 1) & _
                      Mid(x, y - 3, 1))
End Sub
```



In this code, two variables are declared: `x` is a string and `y` is an integer. The variable `x` is used to store the user-entered text from the "TextBox1" text box through the command

```
x = TextBox1.Text
```

Next, the length of the string `x` is computed using the function "`Len()`".

```
y = Len(x)
```

Now, if we want to print out the last character of the string stored in `x`, we can use the command

```
ListBox1.Items.Add( Mid( x, y, 1 ) & _
                   ... )
```

The "`Mid(string, j, k)`" function picks out the `k` characters of the string variable "`string`" starting at location `j`. Note that we are concatenating the characters and using line continuation.

That works well as long as you know exactly the length of the text that the user is going to enter. However, what if you don't? A loop would help!

Loops in VB

One of the most significant advantages of using computers is to do a great many repetitive operations in a fraction of the time it would take a human to do the same thing by hand. Loops enable us to perform those repetitive operations in the programming language.

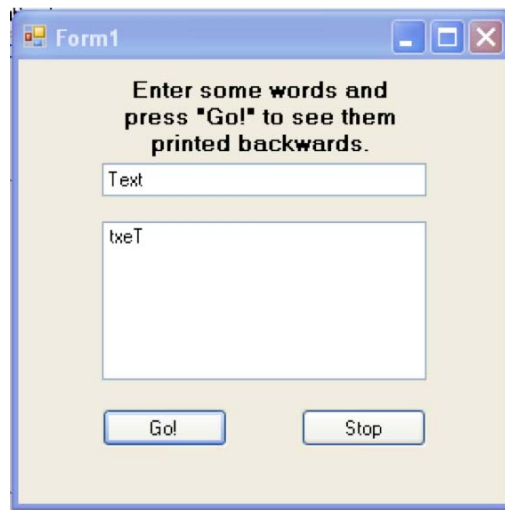
In many cases, it is necessary to exit from a loop when or if a certain condition is satisfied. This exiting process is analogous to the decision-making and branching with “If...Then” statements. The concept of controlling loops and exiting from loops is critical for most programming languages. This concept is often called “Flow of Control” in the programming language.

After this assignment, you will have a basic idea about the two main looping techniques that are most likely to be encountered in computer programs for engineering application.

There are several ways to create loops in Visual Basic. Let’s rewrite the The previous program using a “For Loop”. A For Loop as the general structure:

```
For n = first_n To final_n
...
Next
```

```
Private Sub Button1_Click(ByVal sender As System.Object
    Dim x As String, y As Integer, z As String = " "
    x = TextBox1.Text
    y = Len(x)
    For i As Integer = y To 1 Step -1
        z = z & Mid(x, i, 1)
    Next
    ListBox1.Items.Add(z)
End Sub
```



Now the input text can be any length up to the length of a string variable.

Assignment

1. Implement the reverse text program shown above with the loop structure. Enter some example text in the program to ensure that it

works. (Palindromes are not very good for demonstrating this code, but they are fun. For example, try: "Stanley Yelnats")

- Using the Taylor's (or Maclaurin's) expansion theorem, the function $\sin(x)$ can be approximated as:

$$\sin(x) \approx \sum_{i=0}^n (-1)^i \frac{x^{2i+1}}{(2i+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

The accuracy of the approximation increases as the number of terms in the series, n , increases. In principle, the summation results in an exact answer when n becomes infinite. As a practical matter, the summation is sufficiently accurate for modestly large values of n , say, $n = 5$ or $n = 6$.

Write a VB program to determine $\sin(x)$ using the first n terms of the series expression and then compare this value with the more accurate value returned from the VB "sin" function. The values of x and n will be input values. When evaluating the series expansion, make use of the factorial function discussed in class. Your Graphic User Interface (GUI) should look like the following:

The image shows a screenshot of a Windows-style graphical user interface (GUI) window titled "Form1". The window has a title bar with standard minimize, maximize, and close buttons. The main content area has a light beige background and is titled "Series Approximation of sin(x)". Below the title, there are two input text boxes: one labeled "x =" and another labeled "n =". Below these are three more input text boxes, each with a label to its left: "Series Approximation", "Correct value", and "Error". At the bottom of the window, there are two buttons: "Go!" on the left and "Stop" on the right.

Your program should get the required values of x (the value whose sine is to be approximated) and n (the number of terms to use in the approximation) from the text boxes.

You will need to use a loop to add each term in the series expansion to an accumulation variable "sum". The "factorial" function should be called from this loop to calculate the denominator in each term.

After completing the loop your program should print the value of the approximate value and the correct value (just use the VB $\sin(x)$ function to get this value) in the appropriate text boxes.

Compute the error between the approximate value and the correct value and in display this value in the appropriate text box. The error in the approximation of $\sin(x)$ using n terms in the approximation should be computed from the following formula:

$$\text{error}(n) = \frac{\text{correct_sin}(x) - \text{approximate_sin}(x)_\text{using_n_terms}}{\text{correct_sin}(x)}$$

Compute the results for $x = 1$ and $n = 1, 2, 3, 4, 5, 6, 7, 8$.

Plot (using a spreadsheet) your results from running your program for $x = 1$ and $n = 1, 2, 3, 4, 5, 6, 7, 8$. Your graphs should include plots of:

- (1) n vs *approximation of $\sin(x)$* for $n = 1, 2, 3, 4, 5, 6, 7, 8$; and
- (2) n vs *error* for $n = 1, 2, 3, 4, 5, 6, 7, 8$

Turn in:

1. String code:
 - a. A printout of the VB code used in your program.
 - b. A screen shot of your program running.
2. Sine expansion
 - a. A printout of the VB code used in your program.
 - b. A screen shot of your program running. The screenshot should show the result of testing your program for $x = 1$ and $n = 4$.
 - c. Spreadsheet plots of the results and the errors.