Lab03

Objectives

- to use Visual Studio and Visual Basic 2013 to create and run a simple program

Topics

- String and number variables
- User input: strings and numbers — function int()
- Assignment statements
- Evaluating expressions
- Creating output strings by parsing variables and literals: functions str()
- New material for students to learn via online resources: Using format Strings in Python
- New material for students to learn via online resources: function random()

Preliminaries

1. Get a copy of this lab folder from the N: network drive
    1. Go to folder `N:\Classes\CS130\labs` and copy and paste the folder for today's lab (i.e., Lab03) to `M:\CS130\labs`
    2. You may now work locally by opening the write-up from within the copied folder. (The write-up is the current document you are reading.)
2. For future labs, you may start by copying the proper lab folder right away.

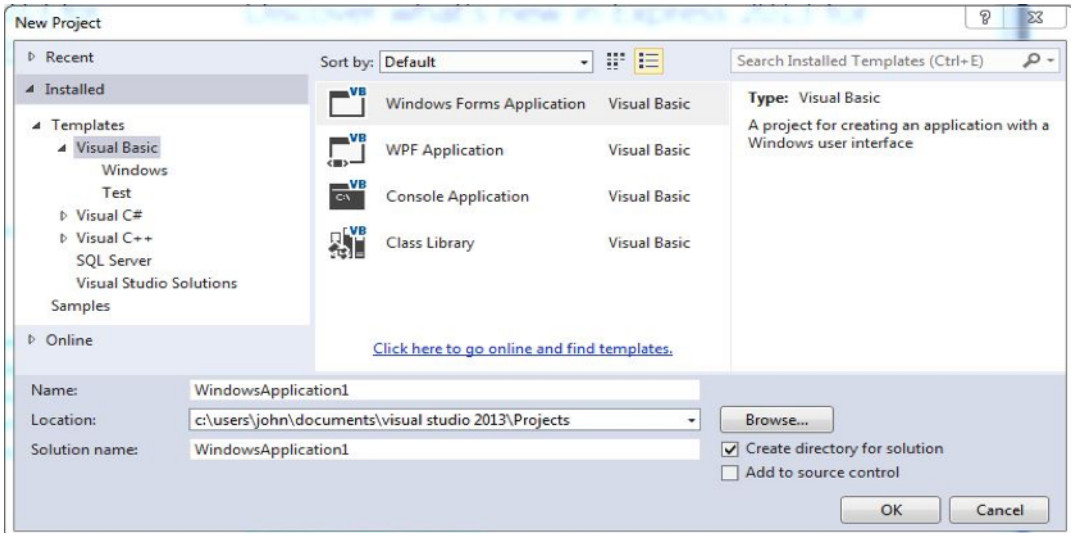
Introduction to Visual Studio

1. Launch Visual Studio 2013 by clicking on the start button. Select All Apps > Visual Studio 2013 > VS Express 2013 for Desktop. Visual Studio is a platform that can be used with many different programming languages. In this course we will be programming in Visual Basic 2013.

2. If you start VS Express 2013 and are asked to sign in, you can sign in using your CSBSJU email address as your username and your CSBSJU Windows account password.

   Make sure that you select work or school account and not personal account.

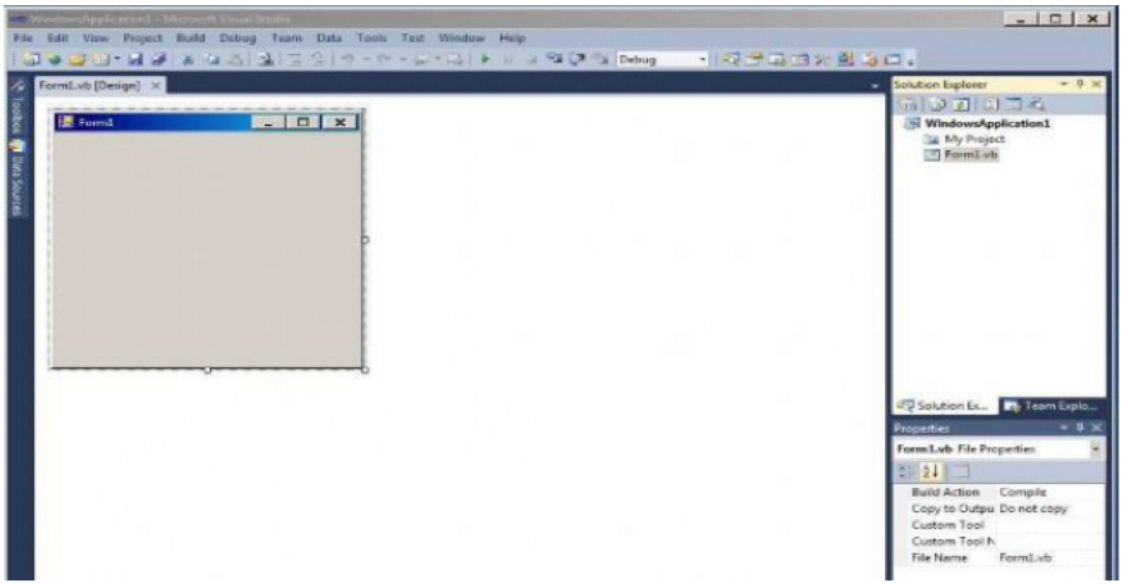
3. To create a new VB project, choose "New Project" from the menu. You will see a screen like the one below.



Make sure that Visual Basic has been highlighted under the Templates tab to indicate the language you will use and Windows Forms Application is chosen as the type of project. Those choices may be the default, but it they are not, you can choose the language and type of project to create. Once you have made your initial choice to work in Visual Basic (VB), the next time you start Visual Studio, it will launch right into the VB environment.

Give your project a name right away in the **Name:** pane at the bottom of the window. For this project, type in Lab03 for the name. Now, use the **Browse** button to select the desired **Location:** (folder), you wish to save your project in. You will always save your projects on your **M:** drive. Do not edit the "Solution name:" pane as it will be automatically filled in with the name of the project. *Uncheck* the **Create directory for solution** box in the lower right. Click **OK** to begin your new project. This will bring up a window with a new, blank form and several other panels with information about the project as you can see in the next figure.

At the top of the window is the menu bar and just below it is the toolbar. Using choices from either the menu or the toolbar will allow you to do a variety of things to your project or form. The tool bar below has icons for the most common functions, such as beginning a new project, adding a new item to your project such as a form, saving your project, and running your project. The icon you will probably use the most is the small green triangle near the middle of the toolbar which you click on to debug and run your programs.



If the toolbox tab, the properties window, or the Solution Explorer panes are not visible, you can use the "View" menu to activate any or all of those to be visible. Refer to chapter 3, pages 3-5 of your text if you have questions.

Your created project will contain a single empty Form object by default. In the Form properties window (if it is not completely visible, click on the *Properties* tab on the right side of the screen), change the Name property to **frmLab03** and Text property to **Lab03**. (PS: the Name property of an object CANNOT contain spaces).

4. Your task at this point is to start building the graphical user interface for your project. We will start with a Quit button.

   Using the Toolbox (if it is not completely visible, click on the *Toolbox* tab on the left side of the screen), drag and drop a single button object onto the blank form. This will be your *Quit* button. For every object we add to the form, we need to set some of the properties. To do this, put the mouse focus on the desired object, in this case the button you just added to the form, (i.e., single-click on the button.) When you click on the button, the Properties window for the button should have appeared on the right side of the screen. (If the Properties window is not already completely visible, click on the *Properties* tab on the right side of the screen). Now you can change two of the properties for the button. Change the Name property to **btnQuit** and the Text property to **QUIT**.

   At this point, please save all your work by selecting **File>Save All**, close Visual Studio altogether, go to where you saved your project folder and double click on the file whose type is **Visual Basic Project File**. If nothing shows up please double click on the form in the menu on the right.

5. Using the Toolbox (if it is not completely visible, click on the Toolbox tab on the left side of the screen) drag and drop the following:

   - 2 Label objects
   - 2 TextBox objects
   - 2 Button objects
   - 1 RichTextBox object

6. Now we need to set the properties for each of our objects. To do this, put the mouse focus on the desired object (i.e., single-click on the object) and in the properties window (again, if it is not already completely visible, click on the Properties tab on the right side of the screen) change the desired properties as directed below:

   - 1st Label object: change the Name property to lblGrossAnnualPay (originally it should have been Label1 or Label2) and the Text property to Gross Annual Pay (again, originally it should have been Label1 or Label2)
   - 2nd Label object: change the Name property to lblMonthlyDeductions and the Text property to Monthly Deductions
   - 1st TextBox object: change the Name property to txtGrossAnnualPay

- 2nd TextBox object: change the Name property to txtMonthlyDeductions
- 1st Button object: change the Name property to btnComputeNetMonthlyPay and the Text property to COMPUTE NET MONTHLY PAY
- 2nd Button object: this is the QUIT button which we already created
- RichTextBox object: change the Name property to outResults

Programming the `Quit` button

1. Double-click on button `btnQuit` to get to the code page which should look something like below.

```
Public Class frmLab02
    Private Sub btnQuit(sender As Object, e As EventArgs) Handles btnQuit.Click

    End
End Class
```

2. Move into the space between `Private Sub...` and `End Sub` and add the command `End` as shown below. Now this button will terminate the VB project (i.e. End it) when clicked.

```
Public Class frmLab02
    'A button that terminates the VB project
    Private Sub btnQuit(sender As Object, e As EventArgs) Handles btnQuit.Click

    End
End Class
```

3. In the image capture above, notice the green line above Private Sub… which is preceded by an apostrophe. This is called a comment line in VB which allows us to write comments for the reader of our program (and for ourselves!). Comments are completely ignored by VB. Add the above comment line to your code.

   Run your program to make sure that the quit button works.

Programming the `Compute` button

1. Switch back to the form design view and double-click on button `btnComputeNetMonthlyPay`.

```
Option Explicit On

Public Class frmLab02
    'A button that computes the monthly net pay given the annual gross pay
    ' and the monthly deductions
    Private Sub btnComputeNetMonthlyPay_Click(sender As Object, e As EventArgs) ...

    End

    'A button that terminates the VB project
    Private Sub btnQuit(sender As Object, e As EventArgs) Handles btnQuit.Click

    End
End Class
```

2. Add the comment line shown above line `Private Sub btnComputeNetMonthlyPay_Click...` and then move into the space between `Private Sub btnComputeNetMonthlyPay_Click...` and `End Sub` and declare the following three variables:

```
Dim grossAnnualPay As Single = 0, netMonthlyPay As Single = 0
Dim monthlyDeductions As Single = 0
```

   Variables `grossAnnualPay` and `monthlyDeductions` will be used to read values input by the user in the corresponding text boxes. Variable `netMonthlyPay` will store the net monthly pay resulting from dividing the gross annual pay (variable `grossAnnualPay`) by 12 and subtracting the monthly deductions (variable `monthlyDeductions`).

   - Reading a value from a text box into a variable is accomplished via statement `variableName = txtTextBoxName.Text`. In our case, you need to add the following two lines after the `Dim` statement.

```
Dim grossAnnualPay As Single = 0, netMonthlyPay As Single = 0
Dim monthlyDeductions As Single = 0

grossAnnualPay = txtGrossAnnualPay.Text
monthlyDeductions = txtGrossAnnualPay.Text
```

   - Computing the net monthly pay requires writing the formula stated above only using our variables as shown below. Add this statement to your program right after the statements that read the values from the textboxes into variables.

```
netMonthlyPay = grossAnnualPay / 12 - monthlyDeductions
```

3. Now that we've computed the value for variable `netMonthlyPay`, we're ready to display its value in our `outResults` RichTextBox object. We can use the `outResults.Text = netMonthlyPay` to do this. Add the following line of code: outResults.Text = netMonthlyPay to your code right after the line that computes the `netMonthlyPay`.

4. Now run your program to make sure that it works. Enter *100000* (with 5 zeros) into `txtGrossAnnualPay` and *289.1* into `txtMonthlyDeductions`. You should see an output of *8044.233*.

   Since this is a monetary value, it would be better to display it with a $ sign and limit the number of decimal

places to 2. Function `FormatCurrency(_value to be formatted, number of desired decimal places)` can help! Change statement `outResults.Text = netMonthlyPay` to `outResults.Text = FormatCurrency(netMonthlyPay, 2))` (we're telling VB to display the value of variable `netMonthlyPay` with exactly two decimal places).

Run your program one more time with the same input values; you should see an output of *$8,044.23*. Experiment with different values for the second parameter of the function which records number of desired decimal places; try 1, 3 and 4 to see what happens. By the way, omitting this second parameter altogether defaults to 2 decimal places which is what I am using henceforth.

5. Eventually, we'd like to be able to display a more meaningful message such as **Your monthly net pay is $8,044.23.** instead of just a meaningless number! In VB, text that needs to be displayed literally must go between double quotes. Let's replace

```
outResults.Text = FormatCurrency(netMonthlyPay))
```

with the following statement:

```
outResults.Text = "Your monthly net pay is FormatCurrency(netMonthlyPay)."
```

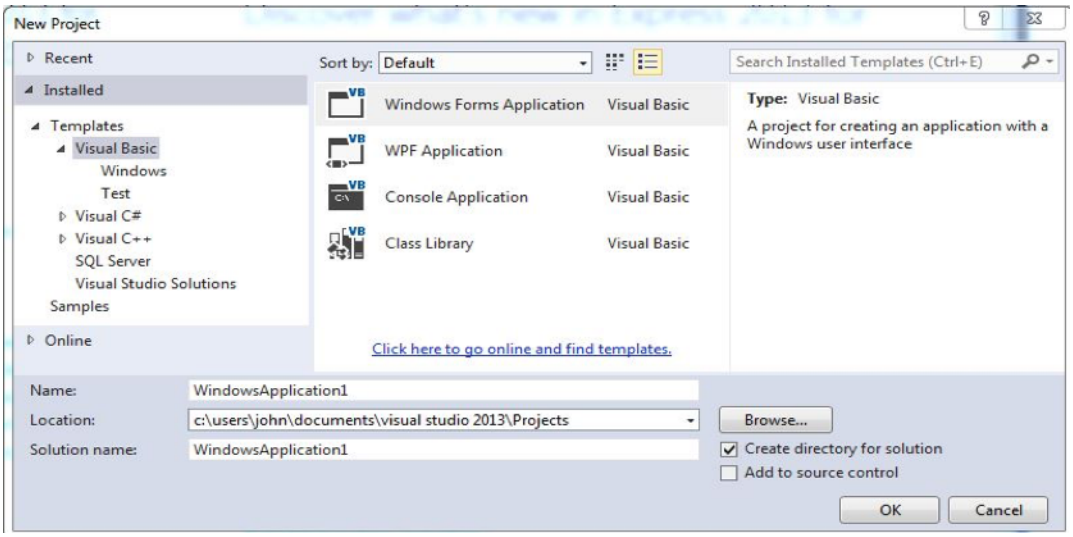
Run your program one more time with the same input values; did you get a meaningful message? The reason this happened is because, even though the text before and after `FormatCurrency(netMonthlyPay)` needs be enclosed between double-quotes, variables and functions should not (without being interpreted as literal text)! Furthermore, VB won't accept something like the following because we are required to explicitly concatenate the literal text and the variables to be displayed. Bummer!

Luckily, VB has the special operator & that does exactly that. Add & before and after `FormatCurrency(netMonthlyPay)` (with a space before and after each &) as shown below and test your program one last time with the same input values. It should work correctly.

```
outResults.Text = "Your monthly net pay is " & FormatCurrency(netMonthlyPay) & "."
```

Part II

Before writing this program, study and run the sample ''General Concepts/FormLevelVariables_GeneralStore" in the VB_Examples folder inside our CS130 folder on the N: drive and then create a program that will allow a customer to make purchases by clicking on buttons as shown below.



In particular, assume there are 4 products available – Soap, Light Bulbs, Candy Bars, and Soda. Your program should have a button for each of these products as seen in the image above. When a customer clicks on one of the buttons the price of the product should be added to a running total and the product and its price will be displayed in a large rich text box on the screen.

Before you begin programming, it is important that you first design your algorithm. Remember, you cannot code the solution to a problem in Visual Basic until you know what solution it is that you are trying to code. A great place to write your algorithm is in a comment at the top of the Visual Basic code for your program. Use examples from class as a guide for the structure of your algorithm. For this particular problem, you will need to write two different algorithms: one that accomplishes the task of each of the 4 product buttons – this algorithm will be the same for each of the buttons, hence, a single algorithm, and one for the calculation of the total.

After you are satisfied with your algorithm, you can begin creating your form and writing your Visual Basic code. Keep in mind that since the running total needs to be accessed by several subroutines on this form, that variable needs to be declared as a form-level (i.e. global) variable, available to all of the subroutines on this form. If you use the variable name, runningTotal, it should be declared near the top of the form just below the class statement and above any sub's. Recall that form-level variables must be declared here to be available to all of the subroutines on this form. Sub-routine (i.e. local) variables are declared within each subroutine. Only declare variables as form-level if they need to be accessed by more than one subroutine. When the user has completed his/her purchase, s/he will click on a Total button that will (1) print out the total so far, (2) compute the sales tax (7%), and (3) then calculate and print the overall total, which should include the 7% tax.

Assume that Soap costs $1.85, Light Bulbs cost $1.25, Candy Bars cost $0.75, and Soda costs $0.50. Use FormatCurrency to print prices and totals in a consistent format.

Here is an example. If the user clicks Soap twice, Light Bulbs once, and Soda once and then clicks the Total button, what should appear in the output box should be exactly as you see below:

```
' Variable frmtStr should be declared at the top of the form as a form-level
' variable The following format string variable displays two strings where
' the format for each is described by two comma-separated numbers enclosed
' between {x,y} where x describes which string is being formatted (0 is
```

```
' first string, 1 is second string, etc.) and y represents the number of
' characters allocated to the string (in the following example, we're using
' 10 characters). Note that a ' negative value indicates align left
Dim frmtStr As String = "{0,-10}{1,10}" & vbNewLine

' using frmtStr in a subroutine later in the code
' In the first example, we're displaying string "CSCI 130" in 10 characters
' left-justified and string "1:00pm" also in 10 characters but
' right-justified
outResults.AppendText(String.Format(frmtStr, "CSCI 130", "1:00pm"))
outResults.AppendText(String.Format(frmtStr, "CSCI 130L", "11:20am"))
```

The above code will produce the following output:

```
CSCI 130       1:00pm
CSCI 130L     11:20am
```

Be sure to use Form-Level variable (i.e. a variable with a global scope) to hold the running total. The subroutine associated with each different button would simply increment the running total by the corresponding item cost and display the item and its cost in the output box. You should also have a Clear button that will clear the output box and reset the Form Level variable holding the running total to 0. Include a QUIT button to stop the program.