Santa Monica College

Computer Science 3

Visual Basic Course Pack

Introduction

VB .NET, short for Visual Basic .NET is a language that was first introduced by Microsoft in 1987. It has gone through many changes to make it easier to learn, and allow its programmers to build applications fast.

Most programming language compilers today, have many tools to help programmers write code fast and easy. VB .NET has what is called the IDE (Integrated Development Environment) that shows typos, color coded keywords, and an auto complete feature.

If you are new to programming in general, the following section explains important concepts.

What is Programming

All the applications you run on a computer have been created by other programmers. Most of these applications allow you to enter data, they process the data and then show you output. For example, in Microsoft Excel, you type numbers in cells; the numbers you type appear in the cells, in most cases formatted. You type a formula to say add those numbers, the formula shows the result. Hence you type data; Excel processes the data (either formatting, and or adding) then displays the result in the cells.

To create an application, programmers write statements that are commands. These commands are then executed by the CPU. The CPU uses binary language, called machine language, which is unreadable by humans. But, what form of language do programmers use to express their commands in? They use one of the available programming languages; in fact to be precise, they use a compiler of a programming language.

A **compiler** then is like a translator; it takes code written in some language, and translates it into the CPU language, Machine Language. So why not use our own English language to write code? Because the natural English language (or any other human language) is currently very difficult to parse, and hence finding a compiler that understands human language is currently not possible. To simplify matters, the people who create compilers, try to find a set of simple rules that makes translation from that language to Machine Language easy. Those people came up with many compilers, such as C, Java and VB.

Now, when a programmer writes code in a programming language, that code is referred to as **source code**. Remember, the CPU can not understand source code, and needs a compiler to translate that code into the Machine Language.

So in order for someone to create their own application to perform a certain task like adding numbers, they need to:

1. Think about what the application should be doing in details.
2. Write down what the steps are so that the user enters data and then the code shows some output.

3. Translate those steps into the programming language of choice.
4. Get the compiler to create the Machine Language version of the code, which is called the executable file, or simply the executable. This file has an extension of .exe - in PC machines, short for executable.
5. Install this executable file (depending on the language used, you may need to install other files) on the user's machine.

We need to stress that steps 3 and 4, involves many other processes that make up what is called Program Development Cycle, which includes Analysis, Writing Code, and Testing among others.

Before we jump into learning VB, let's focus on steps 1 and 2 above.

Flow Charts

Before writing code, a programmer has to think about what the code is supposed to do. Much as when you are presented with a problem like "what is the double of 32 divided by 8, plus -5?" You would do the math in 3 steps: 1. Double 32, giving 64, and then 2. Divide 64 by 8 giving 8, then 3. Subtracting 5 giving 3.

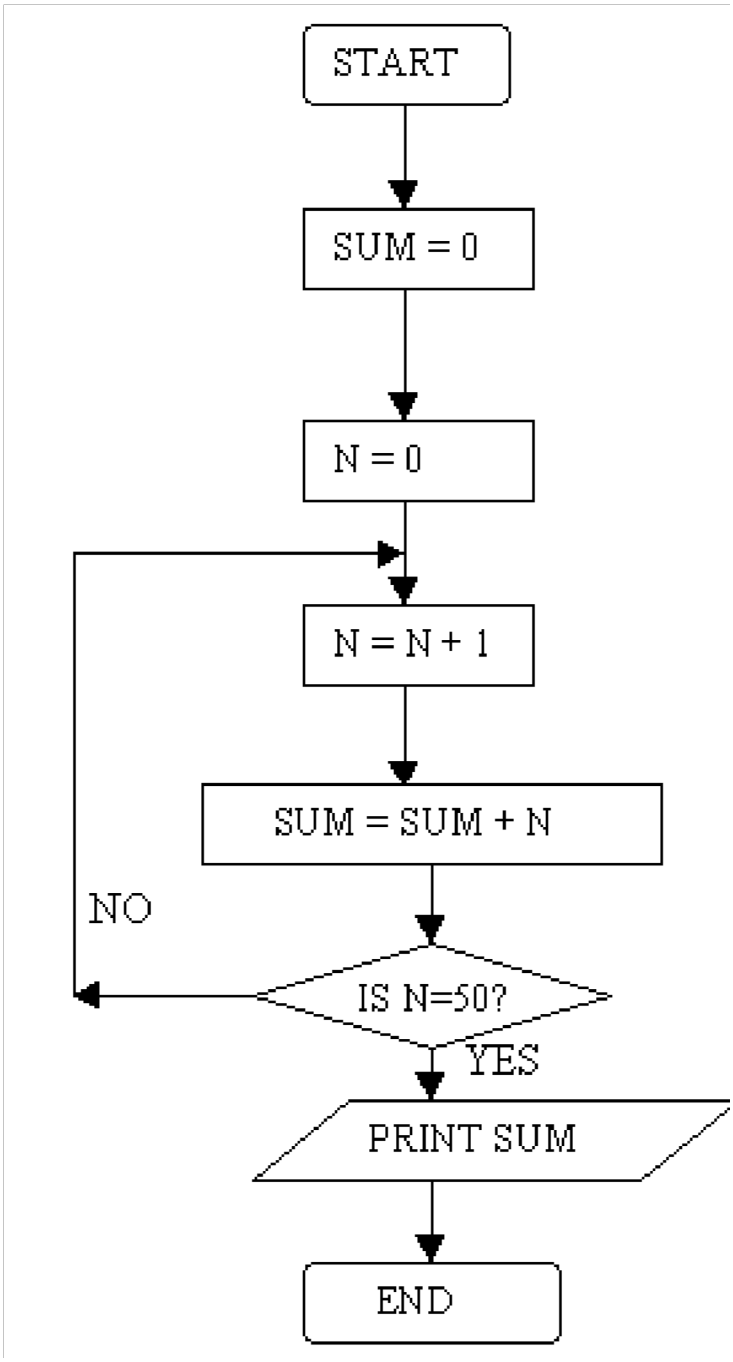
In programming, you do the same. You write down the steps needed to solve a problem. Programmers use a graphical representation for doing so, called flow charts. There is also pseudo code. We will focus on flow charts.

A flowchart is a means of visually presenting the flow of data through an information processing systems, the operations performed within the system and the sequence in which they are performed. **A program flowchart, describes what operations (and in what sequence) are required to solve a given problem.** The program flowchart is like blueprint of a building. The architect draws a blueprint before starting construction on a building. Similarly, a programmer draws a flowchart prior to writing a computer program. As in the case of the drawing of a blueprint, the flowchart is drawn according to defined rules and using standard flowchart symbols prescribed by the American National Standard Institute, Inc.

Meaning of a Flowchart

A flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to get the solution of a problem. Flowcharts are generally drawn in the early stages of formulating computer solutions. Flowcharts facilitate communication between programmers and business people. These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems. Once the flowchart is drawn, it becomes easy to write the program in any high level language. Often we see how flowcharts are helpful in explaining the program to others. Hence, it is correct to say that a flowchart is a must for the better documentation of a complex program.

The figure below shows an example of a flowchart for adding 50 numbers.



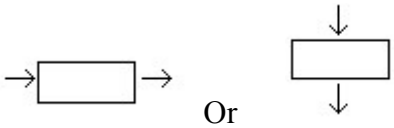
Guidelines for Drawing a Flowchart

Flowcharts are usually drawn using some standard symbols; however, some special symbols can also be developed when required. Some standard symbols which are frequently required for flowcharting many computer programs are described below.

Shape	Task
small circle with S in the middle	**Start** of the program
small circle with E in the middle	**End** of the program (also sometimes oval shape)
Rectangle	Computational steps or **processing** function of a program (mostly called processing)
Diagonal	**Input** or **output** operation
Diamond	**Decision** making and branching
small circle	Connector or joining of two parts of program
Plain Arrow	Flow line
Block Arrow	Connector
dotted line ending at center of 3 side fork	Annotation

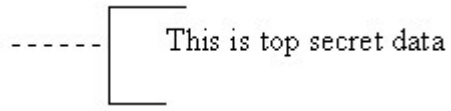
The following are some guidelines in flowcharting:

a. In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
b. The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
c. The usual direction of the flow of a procedure or system is from left to right or top to bottom.
d. Only one flow line should come out from a process symbol.



e. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol.

f. Write within standard symbols briefly. As necessary, you can use the annotation symbol to describe data or computational steps more clearly.



g. If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines. Avoid the intersection of flow lines if you want to make it more effective and better way of communication.
h. Ensure that the flowchart has a logical *start* and *end.*
i. It is useful to test the validity of the flowchart by passing through it with a simple test data.

Advantages of Flowcharts

The benefits of flowcharts are as follows:

1. Communication: Flowcharts are a better way of communicating the logic of a system to all concerned.
2. Effective analysis: With the help of flowchart, problem can be analyzed in more effective way.
3. Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. Proper Debugging: The flowchart helps in debugging process.
6. Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

Limitations of Using Flowcharts

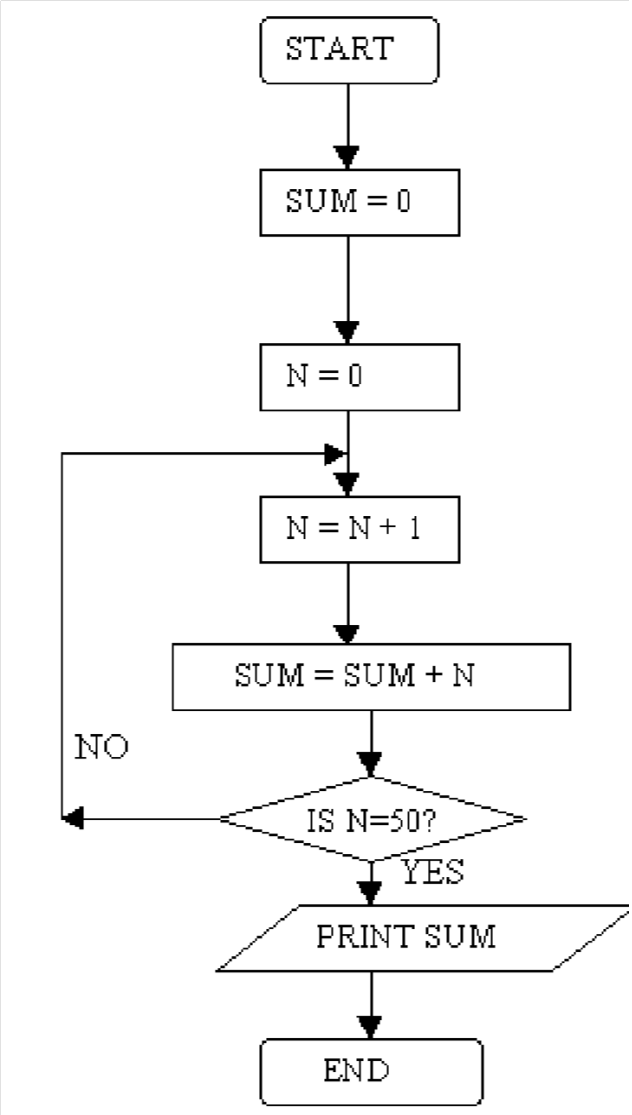
Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.

1. Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely.
2. Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
3. The essentials of what is done can easily be lost in the technical details of how it is done.

Here are a few examples to demonstrate.

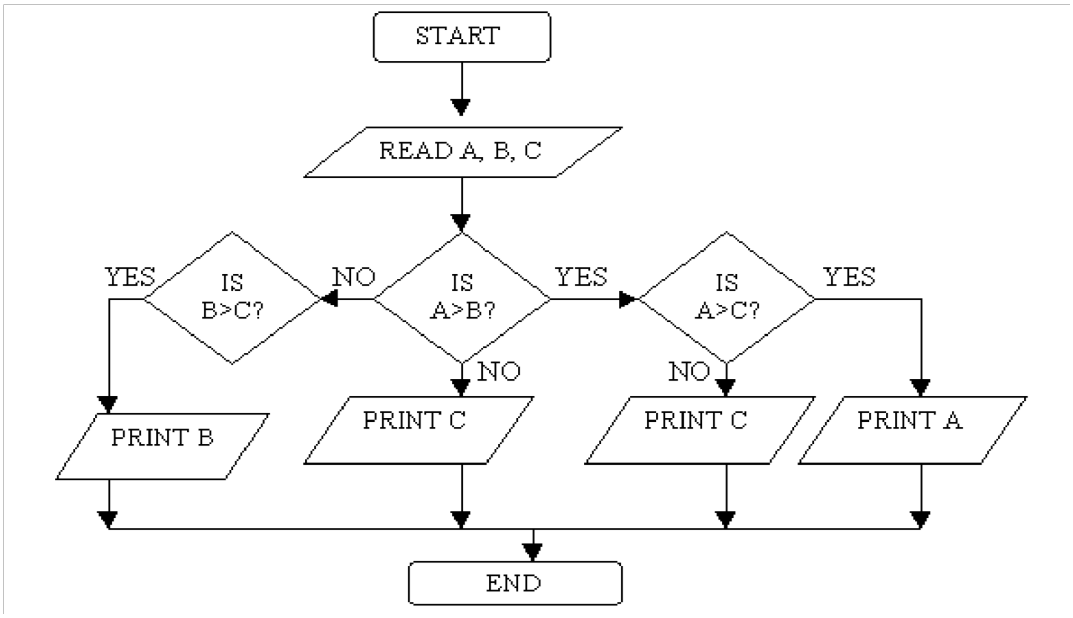
Example 1

The next flowchart is used to find the sum of first 50 natural numbers.



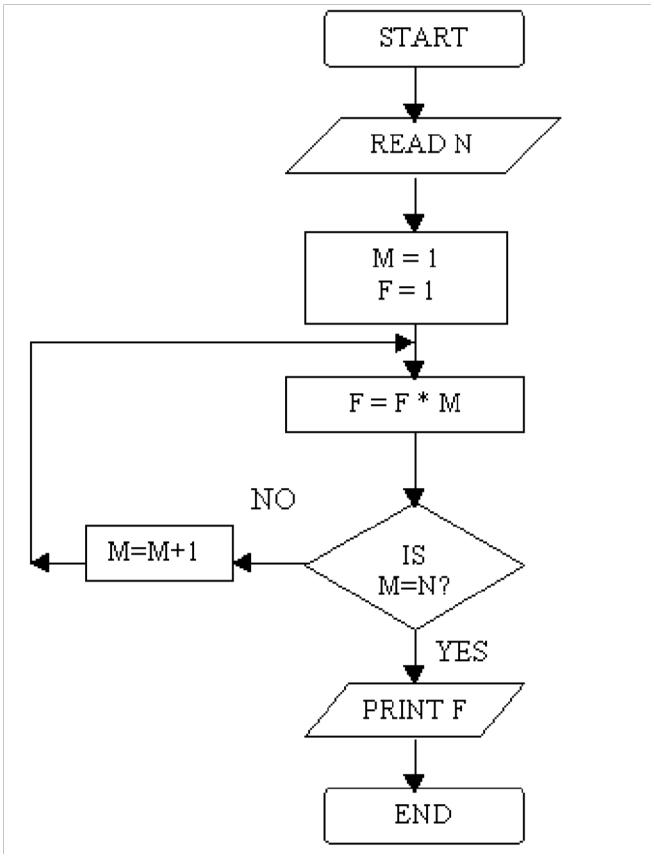
Example 2

The flowchart to find the largest of three numbers A, B, and C shown below has an error. Can you locate that error?



Example 3

In this example, the flowchart shown is for computing factorial N or (N!) Where N! = 1 * 2 * 3 * ...* N



The following exercises are for your review.

1) A program flowchart indicates the_____ to be per formed and the _____ in which they occur.
2) A program flowchart is generally read from _____ to _____
3) Flowcharting symbols are connected together by means of _____
4) A decision symbol may be used in determining the _____ or _____ of two data items.
5) _____ are used to join remote portions of a flowchart
6) _____ connectors are used when a flowchart ends on one page and begins again on other page
7) A _____ symbol in used at the beginning and end of a flowchart.
8) The flowchart is one of the best ways of _____ a program.
9) To construct a flowchart, one must adhere to prescribed symbols provided by the _____.
10) The programmed uses a _____ to aid him in drawing flowchart symbols.

Answers

1) Operations, sequence
2) Top, down
3) Flow line
4) Equality, inequality
5) connectors
6) Off -page
7) Terminal
8) documenting
9) ANSI (American National Standards Institute)
10) Flowcharting template

Learning Visual Basic

Learning a programming language is similar to learning to speak another language. To be fluent, you need to practice and practice and practice. The process of learning is also similar. You learn a few words, and then learn how to put them together to make a sentence. Through practice, you become comfortable with what you've learned. Then you add more words, and sentences. Learning what the words are and putt them into a sentence is called the **syntax** of the language. Learning how to translate your thoughts into sentences is the **semantics** of the language. As you learn any language, programming or other, you will find that learning the syntax is much easier than the semantics. Practice through writing flowcharts then translating them into code makes learning the semantics easier.

In short, as you learn any programming language, you need to learn:

1. What the words of the language are and how to put them together (**syntax**).

2. How to create a sentence that reflects the thought you have (**semantics**).

In programming, words are called **Keywords**, and sentences are called **Statements**. And just like in human languages, when you want to express yourself, you *think* of what you want to say, then you *say* it, in programming, you *think* of what the code should do (called the **logic**), put it down in a flowchart, then using the keywords and statements of the language, you translate the flow chart into code.

Keep in mind that keywords are the words of the compiler. Hence they have a special meaning and must be placed according to the syntax in a certain order to make up statements.

Using the Visual Basic Compiler

As you know the compiler is software that needs to be installed before you can use it. If you are working on your own computer, you must get the VB compiler which comes packaged with other compilers in a suite called Visual Studio .NET.
The VS .NET environment has many tools to help make your code-writing and executing experience easy. As a beginner, you will be creating Windows applications. These are applications that when run by the user cause a Window (as in Microsoft Windows screen) to show on the screen. The user types data and or makes selections from drop down lists or checkboxes, then clicks some button and gets output. Let's walk through an example to help you understand the process.

Creating your first Windows Application

Let's say you want to create an application that plays the Number-Guessing game. To proceed, you need:

1. A detailed description of what the game rules are. This helps you in the next step.
2. Create a flowchart that represents how the game is played.
3. Write the VB statements that are equivalent to the flow chart of step 2.

Detailed Description of the Problem

The game is played between the computer (your code) and a single user. The computer makes a secretive (the user does not know it) guess of a number (whole number) between 0 and 10. The user has one chance to type in the correct guess. Once the user makes a correct guess, the user wins and the game ends. When the user makes an incorrect guess, they loose and the game ends. This problem is simplified to allow us to work through a complete example. We can add more features to it later.

The above is a description of the problem in English 'terms. To help you extract the logic, you can rewrite the description in steps. Here is an example, where the same text above will be rewritten as numbered steps.

1. The game is played between the computer (your code) and a single user.
2. The computer makes a secretive (the user does not about it) guess of a number (whole number) between 0 and 10.
3. The user has one chance to type in the correct guess.
4. Once the user makes a correct guess, the user wins and the game ends.
5. When the user makes an incorrect guess they loose and the game ends.

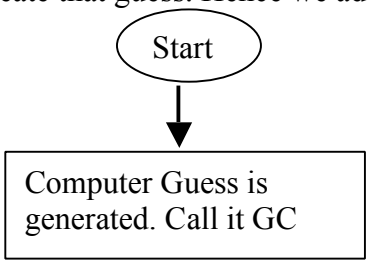
In most cases the steps are not necessarily in the same order they need to come up with the right logic, however, putting paragraphs in numbered steps, helps you get closer to a flowchart.

Trying the flowchart

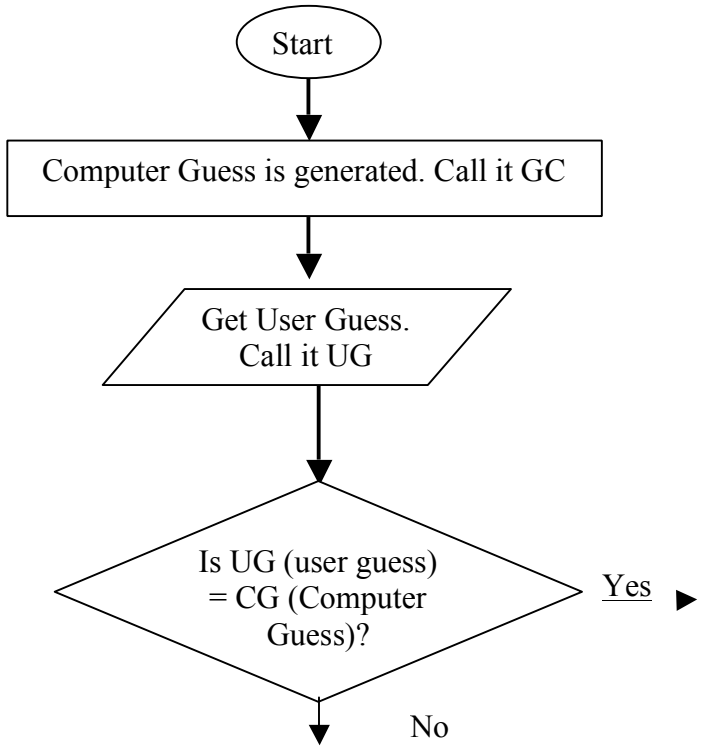
The correct final flowchart will not come to you the first time you attempt it. So with a little patience, you should be able to write one flowchart, test it to see if it works, and make the needed adjustments and test again, till you get it right. To help reduce errors, and locate them fast, try building a small part of the flowchart, test it for completeness and correctness before adding more. You should follow the same approach when writing code. This incremental approach helps minimize errors, and makes locating them an easier task. Errors will appear, so learning how to deal with them efficiently distinguishes the experienced programmer from the novice.

Let's try to put the steps above as they are into a flowchart. We note that some steps are processing, some are input and output and other are Decision. We know every flow chart start with the Start circle, so we have:

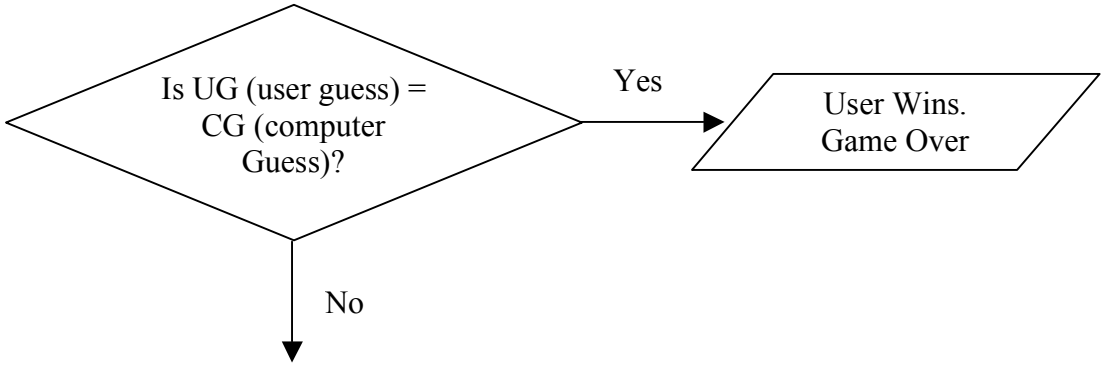
Now we move to step 1. There is no real action to be done here, so we skip to step 2, the computer makes a secretive guess of a whole number between 0 and 10. This means there will be some processing done to create that guess. Hence we add a processing rectangle.



Next we add step 3, in which we take an entry from the user. In step 4, if that entry is the same as the computer's guess, then the user wins. Hence below we added steps 3 (Get user guess) and then step 4 (once the user makes a correct guess…)

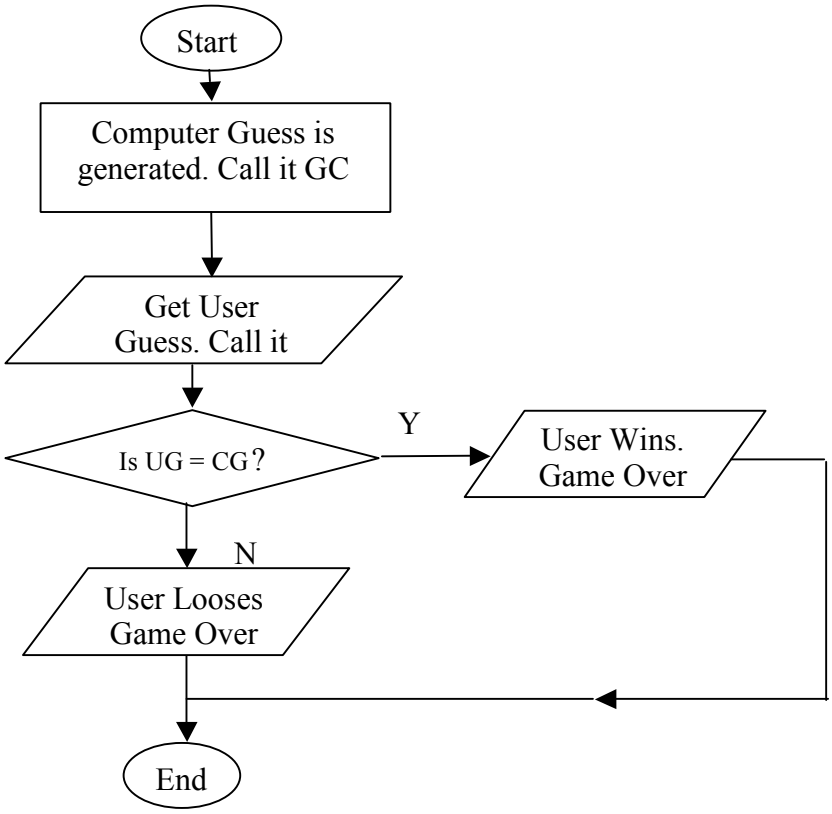


Now to complete step 4 we need to add an output that tells the user they won, and then end the game.



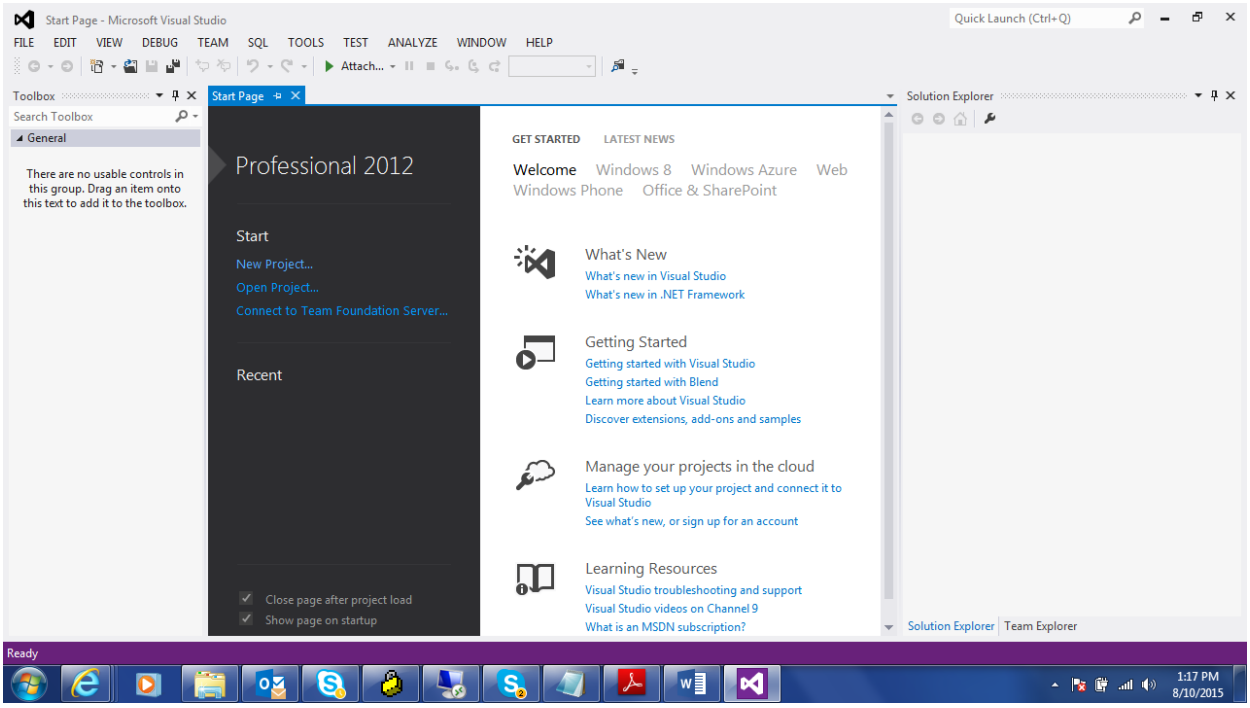
In the above, the user gives the correct answer and the game ends, hence we need to connect that output box to a circle shape with End inside. We'll do that at the end of this exercise.

Moving on to step 5 which completes the 'No branch above. If the user gives an incorrect response, they loose and the game is over. This means we do something similar to the above, except that the output should say 'User Lost, Game Over'. The whole chart is shown below.

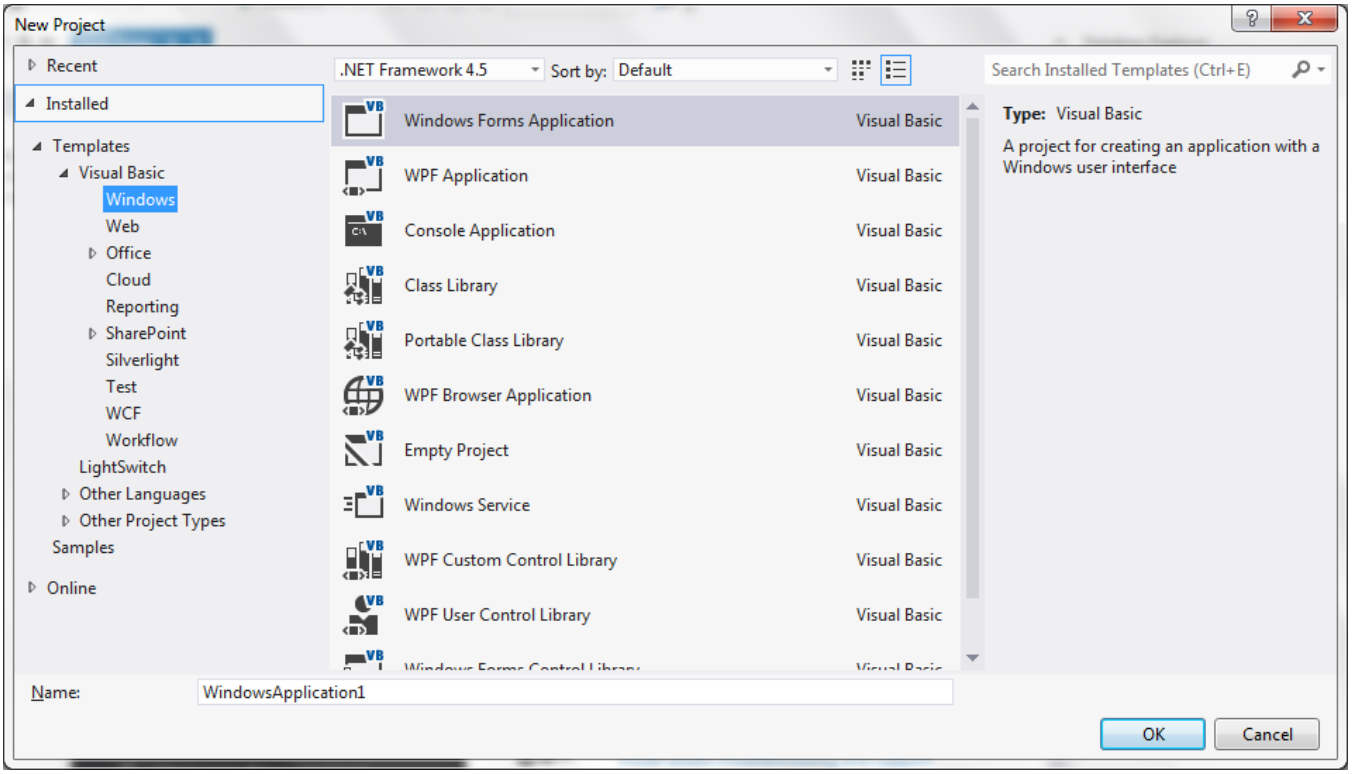


Writing the Code

It's time to use the compiler. Start Visual Studio .NET and then from the File menu choose New Project as shown below. Note that in newer versions of the compiler, the screen below still has the same options.

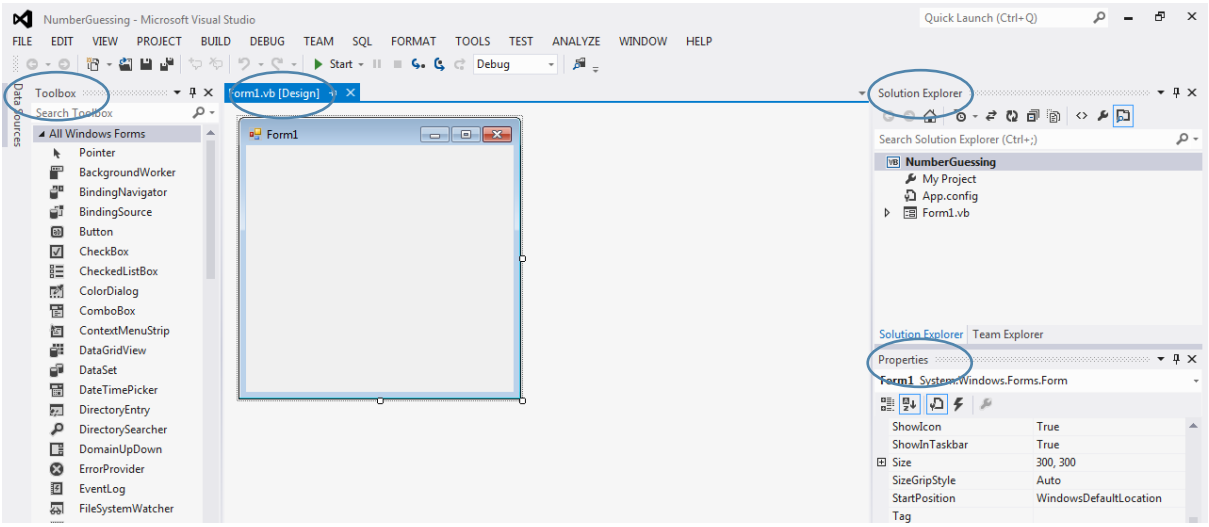


You will then get the screen below:



Make sure **Visual Basic**, and **Windows** is selected from the left side panel, and on the right side panel, under Visual Studio Installed templates, make sure **Windows Forms Application** is selected, as highlighted above. Type a name, NumberGuessing, in the Name box in the bottom of the screen, and then click Ok. This name represents your application. You can change it later.

You will wait for a few seconds, while the compiler generates the needed files for your applications, and then you'll see the next screen.



From the screen above, identify the **Toolbox**, the form **Design**, the **Solution Explorer**, and the **Properties** windows as highlighted above. We will work with these windows later. The above screen may appear overwhelming at first. To simplify things, we will focus on the things we need only, to create the number guessing game program.

Before we start writing the code, let's review our plan. We will be creating an application that displays a window to the screen, on which the user will type the number they are trying to guess, and then click a button to run code to check if their guess is true. In the IDE above, you will follow these steps to achieve that objective:

1. Design the window the user will see, and add the items needed so the user can type somewhere and then click a button.
2. Type the code that should run when the user clicks the button.
3. Test if the application runs correctly.

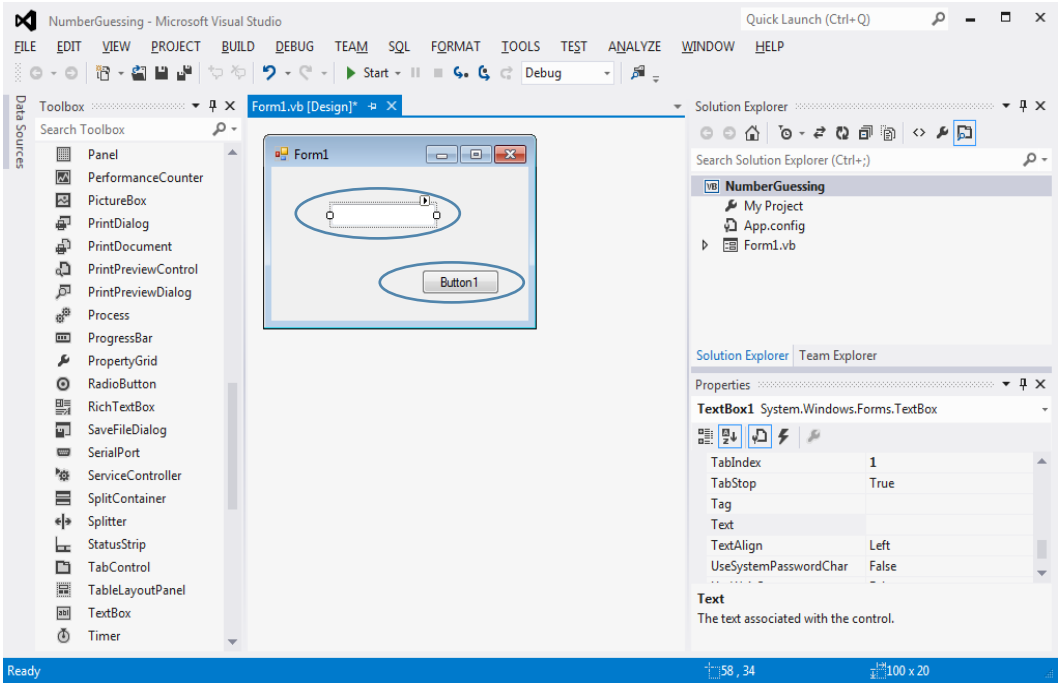
Designing the Window
To do the steps above, the VB IDE makes it easy to have a window show at the screen. It's right there, with the title of Form1. In order to create a writing area, and the button to be clicked, you will need to use the Toolbox. This is where ready made items such as a **Textbox** (where the user can type) and a **Button** (which the user can click), are available. These items are called **controls**. The window you will add these controls to is called a **form**.

Adding Controls
To add a control, double click it in the toolbox and it will appear in the top left side of the form. You can then move it by dragging it to where you think it should be. You can also click and hold the control from the Toolbox, then drag the mouse over to the form area, and release to where you want the control to appear. Note the controls in the Toolbox are alphabetically ordered.

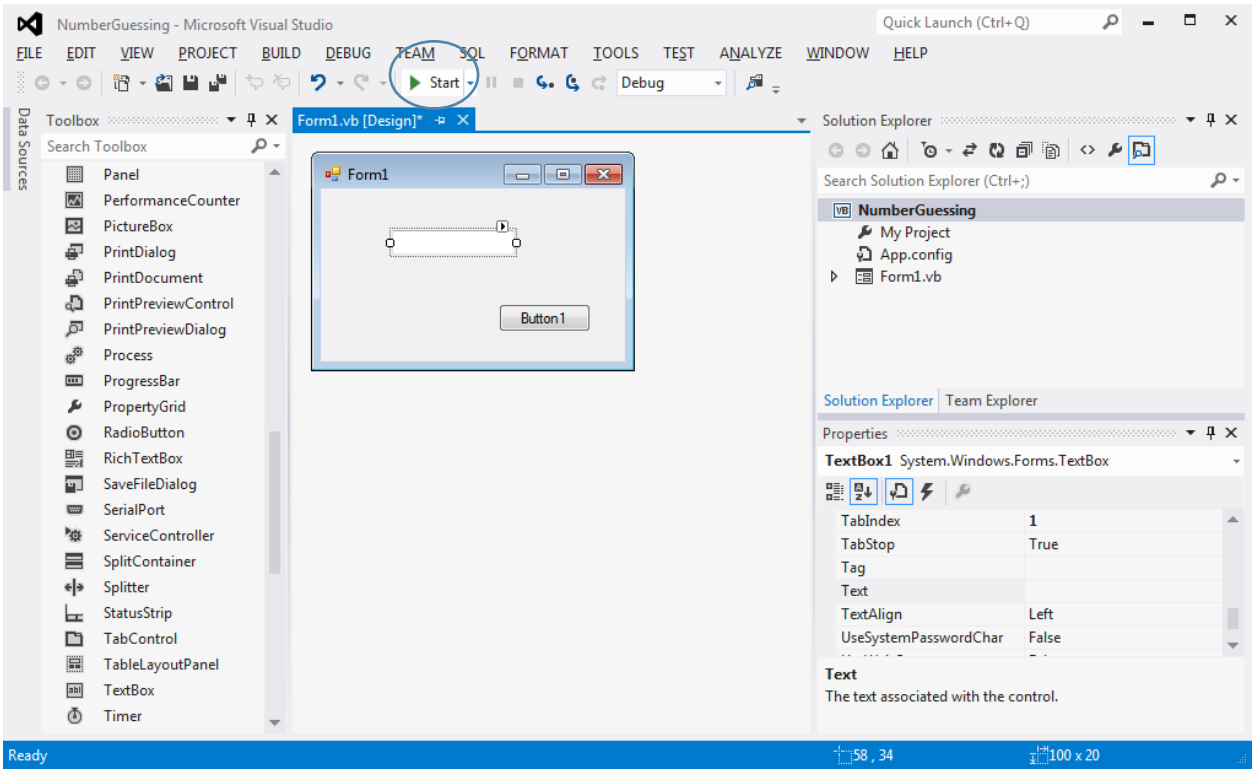
Resizing
To resize a control, click it, then grab the sizing handles to the requested size.

Deleting Controls
Click the control, and then press the Delete button from the keyboard.

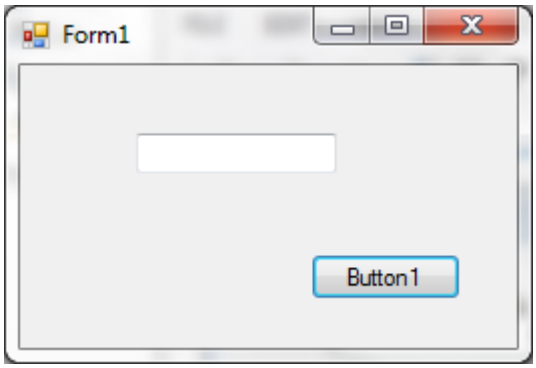
Design vs. Runtime
Now as you add the needed controls, you need to be aware that you are working in what is called the "**Design View**", or **design mode**. In this view, you are the creator of the application. If you want to see what the user of your application sees, you need to have the compiler convert your code into machine language, and then run it on the processor. This is called **Runtime Mode**. While in runtime mode you cannot make changes to the form, as far as moving controls or deleting them. This is the desired effect.
Moving on, we need to add a textbox, and a button as highlighted below.



If you want to see how this looks in runtime mode, click on the Start button shown below. This process is called **testing** and is recommended each time you make a few changes to your work.



You will then see the following:

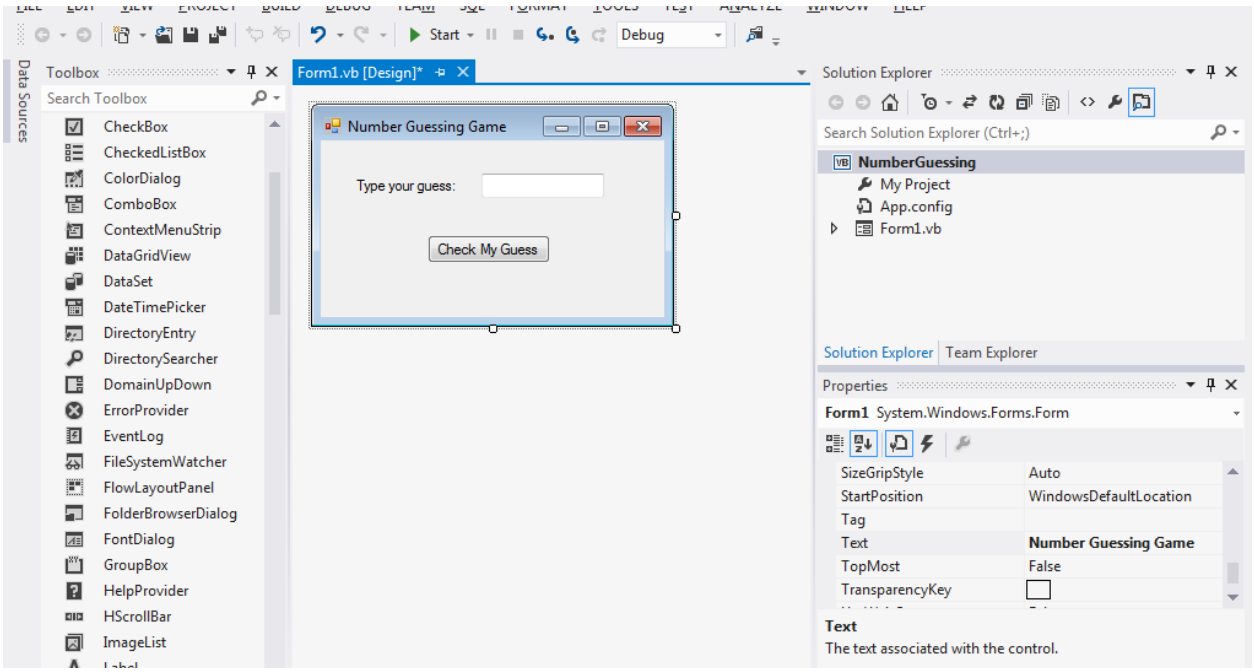


You may think it does not look professional, and it probably does not. To fix it, we may need to change the text shown in the button above, and maybe add a label next to the textbox to indicate what it should be used for. Finally, you may want a title like "Number Guessing Game' in the title bar instead of Form1. To make all these changes, we need to go back to Design mode. This means we have to close the form above, by either clicking the close button (the X on the top right side) of the runtime window, or the Stop button in the IDE, which is 2 buttons to the right of the start button.

Using the Properties Window

To change the text shown in the button above, you will need to change a **property** of the button. This is where you use the properties window. Click the button, and look for the **Text** property in the properties window. Click in the box to the property, and change the text to something like Check My Guess, as shown below.

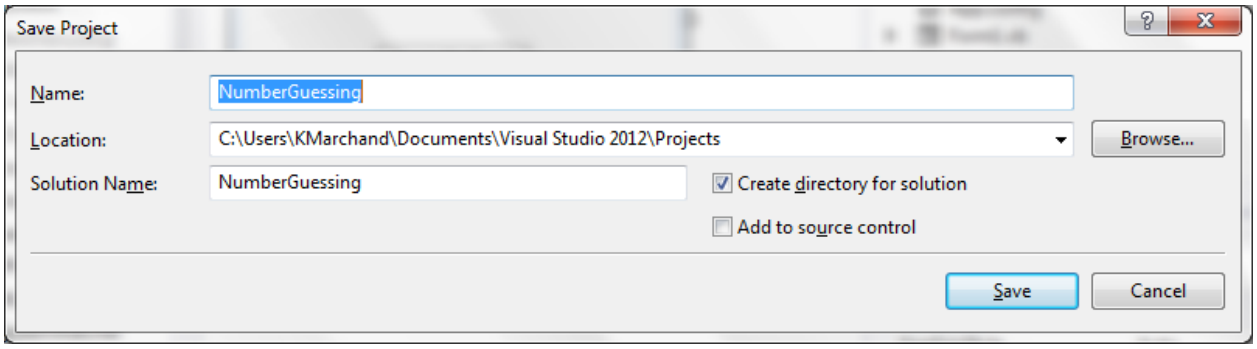
You may also do the same, click the form, and click the Text property to change its title.
Finally, add a label, and change its Text property into something more appropriate. All these changes are shown later. Again, run the form using the Start button to check that it looks ok.



Before we move on, we need to learn how to save and reopen an existing project. To save the work you did above, from the File menu, click Save. This saves only the form you created. But wait, is there more?

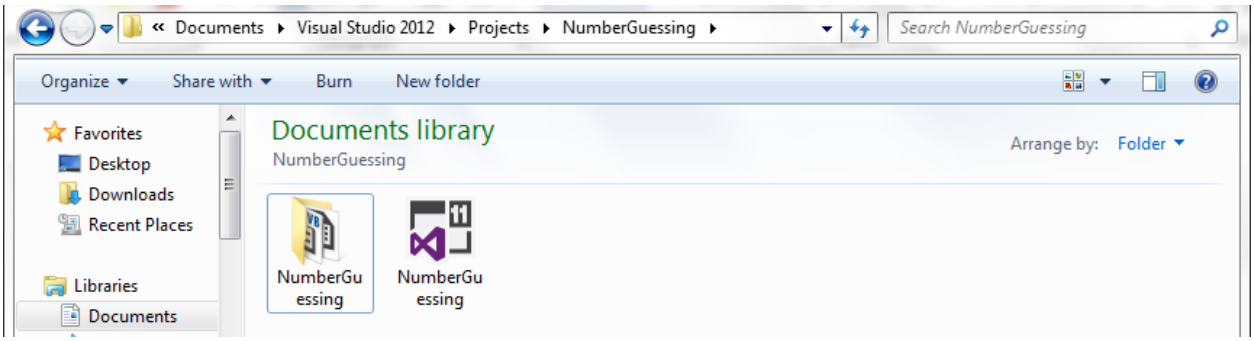
A Windows Application in Visual Basic .NET is made of several files. In CS 15, these files are discussed in more detail. In our class however, we will restrict ourselves to the **Project** file and the **Solution** file. A typical windows application may be made of several forms (a form file will be created for each) and several other modules and classes (beyond our class scope to discuss). A Project file keeps track of what files and forms are in your application; what their names are in addition to other info. In more advanced cases, there may be several projects working together to make up one application. A solution file keeps track of these projects. Note the first icon shown in the Solution Explorer window in the above screen is the Solution file. The second icon is the project file, while the third is the form file. The Solution Explorer is at the top right corner of your screen.

Back to saving; exit using File, Exit. You will get a request to save or discard. Choose save and you'll get the screen below prompting you to change the 'Name (if you wish), 'Location where you want to save your files for this application. VB will create a folder for you, and saves it in the location specified if you check the box shown; hit Save.

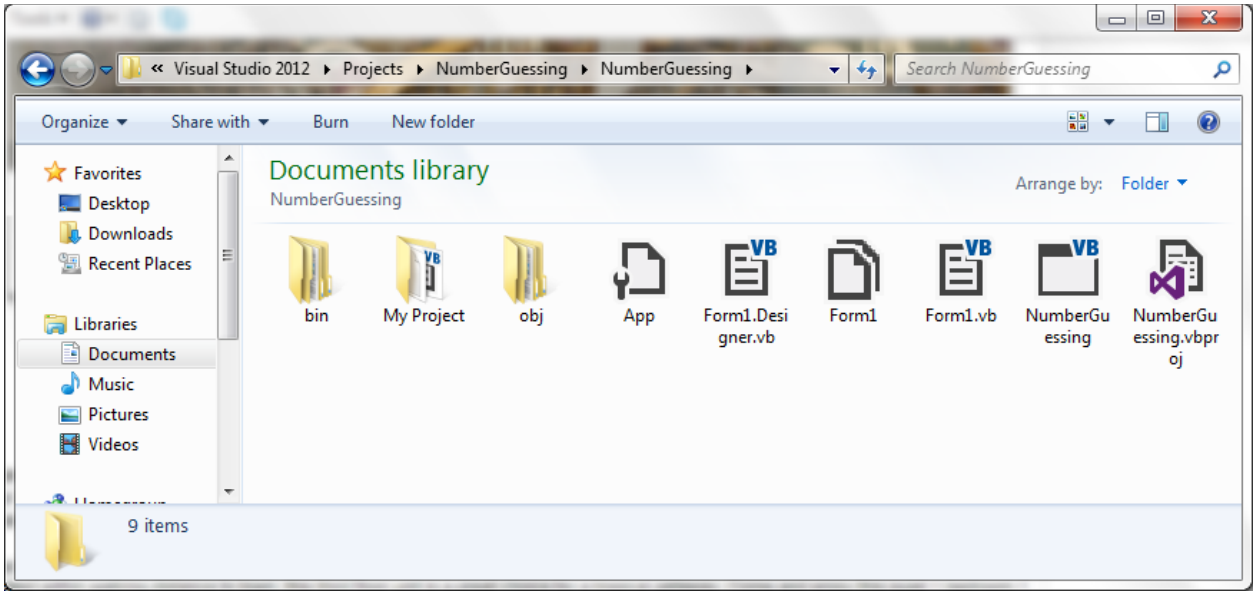


Opening a Project

To open a project, and its form file, you can use 'My Computer and navigate to where you saved your application files. From there double click the Solution file, shown below which starts the IDE with your solution, project and form files opened.



Alternatively, if you double click the folder shown, you will get to see the Project file as shown below. You can double click the Project file for the same effect. Another way to open your project files is to start Visual Studio and choose File, Open Project.



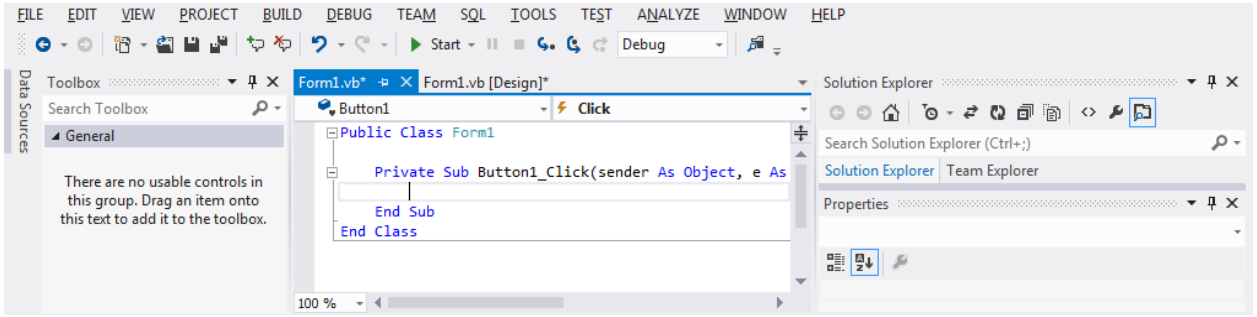
As a side note, the 'bin folder shown above will contain the executable file the compiler creates when you click the start button to test or run your code. Try double clicking that file from and notice that you can see the source code. This is what the end result of your application will look like to end-users.

Writing Code

Generally speaking, whenever you add a button to a form, the intention is that when the user clicks that button, some action is expected. These actions happen due to some code executing which was written and tested by the programmer.

In this case when the user clicks the 'Check My Guess button, code should execute that implements our flowchart that we worked on earlier.

To begin with, you need to know how to add code and make that code run when the user clicks the button at runtime. To add code, double clicking the button at **design** time. This will create a new view for you, where you don't see the form with the controls on it, but a page with text. This is the **Code View** of your form at design time. The snap shot below shows a sample.



You will notice that some code has already been inserted. The code view is a reflection of your form and its content of controls. The visual component of the IDE creates the needed code to display these controls and reflect their properties, without you having to type one line of code. This is to help you create applications faster. What you need to do now is write your code between the line starting with "Private Sub button1…. and the line "End Sub.
Any code you write between these two lines will **run from top to bottom** in that sequence whenever the user clicks the button 'Check My Guess, at runtime.
The two lines referenced above mark the **Click Event Handler**, for the **Button1** control. In CS 15, you will work with Event Handlers in more details.

Now let's write the code. The first line, as per the flowchart, states we need to generate a guess for the computer. We can, as programmers, come up with any number, say 4, and 'save it as the computer's guess, or CG. The question is, how and where do we 'save this number?

Using Variables

Any data that your code works with, whether the data is coming from the user via some input process, or the code (you) generates it, can be saved in memory; the computer's memory RAM. To save data into RAM, you will need to use a **variable.**

Hence, a variable is nothing but some RAM location that you (your code) 'designate, and declare the intention for use. As you know, data has a size. So when you declare the need to use RAM, you will need to decide what the size is. Furthermore, you will need to decide what kind of data will go in there: numbers, names, etc.

The process of declaration use of a variable in VB is done as shown in the example below:

```
Dim  ComputerGuess As Integer
```

Where the words **Dim**, **As** and **Integer** are all keywords (that's why they'll be color codes keywords in blue once you press enter after typing the line above). The above statement is called a declaration statement. This statement tells the compiler to find a memory location, name it ComputerGuess, and reserve a space the size of the variable. What size? The size is decided by the last keyword shown above, Integer. This is referred to as the **Data Type.** Integer is a very common data type, it requires 4 bytes (32 bits), and allows you to store whole numbers ranging from -2,147,483,648 through 2,147,483,647. There are other types that require different RAM sizes, and allow you to store different types of data. More examples should help in a better understanding of variables.

To declare another variable, say to save someone's name in it, you'd use:

Dim UserName As **String**

which causes VB to set aside a different amount of memory space than that needed for the ComputerGuess variable. The String data type can hold text made of alphanumeric characters that are enclosed in quotation marks. For example the following are all strings "0ne", "45", "1/12/1985", while the following are not: one (no quotation used), This String (no quotation used).

Other data types exist, but we'll focus on String and Integer data types. Note that data types are keywords of the language.

Variable Name
You can name a variable any name you like. It could be x or ProductPrice. The second one is preferred as it tells you what the data represents.

Formally, a variable name must start with a letter, can include letters and numbers, but spaces and some other special characters like " are not allowed. Needless to say, keywords are not valid variable names. Hence the following is legal:

```
Dim username as String
Dim EmpSalary1 as Decimal
Dim userAge as Integer
```

But the following is illegal:

```
Dim user name as String          (no space is allowed)
Dim 1EmpSalary as Integer        (starts with a number)
Dim user"Age as Integer          (contains illegal character)
Dim Integer As Integer           (can not use a keyword for a name)
```

In the code for the game, we'll need two variables, one to store the computer's guess, and another to store the user's guess, hence:

```
Dim ComputerGuess As Integer Dim UserGuess As Integer
```

Now, it's time to generate the computer's guess, and get the user's guess.

How To Use Variables
Variables are used for two major operations: to save (or store) values for future use, and to (when needed) 'get their value to show it or just to check it.

For example, the UserGuess variable will first be used to store what the user's guess is.
Later (in the code) we will use it to compare it to the computer's guess.
Storing a value in a variable is a process called **Assignment.** While 'getting what a variable contains is called **evaluation.** Let's see how assignments work first.

The Assignment Statement

After declaring a variable, you will most likely store a value in it. Let's **assign** a value to the ComputerGuess variable. This is how it looks like:

```
ComputerGuess = 4
```

The above code means we have decided that the computer's guess is the number 4. Later we will learn how to make it a randomly generated number. Note that during runtime, the user can not see your source code; hence they can not see this number.

The assignment statement performs the following in that order:

1. Evaluates the expression on the right hand side of the equal sign (RHS for short). In the case above, that would be the number 4. In other cases we may have something like:

   ```
   ComputerGuess = 4 * (2 + 3) - 13
   ```

   where the mathematical expression on the RHS will be evaluated first giving 7. In general, the RHS could be could be another variable, a control content or some mathematical formula.

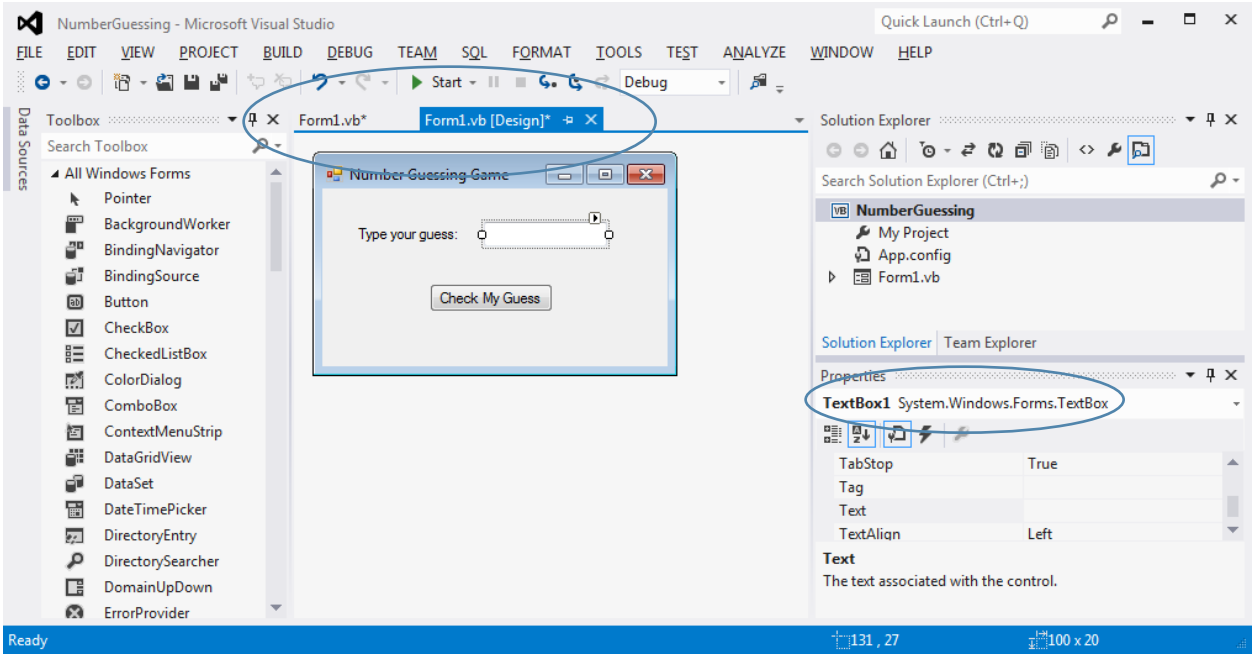
2. Dumps the content of the Left Hand Side (LHS), in this case ComputerGuess.

3. Makes ComputerGuess (the LHS) contain whatever the RHS evaluates to.

For this to work the LHS must be either a variable or a control.property (we'll talk about this later). It may not be an expression. Hence the following produces a syntax error: X * Y * 3 = ComputerGuess because the left hand side is an expression.

Moving on, now we need to get the user's guess, and store that number in the UserGuess variable. We know how to store values in variables, but we need to know, how do we get input from the user.

Recall that the user at runtime gets the form on the screen. They will type a number (representing their guess) in the textbox shown on the form. The number the user typed, was placed into the textbox on the form. To get that value in code we need to refer to the control, namely the textbox. And, we refer to controls using their name. All controls, including the form, have names, just as variables do. These names initially are generated by the compiler. As you place the control (during design time) on the form, the compiler generates code that (among other things) gives a name to the control. Finding out that name is easy. During design time, in Design view, click the control, then look at the Properties window, and look for the Name property. You'll notice for textbox controls, the name is 'TextBox, and then ends with a number that indicates the order of this textbox (1 for the first you add, 2 for the second, and so on). In our case, the textbox will most likely be called *TextBox1*. The screen shot below shows you a sample.



Note the highlighted tabs for Form1.VB (this is the code view) and Form1.Vb [Design] which is the design view of the form. Use these two tabs to switch between the code view and the design view. Remember both of these views are in design time (not runtime).

Going back to getting the user's guess, we know the user will type a guess into the TextBox1 control as shown above. The property of that textbox that tells us what the number (or any data) the user typed there, is the **Text** property. We know how to access this property using the design view's properties window. But how do we access it using the code view? In code view we will have to type code that follows this format:

ControlName**.**Property

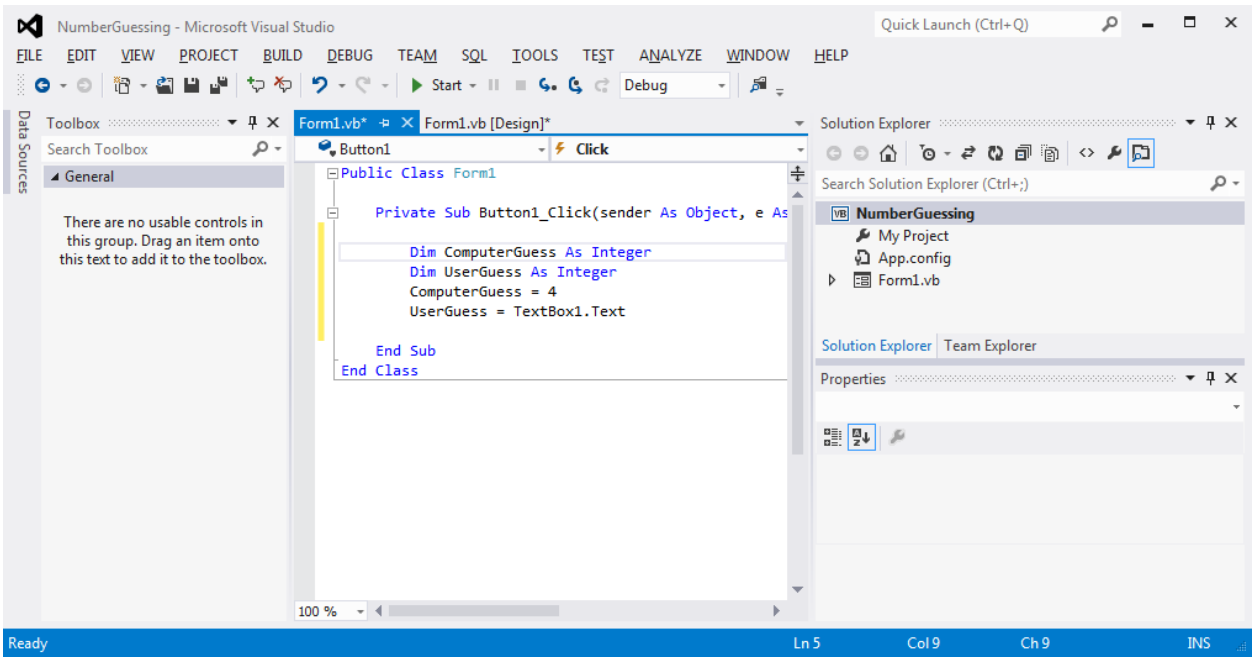
In our case, it should be:

TextBox1.Text

The above value should be stored into the UserGuess variable. A process we now know is called the assignment; here it is:

UserGuess = TextBox1.Text

The screen shot below shows the code so far. Using an incremental approach in building your code, you should test your application by clicking the Start button. In the runtime, type a number in the textbox and click the button. You expect the code below to run, which does not show any output. Hence if nothing happens, you are doing well so far. In the event you got an error message, read what the message states and try to find out how to fix it. Since codes may vary, so do the possible error messages, it is very difficult to go through a list of possible errors to fix. In the next section, we'll give you some background.



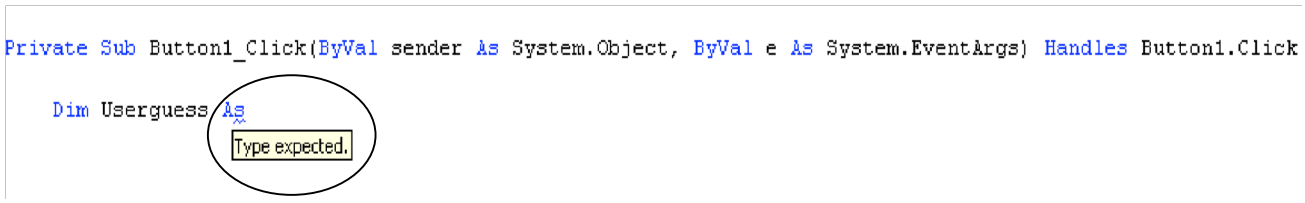
It is worthwhile to mention that that VB is not case sensitive, so you can write in all lowercase or uppercase or a mix of; the compiler formats the statements you type based on the keywords you use. For example, type dim (lowercase d), and finish the declaration statement. When you press enter, the compiler will capitalize the d in dim, the 'a in **As** and the data type's first letter in addition to color-coding the keywords.

Fixing Errors

In programming in general, there are 2 types of errors. Syntax, and Logical. Syntax error occur when the compiler can not translate your code into machine language because it is not conforming to the rules of the language. For example, the following produces a syntax error:

Dim UserGuess As

since it is missing the data type. In this case, the compiler will underline the 'As keyword with blue squiggly line. When the mouse is placed on the line, a tooltip window appears indicating the source of the error, as shown below:



As you can see, this type of syntax error is easy to fix, since: a. it is easily located as the compiler highlights the source of the error, and b. the type of error is known since the tooltip details what it is, hence fixing it is a matter of following syntax rules.

Other types of errors related to syntax may occur as the code runs, hence the name runtime-error. Runtime errors are outside our scope, but are discussed in CS 15.

Logical errors occur when the code runs without any error messages, but it displays unexpected output. For example, it may show the user has won, when they typed the wrong guess. These errors may be due to incorrect translation from flowchart to code, or an error in the logic of the flowchart itself. As you can guess, these types of errors are harder to fix since identify the source of error may be difficult. The VB IDE provides programmer with many tools to aid in locating these errors, such as the use of Breakpoints and stepping into code. This topic, however, is not within our scope.

Going back to our game, we now need to implement the 'Decision boxes of the flowchart, which is done through using the **If** statement.

If Statement Syntax

But first, what is **Syntax**? In any programming language, you will be writing sentences, that we call statements. Each statement you write must comply with the language rules, so that the compiler can translate that statement to machine language. How do you know what the rules are? The syntax tells you. Each statement you use in any language has a syntax rule that tells you how to write the statement and what are its possible forms.

Below is the official syntax of the If statement. When syntax is presented, first the form of the statement and its variations is shown. The minimum required components are in plain text, while optional components are enclosed in square brackets. Then a logical explanation is presented, and finally, non-keywords (shown in the syntax in italics) are explained. Here it is.

If *condition* Then
  statements
  [ElseIf elseifcondition Then
    elseifstatements]
  [Else
     elsestatements]
End If

The If statement conditionally executes a group of statements, depending on the value of condition.

Note everything between square bracket [ ] is optional.
Items in italics mean that you have to replace them with other VB expressions.
Statements could be one statement or more, that depends on the logic of your code. If you have another action to do if the first condition fails, you may use the Else clause and execute other

actions as specified by else if statements. You may also have multiple conditions, and they must be specified by an Else clause for each condition. This concludes the syntax of the If statement.

If the above format sounds confusing (which it probably is at this stage), then focus on the examples discussed below. Working through examples, will help in a better understanding.

```
If ComputerGuess > 0 Then
```

In the above incomplete If statement, the part between 'If and 'Then is called the condition. Each condition you specify must evaluate to True or False. How do you write that condition? Here are some examples. Generally speaking a condition is made of two objects separated by a relational operator. Relational operators include =, <. <=, >, >= and <>. For example:

TextBox1.text = "20"

SomeVariable < 0

A condition may also be connected by a logical operator. For example if you want to test whether a variable is not equal to zero, you may use any of:

ComputerGuess < > 0 Or:

Not (ComputerGuess = 0)

If the condition of an 'If statement is True, then one or more **statements** following 'If...Then, are executed as in the following example, where if the ComputerGuess is > 0, then the textbox named TextBox1 will display the word 'hello:

```
If ComputerGuess > 0 Then

    TextBox1.Text = "hello"
```

Note in the assignment statement above, we are assigning the **Text** property of the textbox to the string value of hello. This is done by using quotes before and after the word (or words) to be placed into the textbox. If you do not use the quotes, the compiler will think that this word 'hello is a variable. When it can not find a declaration for it, the compiler will blue underline it indicating that it is an error (syntax error).
Continuing, if the condition of the IF, is False, then you may have other statements execute if you place them following the Else keyword, or if you have more conditions to test, you can use the 'ElseIf keyword, as in:

```
If ComputerGuess > 0 Then
        Textbox1.Text = "hello"
Else
        Textbox1.Text = "good bye"
EndIf
```

In the above complete example of the If statement, if the computer guess is less than or equal to 0, the user will see the words good bye in the textbox. But what if you want to display one greeting if the guess was 0 and another greeting if it was negative? In that case you'll need further testing prior to the Else portion above, as in:

```
If ComputerGuess > 0 Then
        Textbox1.Text = "hello"
ElseIf ComputerGuess = 0 Then
        Textbox1.Text = "Hi"
Else
       Textbox1.Text = "good bye"
EndIf
```

And you can add further testing as long as it is done before the else statement and after the Then statement of the top 'If as in the following incomplete example:

```
If   condition1  Then
        action1
ElseIf condition2 Then
       action2
ElseIf condition3 Then
      action3
 Else
        action4
EndIf
```

In the above, text in italics means that this is NOT VB code, but some form of 'pseudo code that will be translated later to VB. If condition1 is true, then action1 is executed, and the rest of the conditions are not evaluated, hence none of the actions (from 2 to 4) will be executed. If however, condition1 is False, only then will condition2 be tested. And if it is True, then action2 will be executed, and the rest of the 'If statement is not attempted. Hence, action4 will happen only if conditions 1 through 3 all fail.

Note that depending on the logic, you may not need an **Else** portion, is which case you can skip it as in the following:

```
If   condition1  Then
        action1
ElseIf condition2 Then
      action2
ElseIf condition3 Then
     action3
EndIf
```

Now back to our problem. We are at the stage where we need to test if the user's guess (UserGuess) matches the computer's guess (ComputerGuess). This is translated to:

```
If ComputerGuess = UserGuess Then
```

Note in the above, the condition is comparing (using the equal sign) whether the two sides are the same. This equal sign represents the test for equality, and NOT the assignment. Hence the above does not assign UserGuess to ComputerGuess, just because we used the equal sign.

Now we need to decide what to do if both sides are equal and later what to do if they are not. According to our logic, here is the code needed:

```
If ComputerGuess = UserGuess Then
        'user won, we must display an output message
  Else
        'user lost, we must display an output message
  End If
```

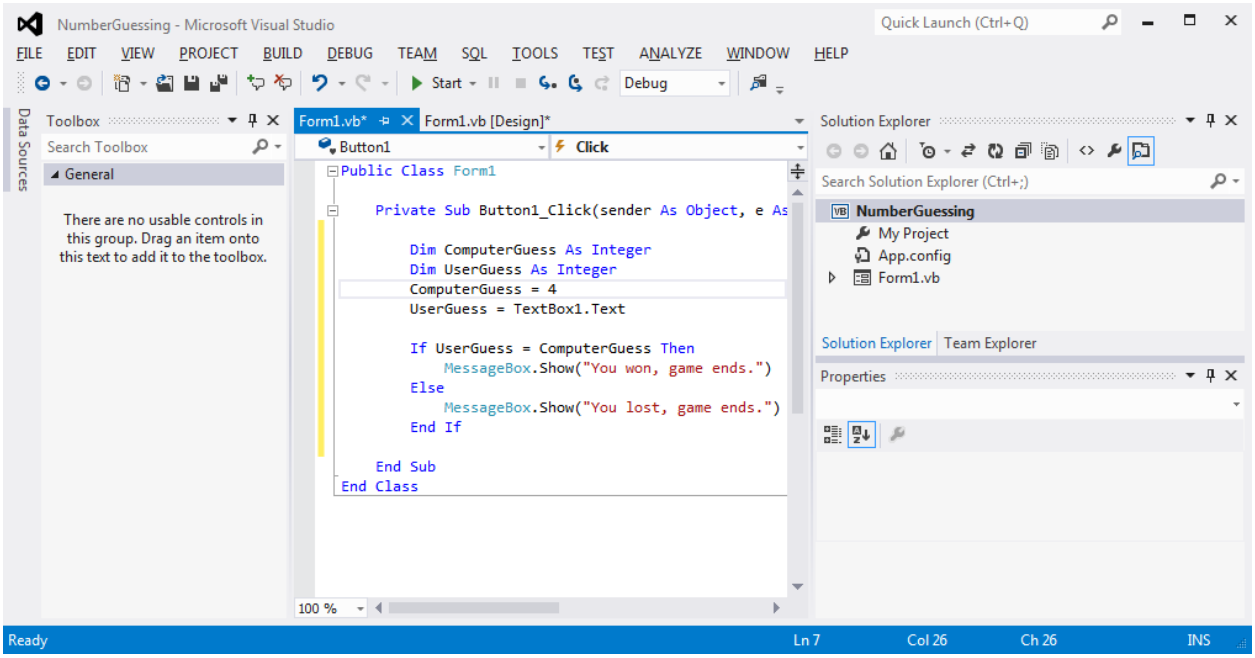
Now we need to change those lines in italics to VB statements. You may use the textbox to show the user the output message, as in:

```
If ComputerGuess = UserGuess Then
        Textbox1.Text = "You won. Game Ends"
Else
        Textbox1.Text = "You lost. Game Ends"
End If
```

You may also use pop up windows to show output messages using the following statement:

```
Messagebox.show("You won. Game Ends")
```

where you'd type your output message between quotes as shown above. The complete final code looks like this:



Note in the above that information of type ‘String is color coded by the compiler in maroon.

As promised, you may make the game more complicated by using a randomly generated number instead of the fixed typed number. You may also allow the user a number of chances to guess the same number.

To create a random number for the computer’s guess use the following:

```
ComputerGuess = Rnd( ) *  10
```

In which the ‘Rnd causes the compiler to generates a random number (not even you the programmer know what it is) between 0 and 1. Since the game assume the number to be guessed is between 0 and 10, and then multiply by 10 to get a whole number between 0 and 10.