Turning pseudocode (or flowchart) into a program in a "real" programming language, Visual Basic.NET.

This semester, you've learned to create and desk-check algorithms, using pseudocode. And you've learned that those same algorithms can be represented as Visual Logic flowcharts. And, furthermore, you've learned that when the algorithm is correct, and correctly represented as a Visual Logic flowchart, you can execute the flowchart, feed it input, and receive output.

You have learned about the "hard part" of computer programming.

This week and next, we'll look at the "easy part", actually writing a program to carry out your algorithm in a real-world programming language, specifically, Visual Basic.NET. You should do these exercises in the lab -- there's no reason to install Visual Basic.NET at home -- at least, not for CSc 10. We're only going to do a couple of exercises.

Visual Basic.NET is one of many programming languages in existence today. It happens to be one of the more popular programming languages. It is particularly good for building very effective and attractive Graphical User Interfaces (GUI). However, we're not going to deal with that in CSc 10. Save that for a nice class in Visual Basic.NET.

We're going to create a very simple, primitive GUI, and then we'll write a program (implement the logic of an algorithm in the syntax of the Visual Basic programming language) and run it from that simple GUI. When you finish these exercises, you'll know how to write computer programs. And I hope you'll see that once you have worked out the algorithm to solve your problem, the translation into the syntax of Visual Basic is really not particularly difficult.

The approach we’ll be using in these exercises will be to create a form which will contain exactly one control (any doo-dads you put on a form are called “controls”), namely, a button to cause your program code to execute. The only thing that will change from one program to the next is the code (programming language statements) that gets executed when the user clicks the button.
Here’s an example of that all-purpose form.

![Visual Basic.NET Form with Execute Button](image)

Figure 1: Visual Basic.NET Form with Execute Button

The button (labeled “Execute”) is the one the user will click to cause the program to run. (Actually, there will be two steps — the first step is to run the whole project, then the second step is to click on the “Execute” button. You'll see how to do those steps in a moment.)

The output that your program produces will be displayed in pop-up message boxes. Here’s an example of the message box which would pop up after you clicked the Execute button of a program which displayed “Hello, World!” in a message box.

![Output message box](image)

Figure 2: Output message box

It is not difficult to obtain printed (hard-copy) output from your programs, but we're not going to bother. (If you really want to know how, I'll include a couple of pages at the end of this handout to show you how to do it.)

A: Exercise One: Your first Visual Basic.NET programs
Part I: Setting up a Visual Basic.NET project

Okay, it’s time to get to a computer in one of our computer labs. We’re going to write a program! (I can see that you are excited beyond all belief.)

1) Start Visual Basic.NET. Most likely you’ll do that by clicking the Windows Start button, selecting Programs, and finding something that looks like “Microsoft Visual Studio.NET” or “Microsoft Visual Basic.NET”. If “Microsoft Visual Studio.NET” is there, you might have to make one more selection to choose “Visual Studio.NET” from the options offered. (Alternatively, there might be a desktop shortcut icon to run VisualStudio.NET or VisualBasic.NET.)

or

2a) Depending on just how your system is configured, the previous step may take you to a screen called the “Start Page”. If that is the case, choose “New Project”. (There are two ways to do that on this screen -- there is a “New Project” button on the “Start Page”, or you can use the File Menu, select New, then Project.) Either of these approaches will show a New Project screen. The default selection is usually just what we want, with “Visual Basic Projects” selected on the left and “Windows Application” selected on the right. (If for some reason, yours has not already chosen those selections, select them -- we’ll use those selections throughout the text.)
2b) If your screen looks different, it probably doesn’t matter. You still need to choose “New Project” (from the File Menu, select New, then Project). The default selection is usually just what we want, with “Visual Basic Projects” selected on the left and “Windows Application” selected on the right. (If for some reason, yours has not already chosen those selections as the default, select them -- we’ll use those selections throughout the text.)

3) Now you have to select the “Location:” (hard disk or floppy disk, which folder) where you want to store this project. (Every program will be a “project”.) If you are working in a University lab, there are restrictions on where you can store the project. Ask your instructor if you do not know where you should put your project. You will also be asked for a project name. The default project names provided by Visual Basic.NET would work, but I suggest you rename the project to something that will help you remember what you were working on. Before you have finished this textbook, you will have **many** different projects.

Here’s what the screen might look like when selecting “Location:” and “Name:”. For this example, I’ve chosen to store the project in the “temp” folder on the “C:” hard drive. And I’ve chosen to name the project “Introduction”.

![New Project screen](image)

Figure 3: New Project screen

4) Once you have a new project ready, your screen should look something like this:
5) To create the “Execute” button, you need to choose from the “Toolbox”. If this Toolbox is not visible on the left side of your window, there’s probably a place with that name, and you can hold your mouse over it to view the Toolbox. Or go to the View menu and click on “Toolbox”.

On the Toolbox, click on the “Button” icon (and release the mouse button).
6) Move the mouse to Form1. Create a small rectangle in the lower right-hand corner of the form. (You do this by moving the mouse to the location where you want the upper left corner of your button, depressing the mouse button, and dragging to the location you want for the lower right corner of your button. Then release the mouse button.)
If you don’t know how to depress a mouse button, just tell it that you’re going to replace the mouse with a more modern pointing device. That always makes my mouse very depressed.

7) Look at the “Properties window”. (It should be in the lower right corner. If it is not displayed, you can find it using the View menu.) The current “Text” for your button is “Button1”. Change that to “Execute”. The current “Name” for your button is also “Button1”. Change that to “btnMain”. (The “Name” property is quite a ways down in the list, under a section named “Design”.)

(Note: It is a convention among Windows programmers to begin the name of each control with a few letters that tell what kind of control it is. Since this is a “Button”, use “btn” as the first three letters. It is also helpful to name controls so that you and others can more easily understand their purpose. I’ve chosen the name “btnMain”, since, for all programs in this book, this will be the “main” or controlling routine. To put it another way, when the user clicks the Execute button, they are running the main program.)
Here’s what things should look like after you’ve made these changes.

![Figure 5: Form after changing button’s text and name.](image)

Notice that on the right side of the window, it says that the name of your current “Solution” is “Introduction”. Visual Basic.NET likes to call these projects “Solutions”. And when I created this one, as you recall, I named it “Introduction”. Notice that the button has the text “Execute” on it. (If you made your button too small and the whole word doesn’t fit on one line, you can change the size of the button by clicking on it, then selecting one of the corners and dragging it out to enlarge the box.) In the Properties window, you see that the (Name) property is set to “btnMain”.

8) Now it is time to write a program! (Nothing you’ve done up to now is part of writing a program — it is all setting up to write a program.) The program’s purpose is to display the message “Hello, World!” The appropriate statement in Visual Basic.NET is:

```vbnet
MsgBox ("Hello, World!")
```

(This statement will cause a message box to pop up on the screen, displaying the message “Hello, World!” We say that “Hello, World!” is the argument to the
MsgBox function. Think of it in the same way you might think of a math function with an argument -- \( \cos(x) \) -- \( x \) is the argument to the \( \cosine \) function.

Ah, but the trick is figuring out where to put this statement. Well, this is really amazingly simple. Just “double-click” on the Execute button. Voila! A place opens up for you to enter your (one-line) program. The screen should look like this:

![Figure 6: The Code Window](image)

The cursor will be on the blank line following the declaration “Private Sub btnMain_Click(…).”. For the first three-fourths of this textbook, you may completely ignore everything else on the window except “Private Sub btnMain_Click” and “End Sub”. Your programs will go in between those two statements.
Furthermore, the weird stuff inside and just after the parentheses after “Private Sub btnMain_Click” should be ignored. In fact, throughout these exercises, when I want to show that line, I’ll always leave out that stuff in the parentheses. (It has to be there in order for the system to work, so don’t remove it. Just ignore it!)

The word “Sub” (and the “End Sub” at the bottom) tell you that this is the “subroutine” which will be executed when btnMain button is “clicked”. So this is the place to put programming statements that you want to have run when the user “clicks” the btnMain button.

9) In between the beginning (Private Sub) and the end (End Sub), write your program. What could be simpler?! (Of course, your programs will get increasingly complicated as the book progresses, but this set-up routine will work throughout.)

Note: VisualBasic.NET tries to offer all sorts of help as you type in stuff. It guesses what you had in mind and offers you menus to save you some keystrokes. I will completely ignore that throughout this book! You may make use of it if you wish.

10) Here’s what the code section should look like once you’ve typed in the program.

Private Sub btnMain_Click ( )

    MsgBox (“Hello, World!”)

End Sub

Note: I left a blank line before and after the MsgBox statement. That was just to make it easier to read. Blank lines have no effect on the program. And, as I said a couple of paragraphs ago, there is a bunch of stuff in between the parentheses after Private Sub btnMain_Click, and even a little bit after the parentheses, still on that same line. We’re ignoring all of that for at least the next couple hundred pages or so.

Note: Also, Visual Basic.NET (unlike some, but not all, other programming languages) will allow you to enter commands in upper or lower case -- it treats upper and lower case as the same characters. So you could have typed “msgbox”. When you move to the next line, Visual Basic.NET will attempt to make things look pretty! (The “case-insensitive” nature of the language does not apply to what’s inside the quotes -- that is going to be displayed in exactly the case you enter it.)
And note yet one more thing: Visual Basic.NET, like most Microsoft products, tries to think for you. In this case, if you forget to include the parentheses around “Hello, World!”, it will put them there for you.

11) Now it’s time to run the program to see if you did everything correctly up to here. We’re going to ask Visual Basic.NET to “build” your project. That is, we want Visual Basic.NET to attempt to translate your code into something the computer can understand, an “executable” program (i.e., machine language). If you’ve done everything correctly up to here, that should work just fine.

To build your project, select the Build menu and choose “Build Solution”. You should get a message at the bottom of the window saying something like:

   Build: 1 succeeded, 0 failed, 0 skipped

(If you got a different message, indicating some sort of error, see if you can figure out what you did wrong and fix it. If not, get help.)

12) Assuming a successful build, it is time to run the program. This is a two-step operation. First, you have to run the Project. To do that, you press the F5 key on your keyboard. (If you’d prefer another approach, there is an icon on the Tool Bar that can be used. It looks like a little blue arrowhead (or triangle) facing right. It is just about at the middle of the Tool Bar, which is just below the Menu Bar. Clicking that icon is equivalent to pressing the F5 function key.)
13) At this point, you should be looking at the form (which looks nicer in this mode than in the design mode you’ve been working in). And the form is looking at you. It looks like this:

![Figure 7: A “running” form.](image)

It is sitting there waiting for you to do something. So, do something! Click once on the Execute button.
14) If all went well, you now have “Hello, World!” displayed in a pop-up message box, like this:

![Message box](image)

Figure 8: Message box

15) You now must click on the “OK” button on your message box -- Visual Basic.NET won’t let you do much else until you’ve closed that box. (Pressing the Enter key has the same effect.)

16) The program code that you wrote has now completed. However, the project is still running. Sometimes it’s hard to tell. You can find out, though, by returning to the code window and attempting to make a change to your program. Visual Basic.NET says “No!”, or something equally unpleasant. (Actually try to change your code, and look at the very bottom of the window. There’s a message that says, “Cannot currently modify this text in the editor. It is read-only.” That means “No!”.)

When you closed the message box, you finished off everything your program wanted done. But Visual Basic.NET is still running the project -- that’s why the form is still displayed in the non-design mode. You can press the “Execute” button again and it will respond.

However, what we really want to do is to end the running of the project. The easiest way to do that is to close the form. Just click on the little “x” in the upper right-hand corner of the form. (Don’t close Visual Basic.NET -- just your little gray form!)

You’re going to have to remember to do that every time you run a program.
Part II: A more interesting program

Considering where we are in the semester, you already have the skills necessary to create a much more complex and interesting program than one that just displays, "Hello, World!" Let's do it.

Here's an algorithm we worked with in one of the loop activities. I'll show it both as a flowchart and as pseudocode. First, the flowchart:
Now as pseudocode:

\[
\begin{align*}
\text{Max} & \leftarrow 3 \quad // \text{Named constant} \\
\text{Total} & \leftarrow 0 \\
\text{FOR} \ Game = 1 \ \text{to} \ \text{Max} \\
& \quad \text{INPUT} \ \text{Score} \\
& \quad \quad \text{WHILE} \ \text{Score} < 0 \ \text{OR} \ \text{Score} > 300 \\
& \quad \quad \quad \text{INPUT} \ \text{Score} \\
& \quad \quad \text{END \ WHILE} \\
& \quad \text{Total} \leftarrow \text{Total} + \text{Score} \\
\text{ENDFOR} \\
\text{DISPLAY} \ \frac{\text{Total}}{\text{Max}}
\end{align*}
\]

So, let's create a Visual Basic program for this algorithm!

Start a new project, or remove the code in the present one (that is, remove everything between Private Sub btnMain_Click and End Sub).

**Converting from pseudocode to Visual Basic.NET code:**

1) There is a way to tell Visual Basic that you want one of your values to be a named constant.

The statement:

\[
\text{Const Max} = 3
\]

will set up a named constant in your Visual Basic program. The constant's name is Max. It's value is 3. (By making this a named constant, we are telling Visual Basic that we do not plan to change the value of Max while the program is running -- and, in fact, Visual Basic won't let us change it while the program is running - -that's what "constant" means!)

**Put that statement as the first statement in your program** (i.e., the first statement between Private Sub btnMain_Click and End Sub).

2) It is necessary to tell the Visual Basic.NET program what kind of data you are going to store in your variables. We call that "declaring variables". In Visual Basic, we do that with \textit{dim} statements.

The statement:

\[
\text{dim Total as Double}
\]

tells the Visual Basic program that the variable \textit{Total} will be storing real numbers. (The total will actually be a whole number, since bowling scores are whole numbers, but since we eventually want to end up with an average, and that might not be a whole number, we'll store the total as a real number instead of as an integer.)
Put that statement in as your second statement.

Then add declarations for the other variables in the algorithm, Game and Score. Declare both of these as whole numbers (in Visual Basic, that's called an integer).

NOTE: Visual Basic (unlike some other programming languages), is case-insensitive. That is, it doesn't matter if you use upper or lower case characters -- they will be treated as the same. (That doesn't apply if you are including a literal string, something in double-quotes, but otherwise, case does not matter.)

NOTE: If you forget to declare (dim) a variable which you use in your program, you'll get an error message when you try to compile the program. That is, you will have made a syntax error, and error in the grammar of Visual Basic.

3) Assignment statements in Visual Basic use an equals sign, just like we did in Visual Net (but unlike what we did in pseudocode, where we use those cute little left-facing arrows).

For your next program statement, assign the value 0 to the variable Total.

4) The FOR statement in Visual Basic looks exactly like it does in pseudocode.

Put the For statement into your program.

(When you press the <Enter> key after typing in the FOR statement, you'll note that the "ending" for the FOR loop is different from our pseudocode. Visual Basic was nice enough to put in the "ending" statement, which, in Visual Basic, is the word Next. That is, the word Next replaces the pseudocode word ENDFOR.)

5) Now it's time for INPUT. This is where Visual Basic is really quite similar to the Visual Logic flowchart tool, but nicer. It provides this very attractive pop-up window for obtaining input -- it's called an InputBox. Here's the statement you would use to input a value for Score, with a nice prompt to the user.

Score = InputBox("Please enter bowling score number " & Game)

(Notice that we're using the value of the FOR loop variable, Game, to make a nice prompt that tells the user which score they are entering.)

Type that statement in as the next statement in your program.
6) Now our pseudocode calls for a WHILE loop (nested within the FOR loop). In Visual Basic, the syntax for a WHILE loop is pretty similar to our pseudocode. However, they use the words *Do While* instead of just *WHILE*, and they end the loop with *Loop* instead of *ENDWHILE*.

**Type in the Do While loop statement:**

```
Do  While Score < 0 Or Score > 300
```

You should notice that after you press the <Enter> key, Visual Basic, again, helps us out by putting in the ending statement (*Loop*), and automatically tabs in for the first statement inside the loop body. (Also notice that, unlike Visual Logic, we don't need parentheses in this particular conditional statement -- as you read when we talked about *AND* and *OR* in the earlier in-lab activities, some languages require parentheses for precedence in that statement, others don't. In Visual Basic, the *OR* has lower precedence than the comparison operators, so we don't need parentheses to change the order.)

6) Now, within the nested WHILE loop, do another input -- use the InputBox. Use a prompt that tells the user they have made a mistake by entering an invalid bowling score, and then ask them to re-enter.

7) Now finish the code in the FOR loop body -- that is, after the WHILE loop ends but before the FOR loop ends, accumulate the Total.

```
Total = Total + Score
```

8) Only one step left in the algorithm. After the end of the FOR loop, we need to display the average. You already know how to do that -- it's just like the "Hello, World!" problem. Use a MsgBox. Include a nice label and the calculated value. (We don't have to worry about division by zero -- the divisor is a constant with the value 3 -- it can't be zero.)

```
MsgBox ("The average of your " & Max & " scores is " & Total/Max)
```

9) The moment of truth!!

Okay, time to see if you typed everything correctly. You need to compile the program (That is, you need to ask Visual Basic to check everything you typed, and if it finds everything syntactically correct, to translate it into a form that can be executed.)

You already saw how to do this in the "setting up" part of the exercise.

**Build/Build Solution**
If you typed everything correctly (your "syntax" was correct), you should get the message that says, "Build: 1 succeeded, 0 failed, 0 skipped"

If you don't get that message, you typed something wrong. Go back and try to find it. (Visual Basic will try to lead you to it and give you some moderately helpful message about it.) Fix any errors, and Build again.

10) Once you get a successful build, it's time to run the program. And you already know how to do that, too. Recall that it is a two-step operation. Press <F5> to start the project, then click the Execute button to run your program.

When your program runs, provide reasonable inputs. Be sure to test that the data validation loop is working -- try several invalid scores.

Assuming all went well, you've now learned how to code a moderately sophisticated program. Of course, the hard part was already done! You had a correct algorithm to begin with. And that's what this course is all about!!!! Learning to write logically correct algorithms to solve problems is the hard part! Translating into the program language, while probably seeming rather foreign and difficult right now, eventually becomes quite easy. (But coming up with algorithms never becomes easy, because the better you get, the harder the problems you try to solve!)

(After you've run your program several times with different date, don't forget to end the process by closing the form. Don't close Visual Basic.NET, though. We're not quite done.)

11) COMMENTS
We have written this entire program without putting in any comments. That is almost never acceptable!

In Visual Basic, you put comments in your program using a single apostrophe as the lead character. Anything that follows the apostrophe (you might think of that as a single-quote… same key) on a single line is a comment. Be sure you use the correct quote key, though. There are two keys on a computer keyboard that sort of look like the single-quote. You want the one on the same key as the double-quote.

At the top of your program, just after Private Sub btnMain_Click, put in a comment saying what the program does (it computes bowling averages), who wrote it (you), and when you wrote it (today's date). In larger programs, particularly those with subroutines, we'd need some more comments, but for this program, those three are probably enough.

TO TURN IN:
Print your program. This is quite easy to do. First, click somewhere inside your program code, so that the proper window is selected. Then just click File/Print.
B: One more Visual Basic.NET program

We're only going to write one more program. You should already see that there is really not all that much difference between pseudocode and Visual Basic.NET. The goal of these two exercises is just to show you the technical steps to create and run the program.

This program will also have loops, and maybe an IF or two. (IF statement in Visual Basic look exactly like IF statements in our pseudocode.)

This program will use an array. The algorithm is one you've dealt with previously (in activity K). It is the one that finds the largest number in a bunch of inputs, then displays the difference between each input and the largest value.

Here is the algorithm, exactly as it was in Activity K.

```
INPUT HowMany
Declare an array named Numbers with HowMany as the upper bound
// Get the first element, outside the loop, and also make it Largest
INPUT Numbers(1);
Largest ← Number(1);
// Fill the array for the rest of the values, and find the highest as we're doing it
FOR i = 2 to HowMany
    INPUT Numbers(i)
    IF Numbers(i) > Largest THEN
        Largest ← Numbers(i)
    ENDIF
ENDFOR
DISPLAY Largest
// Now display the differences.
FOR i = 1 to HowMany
    DISPLAY Largest – Numbers(i)
ENDFOR
```

There's only one thing in this algorithm that you don't know how to do in Visual Basic, and that's how to declare an array. Here is the exact declaration statement you'll need.

```
Dim Numbers(HowMany) as Double
```

Notice that, in the algorithm, we've already input the value for HowMany. That's critical -- when we get to the declaration of the Numbers array, we must have a value for the size of the array. The Dim statement says that Numbers is the name of an array of real numbers (*Double* means *real number*) and that the highest index (subscript) will be the value of HowMany.
Technically, Visual Basic.NET arrays start with a subscript of 0, but we can safely (and logically) ignore that -- the value placed inside the parentheses when you declare the array is the value you want as the highest index. So if you say something like:

```
Dim SomeArray(5) as Double
```

you have told Visual Basic that SomeArray will hold real numbers and that there will be array slots 0, 1, 2, 3, 4, and 5. I recommend at this point that you just **ignore the 0 slot!** The loop in our algorithm does, in fact, ignore that slot.

Next, declare the rest of your variables (**Largest** must be a real number, but \( i \), the loop index which is used as the subscript, must, of course, be a whole number), and then implement the exact logic of the pseudocode. There's nothing new in there -- you did all those things (except the IF) in the previous exercise.

Be sure to use nice prompts to the user and good output labels. (In your input prompt for each number, tell the user which number it is -- just like we did in the previous example. And the final loop output, the differences, should say something like:

```
The difference between the largest number (7.5) and 3.5 is 4.0.
```

Of course, that assumes your largest was 7.5, and that one of your input values was 3.5.

**NOTE:** That message box for the differences is a pretty long line. That's okay, as far as Visual Basic is concerned, but when you print your program, it's going to wrap around and look sort of ugly. Here's a little Visual Basic trick -- it is not one of the important issues, but might come in handy from time to time.

You can split up a single statement into multiple lines by using the `continuation` character, and underscore. This has no effect on the output produced, only on the way the actual Visual Basic statement would look when printed.

Don't ever split up a statement in the middle of a quoted string. Here's an example of the use of the `continuation` character.

```
MsgBox ("Here is a long statement. " & someVariable_
    & " and here’s yet some more " & someOtherVariable_
    & " and let’s quit here.")
```

Because of the space followed by underscore at the end of the first two lines, this becomes a single statement. Be a little careful if you decide to use this -- don’t split a statement in the middle of a quoted string. For example, this would give you a problem.

```
MsgBox ("Here is a bad example. It is an incorrect_
    continuation of a quoted string.")
```

You could correct this as follows:

```
MsgBox ("Here is a bad example. It is an incorrect "_
    & " continuation of a quoted string.")
```
Test your program carefully with several different sets of numbers. For example, try it with the largest number as the first number -- then again with the largest as the last number -- and yet again with largest somewhere in the middle.

Just for fun, try it with zero elements -- that is, answer the first input question (which asks how many numbers there will be) with 0 and see what happens.

OUCH! That didn't work. There's a statement in the algorithm that says to set Numbers(1) to the input value. But if your answer to the input for HowMany was zero, then there is no array element with subscript 1.

Let's fix that!

Modify your program so that it forces the user to enter a value larger than 0 for HowMany. You know how to do that -- use a data validation loop.

Then test your program again -- make sure everything else still works the way it did before, and then see what happens if you try to enter 0 as the value for HowMany. That should get rejected and the user should be forced to enter a new value. Try -1. That should also be rejected. Then try 1. That should work, and the program should work just fine.

TO TURN IN:

Put appropriate comments into your program (purpose of the program, name, date), then print the program and turn that in.
Supplement: How to get "hard-copy" of Visual Basic output.

This is not part of your in-lab activities. I'm just providing this information in case some of you wish to go on with your programming in Visual Basic and want to get "printouts" of your output, rather than having it pop up on the screen in message boxes.

To get printed output from your program, you will have to insert statements into your program to instruct Visual Basic.NET to write the output to a text file on your disk. Then you can simply open that file using Windows Notepad or some other text processor and click the Print option.

Let's return to the "Hello, world!" program you wrote at the beginning of the Visual Basic.NET exercises. This time, we're going to make the program display "Hello, world!" in a message box on the screen, just like it did before. But we're also going to make the program print something to a text file – we'll have it print "Hello, world!" to the file, too. (We could have it print anything we want to the file, but you will usually be printing the same results that you displayed on the screen – that way you can see the results on the screen, and you can also write the results to the file and then print that file to hand in with your assignment.)

Assume you are looking at the screen where you created the "Hello, World!" program. You would be looking at a screen that looks something like this (yours may be slightly different, but the components shown here should be present in yours):

```vbnet
Public Class Form1
Inherits System.Windows.Forms.Form

    Private Sub btnMain_Click ( )

        MsgBox ("Hello, World!")

    End Sub

End Class
```

You do not have to understand what the lines above and below your Private Sub btnMain_Click() are all about.

However, you are going to have to add one statement above Public Class Form1.

1) Above Public Class Form1, type

```vbnet
Imports System.IO
```
2) Your code should now look like this:

```vbnet
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private Sub btnMain_Click()
        MsgBox("Hello, World!")
    End Sub

End Class
```

You have now told Visual Basic.NET that your program is going to use some stuff that is part of a package of additional routines provided by the wonderful people at Microsoft. Specifically, that's where the routines that handle file output are located. If you forget to put in this "Imports" statement, you'll get an error message when you add the commands to write to the file, so don't forget it.

3) Now it's time to add the commands that will write the desired output to the file.
This will require two statements to prepare the file, and then the actual output statements to write to the file. You'll "prepare" the file at the beginning of your code, that is, at the beginning of `Sub btnMain_Click()`.

Add these two lines. **Memorize these lines** – you'll use exactly the same lines (except perhaps for the actual name and folder for the file) any time you want printed output.

```vbnet
Dim writer As StreamWriter
writer = File.CreateText("c:\temp\VBOutput.txt")
```

This tells Visual Basic.NET that you are going to write your output to a file named "VBOutput.txt", which should be placed in a folder named "temp" on the C: drive. You don't need to use that file name, nor that folder, nor even that disk drive. That's entirely up to you – put the file wherever you want it, name it whatever you want to name it. But remember what it was named and where you put it! You'll need to find it after you've run the program, since that is the file you're going to print to the printer, using Notepad.

4) Once you have Imported the System.IO package and you've declared your StreamWriter (the variable name writer was arbitrary, like any variable name), and you've connected your StreamWriter to a particular path and file name, you are ready to actually write output to that file.
There are two commands you can use to print to the file. One is `writer.Write` and the other is `writer.WriteLine`. You use these in exactly the same way you use `MsgBox`. You put parentheses after the command, and inside the parentheses, you put whatever you want to print. Here's how we'd print "Hello, World!" to the file.

After your `MsgBox("Hello, World!")` statement, add the following statement:

```plaintext
writer.WriteLine("Hello, World!")
```

5) One more statement is needed – you need to tell Visual Basic.NET that you are done with all of the writing to the file, and you'd like Visual Basic.NET to "close" the file so. If you forget to do this, it is possible in some cases that you won't see all of your output in the file. The reasons have to do with something called "buffering" – if you want to know what that is all about, by all means ask your instructor.

The last line you need, and it usually comes at the end of the `btnMain_Click()` routine, just before EndSub. The statement is:

```plaintext
writer.Close()
```

So here's what the whole thing should look like. (I've left quite a few blank lines to highlight what you've been doing – you don't need them.)

```plaintext
Imports System.IO
Public Class Form1
    Inherits System.Windows.Forms.Form

    Private Sub btnMain_Click()
        Dim writer As StreamWriter
        writer = File.CreateText("c:\temp\VBOutput.txt")

        MsgBox ("Hello, World!")

        writer.WriteLine("Hello, World!")

        writer.Close()
    End Sub
End Class
```

6) Now try to Build and run (F5) your program. Assuming you typed everything correctly, you should see that everything ran exactly the way it
did before you added all of that stuff. You only see the message box with "Hello, World!".

7) End the program by closing the form.

8) Now open the Notepad program provided by Windows. (It's in the Accessories group, which you reach from the Start button and Programs.) Once in Notepad, open the "VBOutput.txt" file (go to "C:\temp") to find it. It should contain just one line – "Hello, World!" (without the quotes).

If you wanted to print this to the printer, just use Notepad's Print menu selection.

9) Let's see one more point about writing to a file. You were told above that there are two different statements you can use to write to the file – Write and WriteLine. The only difference is that when you use Write, you write to the file, but it does not advance to the next line. So if you did two Writes, one after the other, the output would wind up all on one line. With WriteLine, the output ends by moving to the next line, rather than staying on the same line.

Here's a brief example.

Close Notepad. Then go back to your program and make the following change (the underlined part is what you need to add). Be sure to leave the extra spaces in the " Ho, there." line.

```vbscript
Private Sub btnMain_Click ( )
    Dim writer As StreamWriter
    writer = File.CreateText("c:\temp\VBOutput.txt")

    MsgBox ("Hello, World!")
    writer.WriteLine("Hey, there.")
    writer.WriteLine(" Ho, there.")
    writer.WriteLine(" Hi, there.")
    writer.WriteLine("Hello, World!")
    writer.Close()
End Sub
```

When you run this program, the output will look like:

```
Hey, there. Ho, there. Hi, there.
Hello, World!
```