

Developing Your First Visual Basic Program

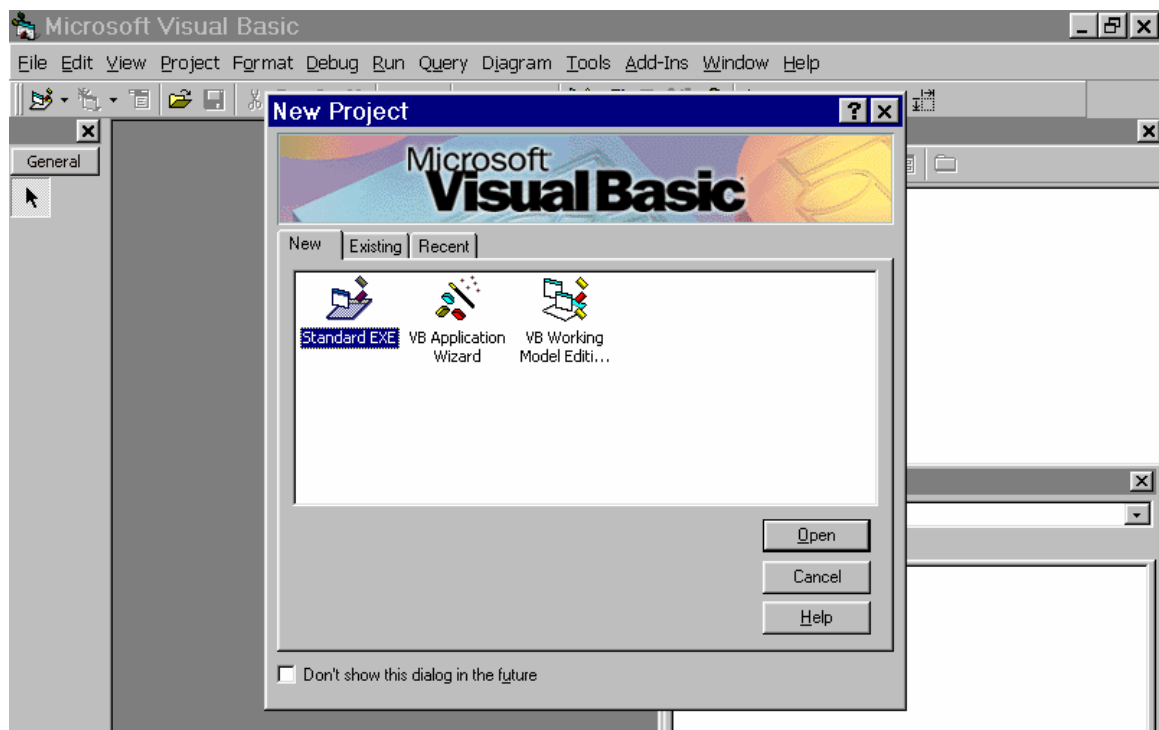
The goal of this tutorial is to enable each student to develop his (or her) first Visual Basic Program. The application involves the creation of a very simple Visual Basic (VB) program in order to compute and display the horizontal distance using stadia theory given user-specified survey data.

This application offers very little in the way of problem solving. The intent of the tutorial is to introduce each student to the VB programming environment. You will find this tutorial much easier if you read Modules 1 and 2 of the course notes prior to attempting this tutorial.

1.0 Starting the Visual Basic (VB6) Program

Log on to your personal account on the NEXUS system. From the Windows **START** command, select: **Programs/Programming/Microsoft Visual Studio 6.0/Microsoft Visual Basic 6.0**

When VB6.0 is first started, your screen will display the **New Project Dialog Box** as shown below:



Leave the default (Standard.EXE) option highlighted and click on the **Open** button.

2.0 The Elements of the Integrated Development Environment (IDE)

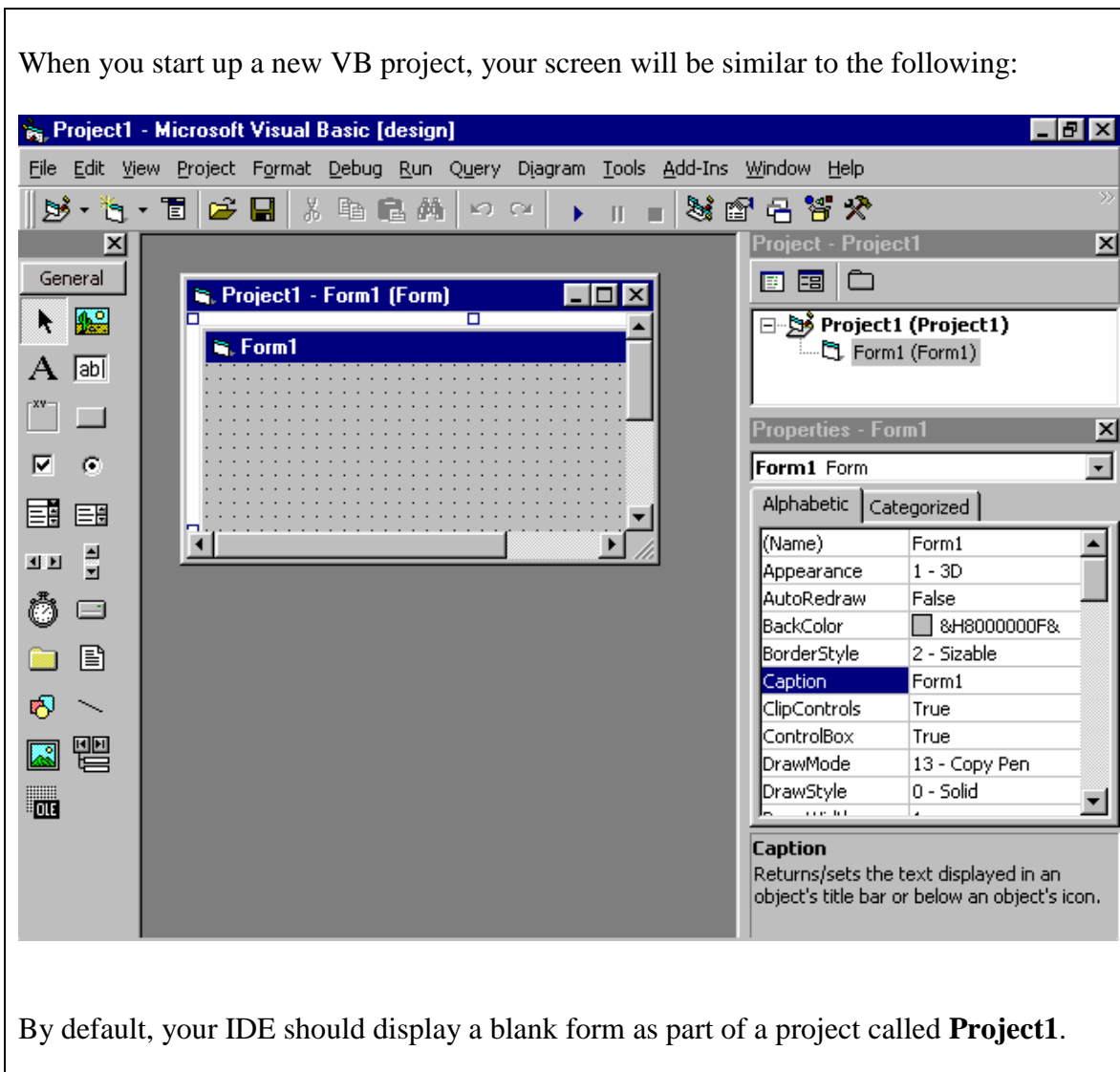
On your screen, you will see the elements of the Visual Basic Integrated Development Environment (IDE).

Near the top of the screen, you should see the familiar Windows pull-down menus and toolbars.

You should also be able to identify:

- 1) A **Form** Window
- 2) A **ToolBox** Window
- 3) A **Project Explorer** Window
- 4) A **Properties** Window

When you start up a new VB project, your screen will be similar to the following:



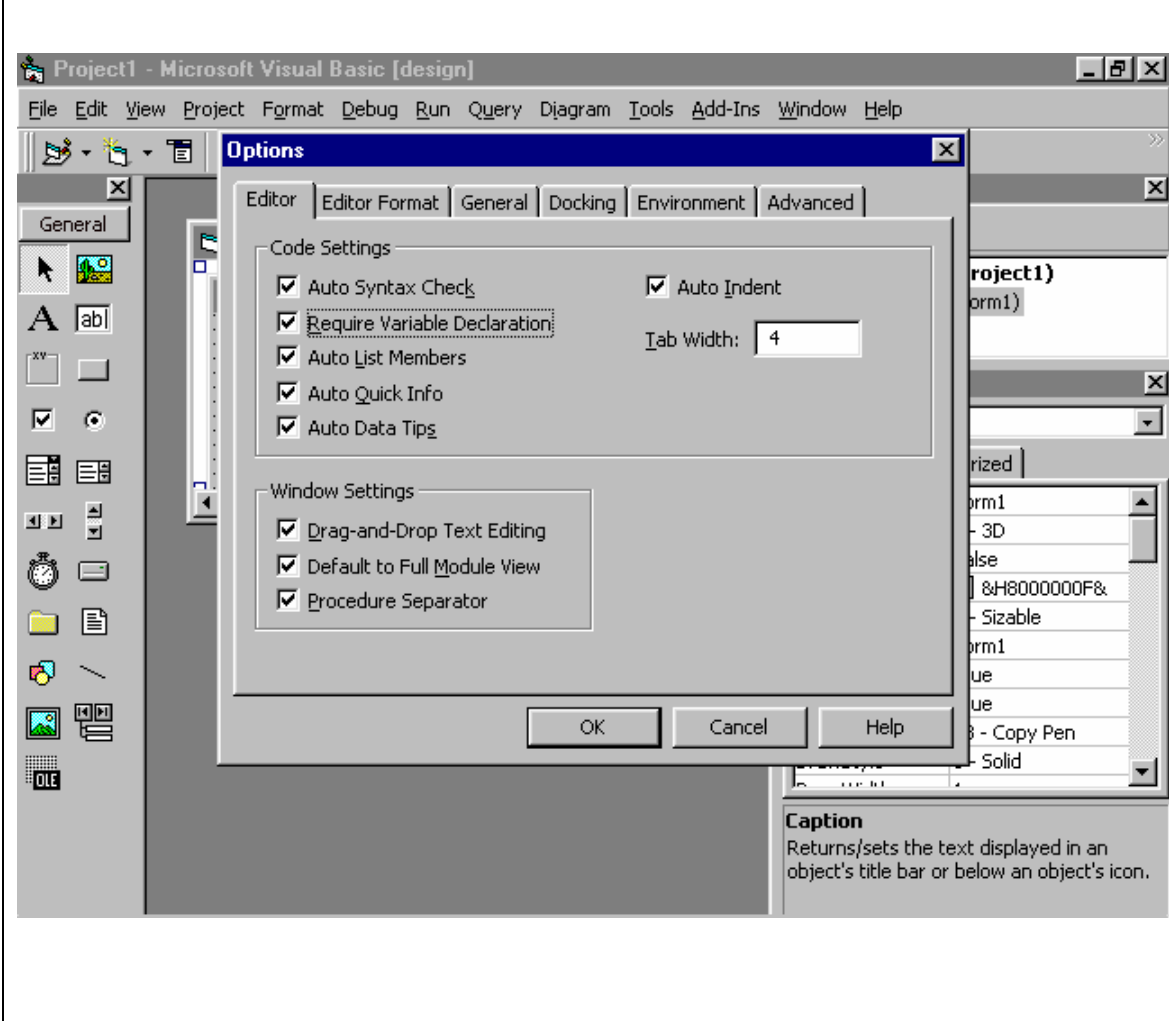
By default, your IDE should display a blank form as part of a project called **Project1**.

3.0 Developing Your First VB Program

3.1 Variable Declaration

Click on the **Tools** menu and select the **Options...** sub-menu.

Under Code Settings, make sure the **“Require Variable Declaration”** is initialized as shown below.



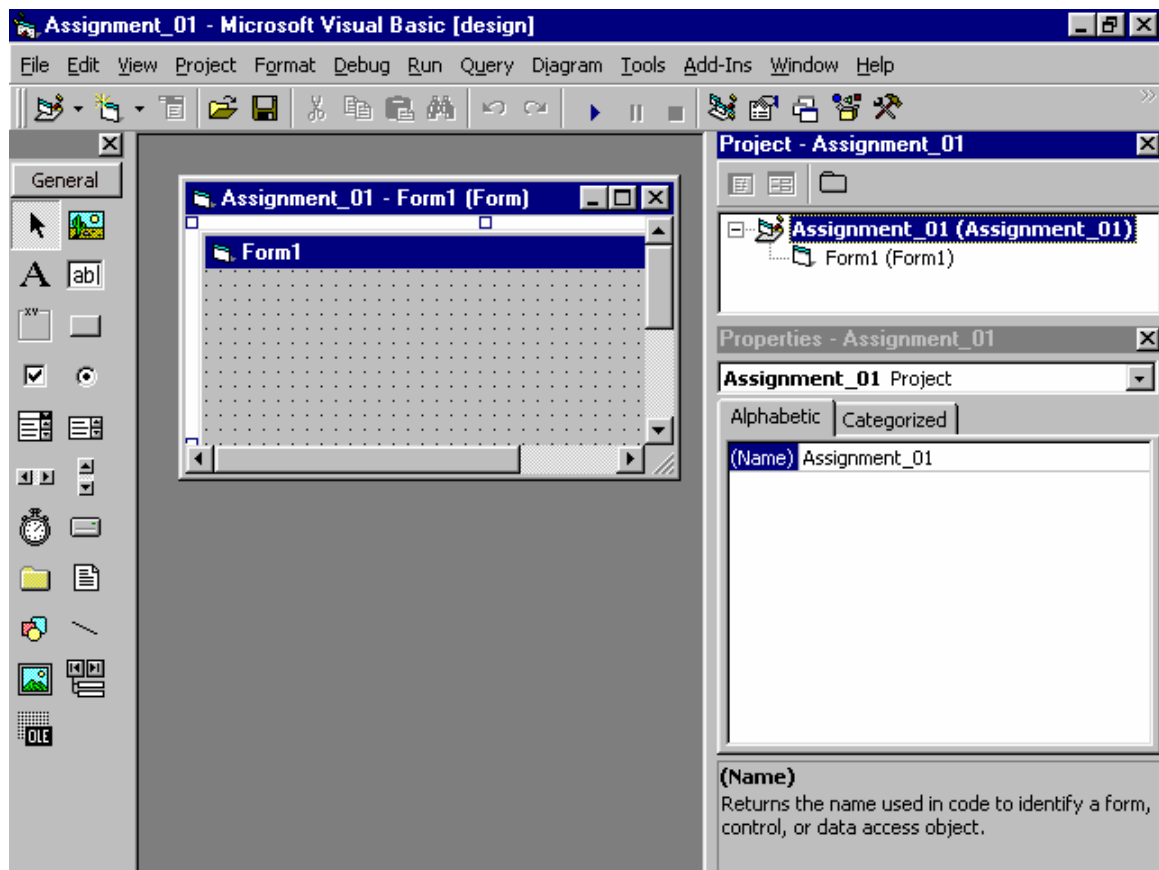
3.2 Name Your Project and Form

In the Project Explorer Window, you should see a project called **Project1** with a form called **Form1**.

In the **Project Explorer Window**, right-click on “**Project1 (Project1)**” and select the “Project1 Properties...” option.

The **Properties Window** should display the default project name. By default, all new projects are initially named Project1.

In the Project Properties Window, change the name of the project to **Assignment_01**.

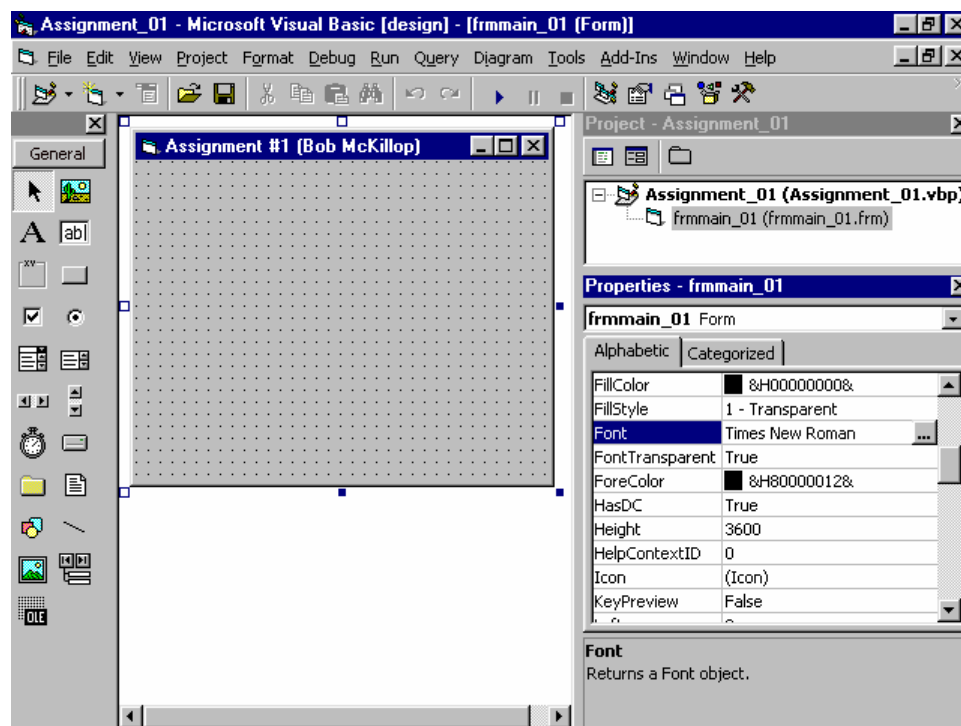


3.3 Specify the Properties of the Main Form

Let's change some of the properties associated with **Form1**. Right-click on the empty form (**Form1**) in the **Form Window** and select the "Properties" option. The properties Window should now display the properties associated with Form1. In the **Properties Window**:

- 1) Change the name of the form to **frmMain_01**. (By convention, we typically begin each control name with a three letter prefix to help identify the type of control. The prefix **frm** indicates that the control is a form). We use the name **Main_01** to indicate that this will be our main form for Assignment 1. (Later in the course, you may develop programs that utilize several forms and it is important that you do not accidentally overwrite your previous forms.
- 2) The caption of the form is displayed at the top of the form in the title bar. In the properties Window, change the caption to display Assignment #1 (your name).
- 3) And finally, in the Properties Window, double click on the Font property. A font Dialog box will be displayed. Specify a 12 point, bold, Times Roman font style.

This is how your IDE screen should look like so far.



3.4 Saving Your Work

So far, we have created a form and assigned several properties to that form. We will soon add controls and all associated code to the form. But first, we should save our work. Visual Basic recognizes both forms and projects. Each program involves a project file (**.vbp**). Each form has its own file (**.frm**). The VB project file keeps track of the form files, their names and where to find them:

```
Type=Exe
Form=frmMain_01.frm
Reference=*G{00020430-0000-0000-C000-
000000000046}#2.0#0#..\..\..\..\WINDOWS\SYSTEM\stdole2.tlb#OLE Automation
IconForm="frmMain_01"
Startup="frmMain_01"
Command32=""
Name="Assignment_01"
HelpContextID="0"
CompatibleMode="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="Civil Engineering"
CompilationType=-1
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
Retained=0
ThreadPerObject=0
MaxNumberOfThreads=1
```

Do not open or edit the .vbp file!

A word of warning...if you change the name of a form file using Windows Explorer, your VB project file will still look for the old filename)

With all your programs in this course, it is very important that you save each VB project in a separate folder. **Every year, we have students overwriting previous forms and losing hours of work.**

For this project, create a suitable subfolder (ie. **cive121\Assign_01**)

Under the Windows "**File**" pull-down menu, use the "**Save Project as..**" and save your new project files (both the main form file and project file) to your newly created folder.

4.0 Placing Controls on the Main Form

4.1 Labels

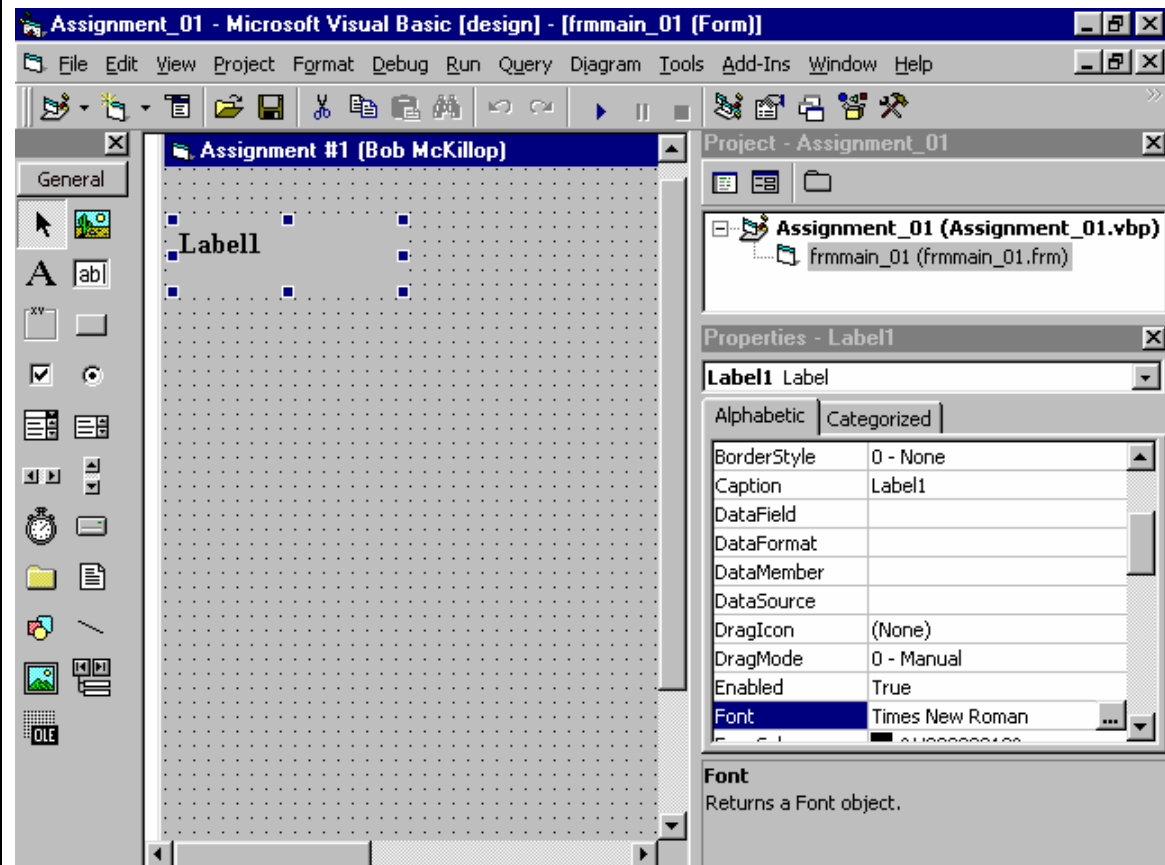
In the **Form Window**, maximize the display of the main form by clicking the maximize button of the Form Window.

Click on the Label Icon (**A**) in the Toolbox Window.

When you move the mouse pointer on top of the main form, the cursor should appear as a small cross.

Drag the mouse pointer on the form and create a label object on the form. When you release the mouse button, the label control will appear on the form.

The caption **Label1** will be displayed as shown. The **Properties Window** will also display the default properties for the label object.



Change the properties of the label.

In the Properties Window, double click each of the following properties and specify the label properties as follows:

Name	lblPrompt1
Caption	Stadia Constant (C)

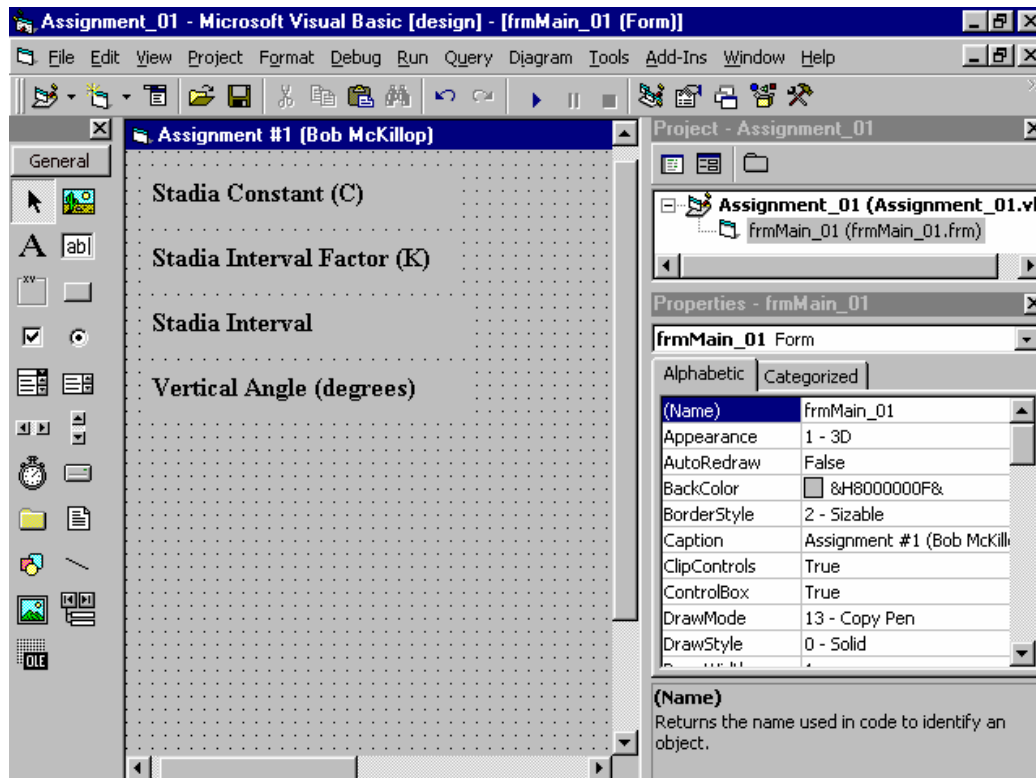
Place three additional labels on the form as shown.

Assign name and caption properties to the labels as follows:

Name	lblPrompt2
Caption	Stadia Interval Factor (K)

Name	lblPrompt3
Caption	Stadia Interval

Name	lblPrompt4
Caption	Vertical Angle (degrees)

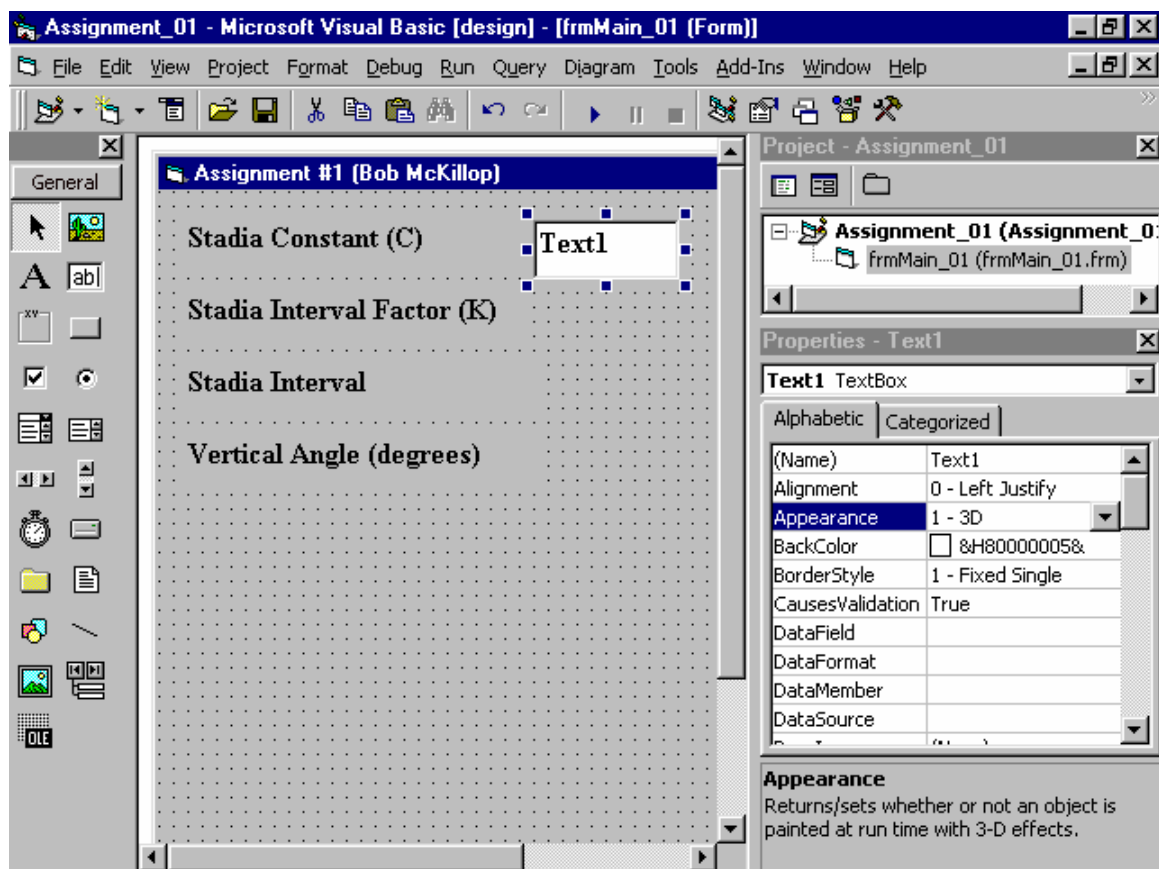


4.2 Text Boxes

The purpose of text boxes is to allow user input.

Click on the TextBox Icon (**AB**) in the Toolbox Window.

Drag the mouse pointer on the form and create a textbox object to the right of your first label object as shown. The caption Text1 will be displayed as shown. The **Properties Window** will also display the default properties for the TextBox object.



Change the properties of the **TextBox**. In the Properties Window, double click the "name" property and specify the name as txtStadiaConstant.

Double click on the "text" property and delete the "**Text1**" string. The TextBox on the form should now be blank.

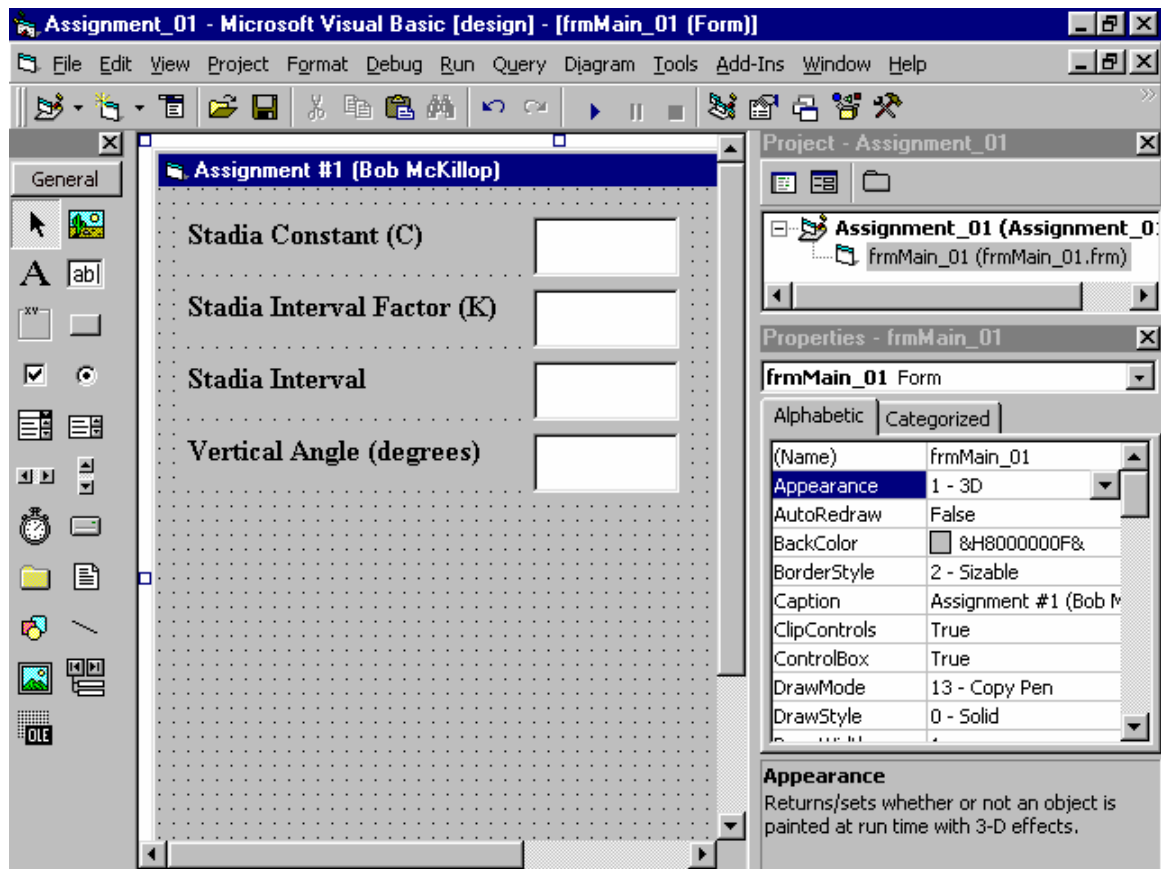
Place three additional textBoxes on the form as shown below.

Assign name properties to the textboxes as follows:

Name	txtStadiaIntervalFactor
Text	[leave blank]

Name	txtStadiaInterval
Text	[leave blank]

Name	txtVerticalAngle
Text	[leave blank]



Save your work.!

4.3 A Command Button

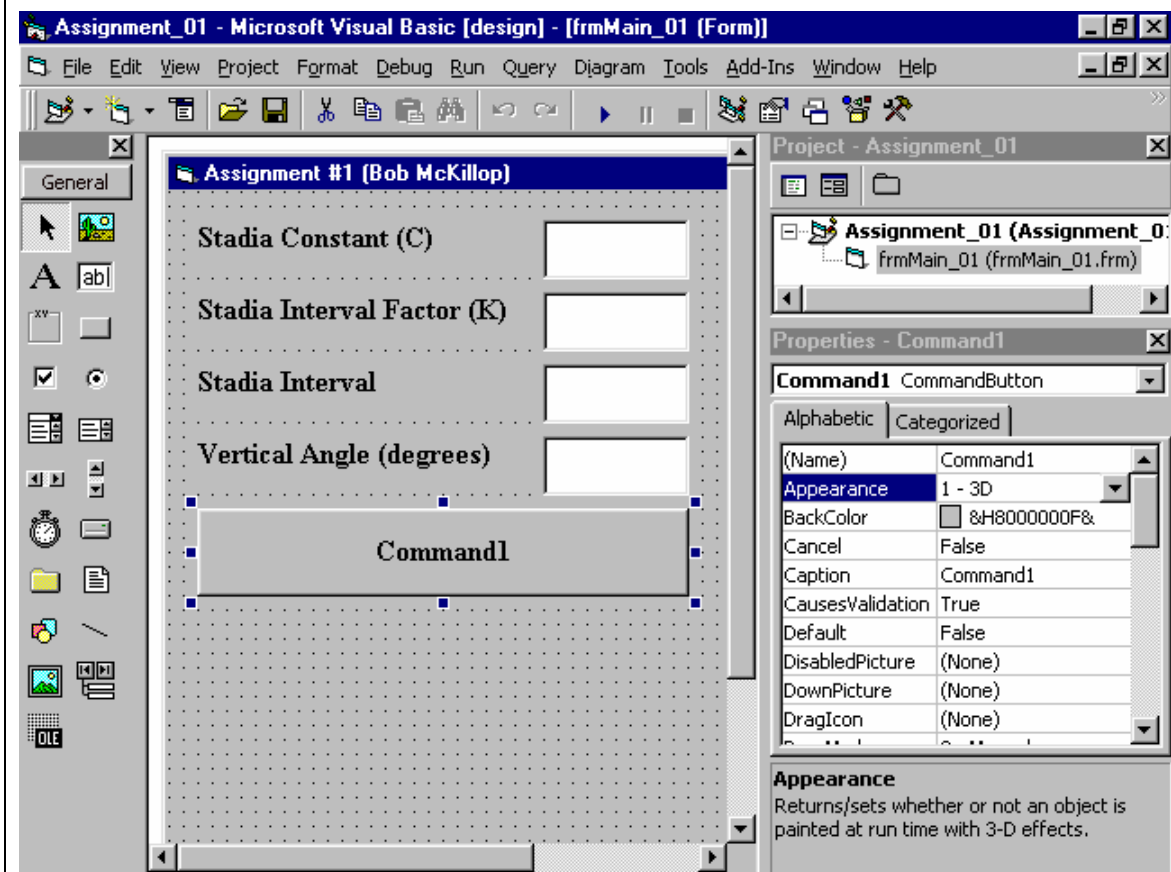
In our program, we will provide a command button on the form.

When the user clicks on the command button, the program will perform its computations and display the answer.

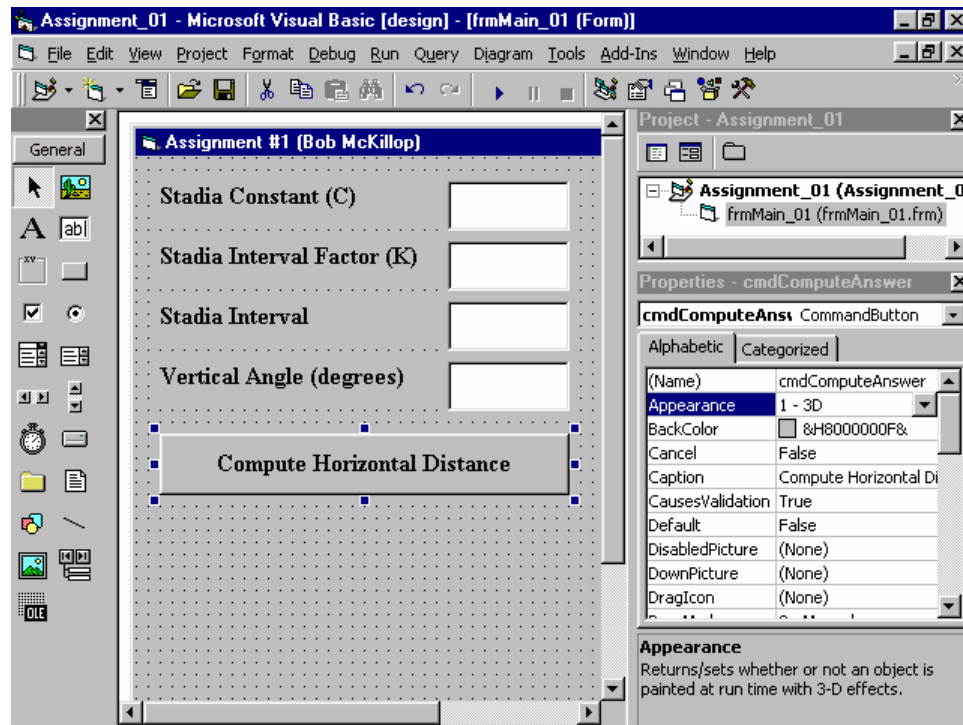
Click on the Command Button Icon () in the Toolbox Window.

Drag the mouse pointer on the form and create a command button object as shown.

The caption **Command1** will be displayed as shown. The **Properties Window** will also display the default properties for the Command Button object.



Now, we will change the properties of the Command Button. In the Properties Window, double click the name property and specify the name as **cmdComputeAnswer**. Double click on the Caption property and specify the caption to be Compute Horizontal Distance. The caption should now be displayed on the button as shown below.



4.4 A Picture Box

It's not much good to have a program compute the answer if it does not tell us the solution. In our program, we need to provide an object on the form in order to display the program output to the user.


Click on the **Picture Box** Icon in the Toolbox Window. Drag the mouse pointer on the form and create a picturebox object on the form as shown on the next page.

As expected, the **Properties Window** will now display the default properties for the PictureBox object.

Change the properties of the PictureBox. In the Properties Window, double click the "name" property and specify the name as **picResults**.

4.5 Stopping the Program

When the user no longer needs the program, we need a button to allow the user to terminate the program execution.

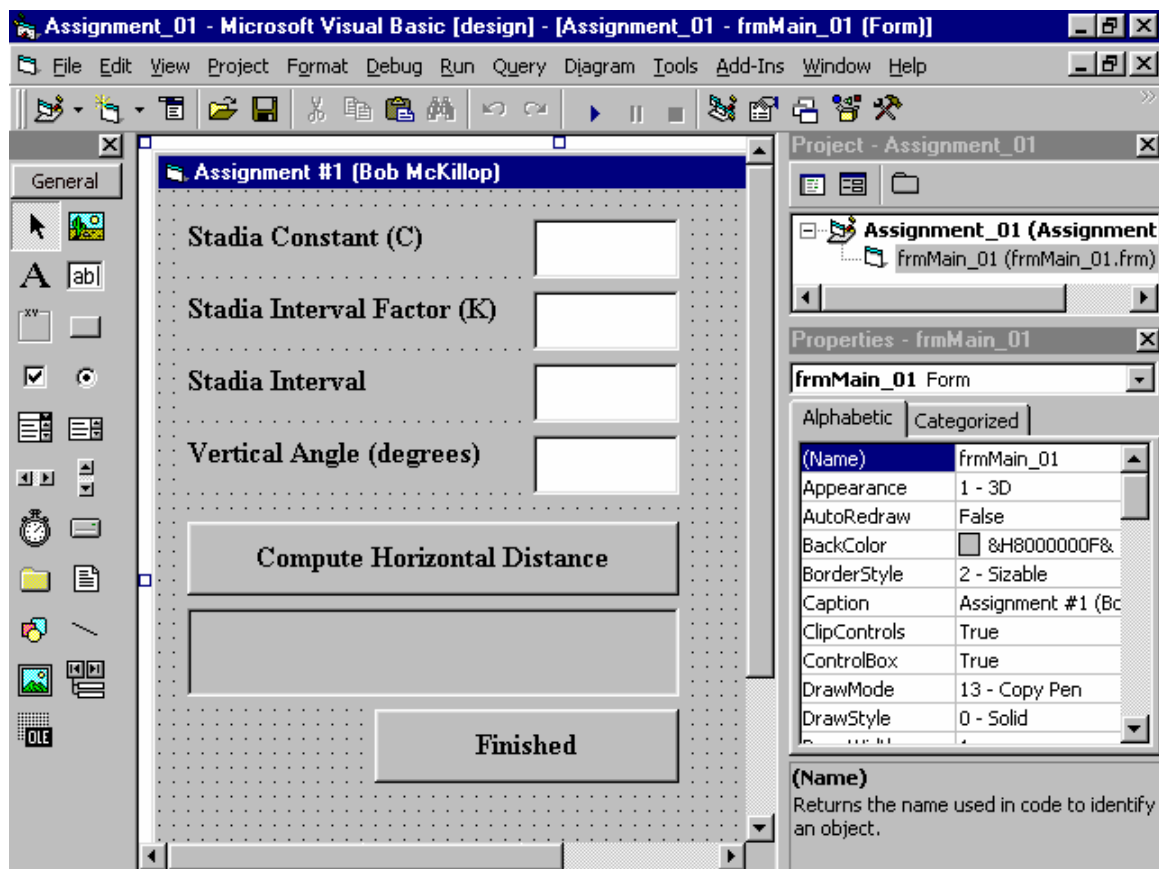
Again, click on the Command Button Icon () in the Toolbox Window.

Drag the mouse pointer on the form and create a command button object as shown below.

Change the properties of the second command button.

In the Properties Window, double click the “name” property and specify the name as **cmdExitProgram**.

Double click on the “caption” property and specify the caption as “**Finished**”. The caption should now be displayed on the button as shown below.



Save your work!

4.6 Let's Summarize Our Controls and Their Properties

The following table provides a brief summary of the controls associated with our program.

Object	Property	Setting
frmMain_01	Caption	Assignment #1 (Bob McKillop)
lblPrompt1	Caption	Stadia Constant (C)
lblPrompt2	Caption	Stadia Interval Factor (K)
lblPrompt3	Caption	Stadia Interval
lblPrompt4	Caption	Vertical Angle (degrees)
txtStadiaConstant	Text	[Blank]
txtStadiaIntervalFactor	Text	[Blank]
txtStadiaInterval	Text	[Blank]
txtVerticalAngle	Text	[Blank]
cmdComputeAnswer	Caption	Compute Horizontal Distance
cmdExitProgram	Caption	Finished
picResults		

5.0 Writing Code For the Events

So far, we have placed numerous objects on our main form and specified some of their properties. In general, these properties influence how the control appears on the screen.

In addition to **properties**, controls also have **events** that it will respond to.

In this course, our most common event will be a **Click** event. If we place the mouse on a command button and left-click the mouse, we expect something to happen.

Each control has numerous **event procedures** associated with that control. These event procedures contain the code instructing the program what actions are to take place if the event is invoked.

5.1 The Main Form

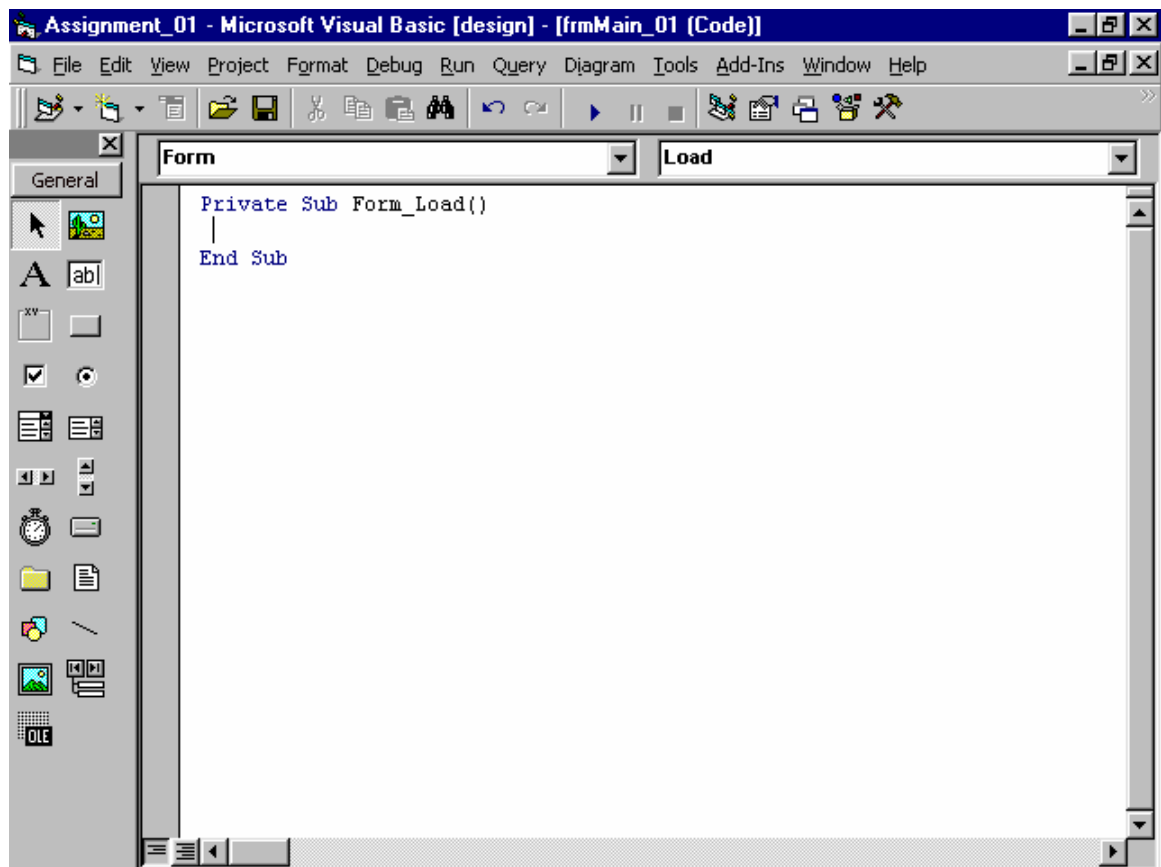
For the main form's default event is **Load**. When a form is initially loaded into memory, a subprocedure associated with the Form_Load event is executed automatically.

We can customize this loading event by placing our computer code in the load subprocedure.

Double click your mouse on the form contained in the Form Window.

Visual Basic will display the code associated with the subprogram executed as part of the default Form_Load event.

For now, no code exists as shown below:



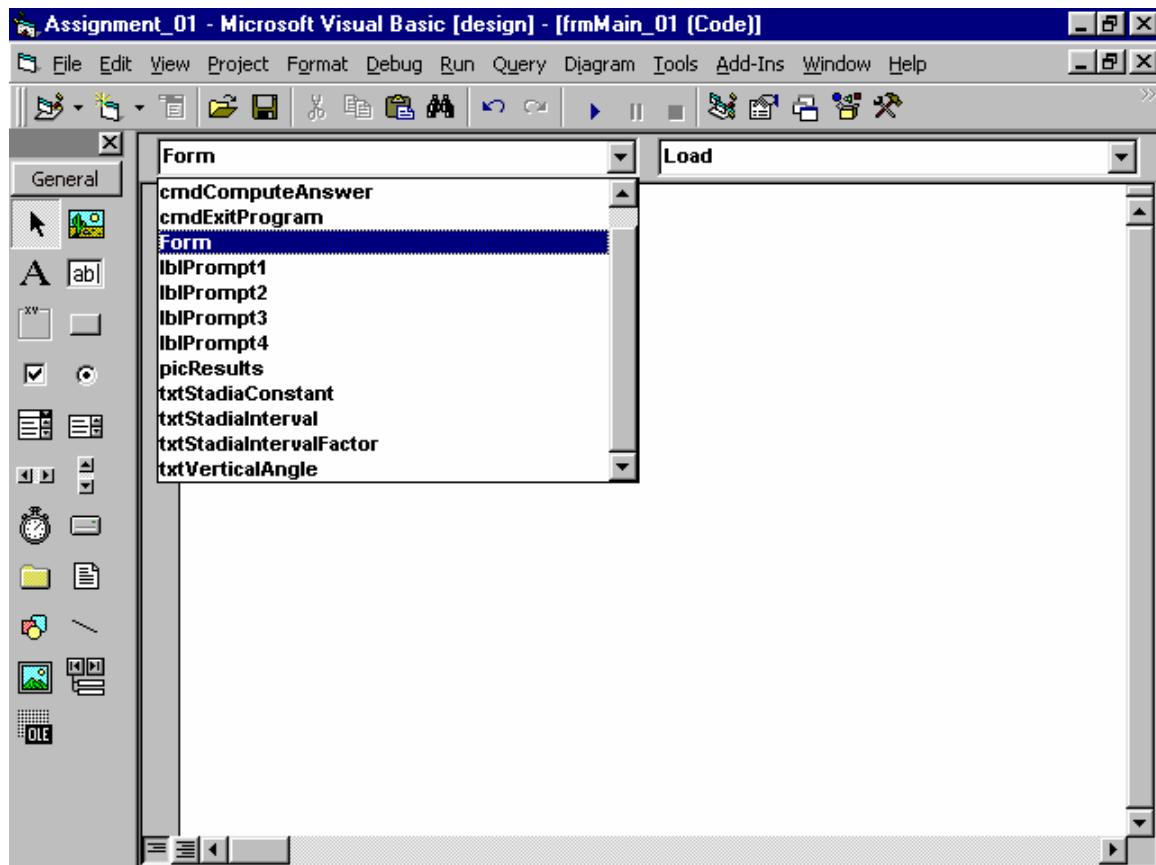
Any code placed in this subprocedure will be executed whenever the form is loaded by the program.

Above the Code Window, you should see two selection boxes.

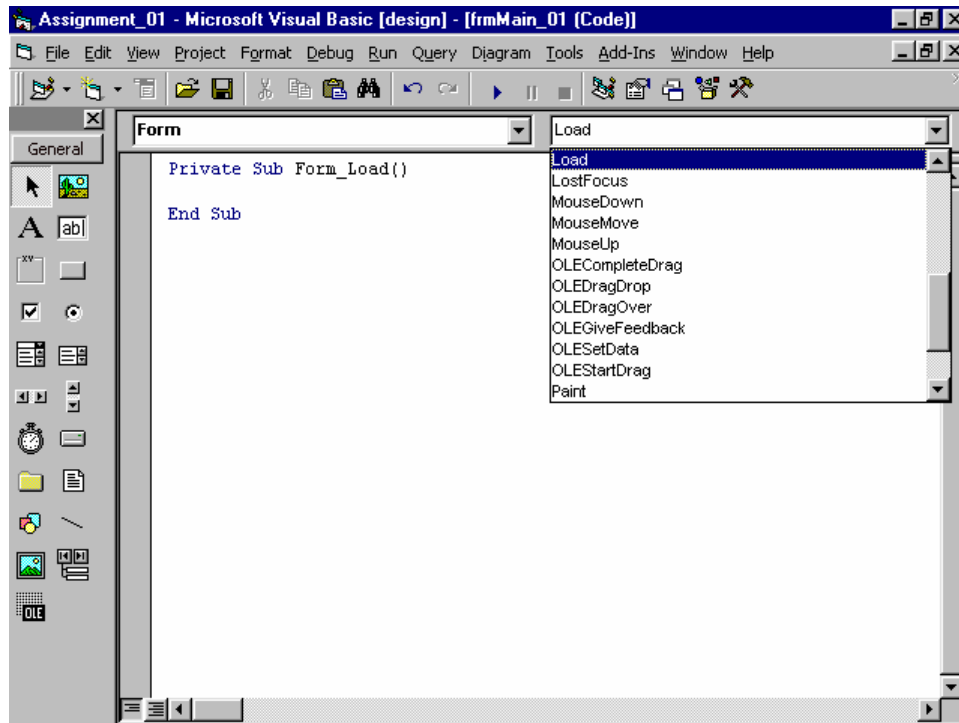
The left box displays the **Object** associated with the code.

In our case, the current object is the **form** itself.

If you click the mouse on the Object list, you should see a list of the objects associated with our project (ie. Form, txtStadiaConstant, etc.)



The right box displays the **Procedures** associated with the Object. In our case, we are interested in the **Form_Load** event. If you click the mouse on the Procedure list, you should see a list of the possible procedures (events) associated with a form.



Let's add code to the **Form_Load** event.

Carefully enter the code shown to the right.

At this stage, you do not need to understand what the code is doing. Just enter the code **EXACTLY** as shown.

```
Private Sub Form_Load()  
  
frmMain_01.Width = 5200  
frmMain_01.Height = 7700  
picResults.AutoRedraw = True  
  
End Sub
```

5.2 The cmdComputeAnswer Button

Okay, we can now begin adding our computer code to the remaining objects. We can start by developing the code that will be executed whenever the command button is clicked.

Click your mouse on the **Object** list and select **cmdComputeAnswer**. Visual Basic will automatically assume the default event for a command button that is a Click Event as shown in the Procedure box shown below.

The subprogram for a click event associated with the button is empty.

We want to add code such that the program computes the horizontal Stadia Distance using our familiar stadia equation (Fraser, 2001). Carefully enter the code exactly as shown below.

```
Private Sub cmdComputeAnswer_Click()  
  
    ' variable declarations  
    Dim sgC As Single  
    Dim sgK As Single  
    Dim sgS As Single  
    Dim sgVertDeg As Single  
    Dim sgVertRad As Single  
    Dim sgHorDist As Single  
  
    ' input surveyed data  
    sgC = CSng(txtStadiaConstant.Text)  
    sgK = CSng(txtStadiaIntervalFactor.Text)  
    sgS = CSng(txtStadiaInterval.Text)  
    sgVertDeg = CSng(txtVerticalAngle.Text)  
  
    ' convert vertical angle to radians  
    sgVertRad = sgVertDeg * 4* Atn (1) / 180  
  
    ' compute final answer  
    sgHorDist = (sgK * sgS * Cos(sgVertRad) + sgC) * Cos(sgVertRad)  
  
    ' display in PictureBox  
    picResults.Cls  
    picResults.Print "Distance= "; sgHorDist  
  
End Sub
```

Save your work!

5.3 The cmdExitProgram Button

Next, we need to provide the code to the **cmdExitProgram** button. When the user is finished with the program, we need to provide an obvious way for the user to terminate the execution of the program.

For the **cmdExitProgram** click event, carefully enter the code shown below.

```
Private Sub cmdExitProgram_Click()  
  
    Unload Me  
  
End Sub
```

5.4 The program header/declaration

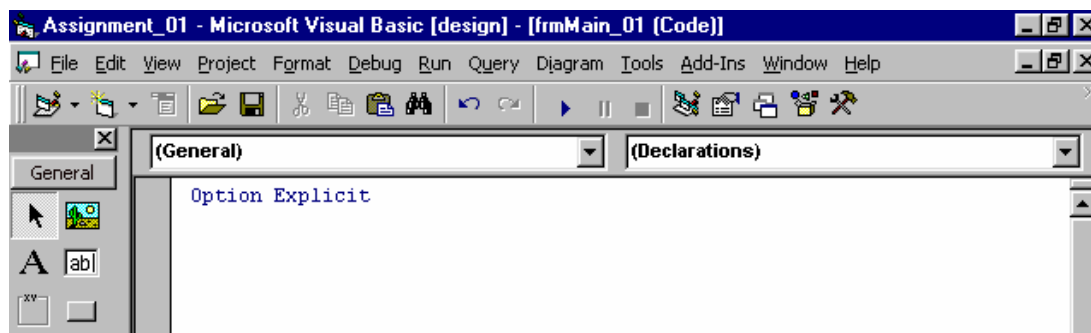
Click your mouse on the **Object** list and select (**General**).

Visual Basic will display the code listing for your general program declarations.

Your code listing should display a single line of code. If it does not, add the line in manually. The next time you start up VB, the *Option Explicit* line will be automatically placed in your general declarations section.

```
Option Explicit
```

This command tells Visual Basic that all variables must be explicitly declared within the code of our program – always good programming practice.



We want to add a declaration of your work in this course. This code **MUST** be included with **EVERY** program you submit for a grade in this course. On the course web site, (<http://www.civil.uwaterloo.ca/courses/Cive121/>) you will find a link entitled “Assignment Header/Declaration”. This link will allow you to download a text listing (Header.txt) of the course declaration. Cut and paste the header text into your program.

You can then edit the generic text to correspond to your assignment as shown for this example.

```
' *****  
' NAME: Bob McKillop  
' STUDENT BLOCK: FL88A  
' Waterloo Network ID: rmckillo@civmail  
' ID: 20123456  
' DEPARTMENT: Civil Engineering  
' TEACHING ASSISTANT: Geoff Milburn  
'  
' ASSIGNMENT: 1  
' QUESTION: Part B Problem 1  
' DATE: May 5, 2005  
'  
' PROGRAM DESCRIPTION:  
'  
' Purpose: Given a set of survey data, this program computes  
' the horizontal distance associated with the survey  
' instrument and the survey rod.  
'  
' Methods: The horizontal distance is computed using equation 7.3  
' of Fraser, 2000  
'  
' Ref: Civ.E. 125 Survey Lecture Notes (2000)  
' D. Fraser, p. 91  
'  
' Inputs: Stadia constant (m)  
' Stadia interval factor  
' Stadia interval (m)  
' Vertical angle associated with the line of sight  
'  
' Outputs: Horizontal distance between transit and survey rod (m)  
'  
' Comments: None  
'  
'  
' I declare that, other than listed below, this program code  
' is my original work.  
'  
' SIGNATURE _____  
'  
' Acknowledgements:  
'  
' This programming code was completed entirely by myself. Initial difficulties  
' with the problem were solved after talking to Geoff and Bob.  
'  
' *****
```

Option Explicit

6.0 Running Your Program

Save your work!

When you have completed constructing your user interface (form) and written all the computer code associated with all anticipated events, you are ready to run your program.

In Visual Basic, pressing the **F5** key will run your program.

Your program should have displayed your main program form as shown below. The cursor should be located in the upper most text box awaiting an entry for the stadia constant.

The screenshot shows a Windows-style application window titled "Assignment #1 (Bob McKillop)". The window contains a form with the following elements:

- Four text boxes for input, labeled from top to bottom: "Stadia Constant (C)", "Stadia Interval Factor (K)", "Stadia Interval", and "Vertical Angle (degrees)". The cursor is positioned in the first text box.
- A button labeled "Compute Horizontal Distance" located below the input fields.
- A large, empty rectangular area below the button.
- A button labeled "Finished" located at the bottom of the form.

You can test your program using the following data.

$$C = 0.300$$

$$K = 100$$

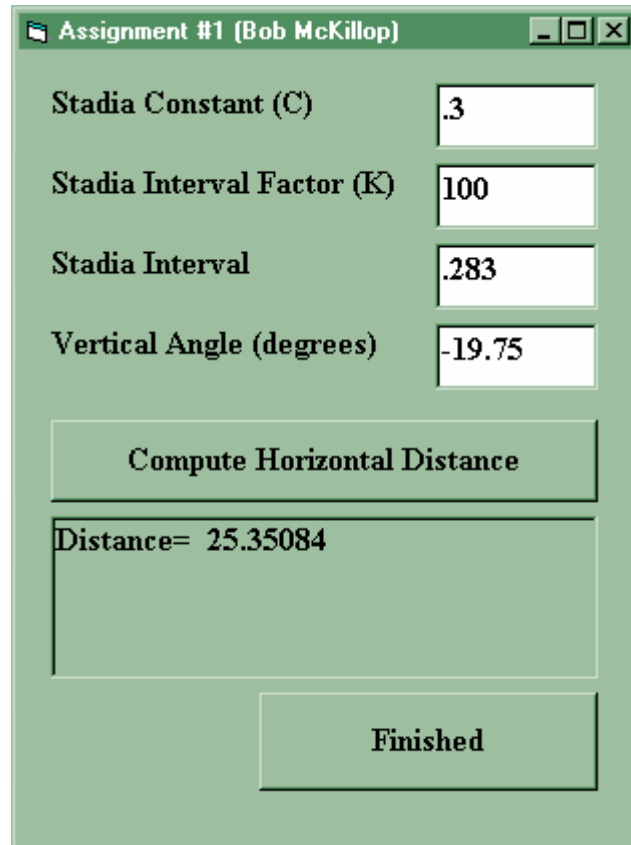
$$S = 0.283$$

$$\alpha = -19.75 \text{ degrees}$$

Enter each value in the appropriate text box as shown below:

The screenshot shows a Windows-style application window with a title bar that reads "Assignment #1 (Bob McKillop)". The window contains four input fields, each with a label to its left and a text box to its right. The labels and their corresponding values are: "Stadia Constant (C)" with ".3", "Stadia Interval Factor (K)" with "100", "Stadia Interval" with ".283", and "Vertical Angle (Degrees)" with "-19.75". Below these fields is a large button labeled "Compute Horizontal Distance". At the bottom of the window is another button labeled "Finished".

Now, click on the **cmdComputeAnswer** button and let's see what happens!



Hopefully, your program has displayed the solution in the picture box and also placed your cursor prompt back into the Stadia Interval TextBox in order to make your next data entry a little more convenient. Your program should have computed an answer of **25.35084** in the picture box.

Note that your program displayed the solution to 5 decimal places! Let's fix our program in order to display only 3 decimal places.

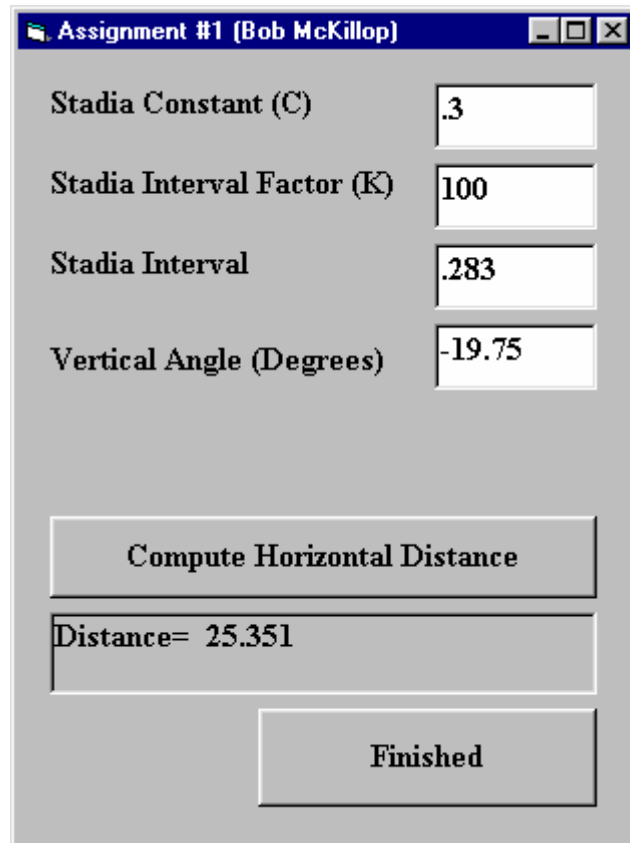
Click on the "**Finished**" command button and modify your code controlling the program output as follows:

```
'      display in PictureBox  
picResults.Cls  
picResults.Print "Distance= "; Round(sgHorDist, 3)
```

In Visual Basic, the **Round** function allows us to control the number of decimal places associated with the program solution. In this case, you have instructed the program to display the computed horizontal distance to 3 decimal places.

Run your program again.

Your program output should be similar to the following:



Your program should have computed an answer of **25.351** as shown in the picture box above.

7.0 What To Do If Your Program Does Not Work (Debugging)

Debugging refers to the process of finding and correcting errors in your program code. Quite honestly, students often waste an enormous amount of time through inefficient debugging of your code.

As you progress through the course, your code will become large and somewhat complex. Effective debugging skills will significantly reduce your frustration level in this course.

Visual Basic provides numerous debugging tools to help you uncover the cause of your program problems. These include:

1) Code Reviews

When your program is not working, this should always be your first approach. A code review involves meticulously going through your code, line by line, looking for any indications of a problem. It is usually easier to work off of a hardcopy (printout) of your program code.

2) Setting Breakpoints

In the VB programming environment, you can instruct the program to stop (temporarily) anywhere you want. More about this in a few moments.

3) Viewing Data Tips

When the computer is in break mode, Visual Basic's data tips windows can be used to display the current value of any variable in your program. More about this later.

4) Executing the Program (One Line at a Time)

When in break mode, you can instruct Visual Basic (using the **F8** key) to execute a single line of code. After the line is executed, you can once again check the values of any variables that are of interest to you. By repeatedly pressing the **F8** key, you can "walk through" your program execution.

5) Displaying Intermediate Results

See page 82 of the class text for more information.

6) Watching Variables

See page 86 of the class text for more information.

In general, errors in your program can be classified into three types:

1) **Syntax Errors**

This error refers to grammar or punctuation errors with your code. Missing parentheses, commas or misspelled code are common syntax errors. In addition, some words in VB are reserved and cannot be used by you as a variable name for instance.

For many syntax errors, the VB compiler will indicate an error message to the user.

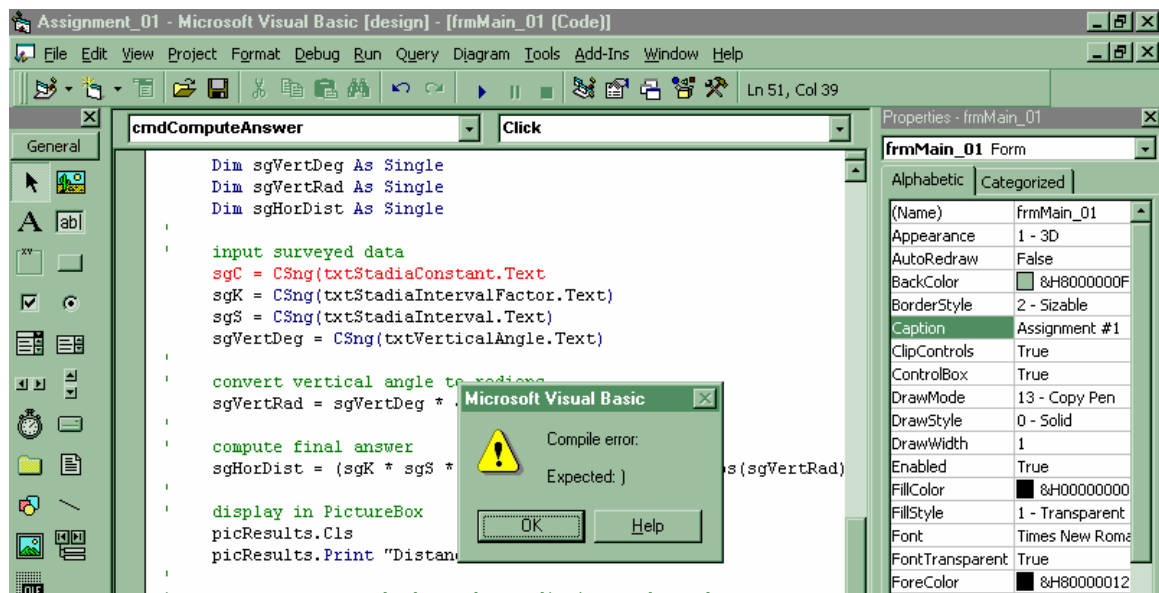
Visual Basic will use a red font to display the line of code containing an error.

Let's deliberately cause a syntax error.

In order to produce a syntax error, remove the parenthesis at the end of the following line in your code:

```
sgC = CSng(txtStadiaConstant.Text
```

Then, move the cursor off the current line of code. Visual Basic will immediately notify you about a **compile error** and indicate that a parenthesis is missing. The code window will display the code using a red font as shown below.



Replace the end parenthesis and your program will remove the red font.

2) Run Time Errors

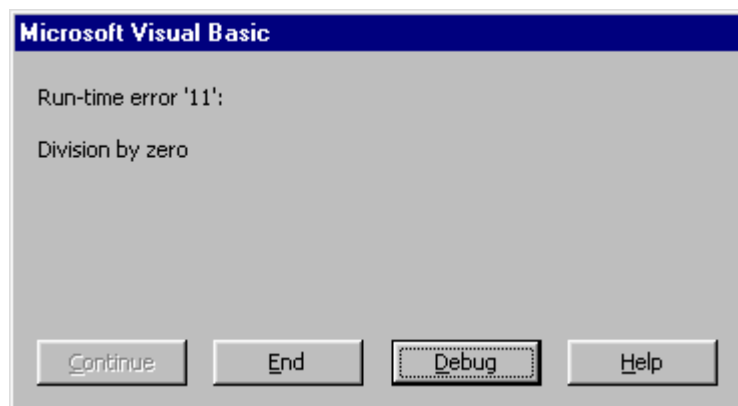
These errors typically occur when the program attempts to perform an illegal operation while it is executing. For example, a common run time error occurs when a program attempts to divide an expression by zero.

Let's deliberately cause a run time error.

In order to produce a run time error, on the following line in your code, replace 180 with zero as shown.

```
sgVertRad = sgVertDeg * 4 * Atn(1) / 0
```

Run your program. Nothing will happen right away. Enter the program data and click the compute button. When your program attempts to execute the line, a run time error is detected by VB and displayed to you in a message box as shown.



Press on the "Debug" button and fix your code.

3) Logic Errors

These errors occur when the program computes an incorrect solution. These errors are due to poor code development by the programmer and can be difficult to fix if the program development phases are not approached in an organized manner.

Much more about this as the course progresses.

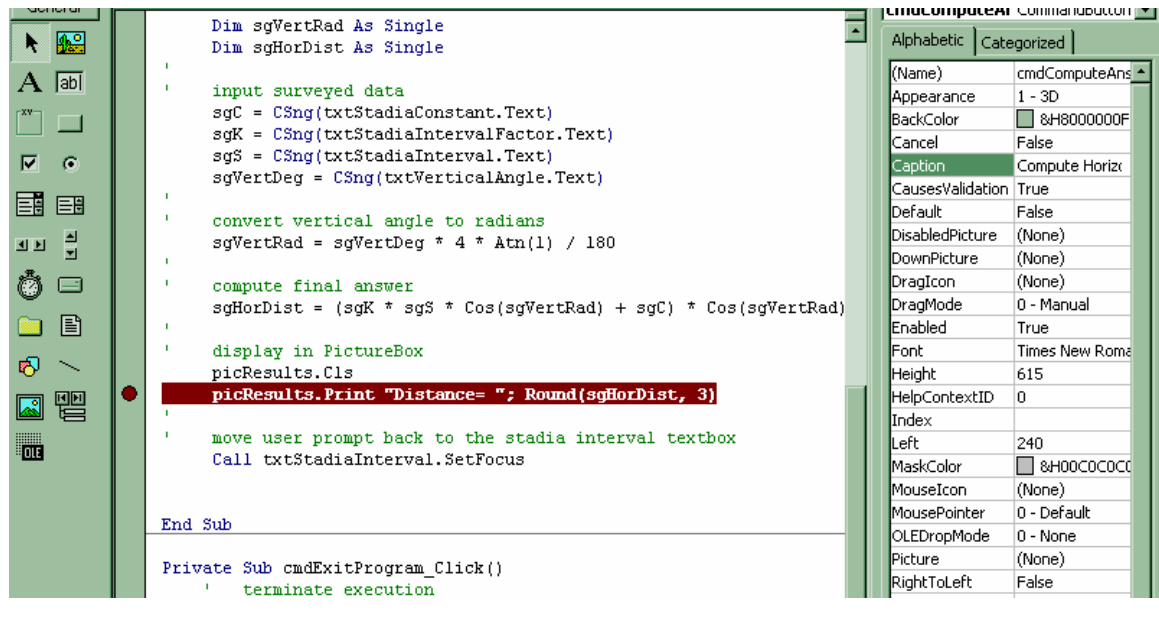
Break Points

Break points are locations in your code where the program execution will temporarily stop.

When your program is not working correctly, you can place breakpoints in your code to help you identify what the program is doing at that location in your code. **In this course, you should make abundant use of breakpoints!!**

In order to use a break point, you simply identify the line of code where you want to stop the program execution. For this example, in your code window, identify the line where printing instructions to the picture box are located. Move the cursor onto the line and then press the **F9** key.

Your code window should display a red dot in the margin and highlight the entire line of code in red as shown below.



The screenshot shows a Visual Basic code editor window with a code window on the left and a Properties window on the right. The code window contains the following code:

```
Dim sgVertRad As Single
Dim sgHorDist As Single

'
' input surveyed data
sgC = CSng(txtStadiaConstant.Text)
sgK = CSng(txtStadiaIntervalFactor.Text)
sgS = CSng(txtStadiaInterval.Text)
sgVertDeg = CSng(txtVerticalAngle.Text)
'
' convert vertical angle to radians
sgVertRad = sgVertDeg * 4 * Atn(1) / 180
'
' compute final answer
sgHorDist = (sgK * sgS * Cos(sgVertRad) + sgC) * Cos(sgVertRad)
'
' display in PictureBox
picResults.Cls
picResults.Print "Distance= "; Round(sgHorDist, 3)
'
' move user prompt back to the stadia interval textbox
Call txtStadiaInterval.SetFocus

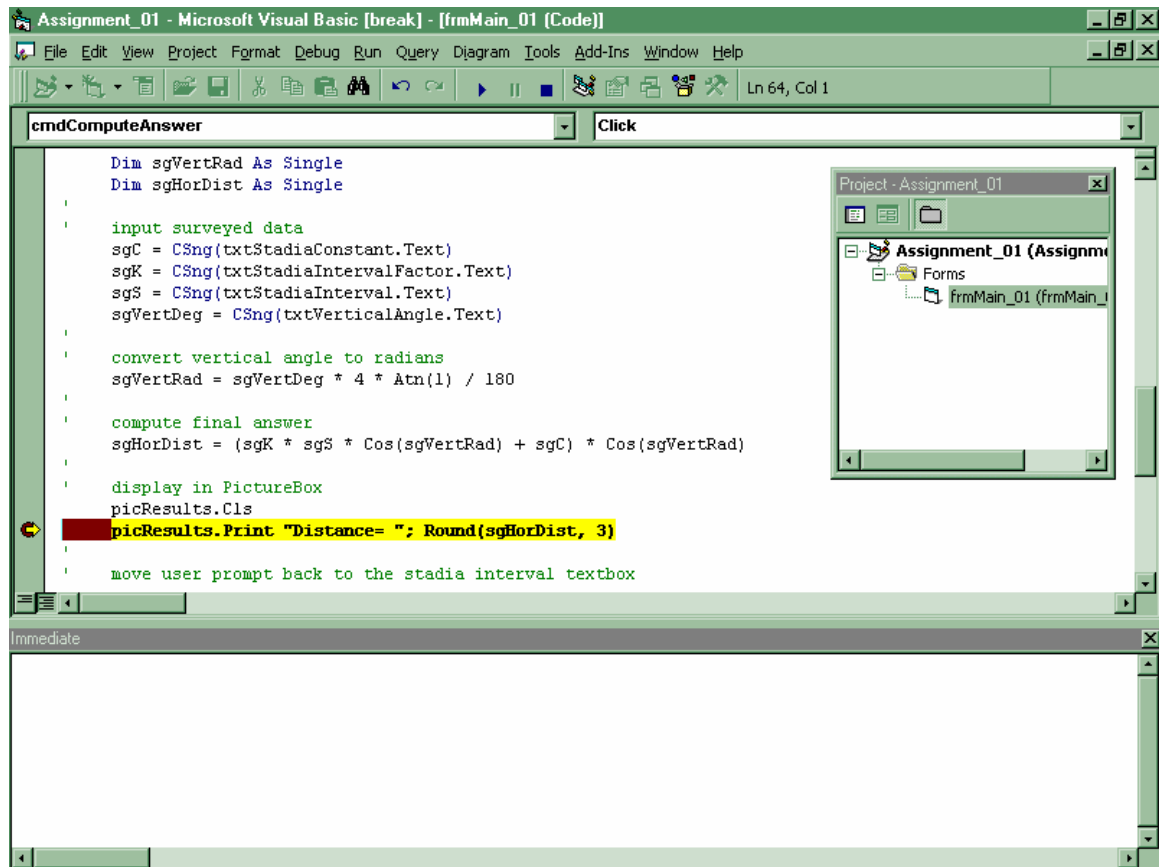
End Sub

Private Sub cmdExitProgram_Click()
    ' terminate execution
End Sub
```

The line `picResults.Print "Distance= "; Round(sgHorDist, 3)` is highlighted in red, and a red dot is visible in the left margin next to it. The Properties window on the right shows the properties for the `cmdComputeAns` control, including `Appearance`, `BackColor`, `Caption`, `CausesValidation`, `Default`, `DisabledPicture`, `DownPicture`, `DragIcon`, `DragMode`, `Enabled`, `Font`, `Height`, `HelpContextID`, `Index`, `Left`, `MaskColor`, `MouseIcon`, `MousePointer`, `OLEDropMode`, `Picture`, and `RightToLeft`.

Now, run your program (**F5**).

Your program should stop when it attempts to execute the print instruction.



When your program suspends its operation, VB goes into **Break Mode**. The word **[Break]** is displayed in the title bar of the project (The word **[run]** used to be there).

The line that will be executed next is highlighted in yellow. Please note: the highlighted line has not been executed yet.

While the program execution has been paused, you can now determine the values of any or all variables that are of interest to you. There are numerous ways to do this.

An **intermediate window** is provided below the code window. The intermediate window can be used to instruct VB to do something. For now, we will focus on instructing VB to print out the value of a variable, allowing us to check and see if it is the value we expect it to be.

Following our example, let's look at the value of the vertical angle after it has been converted to radians by the program. We know the correct answer should be:

$$\alpha = (-19.75) \left(\frac{\pi}{180} \right) = -0.344702527.....$$

Within the intermediate window, instruct the program to print the value of a program variable. For example:

```
print sgvertrad
```

The program should correctly display the current value of the vertical angle (radians) as

-0.3447025

If your result is different, something is wrong with your code! Debugging will be necessary.

The screenshot shows a Visual Basic IDE with a code window and an immediate window. The code window contains the following code:

```
Dim sgVertRad As Single
Dim sgHorDist As Single

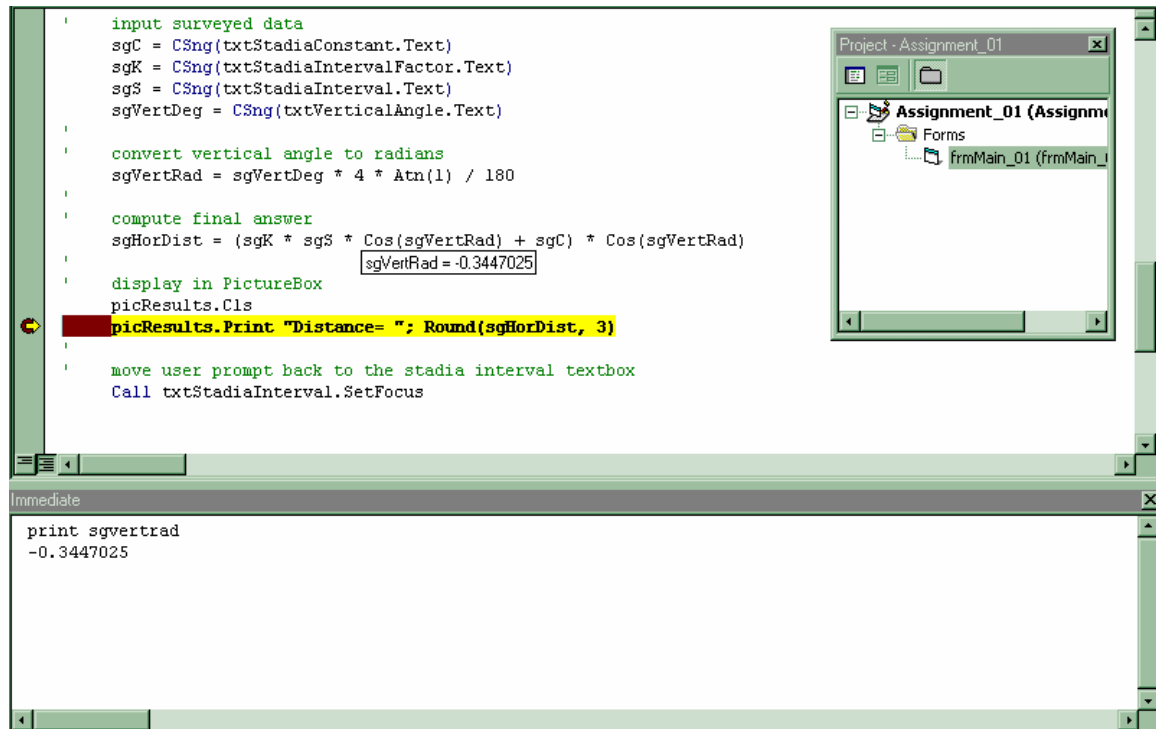
'
' input surveyed data
sgC = CSng(txtStadiaConstant.Text)
sgK = CSng(txtStadiaIntervalFactor.Text)
sgS = CSng(txtStadiaInterval.Text)
sgVertDeg = CSng(txtVerticalAngle.Text)
'
' convert vertical angle to radians
sgVertRad = sgVertDeg * 4 * Atn(1) / 180
'
' compute final answer
sgHorDist = (sgK * sgS * Cos(sgVertRad) + sgC) * Cos(sgVertRad)
'
' display in PictureBox
picResults.Cls
picResults.Print "Distance= "; Round(sgHorDist, 3)
'
' move user prompt back to the stadia interval textbox
```

The immediate window at the bottom shows the output of the `print sgvertrad` command:

```
print sgvertrad
-0.3447025
```

In the background, a project window titled "Project - Assignment_01" is visible, showing a folder structure with "Forms" and "frmMain_01 (frmMain_01)".

Visual Basic also provides a feature involving **DataTips** windows. Simply hover the cursor over a variable name in the code and the current value of the variable will be displayed as shown below.



When debugging your program, you typically know what value should be associated with many variables. In break mode, simply hover the cursor over each variable name in your code listing and verify that the program variable has the proper value. Detecting incorrect variable values is an essential step in debugging programs.

Now, press the **F5** key to instruct the program to continue executing the code.

8.0 Modifying Your Program (Assignment #1)

Based on what you have now learned, it is time to modify your program. Complete Assignment #1 as listed on the course web site and submit it before the end of the lab session.