Chapter 10

Drawing a Complete Assembly

In this chapter, you will learn how to use the following VBA functions to World Class standards:

- **Beginning a New Visual Basic Application**
- **Opening the Visual Basic Editor in AutoCAD**
- **Laying Out a User Input Form in Visual Basic**
- **Creating and Inserting an Image into a Form in Visual Basic**
- **Insert a Label into a Form**
- **Insert a Textbox into a Form**
- **Insert Command Buttons into a Form**
- **Adding a Copyright Statement to a Form**
- **Adding Comments in Visual Basic to Communicate the Copyright**
- **Declaring Variables in a Program with the Dimension Statement**
- **Setting Variables in a Program**
- **Using If-Then Statements when Assigning Variables**
- **Inputting the Code to Set a System Variable**
- **Load AutoCAD Linetypes**
- **Inputting the Code to Create and Set Layers**
- **Inputting the Code to Draw in Visual Basic**
- **Arraying the Block in VBA**
- **Continuing to Draw in VBA**
- **Mirroring the Foundation in VBA**
- **Copying and Moving the Sillplate in VBA**
- **Resetting the Data with the cmdClear Command Button**
- **Exiting the Program with the cmdExit Command Button**
- **Executing a Subroutine with the cmdDraw Command Button**
- **Executing a Subroutine with the cmdPickPoint Command Button**
- **Inserting a Module into a Visual Basic Application**
- **Running the Program**

Beginning a New Visual Basic Application

After learning how to draw multiple orthographic views of products, we will make an assembly drawing of a foundation which has multiple parts. We need every Visual Basic programming skill such as creating forms, drawing arcs, circles and lines, mirroring, arraying and copying to complete the project efficiently. When completed, we will have the ability to continue with other equally large drawings, so that we can finish entire sets of drawings in a few hours where previously the files could take us days to create. At World Class CAD, we have staff members and certified designers who do just that. They develop software that finalizes 90% of the drawings in a project. The assignment in this chapter is an important step in developing the capability to accomplish larger applications.

At the beginning of every chapter, we will start a new Visual Basic Application project, use a sketch to determine the extent of what the program will do, create the form and then write the code. Once the code is finished, we will run the program and multiple orthographic views will appear on the graphical display with dimensions.

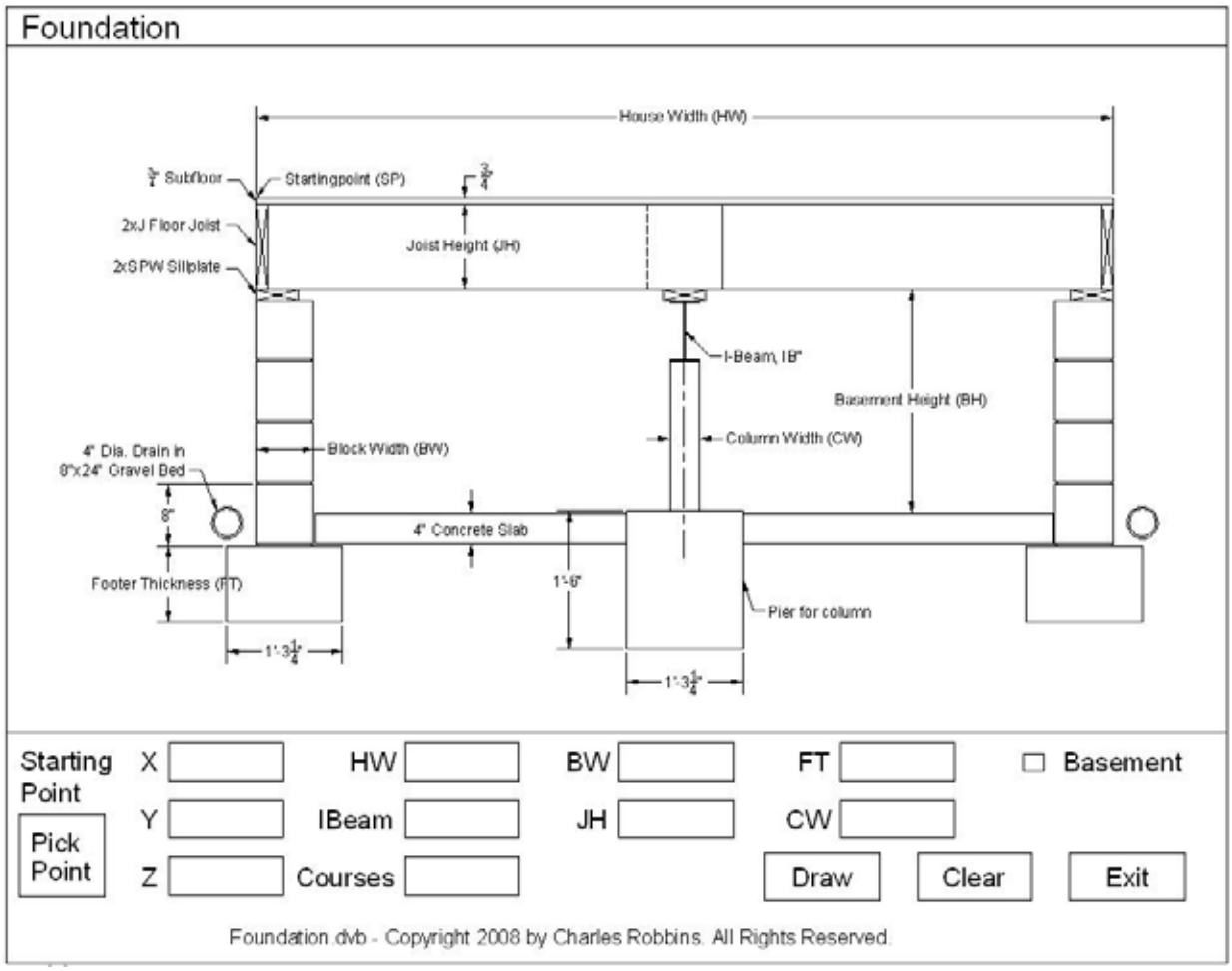

Figure 10.1 – Rough Sketch of the Foundation Form

Remember, that all programming projects begin with one or more sketches, with one portraying the part, detail, or assembly and the other being the user input form. In this Visual Basic Project, the Foundation program, we will be running a user input form inside the AutoCAD application, so we need to sketch the structure of this special dialogue box. We will name the Input form, **Foundation Program**. We will place ten textboxes on the bottom of the form to key in the starting point of the Foundation, and the House Width (HW), the Beam Width (BW), Footer Thickness (FT), I-Beam Height, Joist Height (JH), Column Width dimensions. The last text box allows the user to input the number of courses of block in the foundation. On the top of the form, we will place an image of the Foundation. We will have four command buttons, **Draw**, **Clear, Exit** and **Pick Point**. On the bottom of the form, we will write the copyright statement using another label. In this presentation, we can help ourselves by being as accurate as possible, by displaying sizes, fonts, colors and any other specific details which will enable us to quickly create the form. From the beginning of inserting the form into the project, we need to refer to our sketch. The sketch of the form is shown in Figure 10.1.

Opening the Visual Basic Editor in AutoCAD

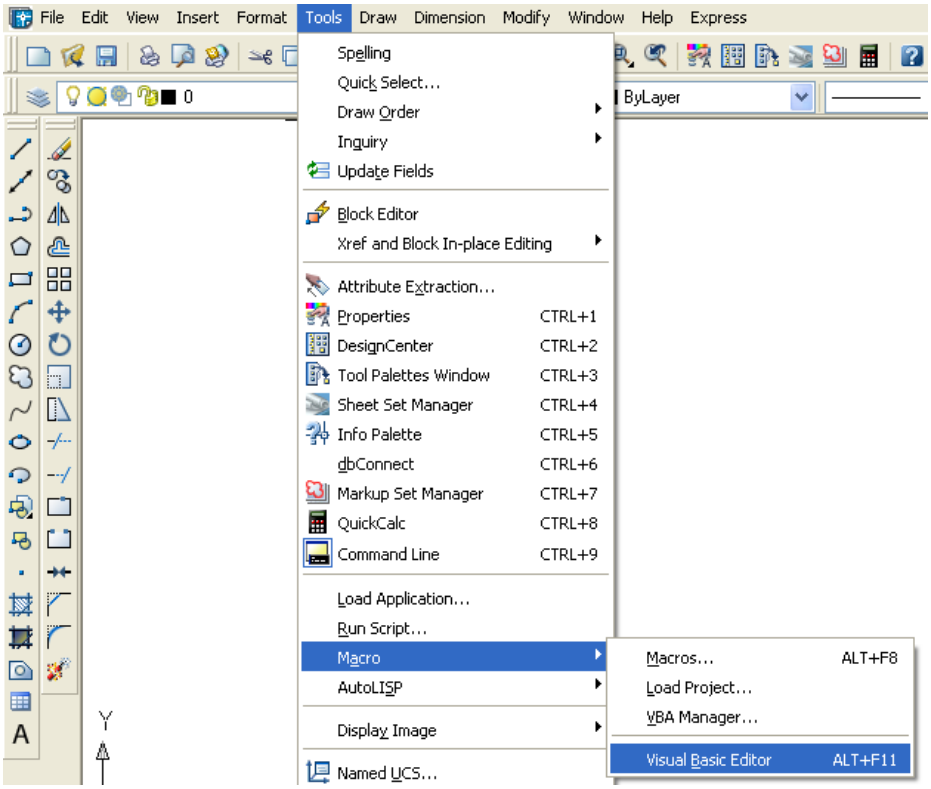

Figure 10.2 – Launching the Visual Basic Editor

Opening the Visual Basic Editor in AutoCAD is essential to creating the program to automate the drawing process. In this version of the World Class CAD – Visual Basic Applications for AutoCAD, we are using AutoCAD 2008, but we just finished using all the programs in this text with a group programming in AutoCAD 2000. Their drawings were automatically made just as efficiently as if they were using the most recent version of the Autodesk software.

Now, select Tools on the Menu bar; then pick Macro and choose the Visual Basic Editor. Look to the right of the phrase, Visual Basic Editor and the shortcut keys Alt – F11 is noted. For quick launching of the editor, press Alt – F11.

The Visual Basic Editor will appear on the computer desktop as a new program application. Looking down on the computer's Taskbar, we can see the AutoCAD and Microsoft Visual Basic Editor program tabs. Just single click either program tab to switch between the applications. However, if we close the AutoCAD drawing, unlike a stand alone version of Visual Basic, the Visual Basic Editor will also close.

For those individuals with previous Visual Basic experience, the Visual Basic Editor in AutoCAD has the same layout as in other VB programs. The Menu Bar contains tools for our use as well as the four toolbars shown in Figure 10.4, which are Standard, Debug, Edit and Userform. Presently, only the Standard toolbar is showing. On the left side of the workspace is the Project menu, which shows the files pertaining to this project. Below the Project menu is the Properties pane. Being familiar with the Properties tool in AutoCAD, makes using this device simple.

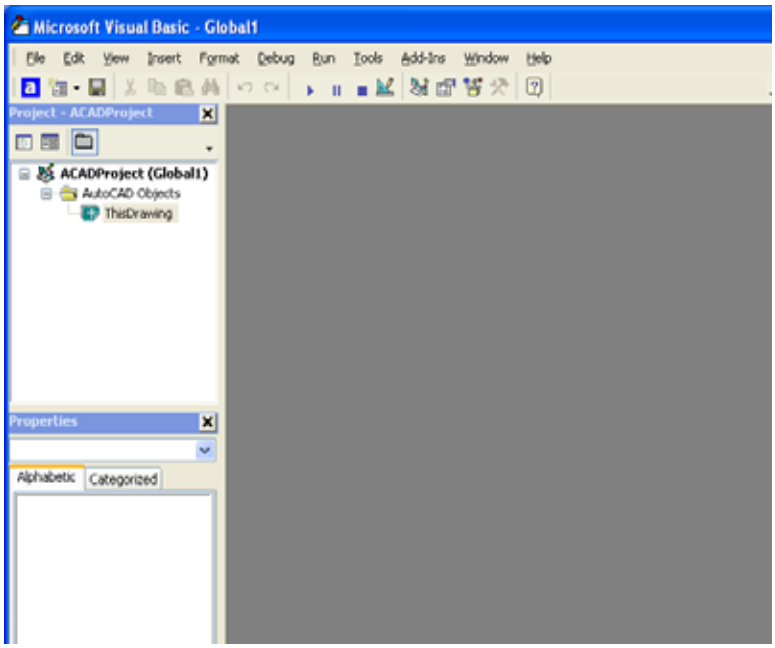

Figure 10.3 – The Visual Basic Editor

Figure 10.4 – Toolbars in the Visual Basic Editor

With the Visual Basic Editor open, select **File** on the Menu Bar and select **Save Project**. Remember, we have a folder on either the desktop or in the My Documents folder called "VBA Programs". Save the project with the filename "Foundation". The file has an extension called *dvb* which means DCL and Visual Basic programs as shown in Figure 10.5.

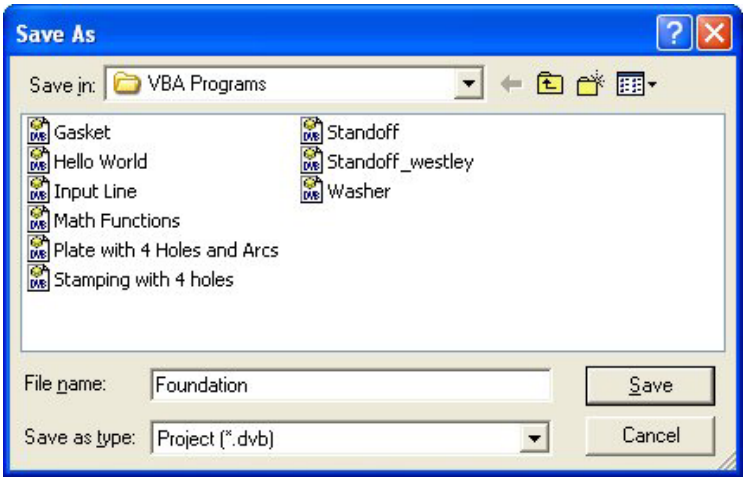

Figure 10.5 – Saving the Foundation Program

Laying Out a User Input Form in Visual Basic

Now that we have an idea of what the dialogue box in our program will look like, select the **Insert UserForm** button on the Standard toolbar to insert a new form as shown in Figure 10.6. Instantaneously, the once grey work area is changed to contain our UserForm1. A Form folder with Userform1 is now in the Project menu and the Properties pane contains the attributes associated with UserForm1 as shown in Figure 10.7.

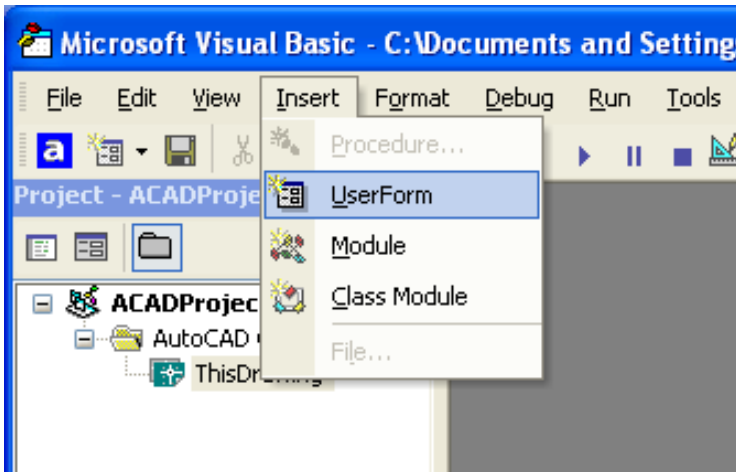

Figure 10.6 – Inserting a User Form

Change the name of the user form to frmFoundation. We use the frm prefix in front of all of the form names in Visual Basic. Change the background of the form to light blue by setting the BackColor in the Properties Pane on the left side of the Visual Basic Application window to "&H80000013&".

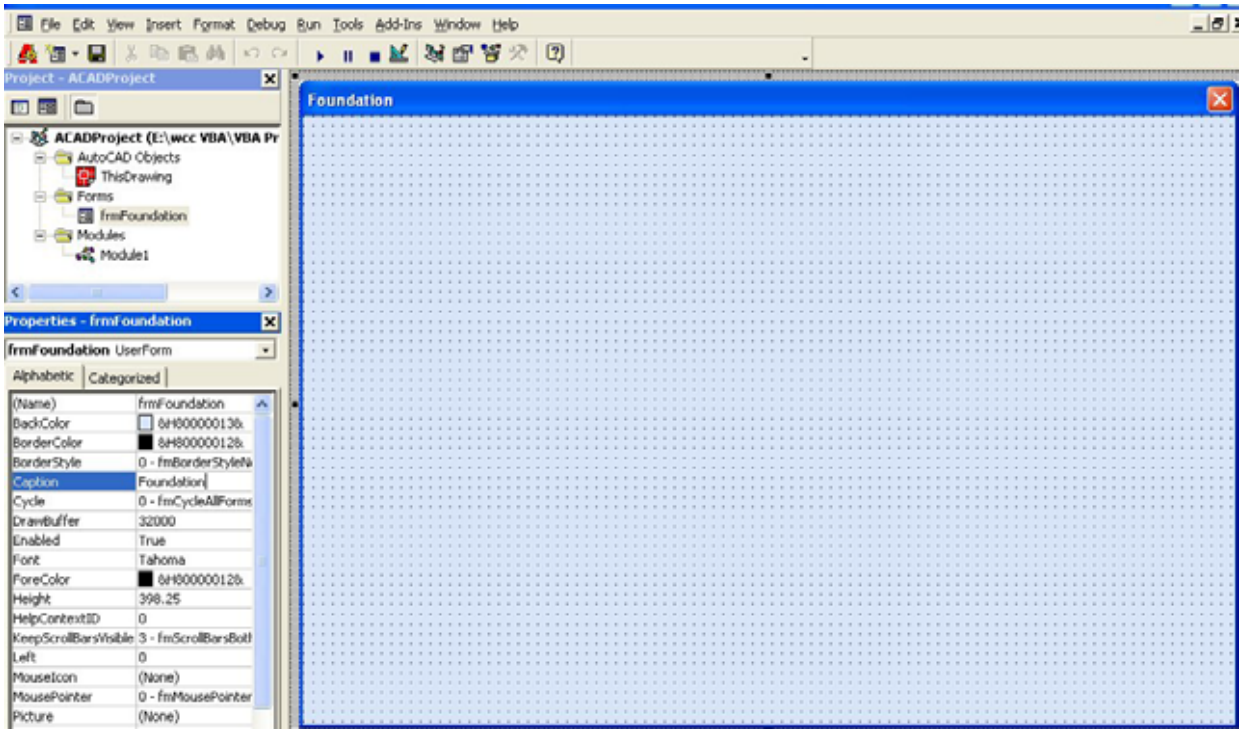

Figure 10.7 – Designing the Foundation Form in Visual Basic

Next, we will change the **Caption** in the Properties pane to **Foundation** to agree with the sketch in Figure 10.1. Go ahead and change the form in two other aspects, Height and Width.

Alphabetic	
(Name)	frmFoundation
BackColor	&H80000013&
Caption	Foundation
Height	480
Width	570

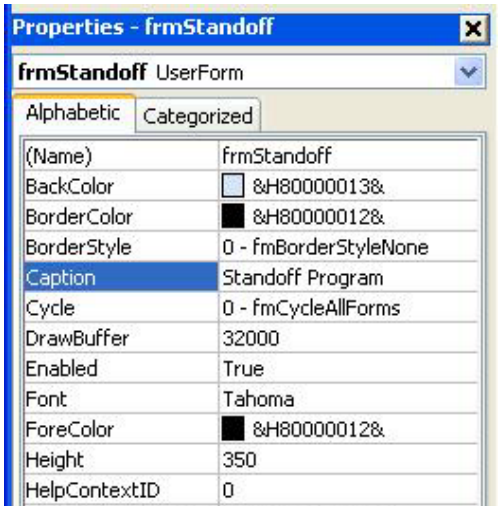

Figure 10.8 – Setting the Caption

The form will change in size to the height and width measurement. The background color will change to a light blue. There are many more attributes in the Properties pane that we will use on future projects.

In previous chapters, we set the Font and Font size for the labels, textboxes and command buttons after creating those specific interfaces. If we set the Font to Arial, Bold and the Font size to 14 on the form, then all of the labels, textboxes and command buttons that we insert from the Control Toolbox will already be set to those attributes.

On the left side of the Visual Basic Editor, locate the property that controls the font and font size in the Properties window. When highlighting the row for Font, a small command button with three small dots appears to the right of the default font name of Tahoma. Click on the three dotted button to open the Visual Basic Font window.

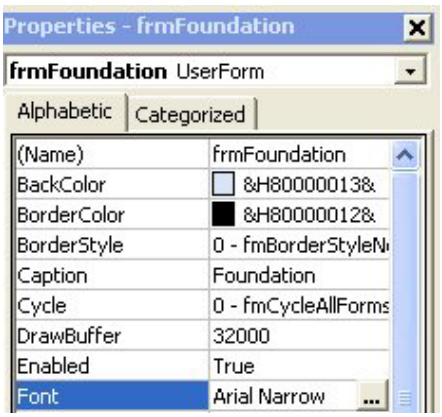

Figure 10.9 – Changing the Font

We will select the Arial Narrow font, Bold font style and 14 size for this project to agree with the initial sketch if the user input form. When we adjust the attributes for the label, these changes do not alter globally for the other objects on the form. If we wish to underline the text or phrase in the label, add a check to the Underline checkbox in the Effects section of the Font window. When we finish making changes to the font property, select the OK command button to return to the work area.

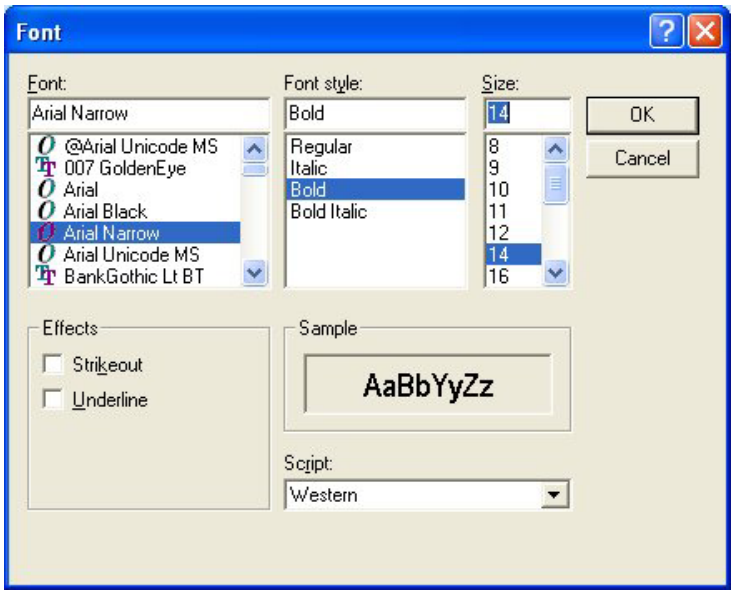

Figure 10.10 – The Font Window in Visual Basic

Creating and Inserting an Image into a Form in Visual Basic

As in previous chapters, this form will have a picture of the part that we will create automatically, so we need to make a drawing of part in AutoCAD. Dimension the drawing as we do in any other drawing, but we will use the Edit Text tool to remove the actual dimension and write in the variable name that matches the textbox label. In Figure 10.11, we show dimensions that correspond with the House Width (HW), the Beam Width (BW), Footer Thickness (FT), I-Beam Height, Joist Height (JH), and Column Width (CW) textboxes. When the drawing is finished, we need to save the drawing as an image file. Use the **Saveimg** command to save file on the VBA Programs folder. Create a folder named Images in the VBA Programs folder and save the file as the same name as the program for matching purposes, Foundation. We saved the file as a Bitmap with a width of 552 pixels and a height of 318 pixels.

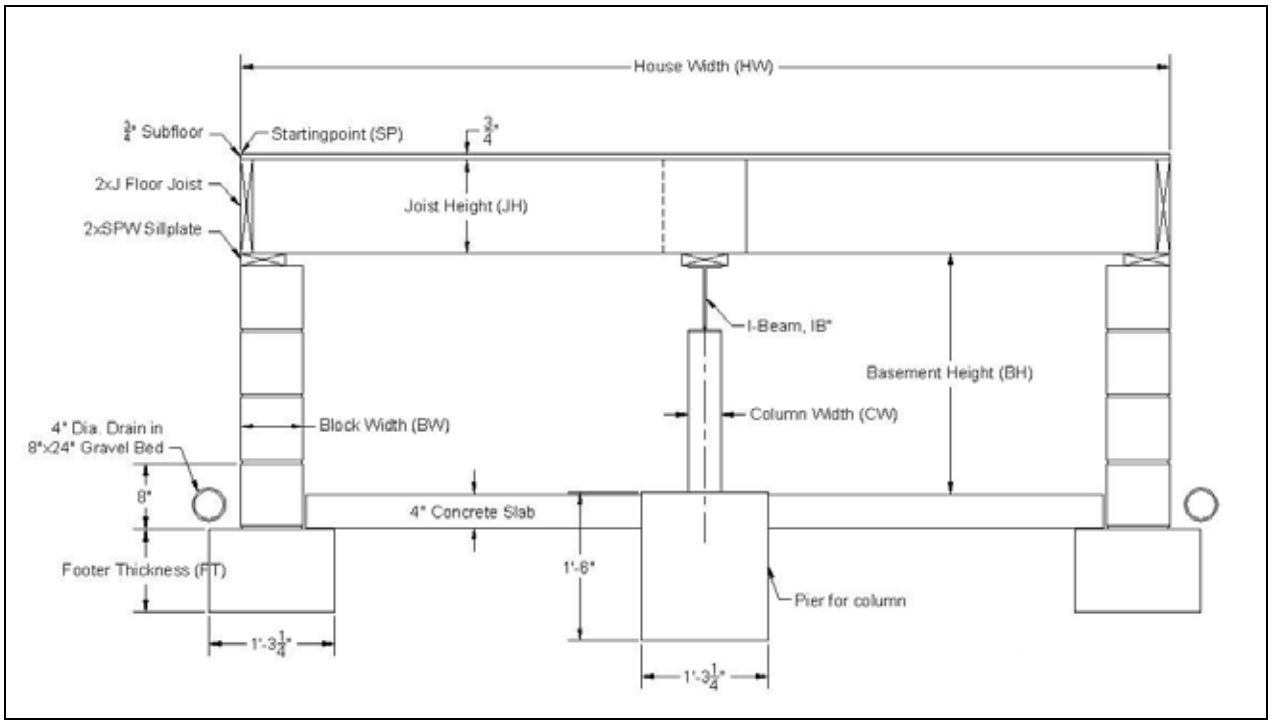

Figure 10.11 – Creating the Foundation Form Image in AutoCAD

On the control toolbox, select the Image tool and then draw a rectangular box on the form in the upper right corner as shown in Figure 10.13. After outlining the size of the image, we will direct the program to the folder and filename of the digital image. In the Properties Image pane, select the attribute named Picture. With the mouse, select the three dot box in the empty cell to the right of Picture. The Load Picture window appears on the screen. Go to the VBA Programs folder and then the Images folder. Select the file, Foundation and it will appear in the picture frame.

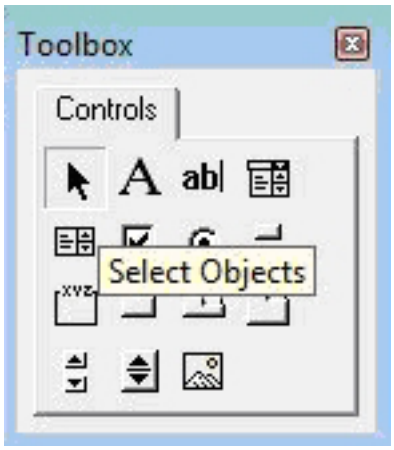

Figure 10.12 – The Control Toolbox

In the Properties pane set the image name to ImgFoundation, the width to 552 and the height to 318. The image will finally appear as shown in Figure 10.14.

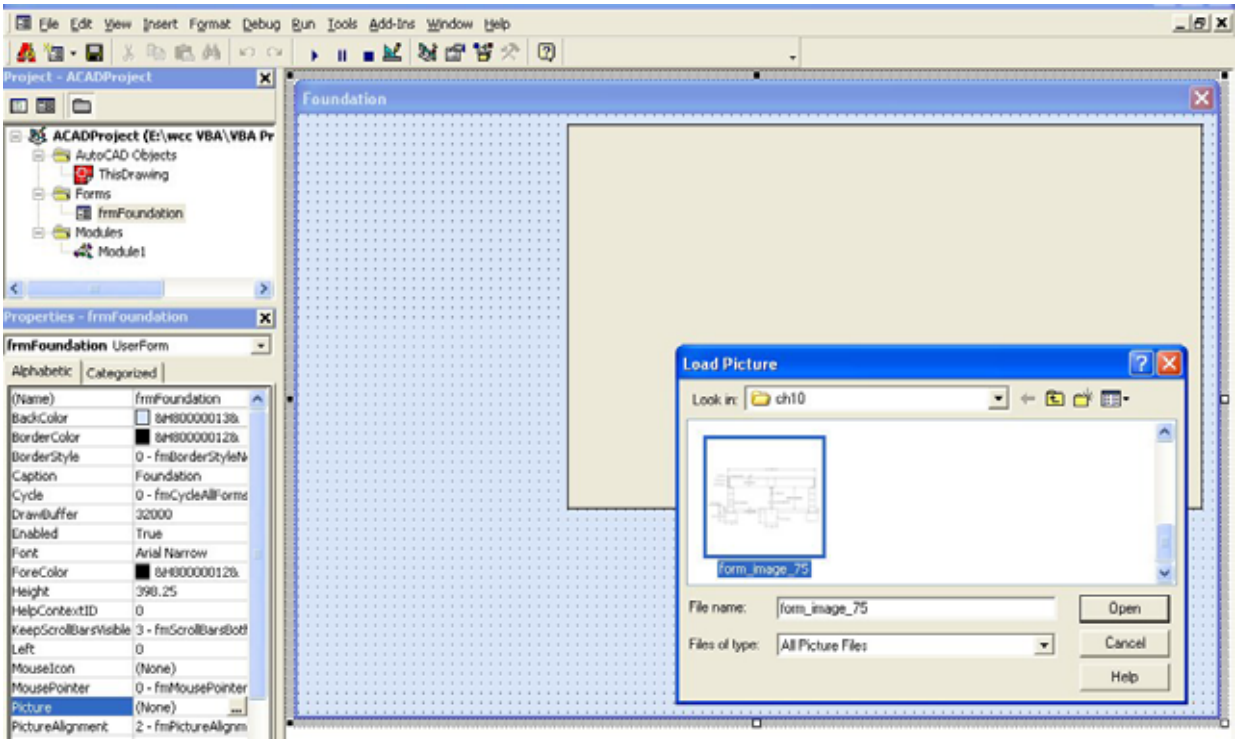

Figure 10.13 – Placing an Image on the Form

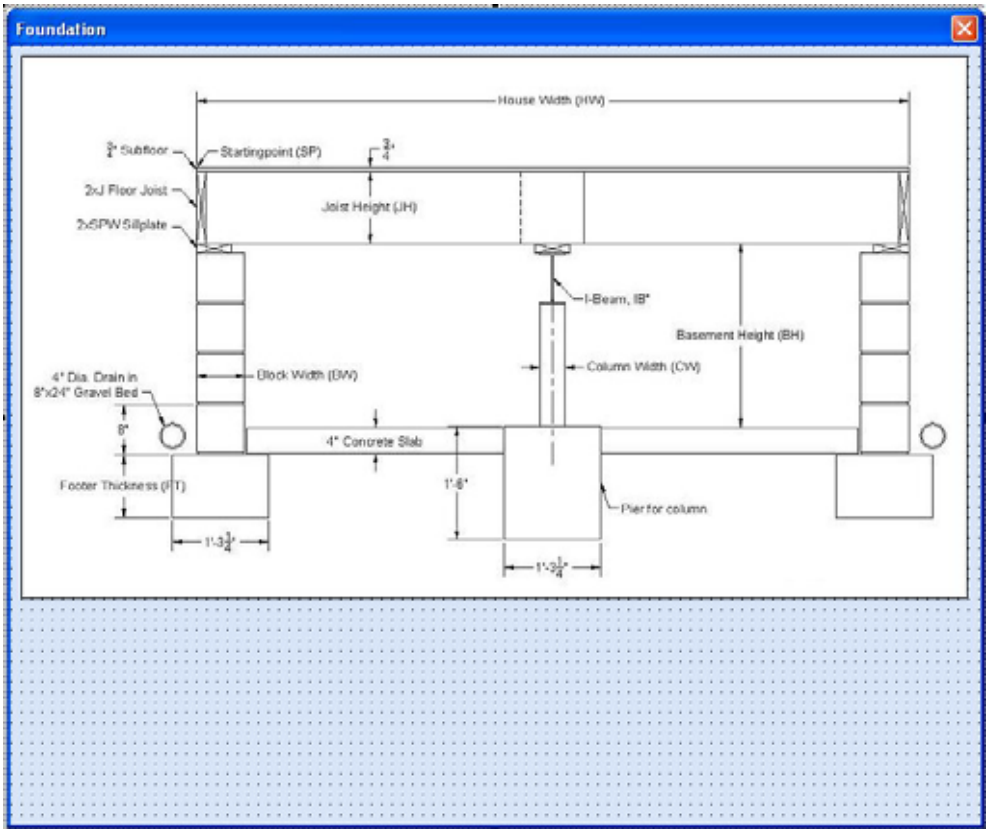

Figure 10.14 – Placing an Image on the Form

Inserting a Label into a Form

A good form is easy to figure out by the user, so when we are attempting to provide information on the window that will run in AutoCAD; we add labels to textboxes to explain our intent. Press the Label (A) button on the Control Toolbar to add a label. To size the label area, click on the upper left area of the form and hold down on the left mouse button, draw the dotted label box as shown in the sketch.

When the first label is done, the background color of the label matches the background color of the form. In many cases that effect is visually pleasing to the eye, versus introducing another color. Both color and shape will direct the user in completing the form along with the explanation we place on the window to guide the designer in using the automated programs. Use colors and shape strategically to communicate well.

For the first label, set the name as **lblStartingpoint** and the caption as Startingpoint. The width of the textbox is 36 and the height is 48. For a label with more than one line of text, left justify the text.

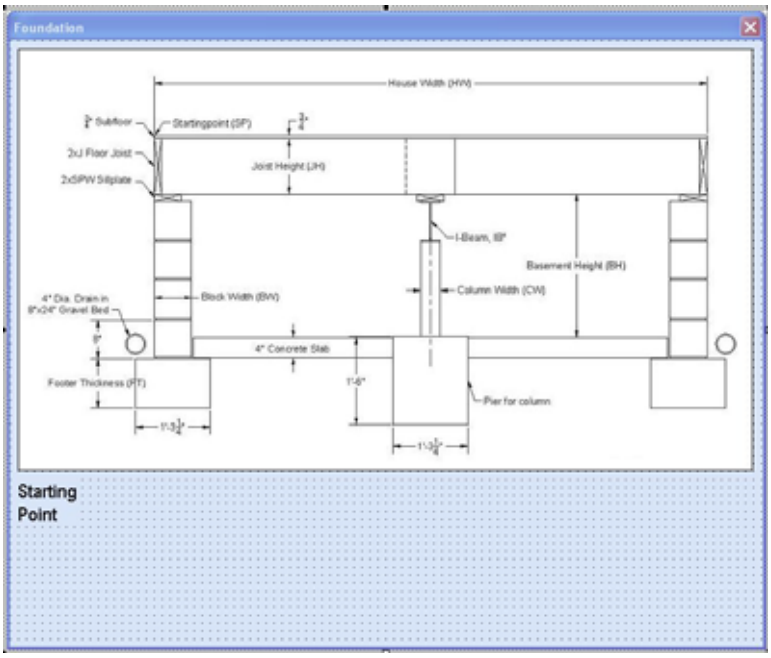

Figure 10.15 – The Finished Label on the Form

Inserting a Textbox into a Form

A textbox is used so that a user of the computer program can input data in the form of words, numbers or a mixture of both. Press the TextBox (ab) button on the Control Toolbar to add a textbox to the form. To size the textbox, click on the upper left area of the form and hold down on the left mouse button, draw the dotted textbox as shown in Figure 10.16.

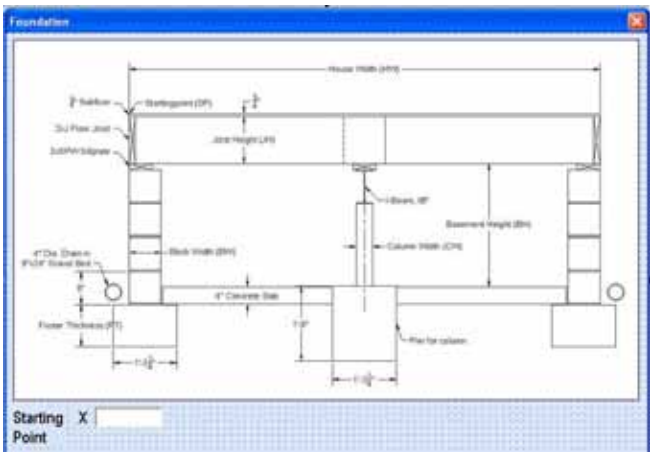

Figure 10.16 – Placing a TextBox on the Form

We will name the textbox using the three letter prefix txt followed by the name or phrase of the tool. For our first textbox, the name is **txtSpX.**

Alphabetic	
(Name)	txtSpX
Font	Arial, 12
Height	18
Width	60

We place a Label using a common Visual Basic naming convention **lblSpX** just to the left of the Textbox. The Caption for the Label will be **X.** On all of the labels that are just to the left of the Textboxes, we will align the text to the right by setting the **TextAlign** property to right align.

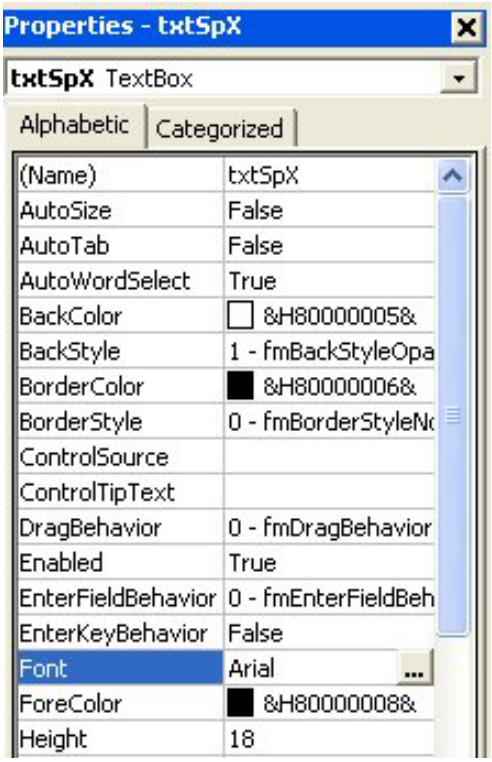

Figure 10.17 – Changing the (Name) to txtName

We will add another TextBox named **txtSpY** under the first one and the Label to the left of the textbox is called **lblSpY.** The Caption for the Label will be **Y.**

We will add yet another TextBox named **txtSpZ** under the first one and the Label to the left of the textbox is called **lblSpZ.** The Caption for the Label will be **Z.**

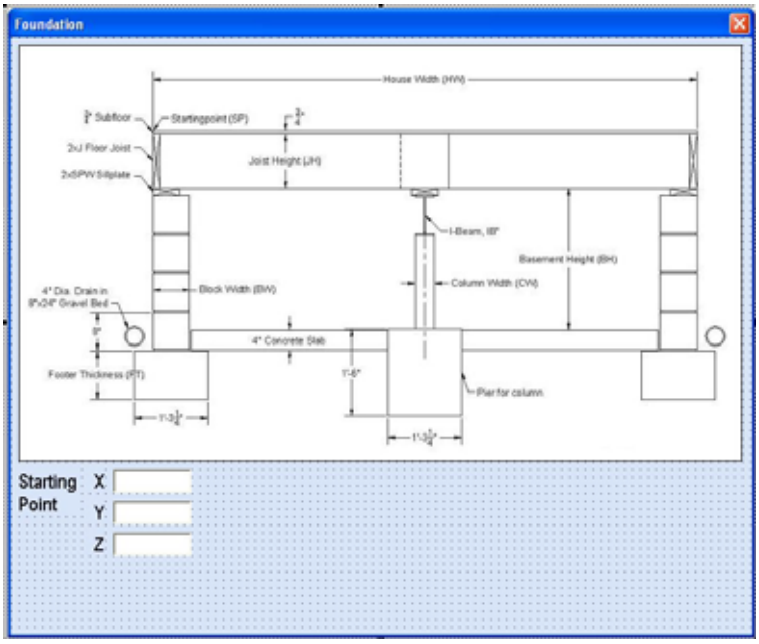

Figure 10.18 – Adding the Y and Z Textboxes

We will add seven more textboxes to the form named **txtHouseWidth, txtBlockWidth, txtFooterThickness, txtIBeam, txtJoistHeight, txtCourses** and **txtColumnWidth.** The labels to the left of the textbox are called **lblHouseWidth, lblBlockWidth, lblFooterThickness, lblIBeam, lblJoistHeight, lblCourses** and **lblColumnWidth.** The Captions for the Labels are shown in Figure 10.19.

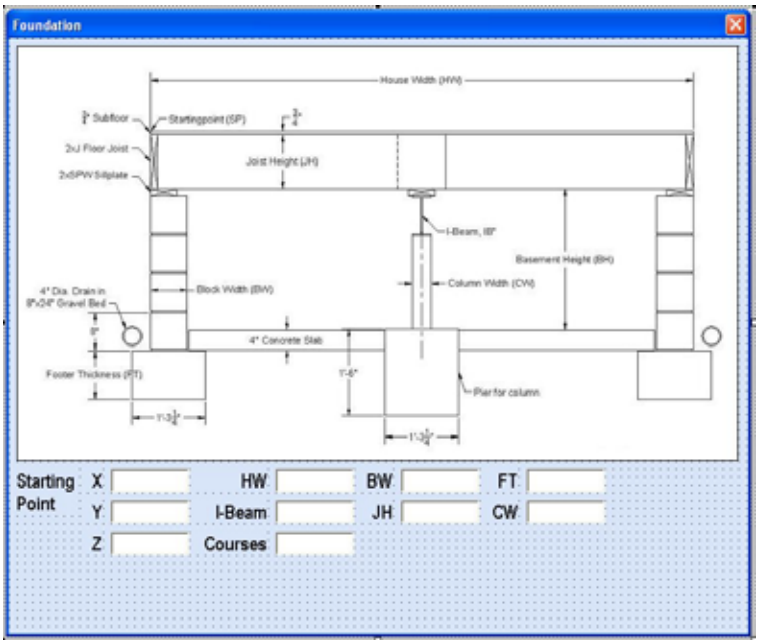

Figure 10.19 – Adding the Last Six Textboxes

Inserting a Checkbox into a Form

We will use a checkbox to indicate an affirmative (yes) or negative (no) response by the user to whether there is a basement floor. By checking the box, the program will insert a 4 inch thick concrete floor. The checkbox tool is on the control toolbox as a checkbox. Select the tool and draw a rectangle on the form in the position shown in the sketch. The checkbox has a label already associated with the function, so type basement for the name. The checkbox should appear as shown in Figure 10.20.

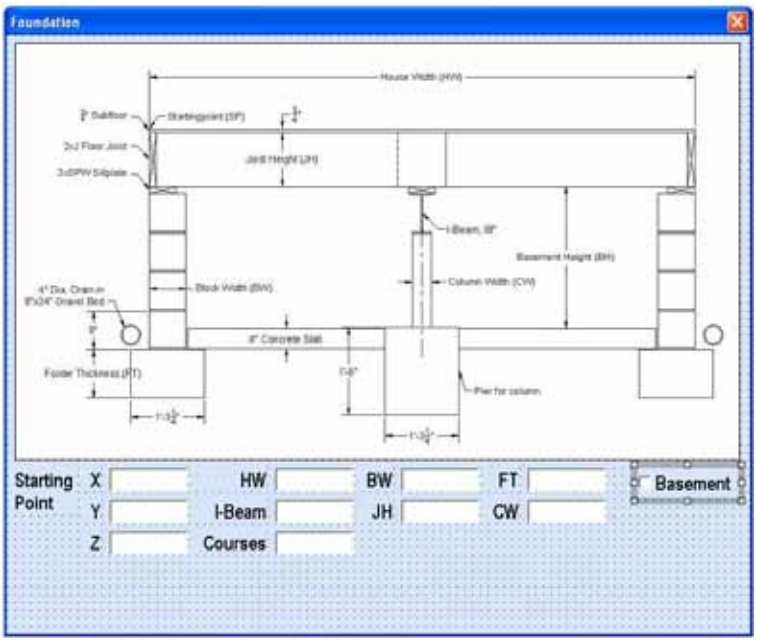

Figure 10.20 – Adding the Basement Checkbox

Inserting Command Buttons into a Form

A command button is used so that a user will execute the application. Press the Command button on the Control Toolbar to add a command button. To size the label area, click on the upper left area of the form and hold down on the left mouse button, draw the command button as shown in Figure 10.21.

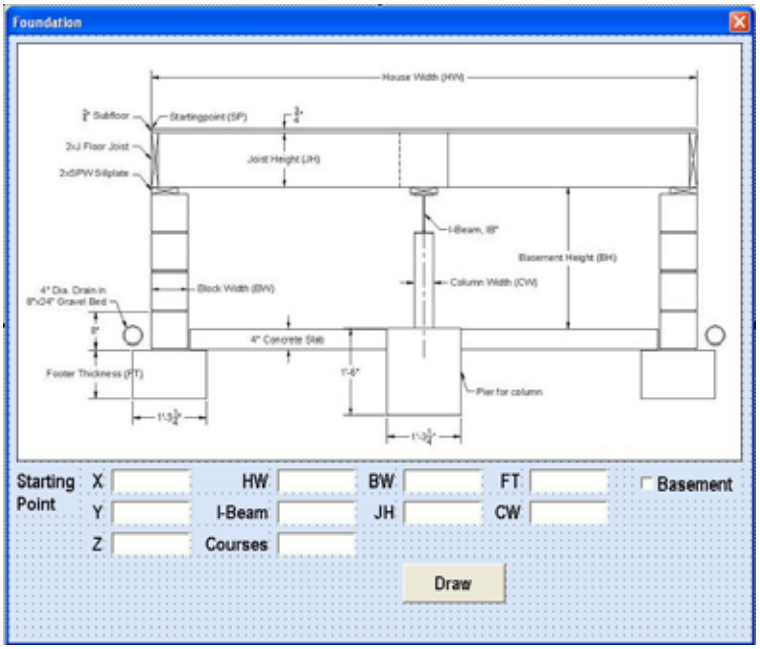

Figure 10.21 – Insert a Command Button onto a Form

We will name the command button using the name is **cmdDraw.**

Alphabetic	
(Name)	cmdDraw
Caption	Draw
Font	Arial
Height	30
Width	78

The font we want for the Command Button is 16 point, Arial Bold. When highlighting the row for Font, a small command button with three small dots appears to the right of the font name of Arial. Click on the three dotted button to open the Visual Basic Font window. Make the changes as we did before and press OK to save the property.

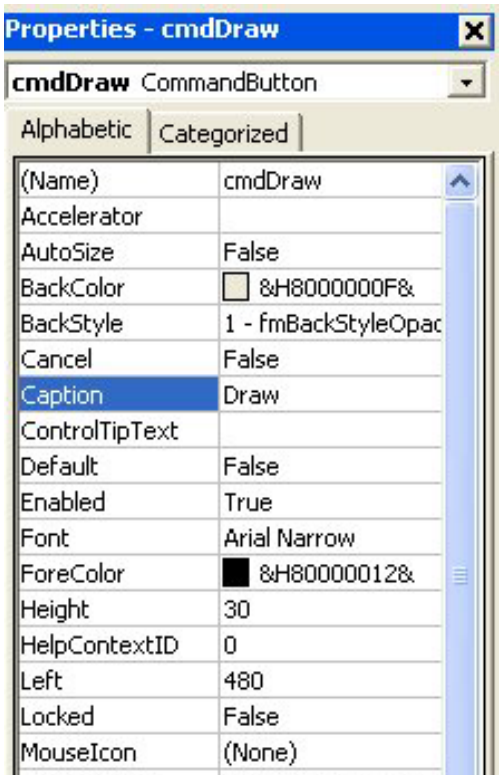

Figure 10.22 – Changing the (Name) to cmdDraw

Add a second Command button; named cmdClear is for clearing the Starting point's X, Y, Z coordinates, AF, ID, and Length textboxes. The third command button is to exit the program. When the user presses the Exit command button, the application closes and full control of the manual AutoCAD program returns to the user. Notice the equal spacing between the command buttons gives a visually friendly appearance.

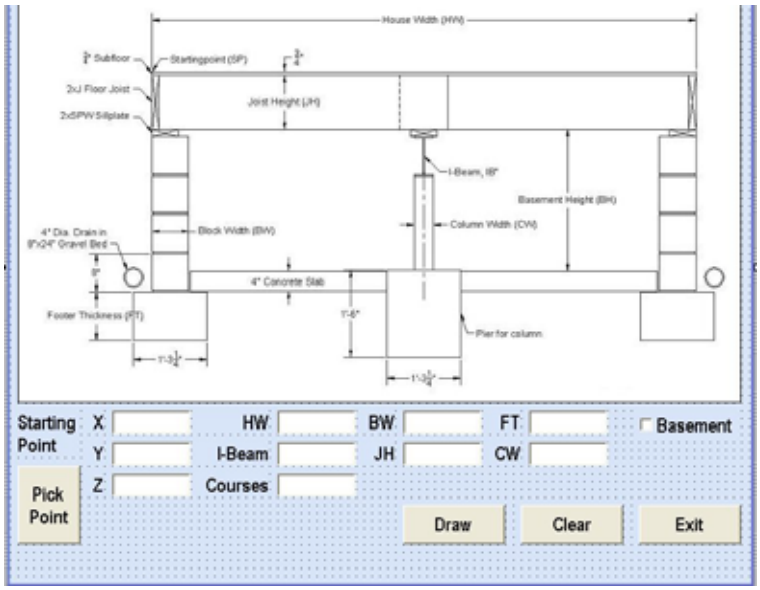

Figure 10.23 – Insert Two More Command Buttons

The fourth command button is Pick Point, which we will name cmdPickPoint. We draw the button as shown in Figure 10.23 and after typing Pick for the caption, press Shift Enter and type Point on the second line. Center the text. When we code for this command button, we will allow the user to select a point on the graphical display and the X, Y and Z coordinates will appear in their specific textboxes.

Adding a Copyright Statement to a Form

At the beginning of a new program, we will expect to see an explanation or any special instructions in the form of comments such as copyright, permissions or other legal notices to inform programmers what are the rules dealing with running the code. Comments at the opening of the code could help an individual determine whether the program is right for their application or is legal to use. The message box is a great tool when properly utilized to inform someone if they are breaking a copyright law when running the code.

Finish the form with the following copyright information.

Foundation Program.dvb - copyright (c) 2008 by Charles Robbins. All Rights Reserved.

If there are special rules or instructions that the user needs to know, place that information on the bottom of the form.

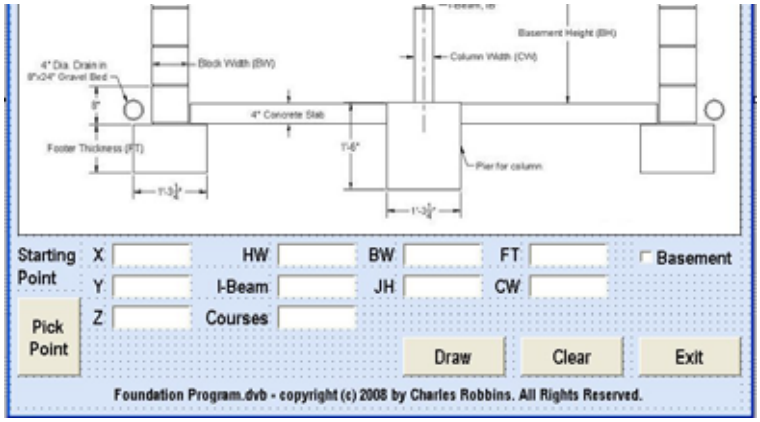

Figure 10.24 – Adding a Copyright Statement

Now that the form is complete, we will begin to write the code that actually interfaces the content of the form using logic and computations to draw the stamping in the AutoCAD graphical display. We will begin the program with comments and place addition phrases throughout the program to assist ourselves or others in the future when modifying the code.

Adding Comments in Visual Basic to Communicate the Copyright

The comments we placed in the first two lines of the program will inform the individual opening and reading the code, but those user that may run the application without checking, the label on the bottom of the form with the copyright information is a great tool to alert the client to the rules of the program and what will the application do.

To begin the actual coding of the program, double click on the Draw command button to enter the programming list. At the top of the program and before the line of code with **Sub CreateFoundation ()**, place the following comments with the single quote (‘) character. Remember, the single quote character (‘) will precede a comment and when the code is compiled, comments are ignored.

Type the following line of code:

```
Sub CreateFoundation()

'Foundation.dvb.copyright (c) 2008 by Charles Robbins
'This program will draws a Foundation from the Footer to the Subfloor
```

Declaring Variables in a Program with the Dimension Statement

When we are going to use a number, text string or object that may change throughout the life of the code, we create a variable to hold the value of that changing entity. In Visual Basic, the dimension or dim statement is one of the ways to declare a variable at the script of procedure level. The other two ways are the Private and Public statements, which we will use in later chapters.

In Figure 10.25, we have a drawing that shows an X and Y grid for each point we need to draw. In this program, we only need points 1 through 44. To create the points, we have made a grid with the values of x1 through x15 horizontally on the bottom and y1 through y16 vertically on the right. In the program, we will define point P1 as coordinate (x1, y15, z1), point P2 as (x7, y15, z1) and so forth. We have found that points are easily defined using this method and therefore explaining the algebra in the program is simpler.

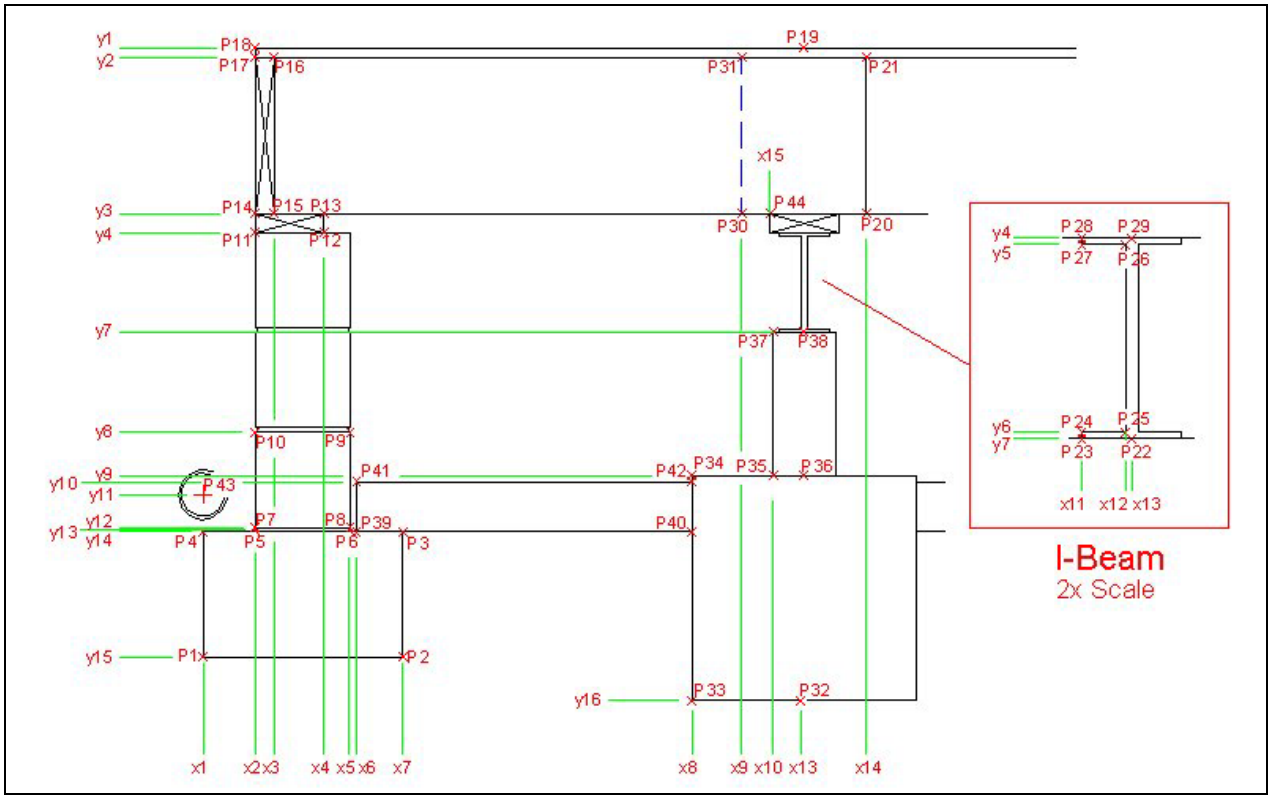

Figure 10.25 – Identifying the Variables for the Foundation Program

In our program, we will declare a variable to enable us to draw lines and circles, a variable for each vertex and a variable for each textbox. As we can see below, the made up name **objCircle** is an AutoCAD Circle by definition and the contrived name **objLine** is a line. To mirror the five lines in the right orthographic view of the drawing, we have made three addition variables, **objSs1, objDrawingObject** and **objMirroredObject**. The first is for creating a selection set, the second is to hold drawing objects and the last is for mirroring the objects. We will cover the selecting and mirroring code later in the chapter.

We will set variables for array (**objArrayedObject**), the linetypes (**objLinetype**) and layers (**objLayers**). Then, we declare variables of the textboxes OD, ID and W as double integers (As Double). For the three dimensions on the drawing we declare **objDimension** as an AutoCAD dimension.

Type the following lines of code after the comment.

```
'identifying variables

Dim objLinetype As AcadLineType
Dim objLayer As AcadLayer
Dim objLine As AcadLine
Dim objArc As AcadArc
Dim objCircle As AcadCircle
Dim objSs1 As AcadSelectionSet
```

```
Dim objSs2 As AcadSelectionSet
Dim objCopiedObject As AcadEntity
Dim objMirroredObject As AcadEntity
Dim objArrayedObject As AcadEntity
Dim objDrawingObject As AcadEntity
Dim objDimension As AcadDimension
Dim HW As Double
Dim BW As Double
Dim FT As Double
Dim IB As Double
Dim JH As Double
Dim CW As Double
Dim CS As Double
Dim pi As Double
Dim FW As Double
Dim P1(0 To 2) As Double
Dim P2(0 To 2) As Double
Dim P3(0 To 2 As Double
Dim P4(0 To 2) As Double
Dim P5(0 To 2) As Double
Dim P6(0 To 2) As Double
Dim P7(0 To 2) As Double
Dim P8(0 To 2) As Double
Dim P9(0 To 2) As Double
Dim P10(0 To 2) As Double
Dim P11(0 To 2) As Double
Dim P12(0 To 2) As Double
Dim P13(0 To 2) As Double
Dim P14(0 To 2) As Double
Dim P15(0 To 2) As Double
Dim P16(0 To 2) As Double
Dim P17(0 To 2) As Double
Dim P18(0 To 2) As Double
Dim P19(0 To 2) As Double
Dim P20(0 To 2) As Double
Dim P21(0 To 2) As Double
Dim P22(0 To 2) As Double
Dim P23(0 To 2) As Double
Dim P24(0 To 2) As Double
Dim P25(0 To 2) As Double
Dim P26(0 To 2) As Double
Dim P27(0 To 2) As Double
Dim P28(0 To 2) As Double
Dim P29(0 To 2) As Double
Dim P30(0 To 2) As Double
Dim P31(0 To 2) As Double
Dim P32(0 To 2) As Double
```

```
Dim P33(0 To 2) As Double
Dim P34(0 To 2) As Double
Dim P35(0 To 2) As Double
Dim P36(0 To 2) As Double
Dim P37(0 To 2) As Double
Dim P38(0 To 2) As Double
Dim P39(0 To 2) As Double
Dim P40(0 To 2) As Double
Dim P41(0 To 2) As Double
Dim P42(0 To 2) As Double
Dim P43(0 To 2) As Double
Dim P44(0 To 2) As Double
Dim x1 As Double
Dim x2 As Double
Dim x3 As Double
Dim x4 As Double
Dim x5 As Double
Dim x6 As Double
Dim x7 As Double
Dim x8 As Double
Dim x9 As Double
Dim x10 As Double
Dim x11 As Double
Dim x12 As Double
Dim x13 As Double
Dim x14 As Double
Dim x15 As Double
Dim y1 As Double
Dim y2 As Double
Dim y3 As Double
Dim y4 As Double
Dim y5 As Double
Dim y6 As Double
Dim y7 As Double
Dim y8 As Double
Dim y9 As Double
Dim y10 As Double
Dim y11 As Double
Dim y12 As Double
Dim y13 As Double
Dim y14 As Double
Dim y15 As Double
Dim y16 As Double
Dim z1 As Double
```

The vertices or points are declared as double integers (As Double) with an array of zero to two (0 to 2). The vertex P1(0) represents the X coordinate, the P1(1) represents the Y coordinate and

P1(2) represents the Z coordinate. Some may think that it is a waste of time to involve the Z-axis in a two dimension drawing, but we will incorporate the Z coordinate for designers that work in all three dimensions. For everyone else, we will just enter zero (0) in the Z coordinate textbox. We will declare points P1 through P18 for the vertices in the drawing in Figure 10.25.

As discussed previously, we have given the Foundation drawing problem a grid, so we declare x1 through x8, y1 through y6, and z1. To compute angles, we declare pi in this program.

Remember, when selecting variable names, they should be a word or a phrase without spaces that represents the value that the variable contains. If we want to hold a value of one's date of birth, we can call the variable, DateofBirth. The keywords Date and Birth are in sentence case with the first letter capitalized. There are no spaces in the name. Some programmers use the underscore character (_) to separate words in phrases. This is acceptable, but a double underscore (__) can cause errors if we do not detect the repeated character.

Using If –Then Statements when Assigning Variables

Many times in the building industry, material is known by a measurement like 2 by 4 yet the actual product does not measure 2 inches by 4 inches, but 1.5 inches by 3.5 inches. When we prompt a user for sizes they can type in the common nomenclature of the item and we will convert to the actual dimension by using an If-then statement.

For instance, if the user types 8 for the block width, in this section of the code we will have the following expression:

```
If BW = 8 Then BW = 7.625
```

In an If-then statement, the sequence starts with a logical argument such as BW=8. If this is true the function will proceed to the **Then** portion and execute that code. If the logical argument is false an **Else** section is added if desired to make a change for a negative response. We will use If-then statements for the block width and joist height variables as shown below.

```
'Assigning values to the variables

pi = 3.14159265358979
HW = txtHouseWidth
BW = txtBlockWidth
FT = txtFooterThickness
IB = txtIbeam
JH = txtJoistHeight
CW = txtColumnWidth
CS = txtCourses
FW = BW * 2
If BW = 8 Then BW = 7.625
If BW = 12 Then BW = 11.625
If JH = 8 Then JH = 7.5
If JH = 10 Then JH = 9.25
```

```
If JH = 12 Then JH = 11.25
If JH = 16 Then JH = 15.25
x2 = txtSpX
x1 = x2 - (FW - BW) / 2
x3 = x2 + 1.5
x4 = x2 + 5.5
x5 = x2 + BW
x6 = x5 + 0.5
x7 = x1 + FW
x13 = x2 + HW / 2
x14 = x13 + 5
x12 = x13 - 0.25
x11 = x13 - 2
x10 = x13 - CW / 2
x15 = x13 - 2.5
x9 = x13 - 5
x8 = x13 - 9
y1 = txtSpY
y2 = y1 - 0.75
y3 = y2 - JH
y4 = y3 - 1.5
y5 = y4 - 0.25
y7 = y4 - IB
y6 = y7 + 0.25
y14 = y4 - CS * 8
y13 = y14 + 0.1875
y12 = y14 + 0.375
y11 = y14 + 3
y10 = y14 + 4
y9 = y10 + 0.5
y8 = y14 + 8
y15 = y14 - FT
y16 = y9 - 18
z1 = txtSpZ
```

We use the same sketch to make the point assignments. Just review the data from Figure 10.25 and make the correct coordinate designation as shown. Remember that P1(0) is the x position, P1(1) is the y position and P1(2) is the z position.

```
'Point assignments

P1(0) = x1
P1(1) = y15
P1(2) = z1
P2(0) = x7
P2(1) = y15
P2(2) = z1
```

P3(0) = x7
P3(1) = y14
P3(2) = z1
P4(0) = x1
P4(1) = y14
P4(2) = z1
P5(0) = x2
P5(1) = y13
P5(2) = z1
P6(0) = x5
P6(1) = y13
P6(2) = z1
P7(0) = x2
P7(1) = y12
P7(2) = z1
P8(0) = x5
P8(1) = y12
P8(2) = z1
P9(0) = x5
P9(1) = y8
P9(2) = z1
P10(0) = x2
P10(1) = y8
P10(2) = z1
P11(0) = x2
P11(1) = y4
P11(2) = z1
P12(0) = x4
P12(1) = y4
P12(2) = z1
P13(0) = x4
P13(1) = y3
P13(2) = z1
P14(0) = x2
P14(1) = y3
P14(2) = z1
P15(0) = x3
P15(1) = y3
P15(2) = z1
P16(0) = x3
P16(1) = y2
P16(2) = z1
P17(0) = x2
P17(1) = y2
P17(2) = z1
P18(0) = x2
P18(1) = y1

P18(2) = z1
P19(0) = x13
P19(1) = y1
P19(2) = z1
P20(0) = x14
P20(1) = y3
P20(2) = z1
P21(0) = x14
P21(1) = y2
P21(2) = z1
P22(0) = x13
P22(1) = y7
P22(2) = z1
P23(0) = x11
P23(1) = y7
P23(2) = z1
P24(0) = x11
P24(1) = y6
P24(2) = z1
P25(0) = x12
P25(1) = y6
P25(2) = z1
P26(0) = x12
P26(1) = y5
P26(2) = z1
P27(0) = x11
P27(1) = y5
P27(2) = z1
P28(0) = x11
P28(1) = y4
P28(2) = z1
P29(0) = x13
P29(1) = y4
P29(2) = z1
P30(0) = x9
P30(1) = y3
P30(2) = z1
P31(0) = x9
P31(1) = y2
P31(2) = z1
P32(0) = x13
P32(1) = y16
P32(2) = z1
P33(0) = x8
P33(1) = y16
P33(2) = z1
P34(0) = x8

P34(1) = y9
P34(2) = z1
P35(0) = x10
P35(1) = y9
P35(2) = z1
P36(0) = x13
P36(1) = y9
P36(2) = z1
P37(0) = x10
P37(1) = y7
P37(2) = z1
P38(0) = x13
P38(1) = y7
P38(2) = z1
P39(0) = x6
P39(1) = y14
P39(2) = z1
P40(0) = x8
P40(1) = y14
P40(2) = z1
P41(0) = x6
P41(1) = y10
P41(2) = z1
P42(0) = x8
P42(1) = y10
P42(2) = z1
P43(0) = x1
P43(1) = y11
P43(2) = z1
P44(0) = x15
P44(1) = y3
P44(2) = z1

Inputting the Code to Set a System Variable

To change a system variable such as the Object Snap Mode, so the Endpoint, Midpoint or other setting cannot interfere with the construction of the orthographic view of the Foundation, we will turn off the Object Snaps. Type **ThisDrawing.SetVariable "osmode", 0** and the system setting for Object Snaps will be turned off.

To change the linetype scale, use the same format, except the variable name is “ltscale” and the new setting is 8.

```
'Set variables

    ThisDrawing.SetVariable "osmode", 0
    ThisDrawing.SetVariable "ltscale", 8
```

Load AutoCAD Linetypes

Before setting a linetype in a layer, we need to load them into the current file. Use the ThisDrawing.Linetypes.Load function with the linetype name followed by a comma and the file holding the linetype definition. Use On Error Resume Next before the expression to avoid errors if the linetype is already loaded.

```
'Load linetypes
   On Error Resume Next
   ThisDrawing.Linetypes.Load "hidden", "acad.lin"
   On Error Resume Next
   ThisDrawing.Linetypes.Load "center", "acad.lin"
```

Inputting the Code to Create and Set a Layer

Many times, we will want to create a layer and then set the layer throughout a program. To create a layer, type **Set objLayer = ThisDrawing.Layers.Add** and in parenthesis place the new layer name in quotes, such as “Foundation”. After making the new layer, set the layer color and linetype by typing **objLayer.Color = acWhite** and **objLayer.Linetype = “Continuous”**. We could make a layer the color blue and with hidden lines if we choose.

In this program, we need to make a Foundation layer for the object with a continuous linetype, a hidden line layer a hidden linetype, a center line layer with a center linetype and a dimension layer with a continuous linetype . We can make as many layers following the format below.

```
'Create and set layer

Set objLayer = ThisDrawing.Layers.Add("Foundation")
   objLayer.Color = acWhite
   objLayer.Linetype = "Continuous"

Set objLayer = ThisDrawing.Layers.Add("Wood")
   objLayer.Color = acBrown
   objLayer.Linetype = "Continuous"

Set objLayer = ThisDrawing.Layers.Add("Hidden")
   objLayer.Color = acBlue
   objLayer.Linetype = "Hidden"
```

To set the layer current, before drawing an entity, we would type:

```
ThisDrawing.ActiveLayer = ThisDrawing.Layers("Foundation")
```

Inputting the Code to Draw in Visual Basic

Now we want to enter the code that will actually draw the two arcs and 4 lines of the block and mortar in AutoCAD Model Space. We use the Set function to draw an arc by typing **Set ObjArc** and giving the center point of the arc, the radius of the arc, then the starting point in radians, and lastly the ending point in radians. We use the Set function to draw a line by typing **Set ObjLine** and then we tell the computer that it will draw in Modelspace by adding a line from point P7 to point P8.

Go ahead and type the following comments and drawing code:

```
'Draw a block with 2 arcs and 4 lines
'Draw 2 arcs

ThisDrawing.ActiveLayer = ThisDrawing.Layers("Foundation")
Set objArc = ThisDrawing.ModelSpace.AddArc(P5, 0.1875, 1.5 * pi, 0.5 * pi)
Set objArc = ThisDrawing.ModelSpace.AddArc(P6, 0.1875, 0.5 * pi, 1.5 * pi)

'Draw the block

Set objLine = ThisDrawing.ModelSpace.AddLine(P7, P8)
Set objLine = ThisDrawing.ModelSpace.AddLine(P8, P9)
Set objLine = ThisDrawing.ModelSpace.AddLine(P9, P10)
Set objLine = ThisDrawing.ModelSpace.AddLine(P10, P7)
```

Drawing and Arraying a Block

We array entities by selecting the object. We zoom all to have all the entities appear in the graphical display, and then use **acSelectionSetLast** to retrieve all six objects. When we array the block, we type:

```
Set objArrayedObject = objDrawingObject.ArrayRectangular(CS, 1, 1, 8, 1, 1)
```

Where the CS is the number of rows in the array, the 1 is the number of columns in the array, the next 1 is the number of levels for 3D arrays. Next we input 8 for the distance between rows, 1 for the distance between columns and 1 for the distance between levels. After the array, we delete the selection set objSs1.

```
'Array the block

ThisDrawing.Application.ZoomAll
Set objSs1 = ThisDrawing.SelectionSets.Add("TempSS")
   objSs1.Select (acSelectionSetAll)

For Each objDrawingObject In objSs1
```

```
Set objArrayedObject = objDrawingObject.ArrayRectangular(CS, 1, 1, 8, 1, 1)
   objArrayedObject.Update
Next
objSs1.Delete
```

Continuing to Draw the Foundation in VBA

Next, we continue to write the code to draw lines and circles before we mirror them across the middle of the foundation. These lines and circles need to be on the appropriate layer and the points need to match the sketch in figure 10.25.

```
'Draw the footer

Set objLine = ThisDrawing.ModelSpace.AddLine(P1, P2)
Set objLine = ThisDrawing.ModelSpace.AddLine(P2, P3)
Set objLine = ThisDrawing.ModelSpace.AddLine(P3, P4)
Set objLine = ThisDrawing.ModelSpace.AddLine(P4, P1)

'Draw the concrete slab

If chkBasement = "True" Then
Set objLine = ThisDrawing.ModelSpace.AddLine(P41, P42)
Set objLine = ThisDrawing.ModelSpace.AddLine(P39, P41)
Set objLine = ThisDrawing.ModelSpace.AddLine(P39, P40)
End If

'Draw the pier

Set objLine = ThisDrawing.ModelSpace.AddLine(P32, P33)
Set objLine = ThisDrawing.ModelSpace.AddLine(P33, P34)
Set objLine = ThisDrawing.ModelSpace.AddLine(P34, P36)

'Draw the sillplate

ThisDrawing.ActiveLayer = ThisDrawing.Layers("wood")
Set objLine = ThisDrawing.ModelSpace.AddLine(P11, P12)
Set objLine = ThisDrawing.ModelSpace.AddLine(P12, P13)
Set objLine = ThisDrawing.ModelSpace.AddLine(P13, P14)
Set objLine = ThisDrawing.ModelSpace.AddLine(P14, P11)
Set objLine = ThisDrawing.ModelSpace.AddLine(P11, P13)
Set objLine = ThisDrawing.ModelSpace.AddLine(P12, P14)

'Draw the joist

Set objLine = ThisDrawing.ModelSpace.AddLine(P14, P15)
```

```
Set objLine = ThisDrawing.ModelSpace.AddLine(P15, P16)
Set objLine = ThisDrawing.ModelSpace.AddLine(P16, P17)
Set objLine = ThisDrawing.ModelSpace.AddLine(P17, P14)
Set objLine = ThisDrawing.ModelSpace.AddLine(P14, P16)
Set objLine = ThisDrawing.ModelSpace.AddLine(P15, P17)

Set objLine = ThisDrawing.ModelSpace.AddLine(P15, P20)
Set objLine = ThisDrawing.ModelSpace.AddLine(P16, P21)

'Draw the subfloor

Set objLine = ThisDrawing.ModelSpace.AddLine(P19, P18)
Set objLine = ThisDrawing.ModelSpace.AddLine(P18, P17)

'Draw the column

Set objLine = ThisDrawing.ModelSpace.AddLine(P35, P37)
Set objLine = ThisDrawing.ModelSpace.AddLine(P37, P38)

'Draw the ibeam

Set objLine = ThisDrawing.ModelSpace.AddLine(P22, P23)
Set objLine = ThisDrawing.ModelSpace.AddLine(P23, P24)
Set objLine = ThisDrawing.ModelSpace.AddLine(P24, P25)
Set objLine = ThisDrawing.ModelSpace.AddLine(P25, P26)
Set objLine = ThisDrawing.ModelSpace.AddLine(P26, P27)
Set objLine = ThisDrawing.ModelSpace.AddLine(P27, P28)
Set objLine = ThisDrawing.ModelSpace.AddLine(P28, P29)

'Draw the drain pipe

Set objCircle = ThisDrawing.ModelSpace.AddCircle(P43, 2)
Set objCircle = ThisDrawing.ModelSpace.AddCircle(P43, 1.75)
```

Mirroring the Foundation with VBA

Before we mirror all the lines, circles and arcs across the vertical centerline, we will select the all of the objects using the Select all function, First, we define a temporary selection set called objSs1 by typing `Set objSs1 = ThisDrawing.SelectionSets.Add("TempSS")` and then we select the five entities using `objSs1.Select (acSelectionSetAll)`.

To mirror all the objects in objSs1, we key the following code:

```
'Mirror the across vertical centerline

Set objSs1 = ThisDrawing.SelectionSets.Add("TempSS")
```

```
    objSs1.Select (acSelectionSetAll)

  For Each objDrawingObject In objSs1
  Set objMirroredObject = objDrawingObject.Mirror(P38, P32)
    objMirroredObject.Update
  Next
  objSs1.Delete
```

Copying and Moving the Sillplate in VBA

To copy the sill plate, we window the entities from P11 to P13.

```
'Copy the sillplate

ThisDrawing.Application.ZoomAll
Set objSs2 = ThisDrawing.SelectionSets.Add("TempSS")
   objSs2.Select acSelectionSetWindow, P11, P13

For Each objDrawingObject In objSs2
Set objCopiedObject = objDrawingObject.Copy
   objCopiedObject.Move P14, P44
   objCopiedObject.Update
Next
objSs2.Delete

Set objLine = ThisDrawing.ModelSpace.AddLine(P20, P21)

'Draw a hidden line

ThisDrawing.ActiveLayer = ThisDrawing.Layers("Hidden")
Set objLine = ThisDrawing.ModelSpace.AddLine(P30, P31)
```

Resetting the Data with the cmdClear Command Button

To clear the textboxes containing the user input, we will first set the textbox for txtXcoord, txtXcoord.text property to a “0.00” entry by using the equal sign “=”.This makes the property equal zero as a default. We do this also for the Y and Z coordinates. We will set the textboxes for txtWidth, txtWidth.text property to a black entry by using the equal sign “=” and the null string “”, and this will make that property blank. Notice that after the control object name the dot (.) separates the suffix which is the name of the property for that object.

Key the following code as a new subroutine Private Sub cmdClear_Click().

```
Private Sub cmdClear_Click()
'clear the form
   txtSpX = "0"
   txtSpY = "0"
   txtSpZ = "0"
   txtHouseWidth = "0"
   txtBlockWidth = "0"
   txtFooterThickness = "0"
   txtIbeam = "0"
   txtJoistHeight = "0"
   txtColumnWidth = "0"
   txtCourses = "0"
End Sub
```

Exiting the Program with the cmdExit Command Button

To exit this program, we will unload the application and end the program.
Type the following code:

```
Private Sub cmdExit_Click()
'unload and end program
   Unload Me
   End
End Sub
```

Executing a Subroutine with the cmdDraw Command Button

In this program, we use a subroutine which is executed by the Draw command button, so type the following code to execute the subroutine, CreateFoundation

```
Private Sub cmdDraw_Click()
'draw the Foundation
   CreateFoundation
End Sub
```

Executing a Subroutine with the cmdPickPoint Command Button

For the cmdPickPoint button, we will write SelectPoint which is a subroutine to allow the user to select a point on the graphical display and the X, Y and Z coordinates will be placed in the appropriate textboxes. A user can now type a starting point manually or has the choice to pick

the initial point with their mouse.

```
Private Sub cmdPickPoint_Click()
   SelectPoint
End Sub
```

The SelectPoint subroutine starts with hiding the Foundation window and the prompt "Pick a Start Point: " is written on the command line. Once the single point is selected, the txtSpX textbox is given the first value of the starting point. After all three txtSP textboxes are filled, the Foundation window reappears.

```
Sub SelectPoint()
Me.hide
Dim StartPoint As Variant
   StartPoint = ThisDrawing.Utility.GetPoint(, vbCr & "Pick a Start Point: ")
   txtSpX = StartPoint(0)
   txtSpY = StartPoint(1)
   txtSpZ = StartPoint(2)
Me.Show
End Sub
```

Inserting a Module into a Visual Basic Application

Insert a Module by selecting Insert on the Menu Bar and select Module as shown in Figure 10.26. In the Project Menu, double click on the Module and type the following code.

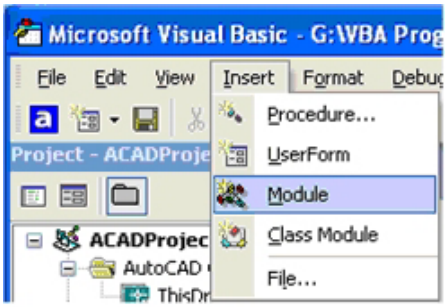

Figure 10.26 – Inserting a Module

```
Sub DrawFoundation ()
'draw the Foundation
   frmFoundation.Show
End Sub
```

The line of code, frmFoundation.Show will display the form at the beginning of the program.

Figure 10.27 – Coding the Module

Running the Program

After noting that the program is saved, press the F5 to run the Foundation application. The Foundation window will appear on the graphical display in AutoCAD as shown in Figure 10.28. Type the data shown in Figure 10.29 or something similar into the textboxes and select the Draw Command Button to execute the program.

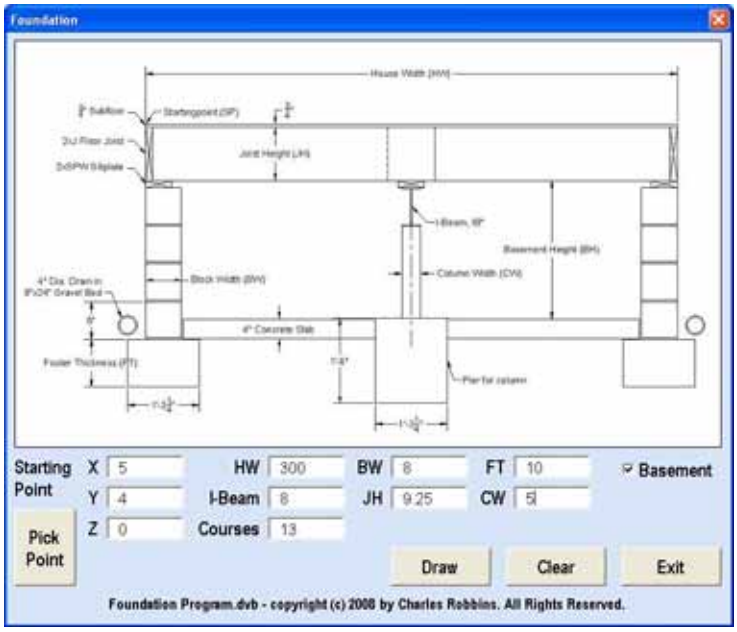

Figure 10.28 – Launching the Program

If we wish to select a point on the graphical display to enter the X, Y and Z values, press the Pick Point command button and do so.

To exit the program, press the Exit command button on the Foundation Program window. In AutoCAD, the drawing of the finished Foundation will appear as shown in Figure 10.37.

X	5
Y	4
Z	0
HW	300
BW	8
FT	10
I-Beam	8
JH	10
CW	5
Courses	13
Basement	☑

Figure 10.29 – Input Data

There are many variations of this Visual Basic Application we can practice and draw many single view orthographic drawings. While we are practicing with forms, we can learn how to use variables, make point assignments and draw just about anything we desire. These are skills that we want to commit to memory.

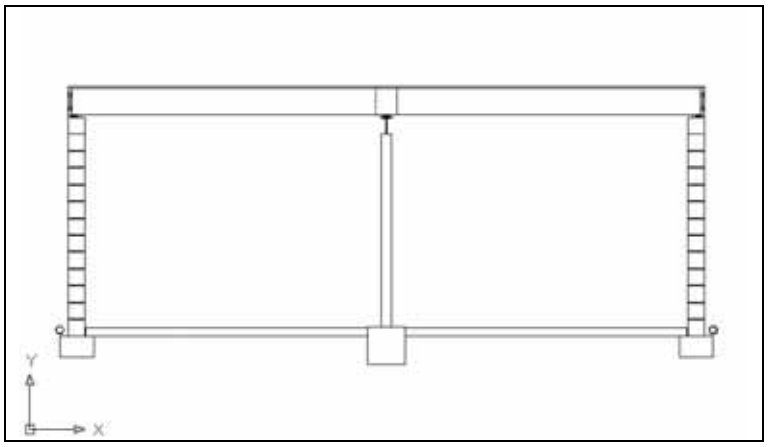

Figure 10.30 – The Finished Draw

*** World Class CAD Challenge 5-10 * - Write a Visual Basic Application that draws a Foundation by inputting data in a form. Complete the program in less than 120 minutes to maintain your World Class ranking.**

Continue this drill four times making other shapes and simple orthographic views with lines and circles, each time completing the Visual Basic Application in less than 120 minutes to maintain your World Class ranking.