

Chapter

9

Programming with Visual AutoLISP

In this chapter, you will learn the following to World Class standards:

- 1. Drawing and Labeling the Boxcircle Programming Sketch**
- 2. Launching the Visual LISP Editor**
- 3. Programming in Visual AutoLISP**
- 4. Loading and Running the AutoLISP Program**

Drawing and Labeling the Boxcircle Programming Sketch

When beginning to create a program using Construction Code, the programmer needs to draw the basic object, including lines, arcs and circles in model space. Go ahead and use your basic mechanical or architectural template so you have a dimensioning, centerline and notes layer. Having these different layers in the drawing will allow you to easily separate the different components of the programming sketch, such as object lines, variables in the dimensions, the x and y grid and point assignments. When one labels a sketch, follow some sort of logical rule, like label the perimeter of the box in a clockwise rotation. (Figure 2.1)

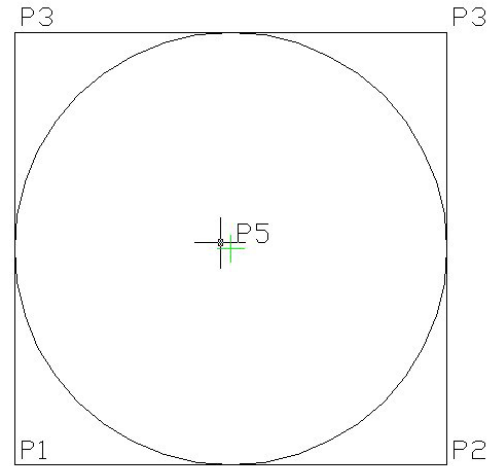


Figure 2.1 – Labeling the Points

After showing the five points on the Boxcircle detail, go ahead and dimension the drawing with the linear dimension tool. Place a dimension from point P1 to P2. By selecting the Edit Text tool on the Text toolbar, one can change the $\langle \rangle$ brackets to the word “Width”. Likewise, when dimensioning from point P1 to P5, one changes in this case the $\langle \rangle$ brackets to “Width / 2” which means this distance will be divided in half of the width of the box. Now, place a leader with SP for starting point at P1, so we know where to start our drawing. (See Figure 2.2)

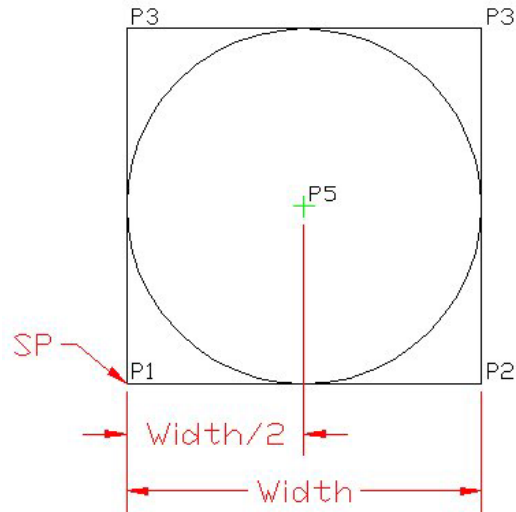


Figure 2.2 – Dimensioning the Variables

WCC Warning: Every once in a while, we find a student that will proceed with the program without a sketch, but we must warn you that to venture forward without documentation can jeopardize the efficiency of an architectural or engineering department since when one opens the programming editor and discovers that they cannot figure out what points or variables are describing what. Especially if they did not write the original code. By having a sketch in the programming folder with the same file name as the program, there should be little difficulty for anyone picking up on the Construction Code. Also, this is the first time that we have changed an auto-dimensioned piece of text by removing the $\langle \rangle$ brackets. Do not do this on a production drawing.

Next, in 2-dimensional orthographic drawings, we have always used the X and Y grid shown in every drawing by the UCS icon. Now we are going to show a grid on the Boxcircle sketch with a line projecting in the positive X and positive Y directions. Label the lines across the bottom in an orderly manner, such as X1, X2 and then X3 (Figure 2.3). After completing the X grid series, then proceed to the Y grid series, starting with the Y1 at the bottom, then Y2 in the middle and finally Y3 at the top. A grid is placed on the drawing, so when point assignments are made, the coder can assign P1 the coordinates (X1, Y1). P2 is assigned (X3, Y1). P3 is assigned (X3, Y3). P4 is assigned (X1, Y3). P5 is assigned (X2, Y2). When finished labeling the sketch, save the drawing as Boxcircle.dwg in your Visual AutoLISP programming folder and exit the AutoCAD drawing. (Figure 2.4)

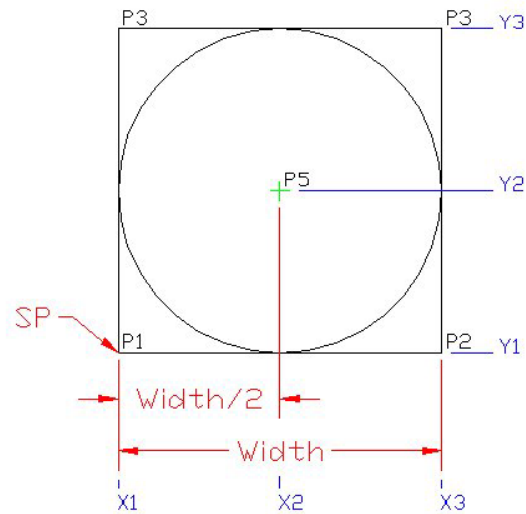


Figure 2.3 – Placing the X and Y Grid

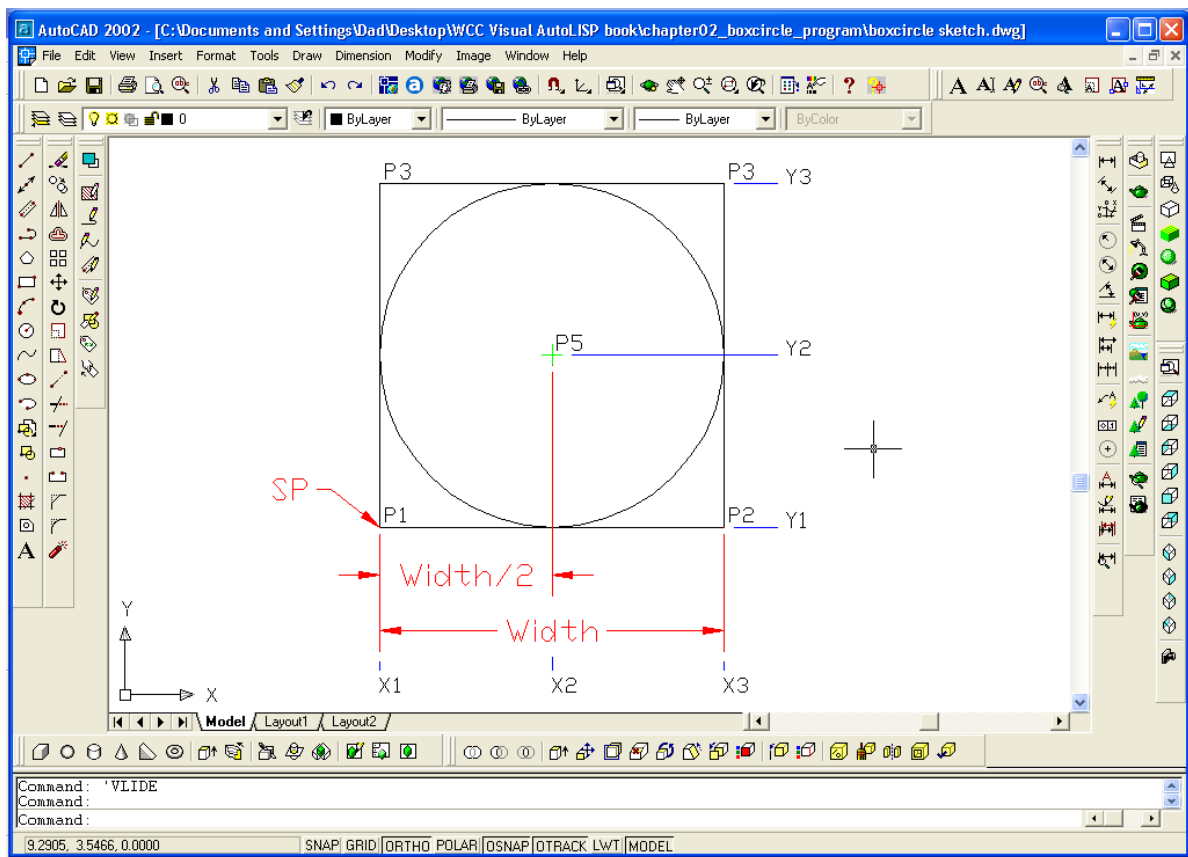


Figure 2.4 – The Finished Boxcircle Programming Sketch

Launching the Visual LISP Editor

Now to start your first engineering program, open a new AutoCAD drawing, select Tools from the popup menu, select AutoLISP, and then Visual LISP Editor (Figure 2.5). A Visual LISP Editor window with a blank page will appear in the middle of your AutoCAD drawing.

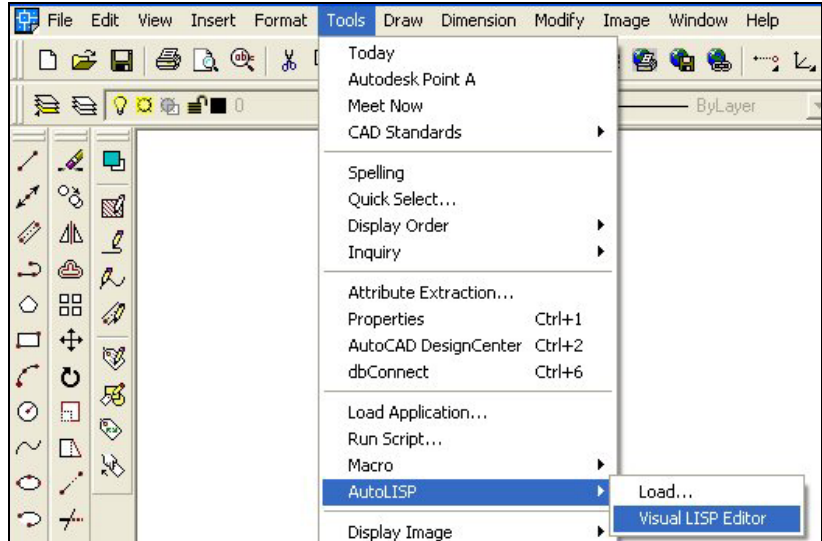


Figure 2.5 – Launching the Visual LISP Editor



New File Tool

The Visual LISP window will open on top of the AutoCAD Model Space window. To create a new program, select the New File tool on the Visual LISP Editor toolbar (figure 2.6). Every once in a while a new programmer in error will start to code on the Visual LISP Console window which you will see when you start the Visual LISP Editor and has the `_ $` symbols at the beginning.

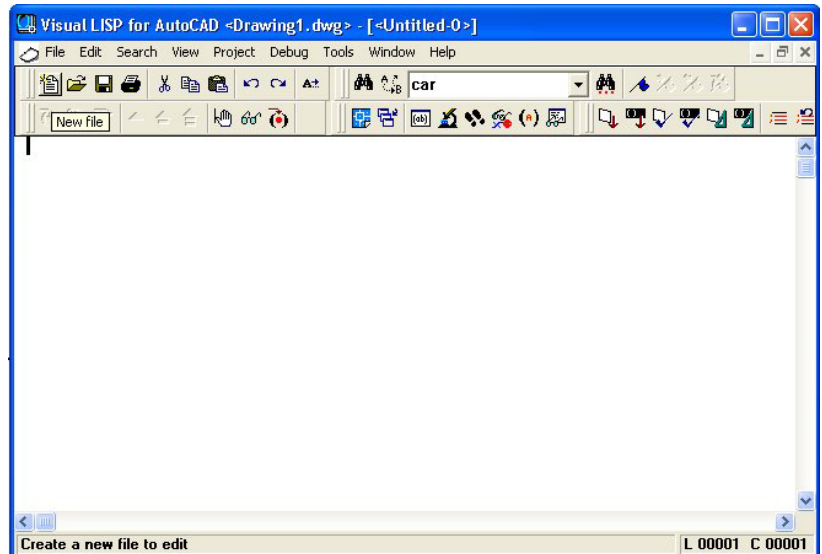
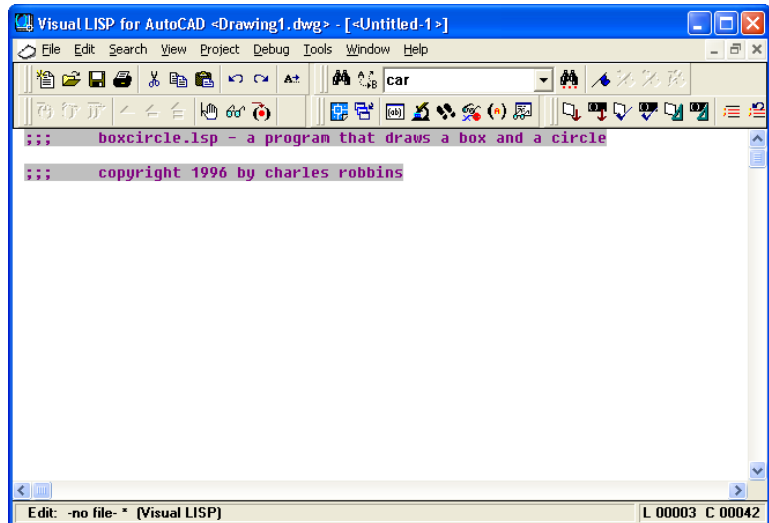


Figure 2.6 – The Visual LISP Editor

Programming in Visual AutoLISP

Every program begins by typing a proper program name as in this case “**Boxcircle.lsp**” followed by a short description of what the program does “- **a program that draws a box and a circle**” (figure 2.7). Start the line of code with a single “;” semicolon to create a comment field shown in a gray shadow. A single semicolon is required to create a comment, but three semicolons are considered as a heading or a 0 column comment which appears on a new line, without any indentation.

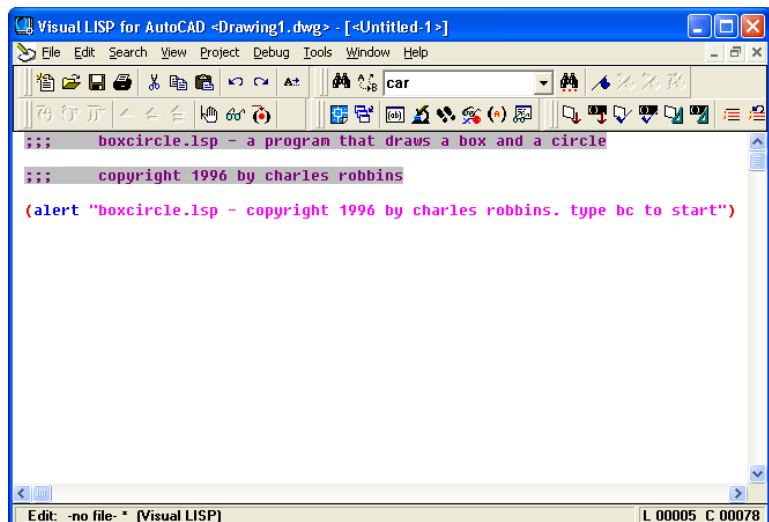
Next, place any special instructions in the form of comments such as copyright, permissions or other legal notices to inform AutoLISP programmers what are the rules dealing with running your code. A program that accidentally or intentionally leaves an organization and is being used by another is a problem. The alert message is a great tool when properly utilized to inform someone if they are breaking a copyright law when running your programs.



```
Visual LISP for AutoCAD <Drawing1.dwg> - [-Untitled-1>]
File Edit Search View Project Debug Tools Window Help
car
;;; boxcircle.lsp - a program that draws a box and a circle
;;; copyright 1996 by charles robbins
Edit: -no file- * [Visual LISP] L 00003 C 00042
```

Figure 2.7 – Naming, Describing and Copyrighting

The Alert function will place an AutoCAD message in the AutoCAD window with an OK button to proceed. If the user does not accept the information by selecting the OK, then the program will not continue. An Alert expression begins with an open parenthesis “(“ and type the function name **alert** and in this case followed by a line of text “**boxcircle.lsp – copyright 1996 by charles robbins. Type bc to start**” (Figure 2.8)



```
Visual LISP for AutoCAD <Drawing1.dwg> - [-Untitled-1>]
File Edit Search View Project Debug Tools Window Help
car
;;; boxcircle.lsp - a program that draws a box and a circle
;;; copyright 1996 by charles robbins
(alert "boxcircle.lsp - copyright 1996 by charles robbins. type bc to start")
Edit: -no file- * [Visual LISP] L 00005 C 00078
```

Figure 2.8 – Alert Message

An expression in AutoLISP begins with an open parenthesis, then the function and after applying their special purpose a closed parenthesis closes the statement. These parentheses may be the source of programming errors as your code grows larger.

Notice that we are using lower case letters in the code. In some programming languages, they are case sensitive, which means that the code will run differently based upon whether the letters are upper or lower case. Visual AutoLISP is not case sensitive and programs can be typed both ways and they will work fine. Lower case text is easier to troubleshoot as the programs become larger.

Add a new comment

;;; start program

Then we start the program with the **defun** function, which means define function. Begin with the open parenthesis then **defun**, then a **c:** which will allow the program to run on the AutoCAD command line. Next type **bc** which will be the execution symbol to start the program. Remember the alert message that stated “type bc to start”. The alert message text and the defun symbol must match.

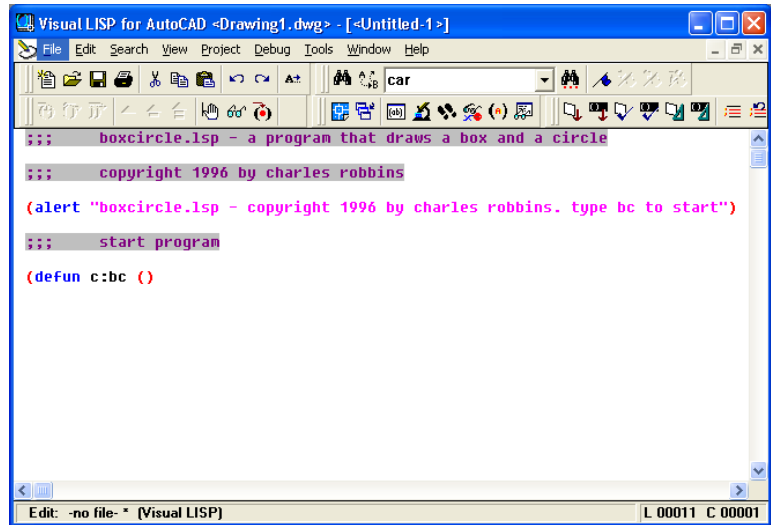


Figure 2.9 – Starting a Program with a Defun Function

The open and closed parenthesis “**()**” following the **bc** enclosing nothing means there will not be any defined arguments or local variables for this program. After that, we need to make changes to the AutoCAD System Variables that may interfere with the running of the code and automatically drawing the lines and circles perfectly. (Figure 2.9)

The programming strategy that we will be using in the setup portion of the code is to find the object snap settings in the AutoCAD System Variables, which you could check yourself manually by going into the AutoCAD drawing and typing “**sysvar**” at the command line and at the prompt, *Enter variable name or [?]:*, type “**osmode**”. A whole number or integer will return that describes the object snap settings presently set in the drawing file.

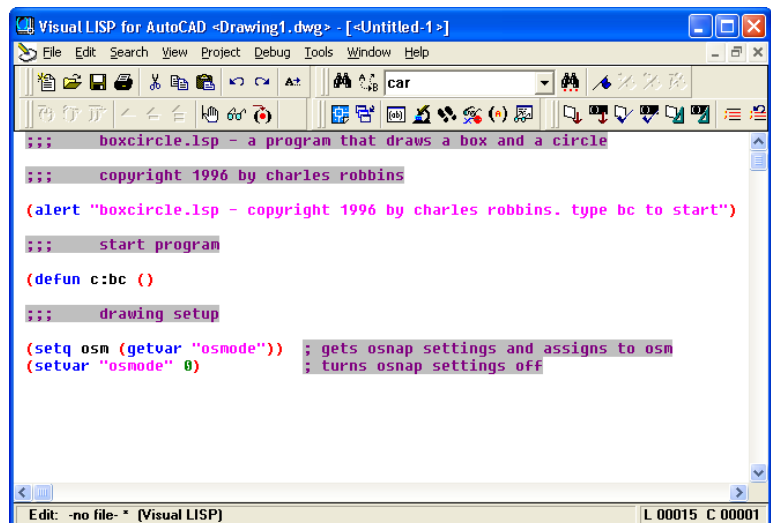


Figure 2.10 – Drawing Setup

Proper programming etiquette states that when you want to modify a drawing to proceed with your program, you need to return the drawing to the same settings at the end of your code. So

we capture the osnap settings with the code and assign the integer to the variable **osm**. Type the compound expression which means that there are two or more functions in the line of code. Place a single semicolon with the comment shown below after the line of code to inform a future programmer of your plans.

Start with a new comment

```
;;; drawing setup
```

And type the code

```
(setq osm (getvar "osmode")) ; gets osnap settings and assigns to osm
```

Next, we will turn off the drawing's object snaps by setting the system variable "osmode" to 0 using this line of code. Add the comment as shown.

```
(setvar "osmode" 0) ; turns osnap settings off
```

Let's talk about the expression, `(setq osm (getvar "osmode"))`. The function **setq** means set quotient and we will use the function to create a variable **osm** which stands for object snap mode, a variable name that we just made up. The variable **osm** will hold the integer representing the "osmode" system variable's setting. To get the number use the function **getvar** followed by the name of system variable inside a set of quotes.

To turn off a system variable in many cases in setting the variable to zero. In the expression, `(setvar "osmode" 0)`, the function **setvar** followed by a system variable inside a set of quotes like "osmode" then a 0 will result in turning off the Object Snap settings.

The next step in the Construction Code strategy is to request two pieces of data required to draw this detail, the starting point and the width of the box. These two pieces of data are shown on the Boxcircle sketch.

Start with the comment

```
;;; user input
```

Then type the following code:

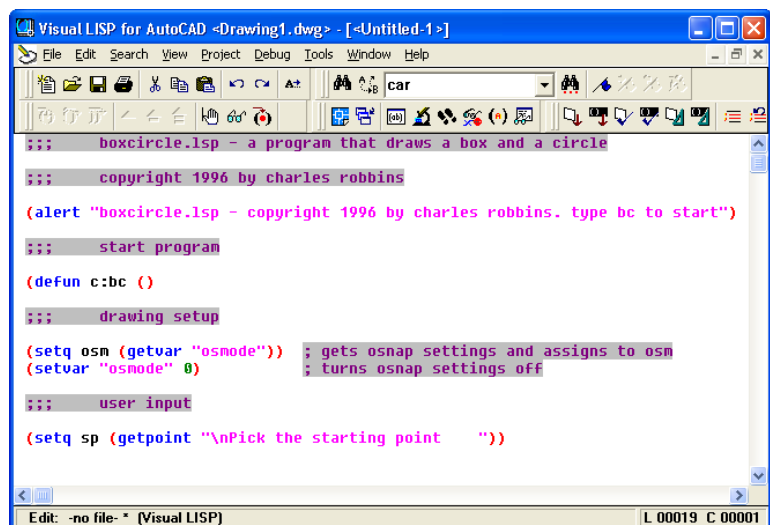


Figure 2.11 – Inquiring about the Starting Point

```
(setq sp (getpoint "\nPick the starting point  "))
```

We are again using the **setq** expression to assign the three point list (X, Y and Z) to the variable **sp** representing the starting point. We use a new function **getpoint** (figure 2.11) to allow the user to select a point on the AutoCAD display screen. A programmer has the option, in which we have chosen, to add a line of text prompting the user to “**Pick the starting point**” and we also modified the prompt in a small way. Notice that in front of the capital P in the word Pick, a “\n” is added. That will place a question containing those two characters to start on a new command line in the AutoCAD program, which allows for a cleaner look to the user when answering the questions.

To ask the question, “what is the width of the box”, there are a couple of changes we will make to the compound AutoLISP expression, the new variable name is **width** and the get function is now changed to **getreal** along with a new prompt. Pick any word for a variable name as long as the name does not match any function names in the program. The **getreal** function will allow the user to type a real number at the command line in the AutoCAD drawing.

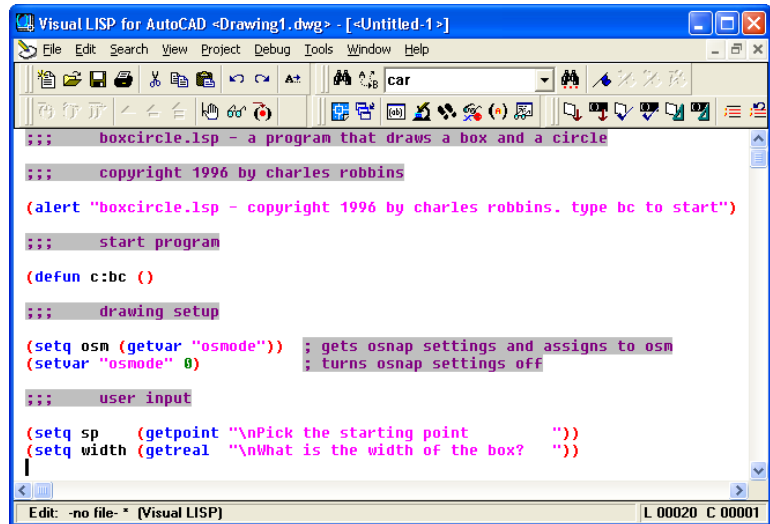


Figure 2.12 – Inquiring about the Width of the Box

So type the following code:

(setq width (getreal “\nWhat is the width of the box ”))

If you look at the Visual LISP Editor in figure 2.12, you will notice that we dressed the last two expressions so that the questions line up perfectly. You will pick up on this characteristic when the program is running and the typed answers to the questions line up neatly.

Many programmers are very apprehensive about the next section of the code, which is the math computations, so in the last two decades we have taught initially trained programmers executing their first fifty programs to just skip this section and return to the math. The math section requires the coder to understand a small amount of algebra, where we will calculate a grid position using variables and a small amount of constants. After some practice, every programmer with whom we have worked has been able to accomplish this section of code adequately.

One of the easiest sections of code for a new or experienced programmer to accomplish is the point assignments, where one assigns X and Y coordinates to the point vertices. Basically, we did the work when we made the Boxcircle sketch. Remember when you read that coordinate P1 is (X1, Y1). P2 is (X3, Y1). P3 is (X3, Y3). P4 is (X1, Y3). P5 is (X2, Y2) earlier in the chapter. Now we write a **setq** expression setting these grid coordinates to points **p1**, **p2**, **p3**, **p4** and **p5**.

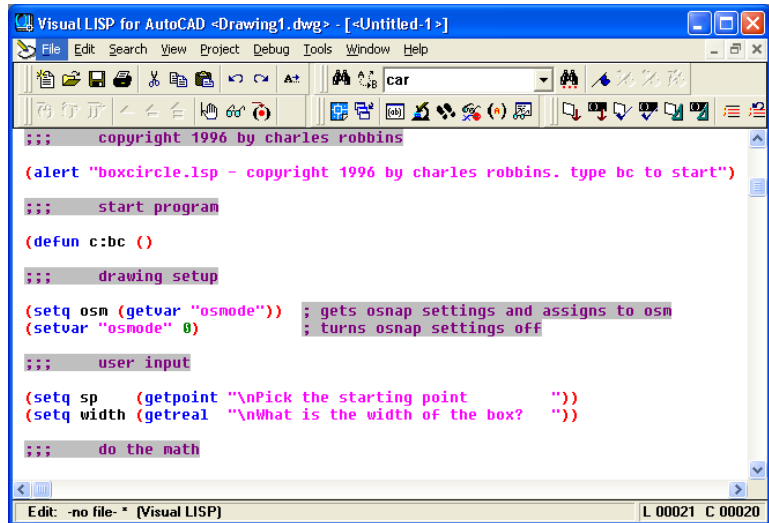


Figure 2.13 – Skip the Math

The **list** function can create an X, Y and Z coordinate by typing the appropriate X and Y values after the function name. We do not need to add the Z coordinate if the value is going to be zero. (Figure 2.14)

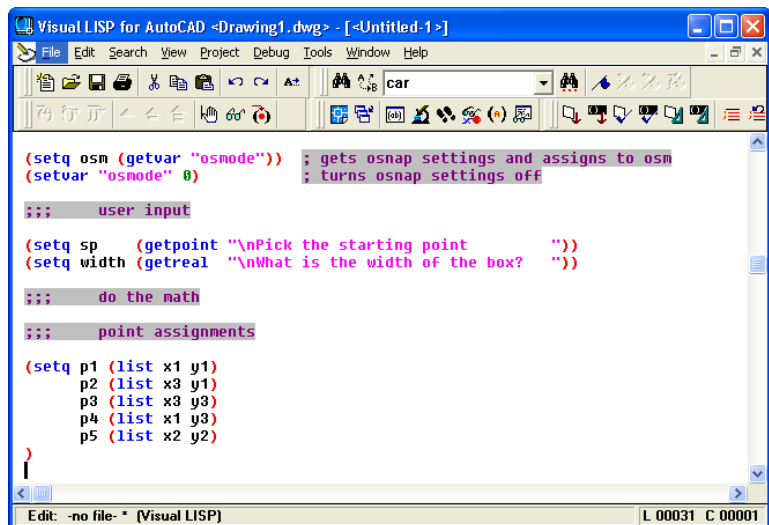


Figure 2.14 - Point Assignments

Type the following code:

;;; point assignments

```

(setq p1 (list x1 y1)
      p2 (list x3 y1)
      p3 (list x3 y3)
      p4 (list x1 y3)
      p5 (list x2 y2)
)

```

When automatically drawing any entity in AutoCAD, the programmer uses the **command** function which evokes any AutoCAD standard command. We have to state this rule, since ARX commands typed at the command line like Render or Rotate3D need to be executed differently, which we will do in later assignments. After the **command** function is typed, the command "line" follows in quotes, then by the point vertices **p1 p2 p3 p4** of the box and finally "c" to close the polygon.

Type the following code:

;;; lets draw

```

(command "line" p1 p2 p3 p4 "c")

```

Think about how many lines are in every architectural and mechanical drawing. When we train an individual in creating Construction Code, there is a transformation that takes place in the first four hours from how they believed that the goal to automate the drawing process to how easy this task really is. How much time will be saved in the organization and how accurate the finished product will be? Drawing circles is just as easy.

```

Visual LISP for AutoCAD <Drawing1.dwg> - [Untitled-1 >]
File Edit Search View Project Debug Tools Window Help
car
::: do the math
::: point assignments
(setq p1 (list x1 y1)
      p2 (list x3 y1)
      p3 (list x3 y3)
      p4 (list x1 y3)
      p5 (list x2 y2)
)
::: lets draw
(command "line" p1 p2 p3 p4 "c")
Edit: -no file- * [Visual LISP] L 00034 C 00033

```

Figure 2.15 – Draw the Four Lines of the Box

To have a circle drawn inside the box composing of four lines, we will use the **circle** command, the next argument is to list the center point of the circle which is the point vertex, p5, and finally to describe the diameter of the circle by requesting the diameter with the text “d” and the real number representing the diameter in the variable width. All of this is brought to together with the following expression.

(command "circle" p5 "d" width)

```

Visual LISP for AutoCAD <Drawing1.dwg> - [Untitled-1 >]
File Edit Search View Project Debug Tools Window Help
car
::: do the math
::: point assignments
(setq p1 (list x1 y1)
      p2 (list x3 y1)
      p3 (list x3 y3)
      p4 (list x1 y3)
      p5 (list x2 y2)
)
::: lets draw
(command "line" p1 p2 p3 p4 "c")
(command "circle" p5 "d" width)
Edit: -no file- * [Visual LISP] L 00035 C 00032

```

Figure 2.16 – Draw the Circle

To end the program, we will set the object snap mode back to the original settings by using the setvar function followed by the variable osm which holds the original integer containing the Osnap settings. Type the following code.

;;; end the program

(setvar "osmode" osm)
; turns osnap settings back on
)

```

Visual LISP for AutoCAD <Drawing1.dwg> - [Untitled-1 >]
File Edit Search View Project Debug Tools Window Help
car
::: point assignments
(setq p1 (list x1 y1)
      p2 (list x3 y1)
      p3 (list x3 y3)
      p4 (list x1 y3)
      p5 (list x2 y2)
)
::: lets draw
(command "line" p1 p2 p3 p4 "c")
(command "circle" p5 "d" width)
::: end the program
(setvar "osmode" osm) ; turns osnap settings back on
)
Edit: -no file- * [Visual LISP] L 00040 C 00002

```

Figure 2.17 – End the Program

Now, we will return back to the middle of the program and finish the math section of the code. Again the `setq` function is the choice for assigning values to the variables X1, X2, X3, Y1, Y2 and Y3.

The `car` function is used with variable `sp` (the starting point) to extract the x-coordinate of the starting point list. If the starting point is (4, 3, 0) then (`car sp`) will return as 4 and be assigned to the variable X1. So the `car` function returns the first number in the list.

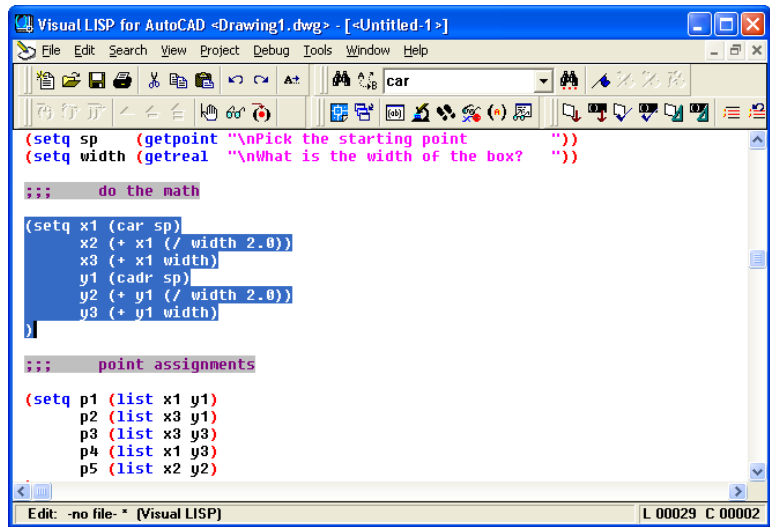


Figure 2.18 – Do the Math

Likewise, the `cadr` function is used with variable `sp` (the starting point) to extract the y-coordinate of the starting point. Again, if the starting point is (4, 3, 0) then (`cadr sp`) will return as 3 and be assigned to the variable `y1`. So the `cadr` function returns the second number in the list.

That explains the use of `car` and `cadr` to find the coordinates `x1` and `y1`, now we have to continue down the X grid to obtain values for `x2` and `x3`. To obtain the `x2` coordinate, use the addition function `+` to add to the `x1` value. To get the number to add to the `x1`, we have to divide `width` by two using the divide function `/` like so. Notice that the function is written first, followed by the numerator `width` and then the denominator `2.0`.

(/ width 2.0)

And place that expression in the addition expression to build a compound expression.

(+ x1 (/ width 2.0))

Now assign the value to `x2`

x2 (+ x1 (/ width 2.0))

To find `x3`, we have a single addition expression where we add the variable `width` to `x1` using the function `+`.

(+ x1 width)

Now assign the value to `x3`

x3 (+ x1 width)

Repeat the coding process with the y-coordinates **y2** and **y3**; making sure your code appears as below.

```
(setq x1 (car sp)
      x2 (+ x1 (/ width 2.0))
      x3 (+ x1 width)
      y1 (cadr sp)
      y2 (+ y1 (/ width 2.0))
      y3 (+ y1 width)
      )
```

Now that the program is finished, double check your typing with the text in this manual and then save your program to your folder named “Visual LISP Programs”. Now back in the AutoCAD program, select Tools, and Load Application (figure 2.18) to open the Load / Unload Application window.

Loading and Running the AutoLISP Program

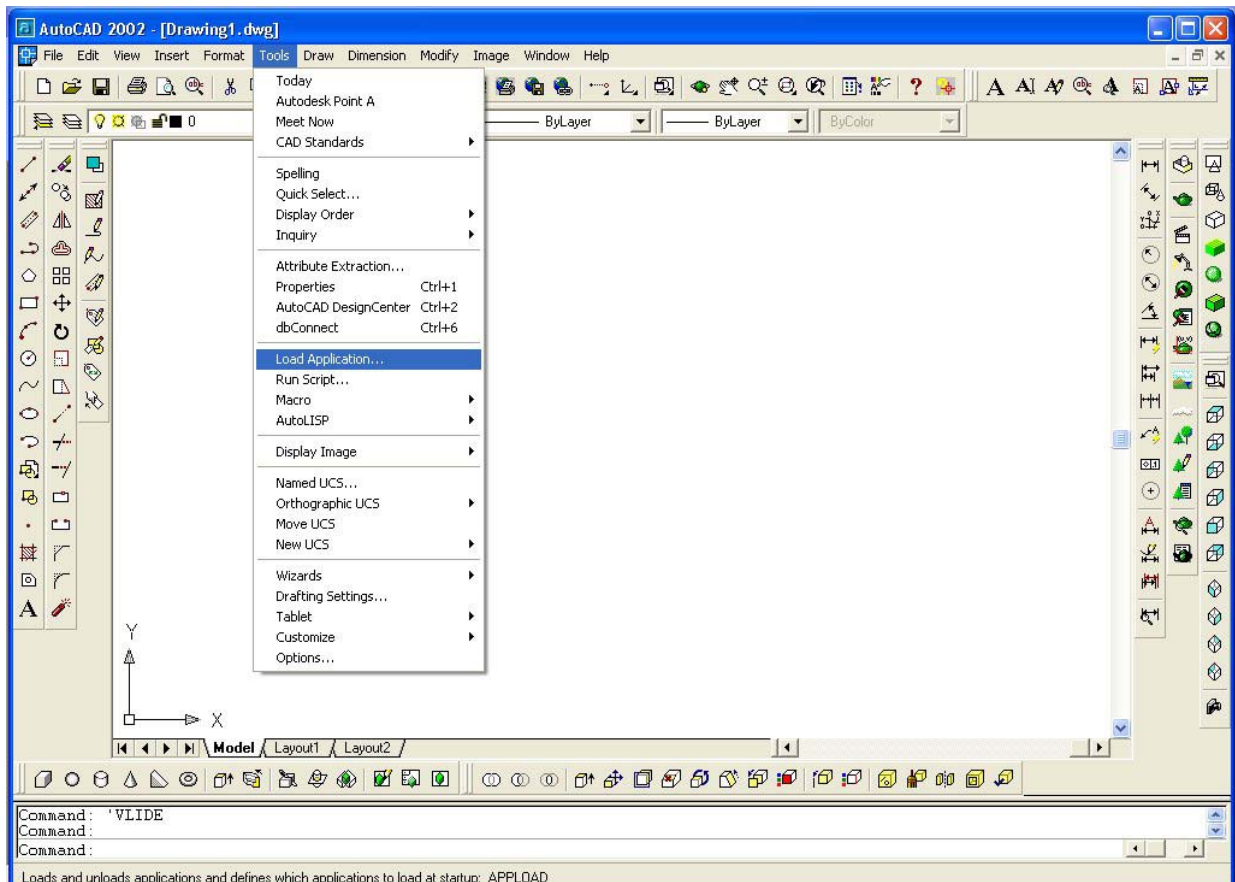


Figure 2.19 – Starting to Load the Application

Make sure the Look in list box is displaying the Visual LISP Programs folder and then select the program “boxcircle” and press the Load button. At the bottom – left corner of the Load / Unload Applications window you will see a small text display that was blank initially but now displays the text, “boxcircle.LSP successfully loaded.” (Figure 2.20)

After noting that the program is loaded, press the Close button and now when you are in the AutoCAD program, an AutoCAD message window appears in the middle of the graphics display stating: “boxcircle.lsp – copyright 1996 by charles robbins. Type bc to start”

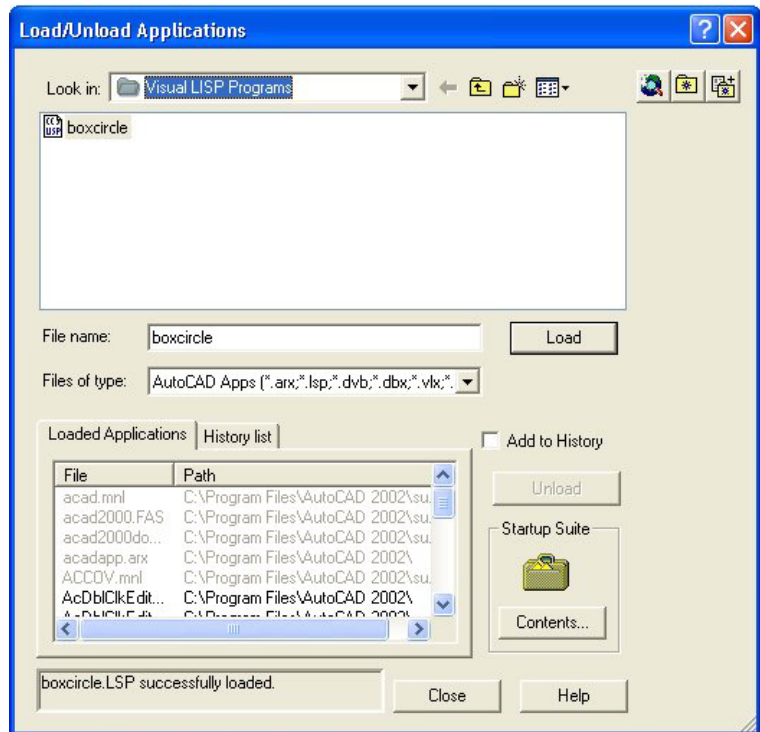


Figure 2.20 – The Load / Unload Applications Window

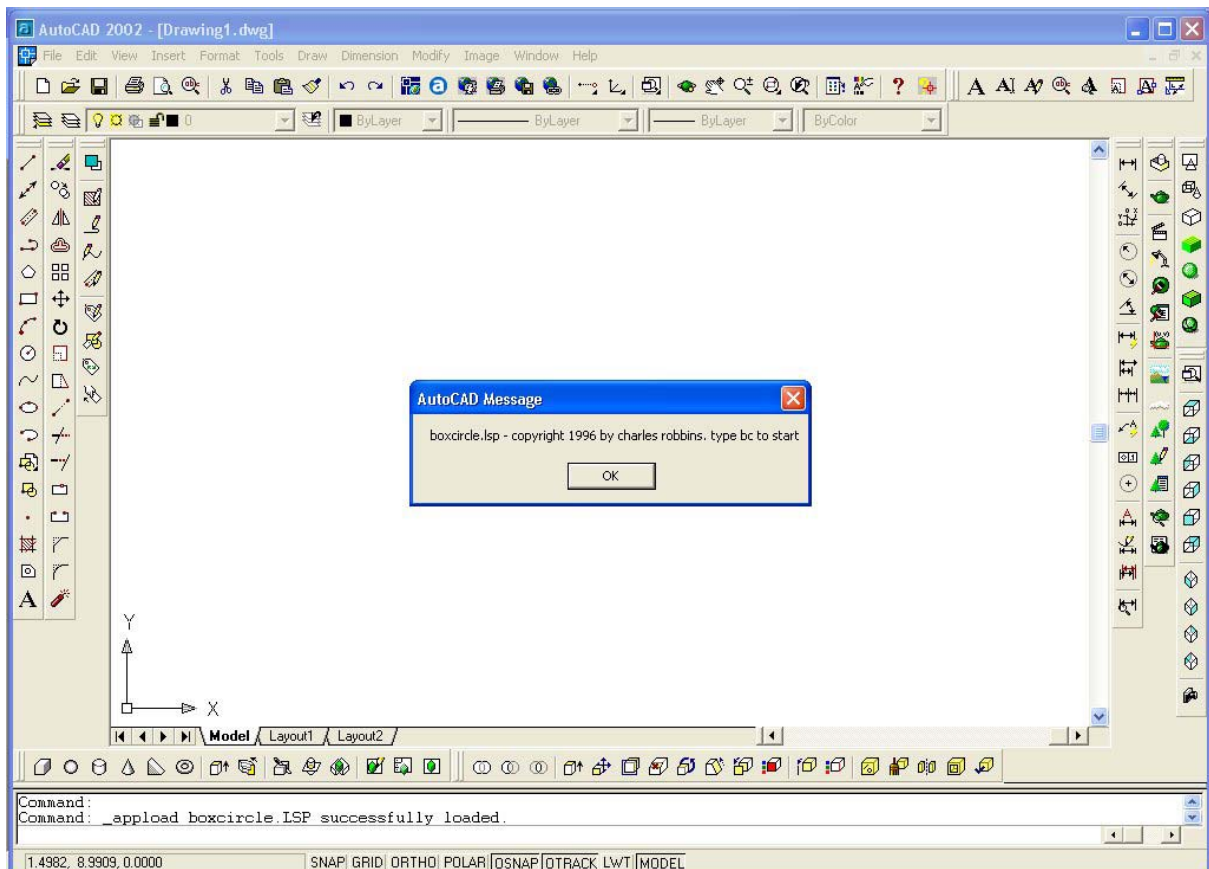


Figure 2.21 – When the Program Loads an AutoCAD Message Appears. Press OK.

Press the OK button if you agree with the message and follow your own instructions by typing **bc** at the command line. The message “Pick the starting point” appears on the command line and the user should select a point at the lower left hand corner of the AutoCAD graphics display.

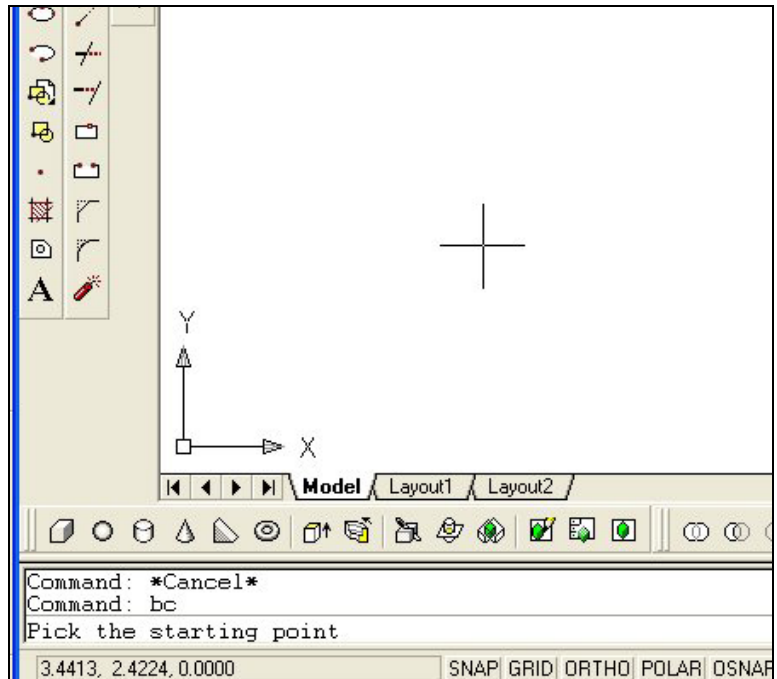


Figure 2.22 – Picking the Starting Point

After the picking the starting point, the user will be prompted, “What is the width of the box?” to which we answered on the command line with 4. (Figure 2.23). After entering the 4 and pressing Enter, the program will draw four lines in a square and place a circle in the middle of the box that is tangent to all four sides. Congratulations, you have completed your first AutoLISP program.

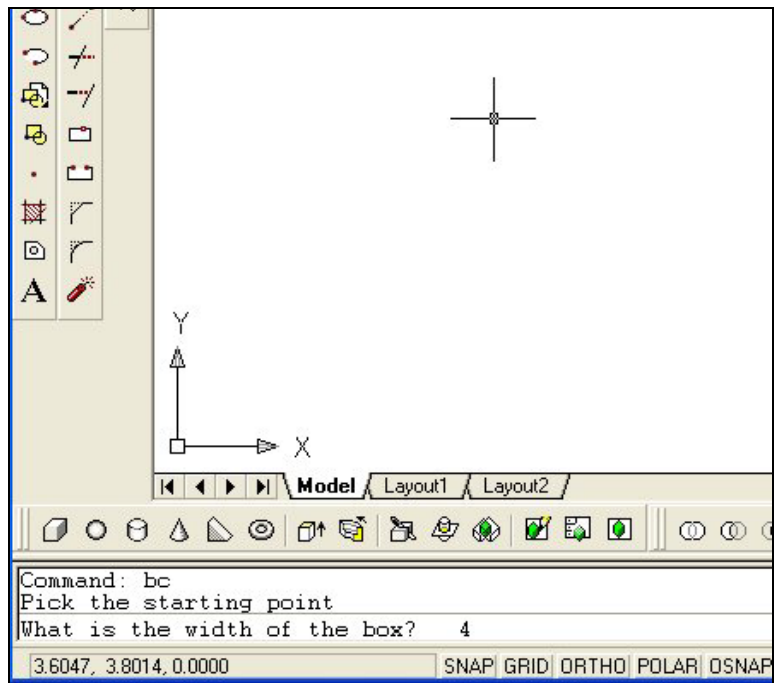


Figure 2.23 – Typing the Width of the Box

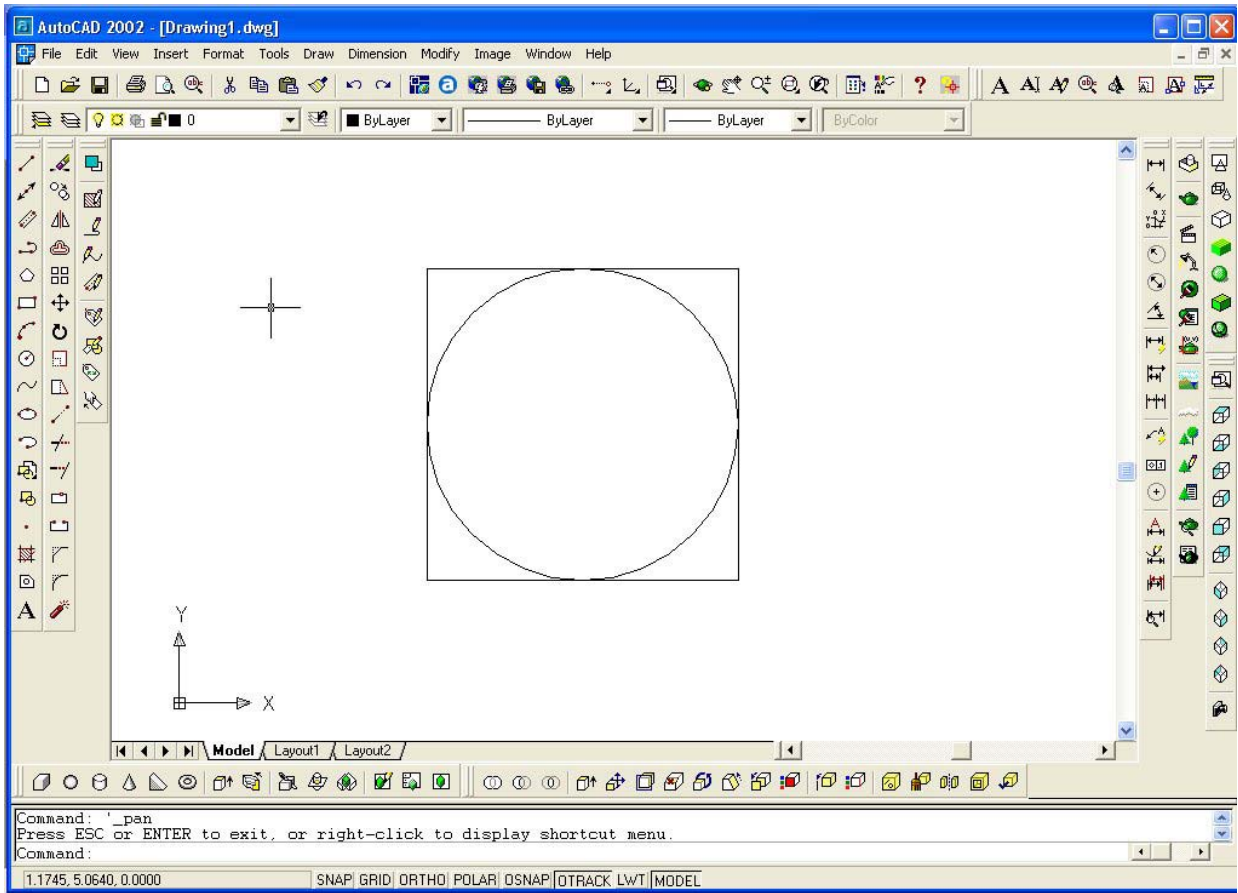


Figure 2.24 – The Finished Drawing