

## The Anglemaker Program

---

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

1. Drawing and Labeling the Anglemaker Programming Sketch
2. Launching the Visual LISP Editor

### The Construction Code Process

3. Step 1 - Start the Program
4. Step 2 - Drawing Setup
5. Step 3 - User Input
6. Step 4 – Do the Math, but Not Right Now
7. Step 5 – Point Assignments
8. Step 6 – Lets Draw
9. Step 7 – End the Program
  
10. Back to Step 4 – Do the Math
11. Loading the AutoLISP Program
12. Troubleshooting the AutoLISP Program
13. Executing the AutoLISP Program

## Drawing and Labeling the Anglemaker Programming Sketch

---

As in the Boxcircle program, we start with a sketch of the problem, the Anglemaker which will automatically draw a side view of an extruded angle typically made from aluminum or steel. As you can observe from the sketch in Figure 3.1 that there are seven points in the construction of the object. From these seven points, we establish a grid of x-coordinates ranging from  $x_1$  to  $x_5$  and of y-coordinates from  $y_1$  to  $y_5$  that will describe each vertex. The variables will be **leg1** which will control the length of the horizontal leg of the angle and with **thk1** defining the thickness. For the second leg of the angle, we use the variables **leg2** and **thk2** to control the size of that detail. The arc that will have to be drawn on the end of each leg will be the same as the thickness of the leg. The center arc joining both legs will be defined by the variable **rad**. The starting point or where the user places the angle in the drawing will be described by **sp**. When you are finished creating the portrayal of the angle in AutoCAD, save your drawing in your Visual LISP Programs folder as Anglemaker Sketch.

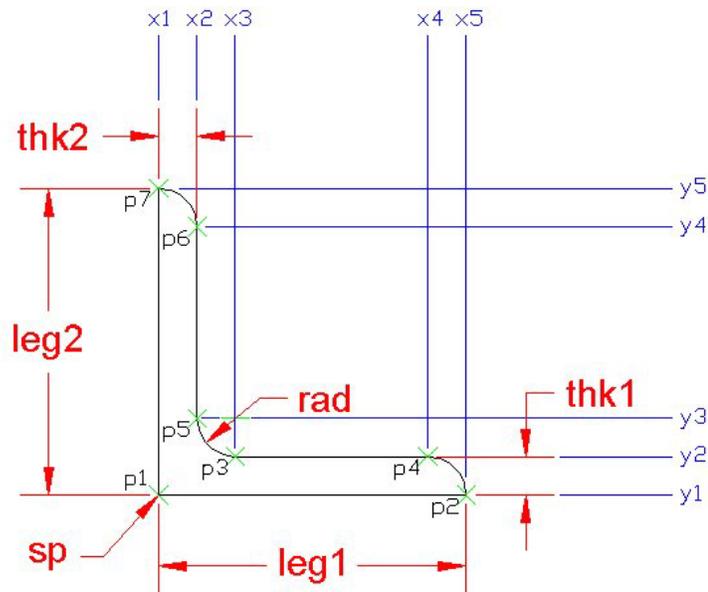


Figure 3.1 – Labeling the Points

We will still use every step of the Construction Code process to create the Anglemaker program, so we will need to remember the steps.

**Start the Program**  
**Drawing Setup**  
**User Input**  
**Do the Math**  
**Point Assignments**  
**Lets Draw**  
**End the Program**

## Launching the Visual LISP Editor

Now to start your second engineering program, open a new AutoCAD drawing, select Tools from the popup menu, select AutoLISP, and then Visual LISP Editor (Figure 3.2). A Visual LISP Editor window with a blank page will appear in the middle of your AutoCAD drawing.

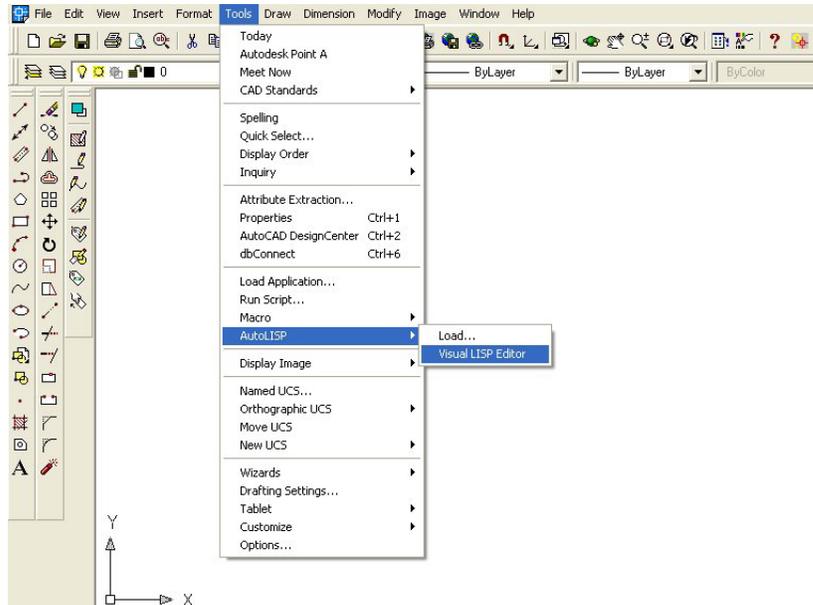


Figure 3.2 – Launching the Visual LISP Editor



### New File Tool

The Visual LISP window will open on top of the AutoCAD Model Space window. The Visual LISP Console will be displaying a `_$_` (figure 3.3), so we know we are not in a new file to start to program. To create a new program, select the New File tool on the Visual LISP Editor toolbar

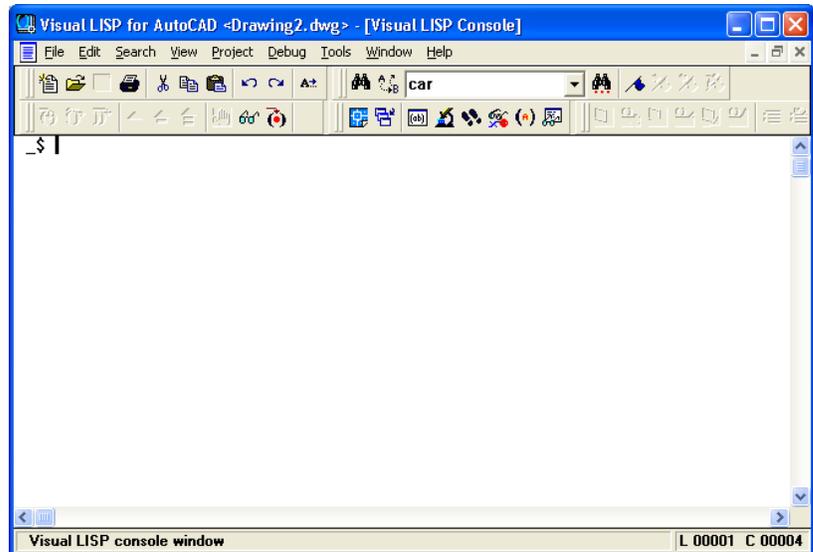
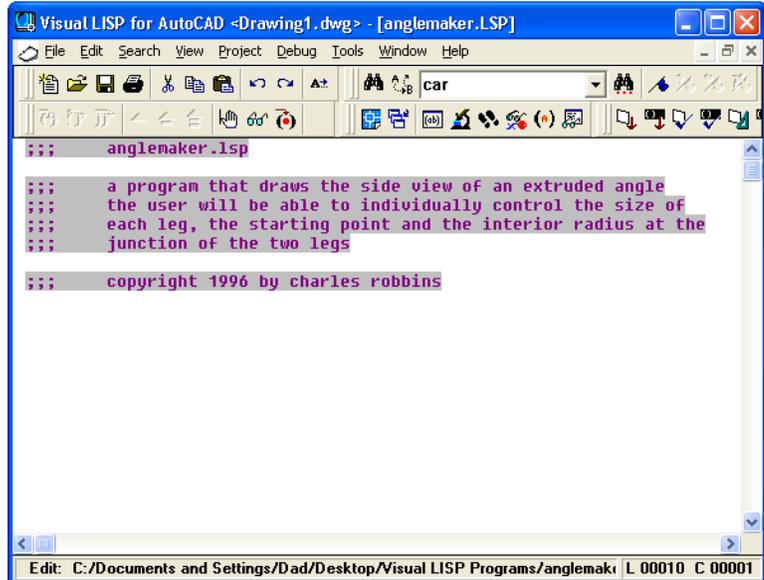


Figure 3.3 – The Visual LISP Console

## Step 1 - Starting the Program

In the discussion of programming the Anglemaker code, we will concentrate on the seven steps of the Construction Code process, so that in the next section of this manual, we will be able to construct six programs by the end of the second week of training.

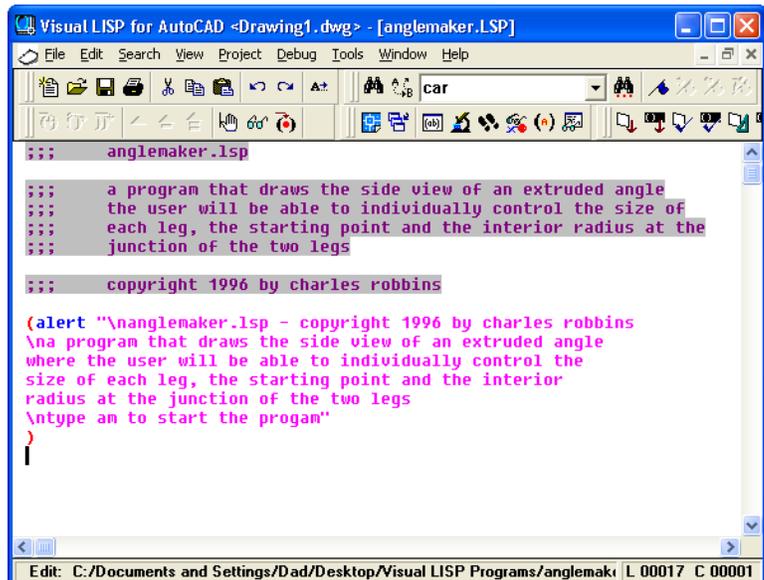
Remember that a program begins with a series of comments, where we communicate with either another coder or ourselves months from now when what was done today will be just a vague memory. Use the semicolons `;;;` to create these three different comments; the program name, a program description, and then the copyright, year of copyright and the name of the copyrighter. (Figure 3.4)



```
Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]
File Edit Search View Project Debug Tools Window Help
car
;;; anglemaker.lsp
;;; a program that draws the side view of an extruded angle
;;; the user will be able to individually control the size of
;;; each leg, the starting point and the interior radius at the
;;; junction of the two legs
;;; copyright 1996 by charles robbins
Edit: C:/Documents and Settings/Dad/Desktop/Visual LISP Programs/anglemak L 00010 C 00001
```

Figure 3.4 – Naming, Describing and Copyrighting

Next we will create an even larger AutoCAD Message by taking the information listed in the comments and placing the text in the `alert` function. On the first line of the alert expression, the program and the copyright information is keyed. The `\n` is typed in the front of `anglemaker` to force a new line at the beginning of the first line. Another is typed at the beginning of the second line, the program description and at the beginning of the last line. (Figure 3.5). The `\n` does not interfere with any of the text as you will see.



```
Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]
File Edit Search View Project Debug Tools Window Help
car
;;; anglemaker.lsp
;;; a program that draws the side view of an extruded angle
;;; the user will be able to individually control the size of
;;; each leg, the starting point and the interior radius at the
;;; junction of the two legs
;;; copyright 1996 by charles robbins
(alert "\nanglemaker.lsp - copyright 1996 by charles robbins
\na program that draws the side view of an extruded angle
where the user will be able to individually control the
size of each leg, the starting point and the interior
radius at the junction of the two legs
\n type an to start the program"
)
Edit: C:/Documents and Settings/Dad/Desktop/Visual LISP Programs/anglemak L 00017 C 00001
```

Figure 3.5 – Alert Message

When we type the message in separate lines the message will appear just as coded, so remember to think about the presentation of text to the user when using the `alert` function and in the `get` functions in the Construction Code user input section (step 3)

When typing the alert expression with some consideration for the individual who runs the AutoLISP program, we will give your AutoCAD Message a clean look and make the information easy to read and understand. (Figure 3.6)



Figure 3.6 – Message using the Alert Function

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 **am** which will be the execution symbol to start the program. Remember the alert message that stated “type am to start the program”. The alert message text and the **defun** symbol must match.

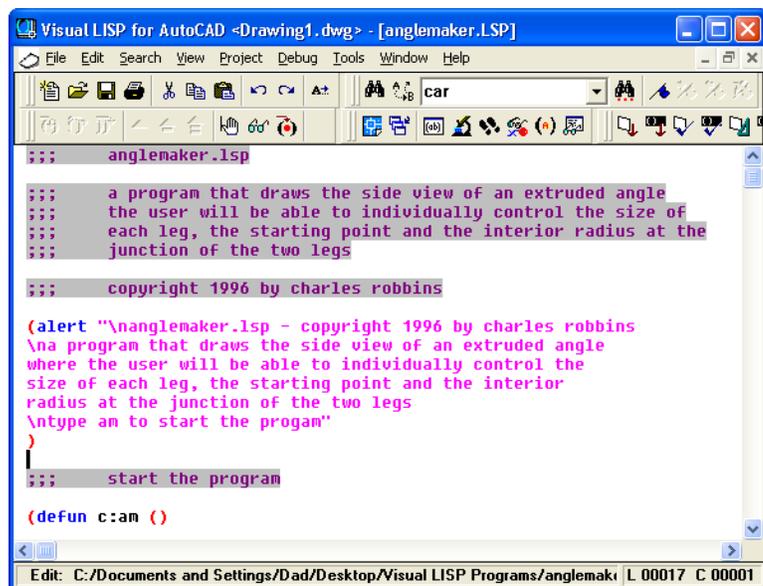


Figure 3.7 – Starting a Program with a Defun Function

The open and closed parenthesis “**()**” following the **am** 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 arcs perfectly. (Figure 3.7)

## Step 2 – Drawing Setup

The second step in the Construction Code process is the Drawing Setup where the AutoCAD system variable are recorded into memory, changed and in the future we will create new layers or text styles. This should be done before getting deep into the code, so we will not have any anxiety about placing different types of expression into other sections of the code which could break your rhythm and speed in keeping programming as simple process

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.

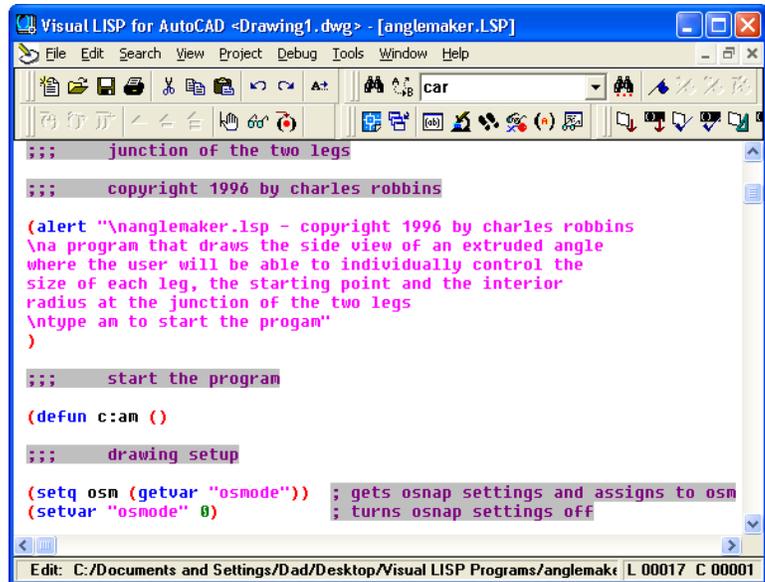


Figure 3.8 – Drawing Setup

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. (Figure 3.8)

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.

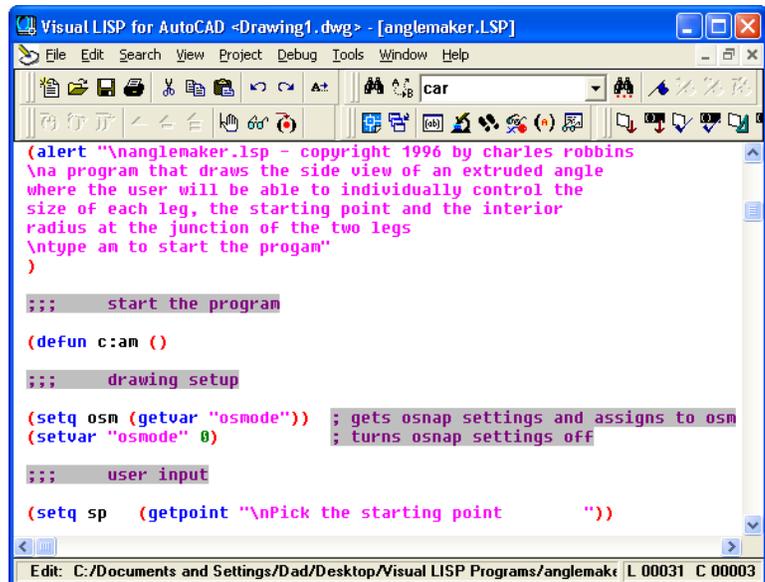
## Step 3 – User Input

The next step in the Construction Code strategy is to request six pieces of data required to draw this detail, the starting point, the leg1 length and thickness, the leg2 length and thickness and the radius of the interior junction of the angle. These six pieces of data are shown on the Anglemaker sketch (Figure 3.1)

Start with the comment

```
;;; user input
```

Then type the following code:



```
Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]
File Edit Search View Project Debug Tools Window Help
car
(alert "\anglemaker.lsp - copyright 1996 by charles robbins
\na program that draws the side view of an extruded angle
\nwhere the user will be able to individually control the
\nsize of each leg, the starting point and the interior
\nradius at the junction of the two legs
\n\ntype an to start the program"
)
;;; start the program
(defun c:am ()
;;; drawing setup
(setq osn (getvar "osmode")) ; gets osnap settings and assigns to osn
(setvar "osmode" 0) ; turns osnap settings off
;;; user input
(setq sp (getpoint "\nPick the starting point "))
Edit: C:/Documents and Settings/Dad/Desktop/Visual LISP Programs/anglemaker.L 00031 C 00003
```

Figure 3.9 – 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 3.8) 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 just as we did in the alert expression. 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 length of the leg1”, there are a couple of changes we will make to the compound AutoLISP expression, the new variable name after the **setq** function is **leg1** and the get function is now changed to **getreal** along with a new prompt **\nWhat is the length of leg1?** (Figure 3.10)

```
(setq leg1 (getreal "\nWhat is the length of leg1? "))
(setq thk1 (getreal "\nWhat is the thickness of leg1? "))
```

After keying the leg1 and thk1 expressions, copy them both by highlighting as shown in figure 3.11 and pasting them below the first two compound expressions. Change the leg1 to leg2 and thk1 to thk2.

```
(setq leg2 (getreal "\nWhat is the length of leg2? "))
(setq thk2 (getreal "\nWhat is the thickness of leg2? "))
```

The Get functions work similarly in Visual AutoLISP, they allow the user to input information into the program while the code is running, so the when training the user at your organization how to run these short and efficient drawing tools that will add much more than a line or are more adaptable than using a block, we need the user to read the questions that appear on the command line.

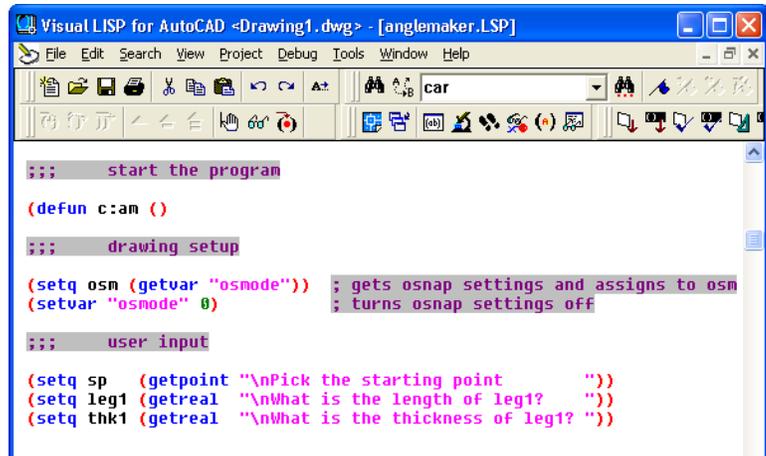


Figure 3.10 – Inquiring about the size of leg1

Programmers will use the **getreal** function to get a response that can be any real number, positive or negative, and in many cases a decimal is required. The getreal function will not accept feet, inches or fractions, but we will adjust the questioning in future project to address the use of English units by Architectural Designers, so we can add another **alert** expression to launch an AutoCAD message stating that the responses need to be in decimal inches or decimal meters for the program.

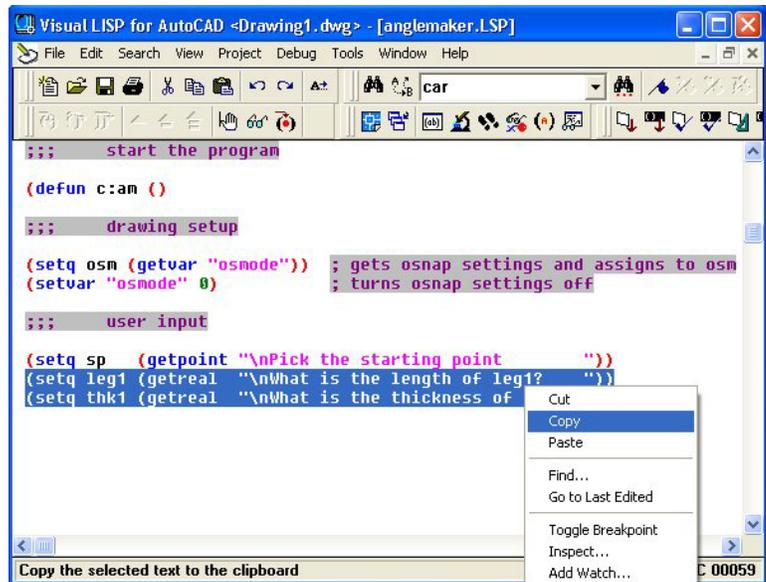


Figure 3.11 – Copying the questions for leg1

If you look at the Visual LISP Editor in figure 3.12, 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. When a designer is checking their responses which they might often do on larger programs, having their answers in a straight line is advantageous.

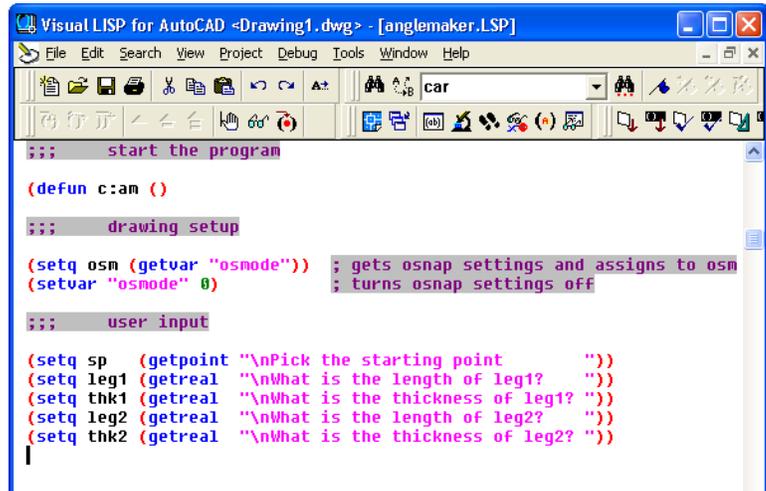


Figure 3.12 – Pasting in the expressions, change 1 to 2

The last `getreal` expression that needs to be included in the code is to inquire about the interior radius of the angle at the junction of the two legs. We are not placing any error checking code in the first two programs, so the responder could key any positive or negative number. We will address those concerns later, but right now we will trust that the designers in your organization will use rational responses, so just type the following code:

```
(setq rad (getreal "\The angle's interior radius is? "))
```

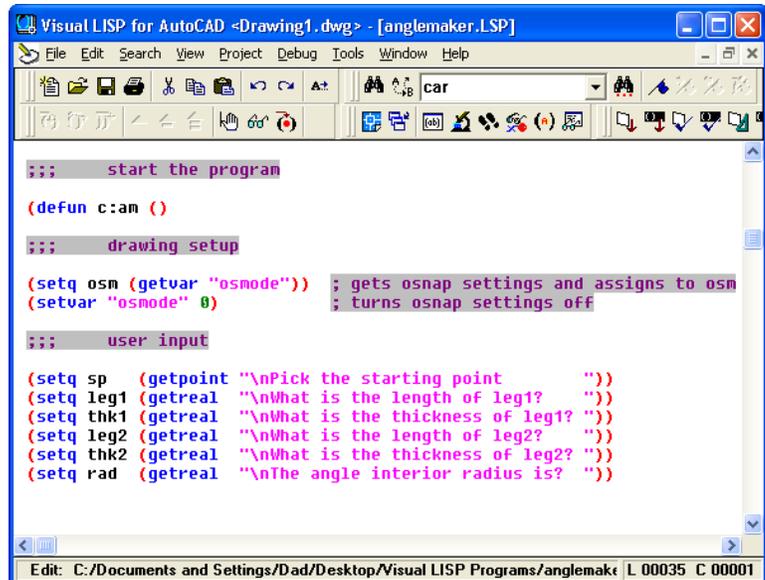


Figure 3.13 – The Rad Getreal Expression

## Step 4 – Do the Math, but Not Right Now

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.

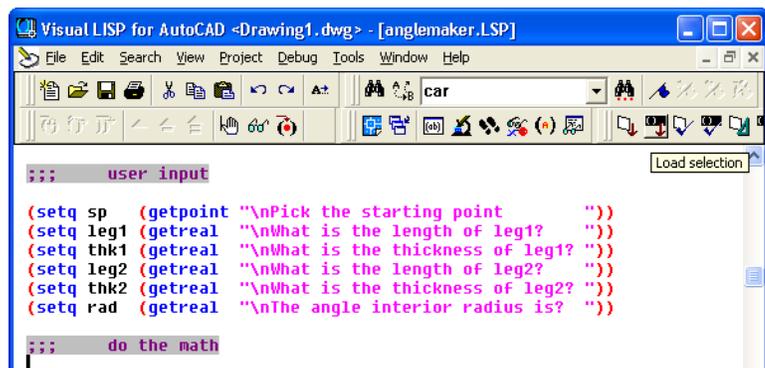


Figure 3.14 – Skip the Do the Math Section

## Step 5 – Point Assignments

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 vertexes. Basically, we did the work when we made the Anglemaker sketch. We set the coordinate P1 is (X1, Y1). P2 is (X5, Y1). P3 is (X3, Y2). P4 is (X4, Y2). P5 is (X2, Y3), P6 is (X2, Y4), and P7 is (X2, Y5). Now we write a `setq` expression setting these grids coordinates to **p1, p2, p3, p4, p5, p6 and p7**.

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, which will work fine for all the 2D orthographic drawings that are being created. (Figure 3.16) In the first line of the Point Assignments code, the **list** expression joins the **x1** and **y1** coordinates creating a vertex which is set to the variable like **p1** using the **setq** function. See figure 3.15 to check the coordinates.

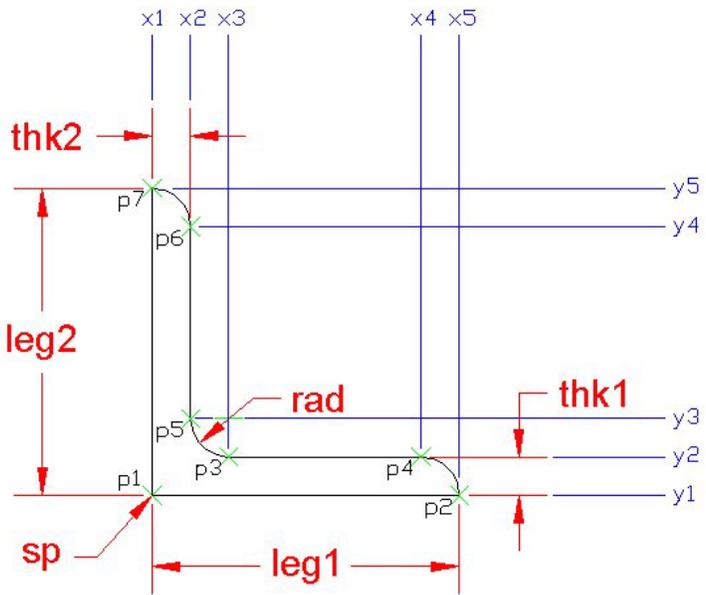


Figure 3.15 – Matching Points with X, Y Grid

Type the following comment:

```
;;; point assignments
```

and the following code:

```
(setq p1 (list x1 y1)
      p2 (list x5 y1)
      p3 (list x3 y2)
      p4 (list x4 y2)
      p5 (list x2 y3)
      p6 (list x2 y4)
      p7 (list x1 y5)
)
```

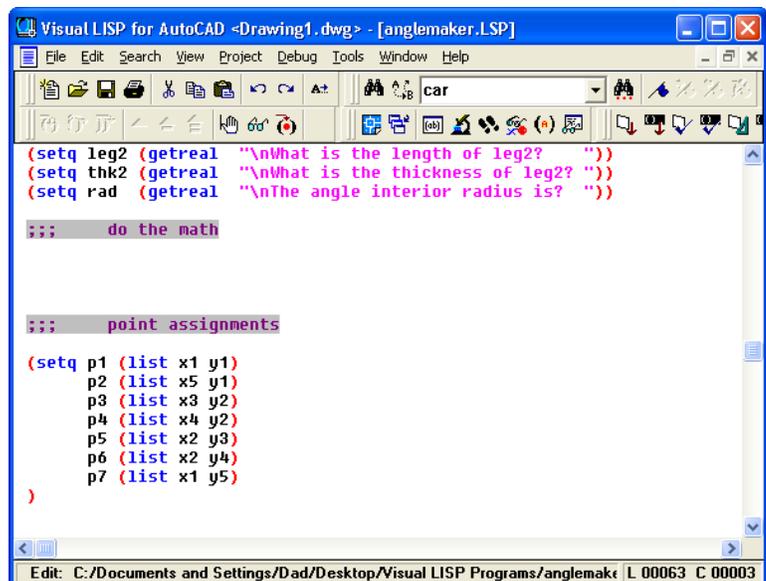


Figure 3.16 - Point Assignments

There will be times when we feel that our present CAD programming knowledge is not adequate to continue with the code, like not having the algebra skills to do the math, or not understanding how to execute the a command required to make the three arcs in this problem. The Point Assignments section of the Construction Code is a perfect example of how to make progress in the writing of a program when we need to work around an area lacking in familiarity. The sketch is critical to the Construction Code process to identify and name each of the variables, to identify and name each of the points and finally to identify and name the grid system used to identify those points. So when we start using a Construction Code template on future program, some inexperienced coders go right to the Point Assignments section first.

## Step 6 – Lets Draw

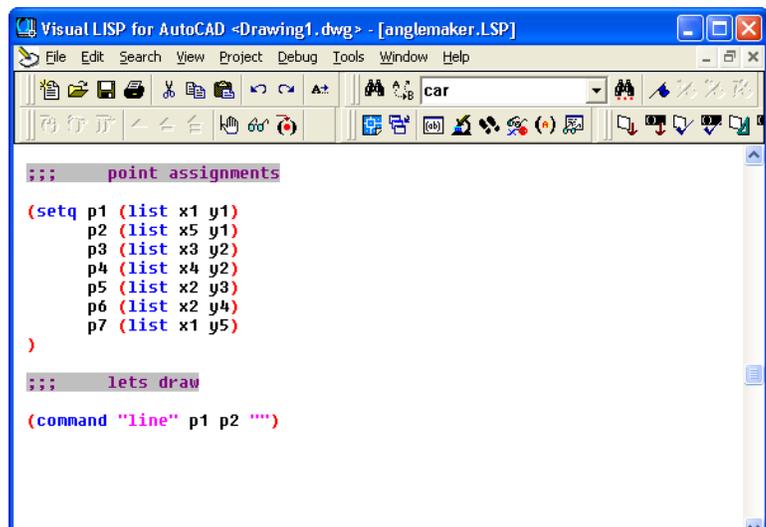
When automatically drawing any entity in AutoCAD, the programmer uses the **command** function which evokes any AutoCAD standard command. After the **command** function is typed, the command **"line"** follows in quotes, then by the point vertexes **p1 p2** of the first line and finally **""** to end the line command.

Type the following code:

```
;;; lets draw
```

```
(command "line" p1 p2 "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 lines is just too easy.



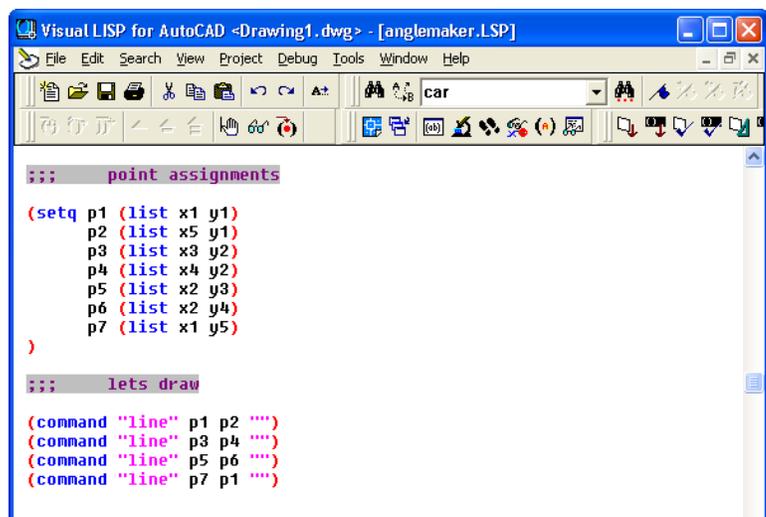
The screenshot shows the Visual LISP for AutoCAD interface. The title bar reads "Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]". The menu bar includes File, Edit, Search, View, Project, Debug, Tools, Window, and Help. The toolbar shows various icons, including a "car" button. The command line area contains the following code:

```
;;; point assignments
(setq p1 (list x1 y1)
      p2 (list x5 y1)
      p3 (list x3 y2)
      p4 (list x4 y2)
      p5 (list x2 y3)
      p6 (list x2 y4)
      p7 (list x1 y5)
)
;;; lets draw
(command "line" p1 p2 "c")
```

Figure 3.17 – Drawing a single line

Unlike in the Boxcircle program, where we gave command instructions to make a square in this code we draw four separate lines end each line command with an Enter by keying a double quote **""** in the command expression. So again check the sketch in figure 3.15 to get the starting and ending points of each line and type the last three expressions of code:

```
(command "line" p3 p4 "")
(command "line" p5 p6 "")
(command "line" p7 p1 "")
```



The screenshot shows the Visual LISP for AutoCAD interface. The title bar reads "Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]". The menu bar includes File, Edit, Search, View, Project, Debug, Tools, Window, and Help. The toolbar shows various icons, including a "car" button. The command line area contains the following code:

```
;;; point assignments
(setq p1 (list x1 y1)
      p2 (list x5 y1)
      p3 (list x3 y2)
      p4 (list x4 y2)
      p5 (list x2 y3)
      p6 (list x2 y4)
      p7 (list x1 y5)
)
;;; lets draw
(command "line" p1 p2 "")
(command "line" p3 p4 "")
(command "line" p5 p6 "")
(command "line" p7 p1 "")
```

Figure 3.18 – Draw all Four Lines

Now, we will be using a command that is rarely selected among the tools in the draw toolbar, Arc. In AutoCAD, arcs are drawn in a counterclockwise direction so we need to select **p2** as the starting point of the first arc. We chose to use the starting point, end point and radius definition to draw the arcs, but other coders may want to use another drawing strategy which will be fine. Keying an “e” for endpoint, we follow with point **p4**. Finally, we key “r” for radius and type **thk1** to supply the radius. (Figure 3.19)

```

Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]
File Edit Search View Project Debug Tools Window Help
car
;;; point assignments
(setq p1 (list x1 y1)
      p2 (list x5 y1)
      p3 (list x3 y2)
      p4 (list x4 y2)
      p5 (list x2 y3)
      p6 (list x2 y4)
      p7 (list x1 y5)
)
;;; lets draw
(command "line" p1 p2 "")
(command "line" p3 p4 "")
(command "line" p5 p6 "")
(command "line" p7 p1 "")
(command "arc" p2 "e" p4 "r" thk1)

```

Figure 3.19 – Draw an Arc

Arcs are less forgiving if we make an error in writing the code since if the AutoCAD program finds that creating the curved entity is impossible, nothing will be placed on the graphics screen to give us any clue what went wrong. When making a mistake with a point in the line command will have the CAD program drawing a point from lets say p1 to p3 and we can see that there is an error and that the line should go from p1 to p2. Check your beginning and ending arc points carefully and use the right radius variable.

```

Visual LISP for AutoCAD <Drawing1.dwg> - [anglemaker.LSP]
File Edit Search View Project Debug Tools Window Help
car
)
;;; lets draw
(command "line" p1 p2 "")
(command "line" p3 p4 "")
(command "line" p5 p6 "")
(command "line" p7 p1 "")
(command "arc" p2 "e" p4 "r" thk1)
(command "arc" p6 "e" p7 "r" thk2)
(command "arc" p5 "e" p3 "r" rad )

```

Figure 3.20 – Draw all the Arcs

So key the following code to create the three arcs:

```

(command "arc" p2 "e" p4 "r" thk1)
(command "arc" p6 "e" p7 "r" thk2)
(command "arc" p5 "e" p3 "r" rad)

```

Again the most common error in placing arcs in a drawing is changing the order of the points. Once your program is running, go ahead and change point p2 and p4 and we will see the arc turn into an interior direction similar to the third expression.

## Step 7 – End the Program

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  
)
```

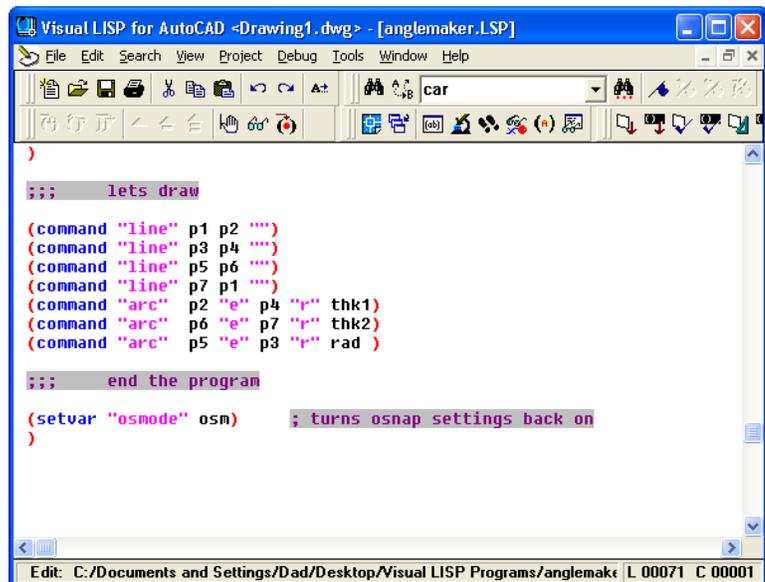


Figure 3.21 – End the Program

## Return to Step 4 – Do the Math

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`, `x4`, `x5`, `y1`, `y2`, `y3`, `y4` and `y5`.

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 (7, 2, 0) then `(car sp)` will return as 7 and be assigned to the variable `x1`. So the `car` function returns the first number in the list.

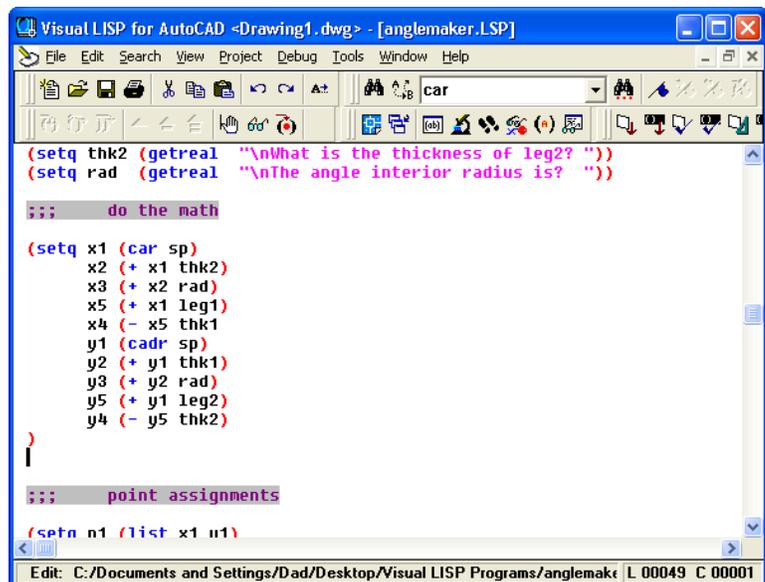


Figure 3.22 – Do the Math

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 (7, 2, 0) then `(cadr sp)` will return as 2 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**, **x3**, **x4** and **x5**. To obtain the **x2** coordinate, use the addition function **+** to add the variable **thk2** to the **x1** value to build the expression:

$$(+ \text{ x1 thk2})$$

Now assign the value to **x2**

$$\text{x2 } (+ \text{ x1 thk2})$$

To find **x3**, we have a single addition expression where we add the variable **rad** to **x2** using the function **+**.

$$(+ \text{ x2 rad})$$

Now assign the value to **x3**

$$\text{x3 } (+ \text{ x2 rad})$$

Now here is a different strategy to keep the math expressions shorter. Go ahead and find **x5**, we have a single addition expression where we add the variable **leg1** to **x1** using the function **+**.

$$(+ \text{ x1 leg1})$$

Now assign the value to **x5**

$$\text{x5 } (+ \text{ x1 leg1})$$

Now find **x4**, we have a single addition expression where we subtract the variable **thk1** from **x5** using the function **-**.

$$(- \text{ x5 thk1})$$

Now assign the value to **x4**

$$\text{x4 } (- \text{ x5 thk1})$$

If we did not change tactics and us the subtract function, the expression to find **x4** would have been something like this:

$$\text{x4 } (+ \text{ x1 } (- \text{ leg1 thk1}))$$

So changing the order in which we define the variables of the grid coordinates can simplify the algebraic formula to compute a point's position.

Repeat the coding process with the y-coordinates y2, y3, y4 and y5; making sure your code appears as below.

```
(setq x1 (car sp)
      x2 (+ x1 thk2)
      x3 (+ x2 rad)
      x5 (+ x1 leg1)
      x4 (- x5 thk1)
      y1 (cadr sp)
      y2 (+ y1 thk1)
      y3 (+ y2 rad)
      y5 (+ y1 leg2)
      y4 (- y5 thk2)
      )
```

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 3.23) to open the Load / Unload Application window.

## Load the AutoLISP Program

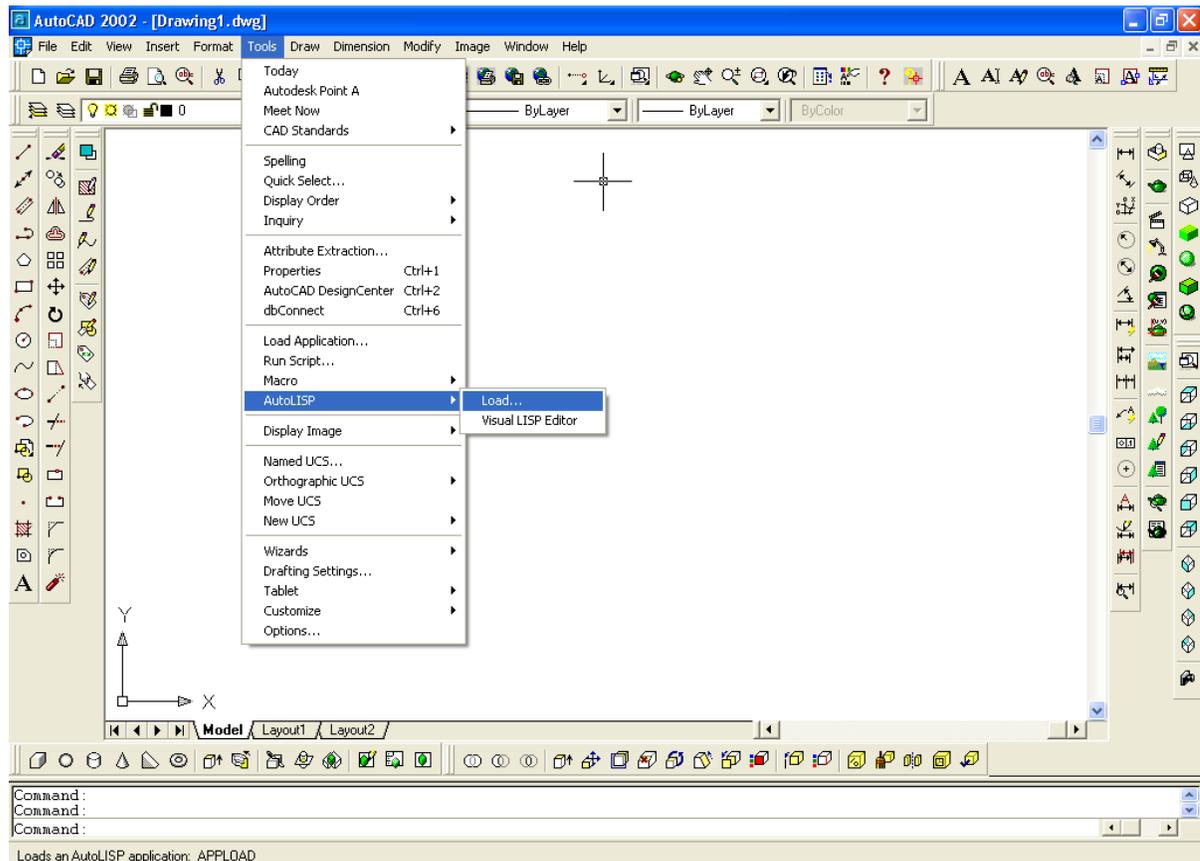


Figure 3.23 – Starting to Load the Application

The Look in list box should be displaying the Visual LISP Programs folder so then select the program “anglemaker” 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, “anglemaker.LSP successfully loaded.” (Figure 3.24)

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. The message ends with “type am to start the program” (Figure 3.25)

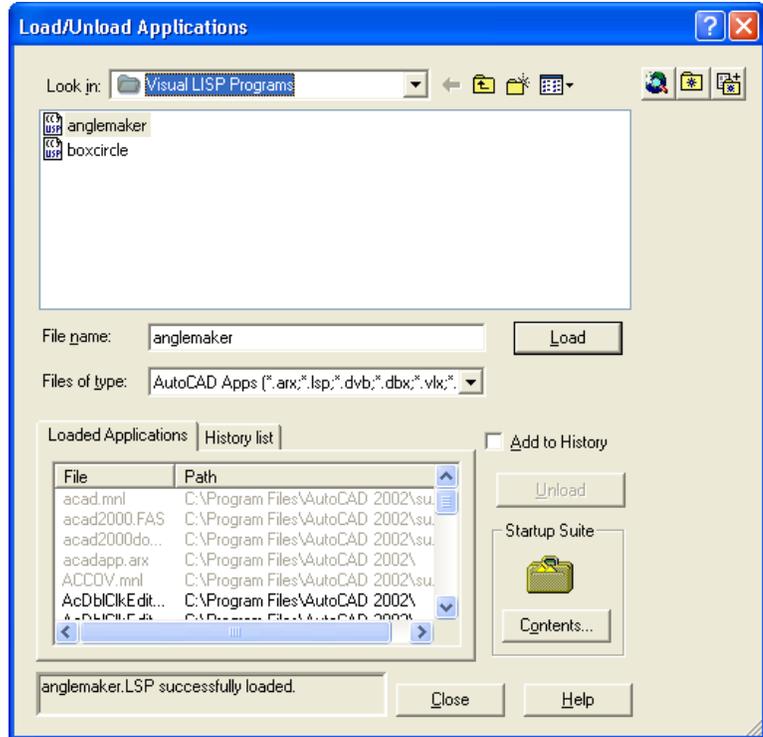


Figure 3.24 – The Load / Unload Applications Window

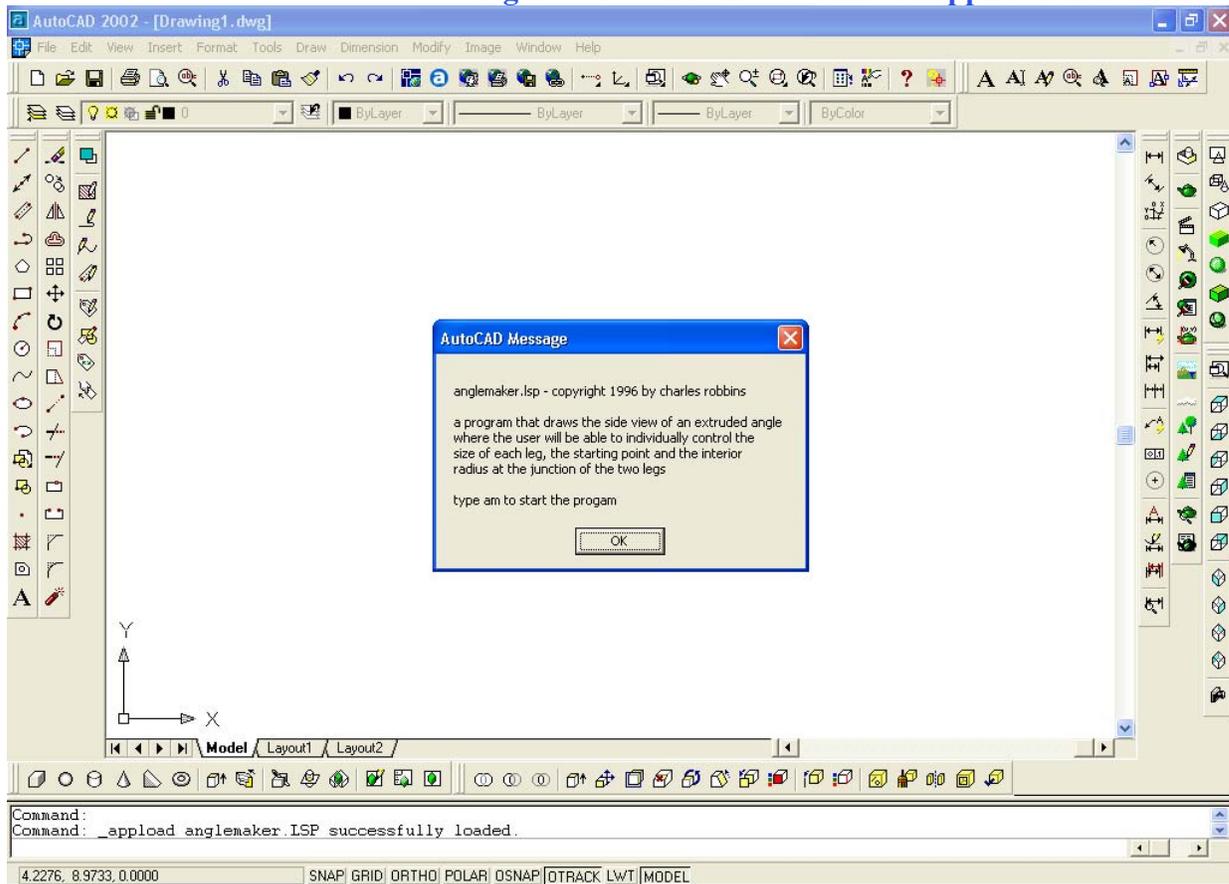
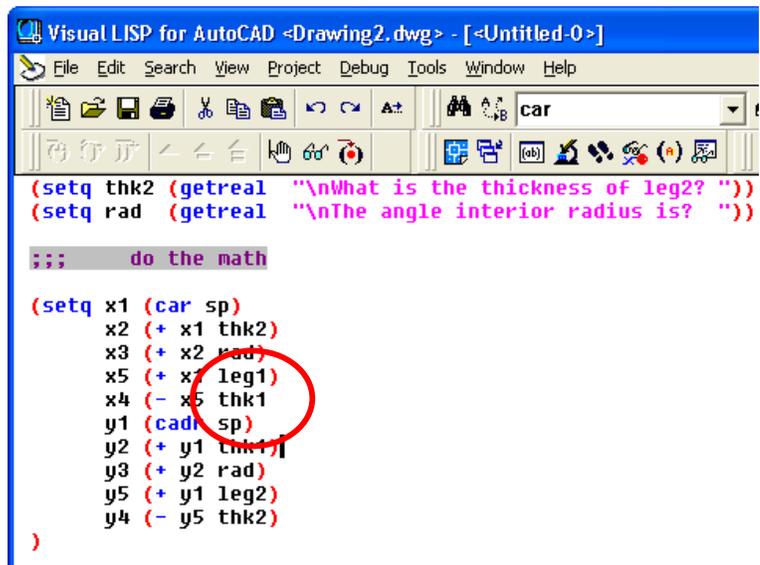


Figure 3.25 – When the Program Loads an AutoCAD Message Appears. Press OK.

## Troubleshooting the AutoLISP Program

Press the OK button if you agree with the message and follow your own instructions by typing **am** at the command line. When my program was executed, a malformed list error message was present on the command line. In most cases the error is a missing parenthesis which will cause the program to malfunction. As you see in figure 3.26, a right parenthesis was missing after the variable **thk1** in the **x4 (- x5 thk1** expression. To fix the problem, a right parenthesis was added.



```
(setq thk2 (getreal "\nWhat is the thickness of leg2? "))
(setq rad (getreal "\nThe angle interior radius is? "))

;;; do the math

(setq x1 (car sp)
      x2 (+ x1 thk2)
      x3 (+ x2 rad)
      x5 (+ x1 leg1)
      x4 (- x5 thk1
      y1 (cad sp)
      y2 (+ y1 thk1)
      y3 (+ y2 rad)
      y5 (+ y1 leg2)
      y4 (- y5 thk2)
)
```

Figure 3.26 – Malformed List Error with Missing )

## Executing the AutoLISP Program

Pick the starting point at the lower left hand corner of the graphical screen or we can place the angle in a drawing where the side view is desired. Next, there are five questions asking the length of leg1, the thickness of leg1, the length of leg2, the thickness of leg2 and finally the interior radius of the angle.

We answered with 3.50, 0.375, 2.875, 0.3125 and 0.344 and after entering the data shown in figure 3.28; the AutoCAD program drew the angle perfectly. The finished drawing can be seen in figure 3.29.

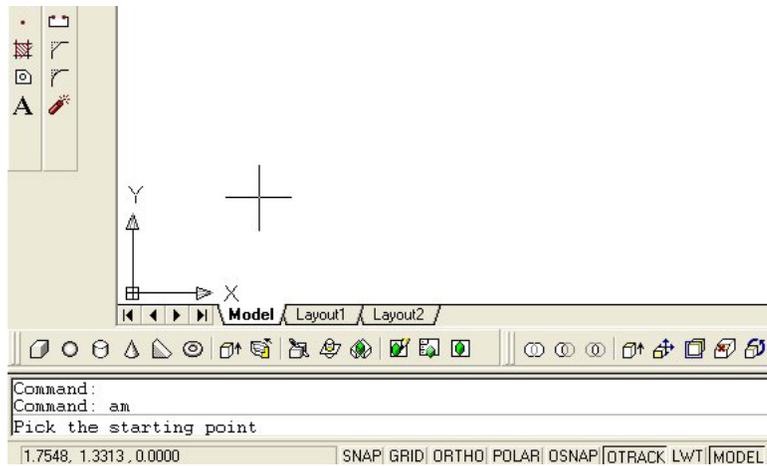
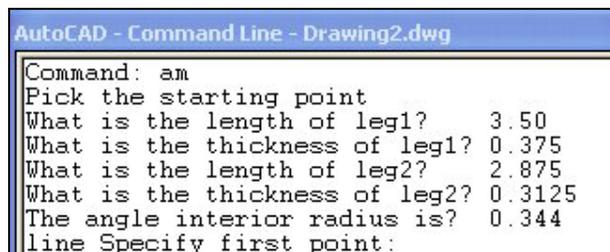
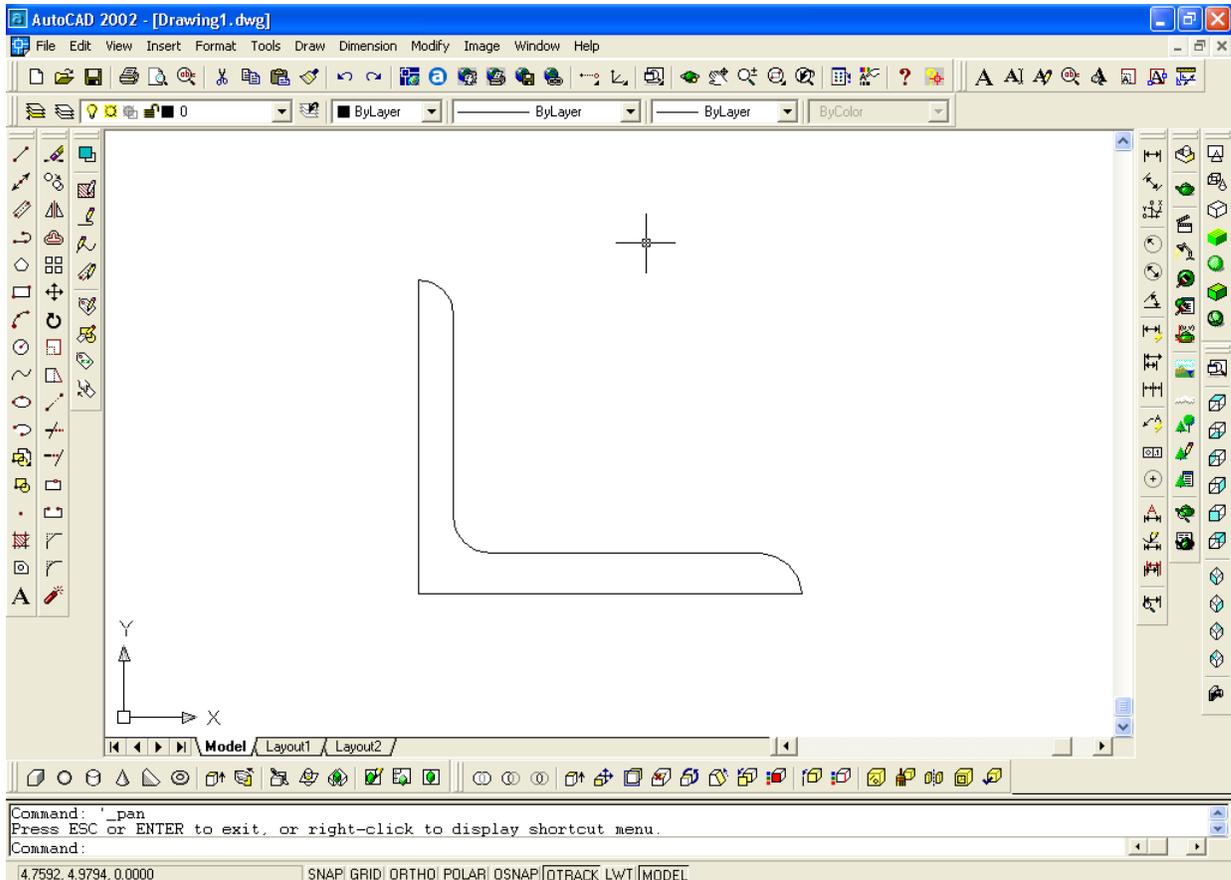


Figure 3.27 – Picking the Starting Point of the Angle



```
AutoCAD - Command Line - Drawing2.dwg
Command: am
Pick the starting point
What is the length of leg1? 3.50
What is the thickness of leg1? 0.375
What is the length of leg2? 2.875
What is the thickness of leg2? 0.3125
The angle interior radius is? 0.344
line Specify first point:
```

Figure 3.28 – Input at the AutoCAD Command Line



**Figure 3.29 – The Finished Drawing**

This is the second time that we have completed all seven steps of the Construction Code and finished a workable program that will run in any version of AutoCAD. As we said in the beginning, most engineering programming students young or experienced expect the process to fill them with anxiety, since every project will require an extensive re-learning of the methods or most surveyed expected little or no procedures at all, but many do expect having to discover rules of executing different programming functions. There are many who become skilled at the Construction Code procedure and find that the other clever functions and strategies that are revealed in the following chapters to be more than they what they need at this time.

The next chapter will take you through six projects where we will reinforce the skills that were gained in Chapters 1, 2 and 3.