Chapter 7A

Making a Multiview Drawing

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

1. **Learn How to Make Multiple Views with Visual LISP**
2. **Starting the Code by Launching the Visual LISP Editor**
3. **Saving the Object Snap Settings and Then Turning Them Off**
4. **Using Getpoint to Obtain a Point on the Graphical Display**
5. **Using Getreal to Obtain a Real Number from the Keyboard**
6. **Creating Layers with the Visual AutoLISP Command Function**
7. **Doing the Math in Visual AutoLISP**
8. **Making Point Assignments in Visual AutoLISP**
9. **Drawing in Visual AutoLISP**
10. **Ending the Program**
11. **Saving the Program**
12. **Loading the Program**
13. **Running the Program**

Learn to Make Multiple Views with Visual LISP

In this chapter, we will create more than one view so that by the end of this textbook, we will be making entire drawings with the Visual AutoLISP code. In Figure 7A.1, we see a sketch of a bracket. We see that there is a front and a top view. There are four mounting holes on the base of the bracket and a single through hole at the top of the metal development. We define the part by a length (lg), a width (wd), a height (hgt), a platform width (pf), metal thickness (thk), the mounting hole diameter (mh) and the through hole diameter (hl). We can observe three types of linetypes; continuous, hidden and center lines.

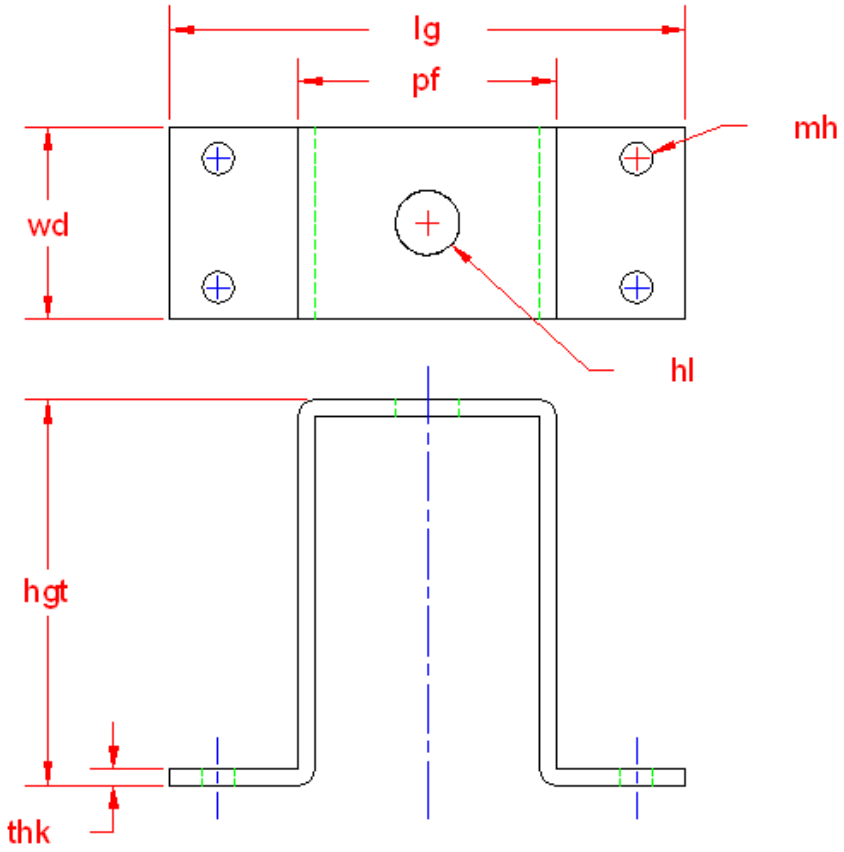

Figure 7A.1 – Sketch of the Bracket

After making a sketch, we will identify every endpoint of the lines in the bracket with a "P" and a number. In our problem, we will use the mirror command multiple times, so some lines, arcs and circles will not be drawn but copied using modify commands. After the points are shown, we then visually define the points by imposing an X and Y grid. Later, when we code the bracket routine, we will define P1 as X1 and Y2. We have found that this technique makes the program a little longer, but takes the complexity out of the algebra.

The next sketch shown in Figure 7A.2 illustrates the x and y grid that will make up the coordinates for every point.

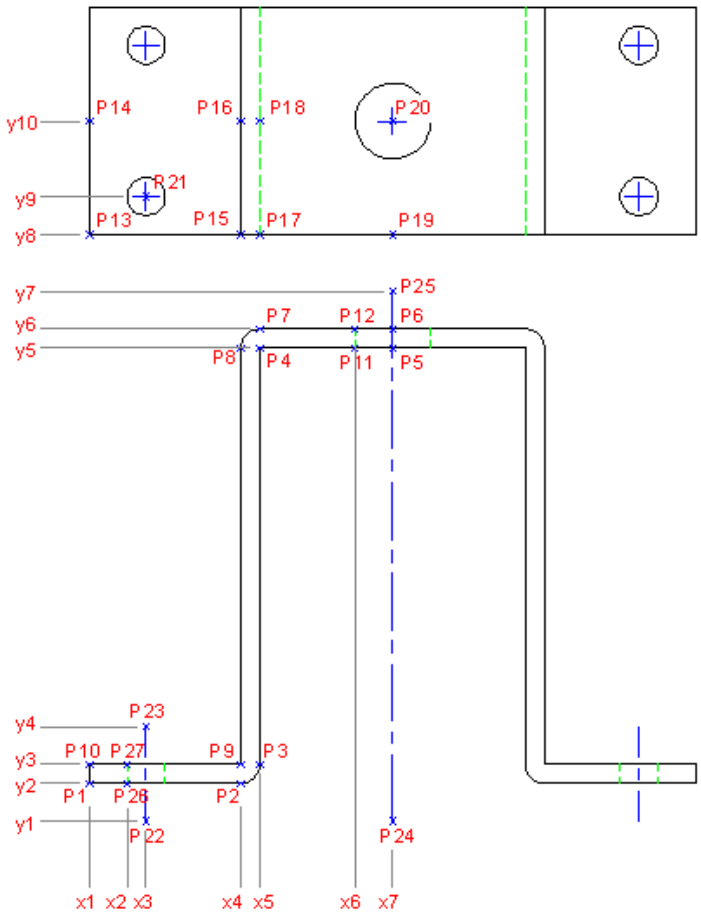

Figure 7A.2 – X and Y Grid

This program will have every section of the Construction Code. We will create layers to place the different types of lines on the drawing. We will draw lines and arcs. We will use the modifying tool, Mirror command with the Crossing selection technique to reduce the amount of the entities to create the bracket. In the table below, we can see the Construction steps listed for our benefit.

Step 1	**Start the program**
Step 2	**Drawing setup**
Step 3	**User input**
Step 4	**Do the math**
Step 5	**Point assignments**
Step 6	**Lets draw**
Step 7	**End the program**

The math on this problem is very simple, so we will do all the procedures in order on this exercise. The first step we need to take is to launch the Visual LISP Editor in AutoCAD.

Starting the Code by Launching the Visual LISP Editor

Open the Visual LISP Editor and on the first line type the comment

;;; bracket.lsp

The program name is always on the first line of the code. The semicolons cause the statement to become a comment so the line of code will not be read.

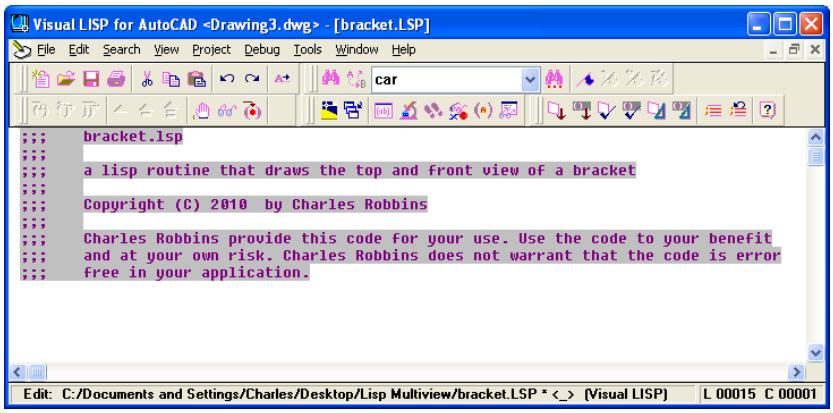

Figure 7A.3 – Starting the Bracket Program

The next comments in the program will be the details concerning what the routine will do. In this program, there are comments after almost every line of code.

Next, we will create an 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.

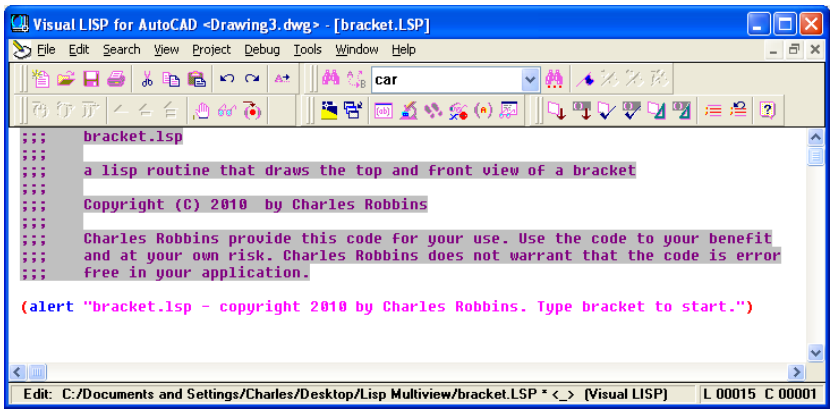

Figure 7A.4 – Adding the Alert Expression

Add a new comment

;;; start the 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.

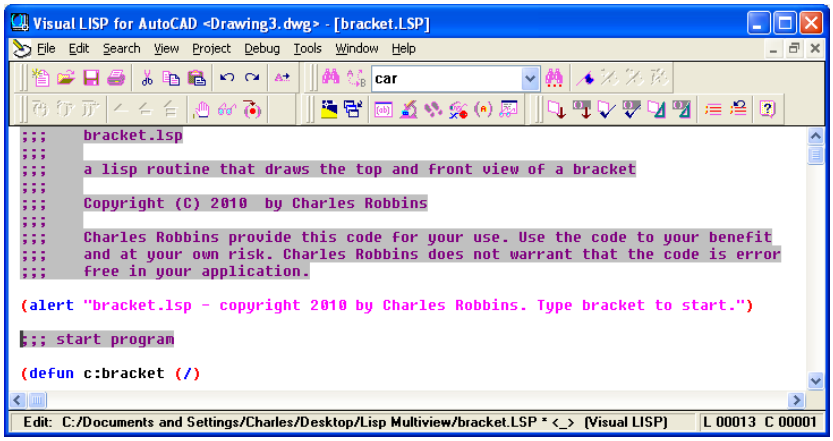

Figure 7A.5 – The Defun Expression

Practice typing the following examples of the **alert** function at the command line of AutoCAD.

Function	Name	Description
alert	**AutoCAD Message**	**The alert function will create an AutoCAD message window appear on the graphical display with an OK button to close the message window.**
Examples		
At the beginning of the program	**(alert "bracket.lsp - copyright 1999 by charles robbins. type bracket to start")**	Window appears on the graphical display
As an error prompt	**(alert "Error: Type units in inches")**	Window appears on the graphical display

Next, we type **bracket** which will be the execution symbol to start the program. Keep in mind the alert message that stated "type bracket to start". The alert message text and the **defun** symbol must match. The open and closed parenthesis "**()**"following the **bracket** 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.

Practice typing the following examples of the **defun** function at the command line of AutoCAD.

Function	Name	Description
defun	**Define Function**	**The define function leads off the beginning of the program**
Examples		
Place a **c:** in front of the program, hello. Allows **hello** to be typed at the keyboard to execute the code	**(defun c:hello (/) (print "Hi Ya All"))**	Answer: **C:HELLO** Type: **hello** Returns: **"Hi Ya All"**
When the code is used inside another program, do not place the c: in front of the program name	**(defun hello (/) (print "Hi Ya All"))**	Answer: **HELLO** Type: **(hello)** Returns: **"Hi Ya All"**

Saving the Object Snap Settings and then Turning Them Off

In the next section of the code, we will turn off the drawing Object Snaps so they cannot possibly interfere with the insertion of the drawing notes. In order to accomplish this task, we you need to understand the **getvar** and the **setvar** functions. The **getvar** function will obtain a drawing setting, so we can save the number or text string for future use. The **setvar** function will allow us to change a system variable, like turning off the Object Snaps.

Start with a new comment

;;; drawing setup

And type the code

(setq osm (getvar "osmode"))

; gets osnap settings and assigns to osm

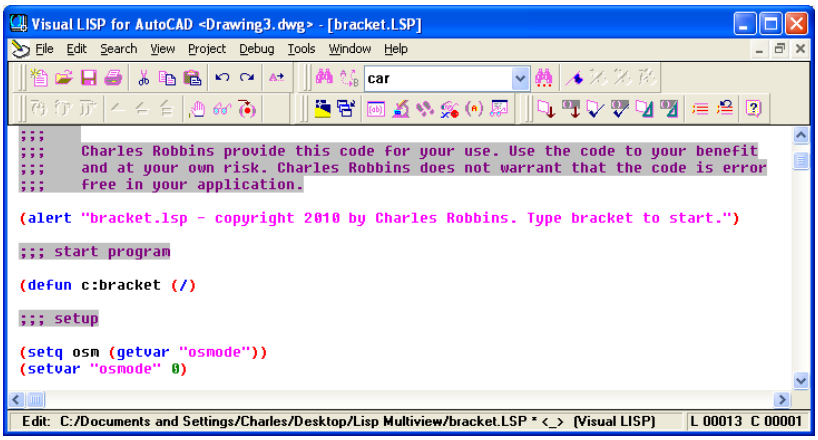

Figure 7A.6 – Saving and Turning Off Object Snaps

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.

Practice typing the following examples of the **setq, getvar** and **setvar** functions at the command line of AutoCAD.

Function	Name	Description
setq	**Set Quotient**	**Allows the user to assign a real number, integer, string or list to a variable**
Examples		
Set the variable **a** the text string **World Class CAD**	**(setq a "World Class CAD")**	Answer: **"World Class CAD"**
Set the variable **counter** the integer **0**	**(setq counter 0)**	Answer: **0**
Set the text height variable **txtht** the real number **0.125**	**(setq txtht 0.125)**	Answer: **0.1250**
Set the point variable **sp** the list of 0,0,0	**(setq sp (list 0.0 0.0 0.0))**	Answer: **(0,0,0)**

Function	Name	Description
getvar	**Get a variable**	**Allows the user to obtain a system variable setting from an AutoCAD drawing**
Examples		
Turn on the endpoint, midpoint, quadrant, intersection and perpendicular Object Snaps	**(setq osm (getvar "osmode"))**	Answer: **179**
Get the AutoCAD version number	**(setq osm (getvar "acadver"))**	Answer: **"16.2s (LMS Tech)"**

Function	Name	Description
setvar	**Get a variable**	**Allows the user to obtain a system variable setting from an AutoCAD drawing**
Examples		
Turn off the Object Snaps	**(setvar "osmode" 0))**	Answer: **0**

Using Getpoint to Obtain a Point on the Graphical Display

In the User Input section of the Construction Code, we need to expand into new areas besides just requesting the starting point and the getting a measurement using the **getreal** function as we did in the first eight programs.

The first function we will examine together is **getpoint**. This tool will allow the program user to select a point on the graphical display with their mouse. Following **getpoint** is a text string usually written is a commanding or questioning format.

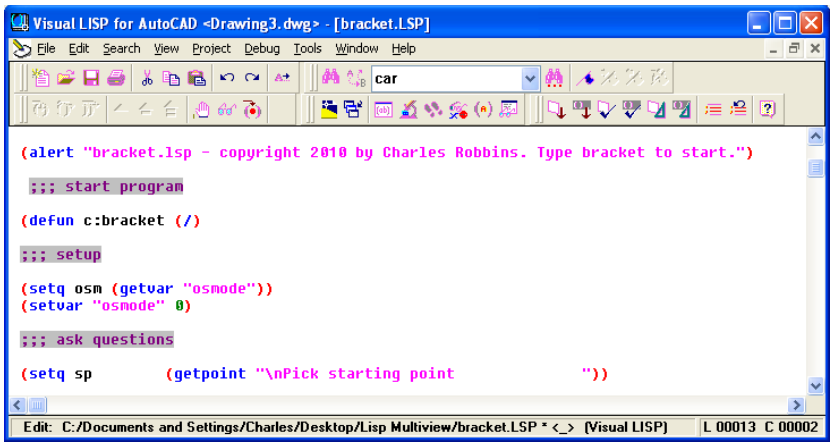

Figure 7A.7 – Using the Getpoint Function

The user input of selecting a point begins with a comment.

;;; user input

Then type the following code:

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

We use the **setq** expression to assign the three point list (X, Y and Z) to the variable **sp** representing the starting point. After the function **getpoint**, 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 the command **"Pick the starting point"** without containing those two characters to start on a new command line in the AutoCAD program.

Periodically we will work at an organization that wants their details in the exact location of their drawing. When we face a programming problem such as this the starting point expression will change. First instance, let us make believe that the detail at this company starts at the X and Y coordinates 14, 10. Then in this bracket code, we would change the starting point expression to:

(setq sp (list 14 10 0))

This will place the beginning of the notes in an exact position for every occasion.

Practice typing the following examples of the **setq** and **getpoint** functions at the command line of AutoCAD.

Function	Name	Description
setq	**Set Quotient**	**Allows the user to assign a real number, integer, string or list to a variable**
Examples		
Set the variable **a** the text string **World Class CAD**	**(setq a "World Class CAD")**	Answer: **"World Class CAD"**
Set the variable **counter** the integer **0**	**(setq counter 0)**	Answer: **0**
Set the text height variable **txtht** the real number **0.125**	**(setq txtht 0.125)**	Answer: **0.1250**
Set the point variable **sp** the list of 0,0,0	**(setq sp (list 0.0 0.0 0.0))**	Answer: **(0,0,0)**

Function	Name	Description
getpoint	**Get a Point**	**Allows the user to obtain a point on the graphical display by selecting with a mouse**
Examples		
Get a starting point	**(setq sp (getpoint "\nPick starting point"))**	Answer: **Pick starting point** Then select a point and the will return a list like: **(30.471 28.4052 0.0)**

Using Getreal to Obtain a Real Number from the Keyboard

To ask the question, "What is the bracket's length?", we will use the **getreal** function. We use **getreal** to allow the LISP program user to type a number containing decimals with their keyboard. The **getreal** expression is set within the **(setq lg …….)** code.

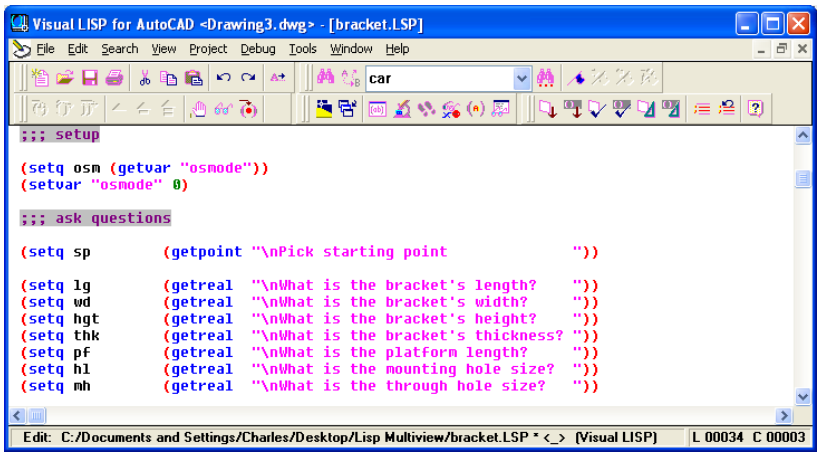

Figure 7A.8 – Using the Getreal Function

So type the following compound expression:

(setq lg (getreal "\nWhat is the bracket's length? "))

The information that the user types with the keyboard is stored in the variable name **lg**. We will never pick a variable name that matches an AutoCAD command.

Whenever we are not quite sure whether the answer is going to be a whole number or a decimal, we will use the **getreal** function. Using another function which will only allow whole numbers will never allow the acceptance of a decimal.

If you look at the Visual LISP Editor in Figure 7A.8, 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.

Here are the user prompts for all of the critical dimensions of the bracket.

```
(setq lg    (getreal "\nWhat is the bracket's length?      "))
(setq wd    (getreal "\nWhat is the bracket's width?       "))
(setq hgt   (getreal "\nWhat is the bracket's height?      "))
(setq thk   (getreal "\nWhat is the bracket's thickness? "))
(setq pf    (getreal "\nWhat is the platform length?      "))
(setq hl    (getreal "\nWhat is the mounting hole size?   "))
(setq mh    (getreal "\nWhat is the through hole size?    "))
```

Practice typing the following examples of the **getreal** function at the command line of AutoCAD.

Function	Name	Description
getreal	**Get a Real Number**	**Allows the user to obtain a real number by allowing the user to type at the keyboard**
Examples		
Ask for a number, user types a whole number and the reply is changed to a real number	**(setq txtht (getreal "\nWhat is the text height?"))**	Answer: **What is the text height?** Then type: **1** **1.0**
Ask for a number, user types a fraction and the reply is changed to a real number	**(setq txtht (getreal "\nWhat is the text height?"))**	Answer: **What is the text height?** Then type: **1/8** **0.125**

Now we will create the layers we need for this project.

Creating Layers with the Visual AutoLISP Command Function

We will control the type of lines and their color with layers. In this problem, we will create four layers and use only two. Two layers, dimension and text will be used in other chapters to dimension and add notes.

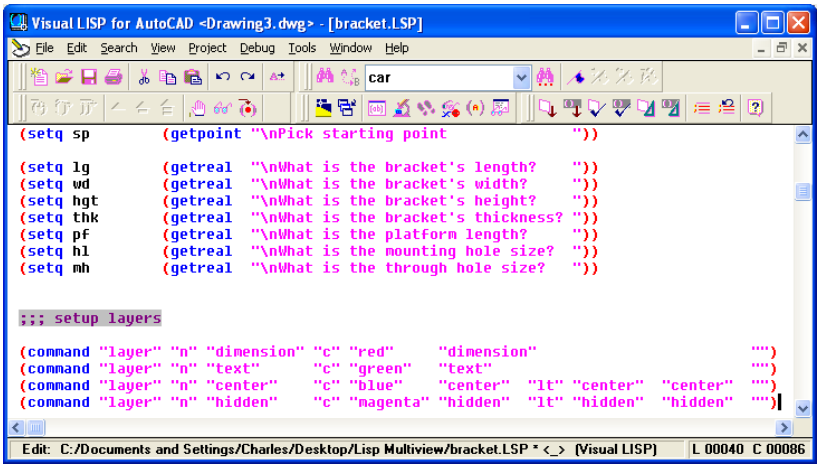

Figure 7A.9 – Creating Layer in AutoLISP

An easy way to make a layer for a detail is to use the command function and to follow the creating a new layer process. This can be harder for individuals just newly training with AutoCAD, since many computer aided design tools are now in dialogue boxes and we cannot easily view all the options that are available with a command function. We will share the most common options in the table below.

Command Layer Function	Command Layer Function
(command "layer" "n" "hidden" “”)	**Makes a new layer named “hidden”**
(command "layer" "c" "8" “hidden" “”)	**Sets the layer color to “color 8” for layer named “hidden”**
(command "layer" "lt" "center" ”center" "")	**Sets the layer linetype to “center” for layer named “center”**
(command "layer" "s" "hidden" “”)	**Sets the current layer as “hidden”**
(command "layer" "f" "hidden" “”)	**Freezes the layer named “hidden”**
(command "layer" "t" "hidden" “”)	**Thaws the layer named “hidden”**
(command "layer" "on" "hidden" “”)	**Turns the layer named “hidden” on**
(command "layer" "off" "hidden" “”)	**Turns the layer named “hidden” off**
(command "layer" "lo" "hidden" “”)	**Locks the layer named “hidden”**
(command "layer" "u" "hidden" “”)	**Unlocks the layer named “hidden”**

We can combine layer options such as new and color and create a command line expression that will both create a new layer and set the color for that layer. When we use the color option **"c"**, the next item is the name or number of the color, followed by the name of the layer. To end the layer command expression, place an open and closed quote **""** at the end of the code and then a closed parenthesis.

Type the following lines in the bracket program.

;;; setup layers

```
(command "layer" "n" "dimension" "c" "red"    "dimension"              "")
(command "layer" "n" "text"       "c" "green"  "text"                   "")
```

Now we will make the center, hidden and section layers in the routine. We may be using different colors or layer names than what your organization uses, so feel to free to make changes to the layer name or color that defines your group's standard layers. changes

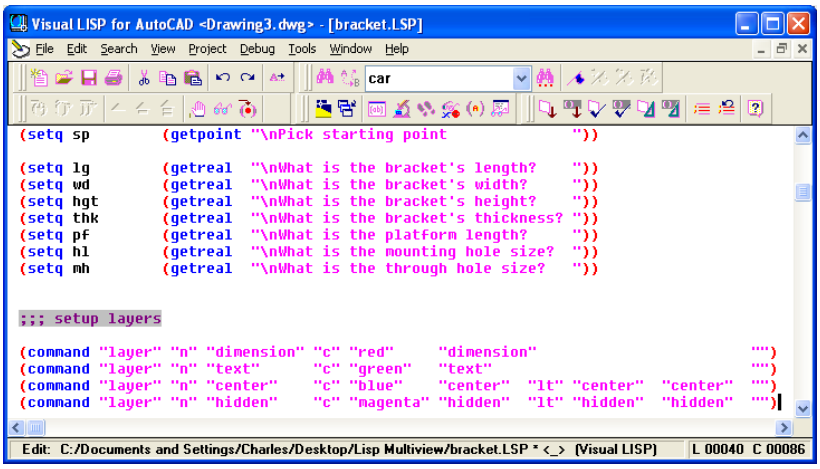

Figure 7A.10 – Creating Layer with Special Linetypes

If the layer name, color and linetype already exist in the drawing, nothing will change when these lines of code execute.

Type the following lines in the bracket program.

```
(command "layer" "n" "center"  "c" "blue"    "center"  "lt" "center"  "center"  "")
(command "layer" "n" "hidden"  "c" "magenta" "hidden"  "lt" "hidden"  "hidden"  "")
```

Doing the Math in Visual AutoLISP

Now, we will do the math section of the code. Again the **setq** function is the choice for assigning values to the variables X1, X2, X3, X4, X5, X6, X7, Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Y9, Y10, and Y11.

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.

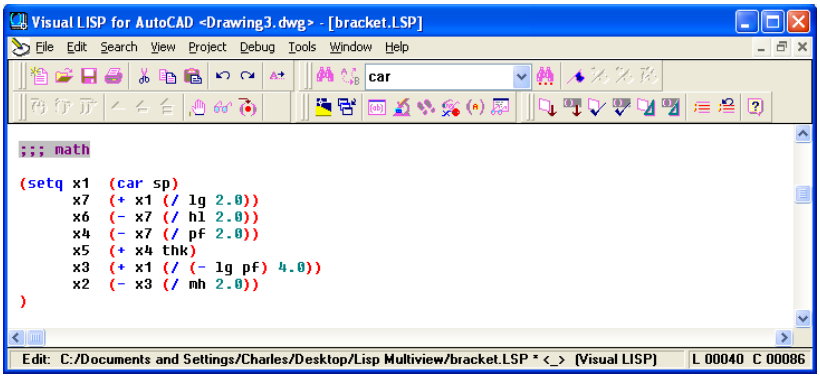

Figure 7A.11 – Defining the X-Ordinates

That explains the use of **car** to find the coordinates x1, now we have to continue down the X grid to obtain value for **x7**. To obtain the **x7** coordinate, use the addition function **+** to add to the **x1** value. To get the number to add to the **x1**, we have to divide the bracket width **LG** by two using the divide function **/** like so. Notice that the function is written first, followed by the numerator **LG** and then the denominator **2.0**.

(/ LG 2.0)

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

(+ x1 (/ LG 2.0))

Now assign the value to x2

x2 (+ x1 (/ LG 2.0))

Type the following lines in the bracket program using the sketch in Figure 7A.2 to find the X ordinate the dimension that defines the horizontal measurements.

```
;;; math

(setq x1  (car sp)
      x7  (+ x1 (/ lg 2.0))
      x6  (- x7 (/ hl 2.0))
      x4  (- x7 (/ pf 2.0))
      x5  (+ x4 thk)
      x3  (+ x1 (/ (- lg pf) 4.0))
      x2  (- x3 (/ mh 2.0))
)
```

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.

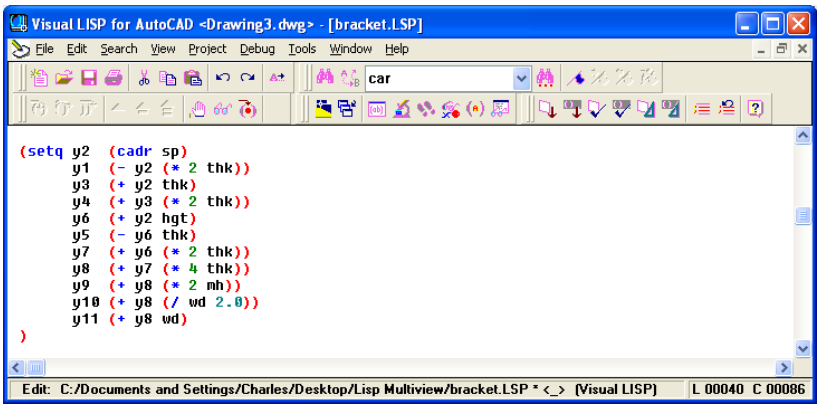

Figure 7A.12 – Defining the Y-Ordinates

To find the value for the variable **y3**, we add the thickness **thk** to the variable **y2**. We use the adding LISP function **+** to make the expression

y3 (+ y2 thk)

Other than for the measurement **y2**, all of the vertical distances are compiled by using the adding or subtracting functions. Type the following lines in the bracket program using the sketch in Figure 7A.2 to find the Y ordinate the dimension that defines the vertical measurements.

```
(setq y2  (cadr sp)
      y1  (- y2 (* 2 thk))
      y3  (+ y2 thk)
      y4  (+ y3 (* 2 thk))
      y6  (+ y2 hgt)
      y5  (- y6 thk)
      y7  (+ y6 (* 2 thk))
      y8  (+ y7 (* 4 thk))
      y9  (+ y8 (* 2 mh))
      y10 (+ y8 (/ wd 2.0))
      y11 (+ y8 wd)
)
```

Making Point Assignments in Visual AutoLISP

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 Bracket sketch. When we define the points for the footer, remember when we read that coordinate P0 is (X1, Y1). P1 is (X1, Y2). P2 is (X4, Y2). P3 is (X5, Y3). Now we write a **setq** expression setting these grids coordinates to the points **p1**, **p2**, **p3**, **p4** through **p28**.

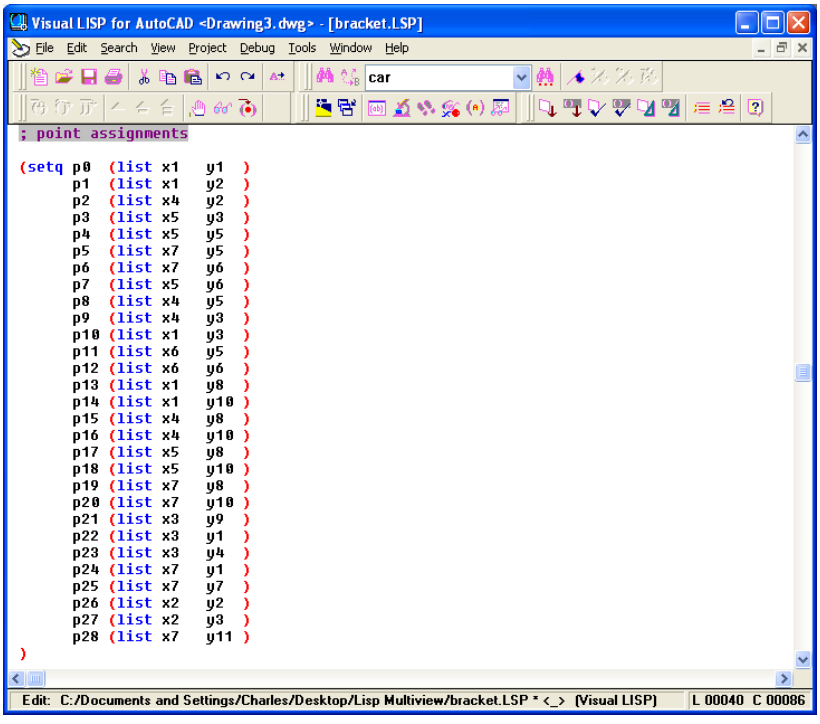

Figure 7A.13 – Defining the Point Assignments

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. (See Figure 7A.13)

Type the following lines in the bracket program using the sketch in Figure 7A.2 to find the X and Y coordinate for each point.

```
; point assignments

(setq p0  (list x1  y1 )
      p1  (list x1  y2 )
      p2  (list x4  y2 )
      p3  (list x5  y3 )
      p4  (list x5  y5 )
      p5  (list x7  y5 )
      p6  (list x7  y6 )
      p7  (list x5  y6 )
      p8  (list x4  y5 )
      p9  (list x4  y3 )
      p10 (list x1  y3 )
      p11 (list x6  y5 )
      p12 (list x6  y6 )
      p13 (list x1  y8 )
      p14 (list x1  y10 )
      p15 (list x4  y8 )
      p16 (list x4  y10 )
      p17 (list x5  y8 )
      p18 (list x5  y10 )
      p19 (list x7  y8 )
      p20 (list x7  y10 )
      p21 (list x3  y9 )
      p22 (list x3  y1 )
      p23 (list x3  y4 )
      p24 (list x7  y1 )
      p25 (list x7  y7 )
      p26 (list x2  y2 )
      p27 (list x2  y3 )
      p28 (list x7  y11 )
)
```

Drawing in Visual AutoLISP

Before drawing the first line in the Bracket detail, we will set the current layer as "0". Before we draw an entity using the command functions of line, circle and arc tools, we will continually place the article on the precise drawing layer. Type the following code to set the current layer to "0".

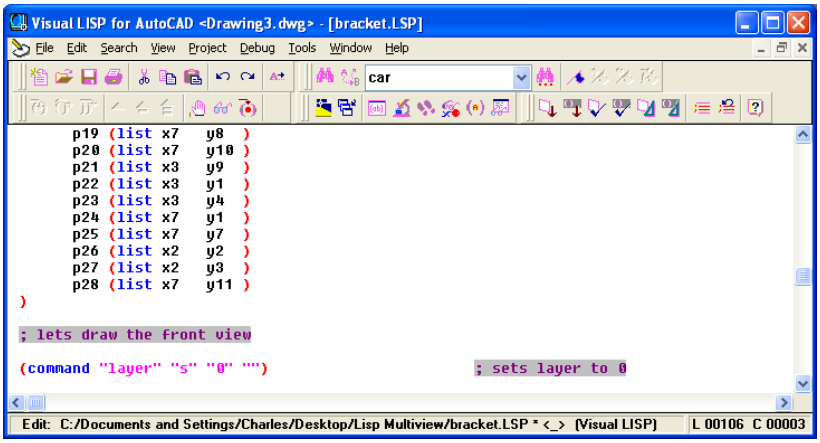

Figure 7A.14 – Setting the 0 Layer as Current

(command "layer" "s" "0" "")

Now that the layer is set to 0, we will proceed to draw the front view of the bracket. 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 did in Chapter 6 with the **saveimg** function. After the **command** function is typed, the command **"line"** follows in quotes, then by the point vertexes **p2 p1 p10 p9 p8** of the line segment and finally **""** to end the command.

Type the following code:

```
;;;   lets draw

(command "line" p2 p1 p10 p9 p8 "")
```

And the LISP routine draws four lines representing the perimeter of the bracket on the AutoCAD graphical display. We draw three more lines using the line command twice more.

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

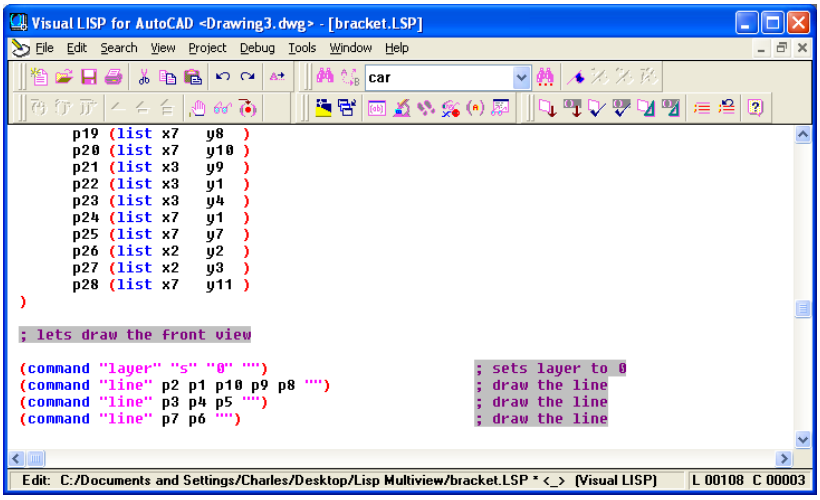

Figure 7A.15 – Drawing Lines with LISP Commands

Now, we will draw the two arcs. 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.

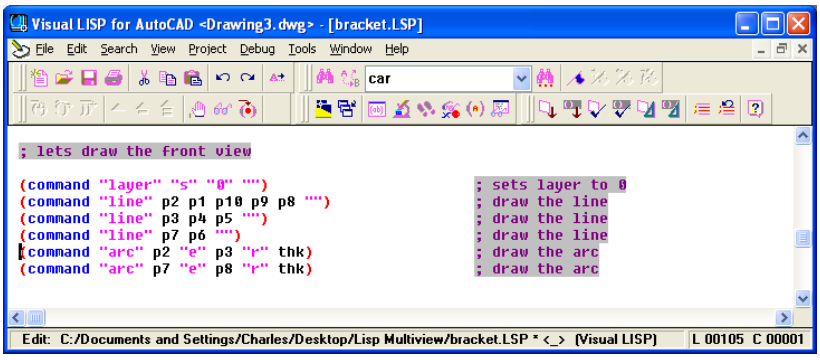

Figure 7A.16 – Drawing Arcs with LISP Commands

Keying an **"e"** for endpoint, we follow with point **p3**. Finally, we key **"r"** for radius and type **thk** to supply the radius. (Figure 7A.16)

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 let's 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.

So key the following code to create the three arcs:

```
(command "arc" p2 "e" p3 "r"  thk)
(command "arc" p7 "e" p8 "r"  thk)
```

We will draw a hidden line after setting the current layer to "hidden".

```
(command "layer" "s" "hidden" "" )
(command "line" p26 p27 "")
```

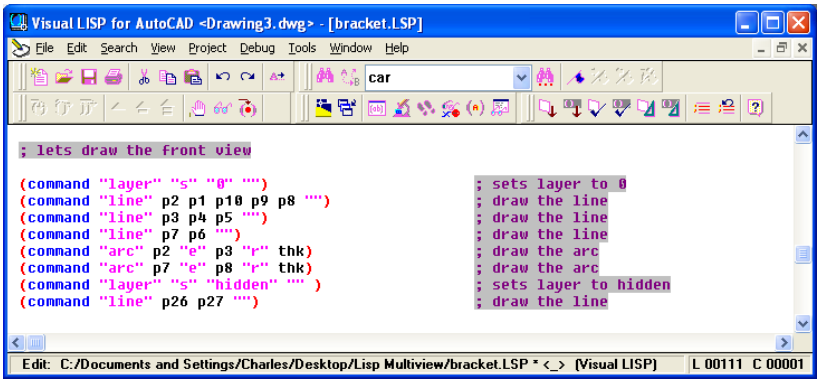

Figure 7A.17 – Draw a Hidden Line

We will mirror the left side of the front view of the bracket to the right across a vertical centerline from **p22** to **p23**. For the select last function to work properly, all the entities need to be visible, so we first perform the zoom extents command. Then, we mirror the last line we drew across the centerline as shown.

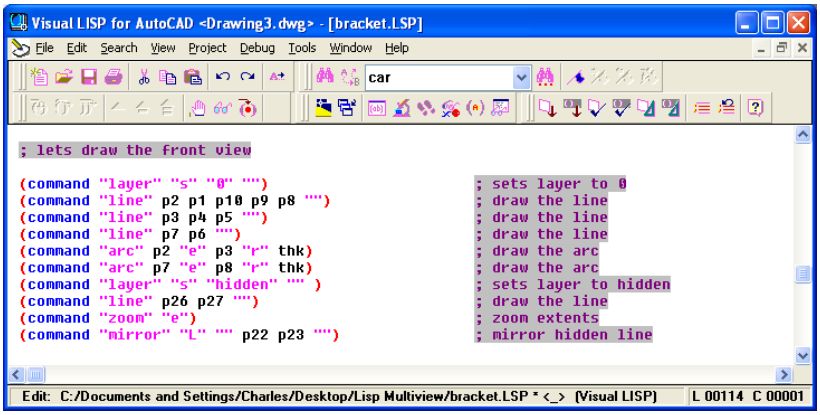

Figure 7A.18 – Mirror a Hidden Line

```
(command "zoom" "e")                    ; zoom extents
(command "mirror" "L" "" p22 p23 "")    ; mirror hidden line
```

We draw another hidden line and then a centerline after setting the layer to “center”. We finish the front view with another mirror using the crossing selection technique. The last line drawn in the front view is the main vertical centerline.

We now will draw the top view.

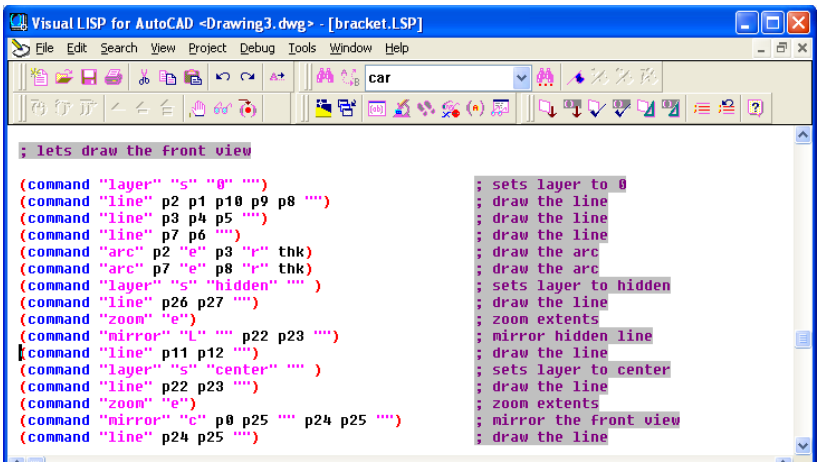

Figure 7A.19 – Finish the Front View

We can draw the top view using the line, circle, zoom, mirror and layer commands. Some programmers will draw entities in different order and this does really matter. The essential task is to use the correct points and put the lines and circles in the correct position.

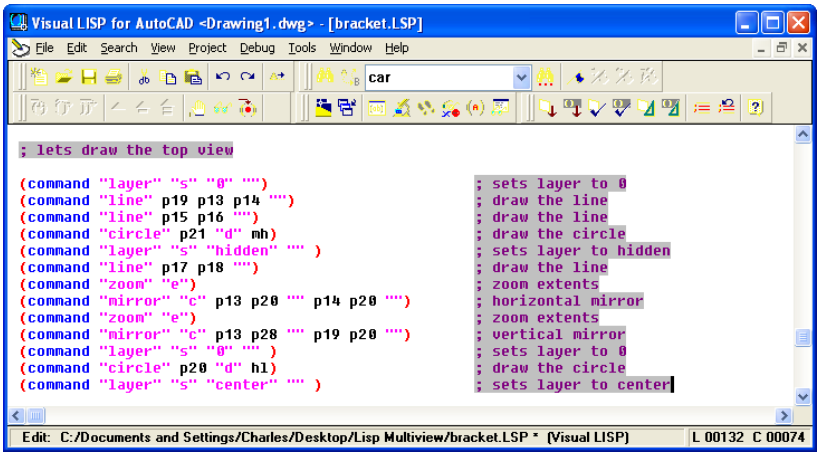

Figure 7A.20 – Draws the Wood Components

Ending 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.

(setvar "osmode" osm)

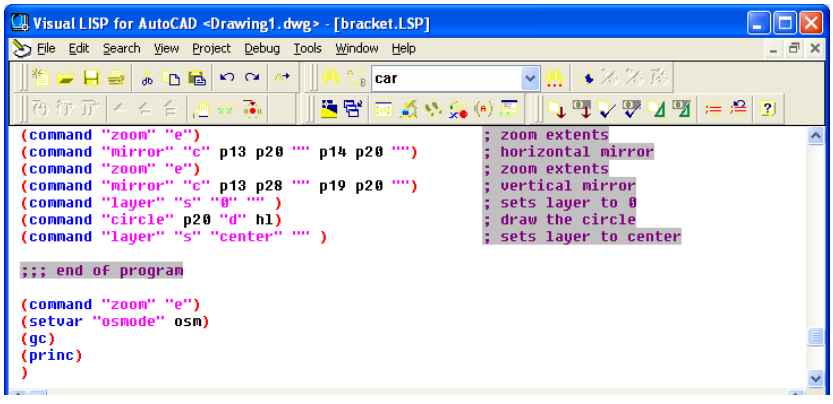

Figure 7A.21 – End of Program

To end the program, we will need to place a parenthesis at the end of the code to close the **defun c:bracket** function. Type the following code.

```
(princ)
)
```

The **princ** function used in this routine will allow the program to end without printing the last line of the program to the command line. Without this function the command line can show a number or text that may not make sense to the use. This function is used to keep your code neat.

Practice typing the following examples of the **princ** function at the command line of AutoCAD.

Function	Name	Description
princ	**Princ Function**	**Will allow the program to run without printing the last line of the code to the command line**
Example		
Typing an expression at the command line without the princ function	**(setq a "Hello")**	Answer: **"Hello"**
Typing an expression at the command line without the princ function	**(setq a "Hello")(princ)**	Answer: **nothing**

Saving and Running the Program

Now that the program is finished, we need save our program to our folder.

Then to run the program, we will make sure the Look in list box is displaying the Visual LISP Programs folder and then select the program "bracket" 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 that the program is loaded.

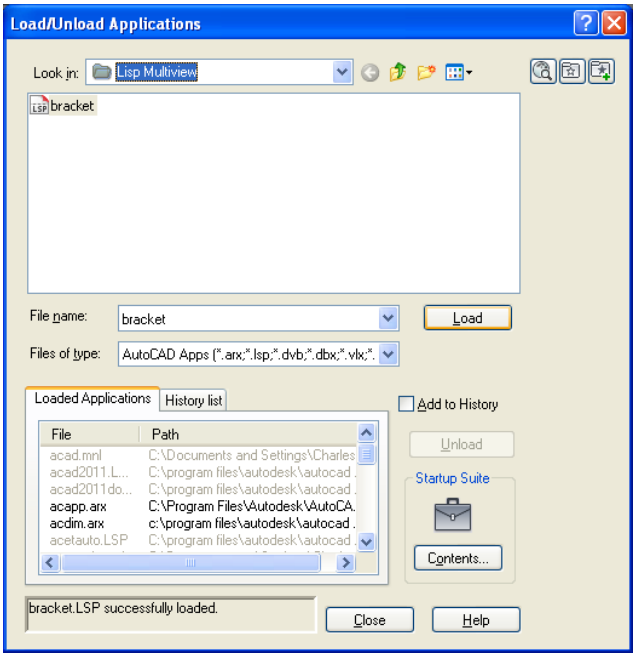

Figure 7A.22 – Loading the Bracket Program

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 copyright and information to start the program is shown.

Figure 7A.23 – The Alert Message

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

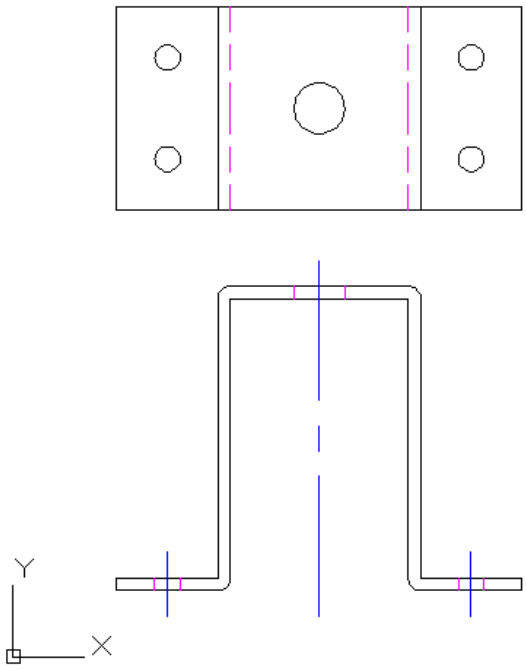

Figure 7A.24 – Starting the Program

Programs creating and placing text on a drawing are very easy to write once we have achieved writing the first program with these new functions. There are addition exercises for text based routines in the appendixes of this manual. Written below is the entire bracket.LSP code for your benefit.

```
;;;   bracket.lsp
;;;
;;;   a lisp routine that draws the top and front view of a bracket
;;;
;;;   Copyright (C) 2010  by Charles Robbins
;;;
;;;   Charles Robbins provide this code for your use. Use the code to your benefit
;;;   and at your own risk. Charles Robbins does not warrant that the code is error
;;;   free in your application.
```

```
(alert "bracket.lsp - copyright 2010 by Charles Robbins. Type bracket to start.")

;;; start program

(defun c:bracket (/)

;;; setup

(setq osm (getvar "osmode"))
(setvar "osmode" 0)

;;; ask questions

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

(setq lg       (getreal "\nWhat is the bracket's length?      "))
(setq wd       (getreal "\nWhat is the bracket's width?       "))
(setq hgt      (getreal "\nWhat is the bracket's height?      "))
(setq thk      (getreal "\nWhat is the bracket's thickness? "))
(setq pf       (getreal "\nWhat is the platform length?      "))
(setq hl       (getreal "\nWhat is the mounting hole size?   "))
(setq mh       (getreal "\nWhat is the through hole size?     "))

;;; setup layers

(command "layer" "n" "dimension" "c" "red"     "dimension"                "")
(command "layer" "n" "text"          "c" "green"  "text"                        "")
(command "layer" "n" "center"  "c" "blue"  "center"  "lt" "center"  "center"  "")
(command "layer" "n" "hidden" "c" "magenta" "hidden"  "lt" "hidden"  "hidden"  "")

;;; math

(setq x1  (car sp)
      x7  (+ x1 (/ lg 2.0))
      x6  (- x7 (/ hl 2.0))
      x4  (- x7 (/ pf 2.0))
      x5  (+ x4 thk)
      x3  (+ x1 (/ (- lg pf) 4.0))
      x2  (- x3 (/ mh 2.0))
)

(setq y2   (cadr sp)
      y1  (- y2 (* 2 thk))
      y3  (+ y2 thk)
      y4  (+ y3 (* 2 thk))
      y6  (+ y2 hgt)
      y5  (- y6 thk)
      y7  (+ y6 (* 2 thk))
      y8  (+ y7 (* 4 thk))
      y9  (+ y8 (* 2 mh))
      y10 (+ y8 (/ wd 2.0))
      y11 (+ y8 wd)
)
```

```
; point assignments

(setq p0   (list x1   y1  )
      p1   (list x1   y2  )
      p2   (list x4   y2  )
      p3   (list x5   y3  )
      p4   (list x5   y5  )
      p5   (list x7   y5  )
      p6   (list x7   y6  )
      p7   (list x5   y6  )
      p8   (list x4   y5  )
      p9   (list x4   y3  )
      p10 (list x1   y3  )
      p11 (list x6   y5  )
      p12 (list x6   y6  )
      p13 (list x1   y8  )
      p14 (list x1   y10 )
      p15 (list x4   y8  )
      p16 (list x4   y10 )
      p17 (list x5   y8  )
      p18 (list x5   y10 )
      p19 (list x7   y8  )
      p20 (list x7   y10 )
      p21 (list x3   y9  )
      p22 (list x3   y1  )
      p23 (list x3   y4  )
      p24 (list x7   y1  )
      p25 (list x7   y7  )
      p26 (list x2   y2  )
      p27 (list x2   y3  )
      p28 (list x7   y11 )
)

; lets draw the front view

 (command "layer" "s" "0" "")                ; sets layer to 0
(command "line" p2 p1 p10 p9 p8 "")          ; draw the line
 (command "line" p3 p4 p5 "")                ; draw the line
(command "line" p7 p6 "")                    ; draw the line
(command "arc" p2 "e" p3 "r"  thk)           ; draw the arc
(command "arc" p7 "e" p8 "r"  thk)           ; draw the arc
(command "layer" "s" "hidden" "" )           ; sets layer to hidden
(command "line" p26 p27 "")                  ; draw the line
(command "zoom" "e")                         ; zoom extents
(command "mirror" "L" "" p22 p23 "")         ; mirror hidden line
(command "line" p11 p12 "")                  ; draw the line
(command "layer" "s" "center" "" )           ; sets layer to center
(command "line" p22 p23 "")                  ; draw the line
(command "zoom" "e")                         ; zoom extents
(command "mirror" "c" p0 p25 "" p24 p25 "")      ; mirror the front view
(command "line" p24 p25 "")                      ; draw the line

; lets draw the top view

(command "layer" "s" "0" "")                 ; sets layer to 0
(command "line" p19 p13 p14 "")              ; draw the line
```

```
(command "line" p15 p16 "")                          ; draw the line
(command "circle" p21 "d" mh)                        ; draw the circle
 (command "layer" "s" "hidden" "" )                  ; sets layer to hidden
(command "line" p17 p18 "")                          ; draw the line
(command "zoom" "e")                                 ; zoom extents
(command "mirror" "c" p13 p20 "" p14 p20 "")         ; horizontal mirror
(command "zoom" "e")                                 ; zoom extents
(command "mirror" "c" p13 p28 "" p19 p20 "")         ; vertical mirror
(command "layer" "s" "0" "")                         ; sets layer to 0
(command "circle" p20 "d" hl)                        ; draw the circle
(command "layer" "s" "center" "")                    ; sets layer to 0

;;; end of program

 (command "zoom" "e")
(setvar "osmode" osm)
(gc)
(princ)
)
```