### Learning Objectives

*After completing this chapter, you will be able to:*

- Identify ways to provide for user input.
- Retrieve and use system variable values in AutoLISP programs.
- Work with lists using AutoLISP.
- Use angular input in AutoLISP programs.

As you practice using AutoLISP, you will develop ideas for programs that require additional commands and functions. Some of these programs may require that the user pick two corners of a windowed selection set. Another program may use existing points to draw a shape. You may also need to locate a point using polar coordinate notation or determine the angle of a line. All of these drawing tasks can be done with AutoLISP programs.

#### Providing for Additional User Input

The `(getreal)` function allows you to define a variable value by entering a real number at the keyboard. Remember, as defined by AutoLISP, real numbers are classified separately from integers. A real number is considered to be more precise than an integer because it has a decimal value.

The `(getreal)` function is similar to the `(getdist)` function covered in the previous chapter, except that it works strictly with real numbers. You cannot respond with a value of feet and inches, nor can you respond by picking points. Once issued, the function waits for user input. As with most of the `(setq)` functions, a prompt can be included. The real number is returned after a response is entered. When nested within the `(setq)` function, the `(getreal)` function can be used to set the value of a variable as follows.

Command: `(setq X (getreal "Enter number: "))`

Enter number: 34

34.0
The `getcorner` function allows the user to pick the opposite corner of a rectangle and define it as a point value. It is similar to the `getpoint` function covered in the previous chapter in that it also returns a point list. This is similar to placing a window around objects to define a selection set in a drawing. An existing point serves as the first corner of the rectangle. When locating the opposite corner, the screen cursor appears as a “rubber band” box similar to the window used when defining a selection set.

The `getcorner` function can also be used to set the value of a variable when nested within the `setq` function. The second corner can be picked with the pointing device or entered at the keyboard. The following is an example of using the `getcorner` function.

```
Command: (setq PT1 (getpoint "Pick a point:"))
```

```
Pick a point:
```

```
Command: (setq PT2 (getcorner PT1 "Pick the second corner:"))
```

```
Pick the second corner: 
```

Notice that the value of `PT1` is set first. The point represented by `PT1` becomes the base point for locating `PT2`. The two points (corners) located in this example can be used to construct an angled line, rectangle, or other shape. The points can also be applied to other functions.

---

### Using the Values of System Variables

AutoCAD’s system variables can be read and changed from within AutoLISP applications with the `getvar` and `setvar` functions. These functions can be useful if you are writing a program that requires you to change the value of a system variable. It is typically a good practice to get the current setting of the variable and store it in an AutoLISP variable, change the system variable setting for your program, and then reset the system variable to its original value.

The `getvar` function is used to return the value of a system variable. In the following example, two system variable settings are saved as AutoLISP variable values.

```
Command: (setq V1 (getvar "TEXTSIZE"))
```

```
current value of the TEXTSIZE system variable
```

```
Command: (setq V2 (getvar "FILLETRAD"))
```

```
current value of the FILLETRAD system variable
```

The `setvar` function is used to change an AutoCAD system variable setting. You can assign a new value to a variable as follows.

```
Command: (setvar "TEXTSIZE" 0.25)
```

```
0.25
```

```
Command: (setvar "FILLETRAD" 0.25)
```

```
0.25
```

Remember, you need to add the 0 in front of .25 or an error is generated.

Make it a point to remember that whenever you are going to change a system variable setting, you should save the current system variable setting, change the variable, and then reset the variable to its original value after the command is executed. The `getvar` function, nested within a `setq` function, can be used to assign the original value to a variable, such as `V1` shown in the first example on the `TEXTSIZE` system variable above. When the program is complete, the `setvar` function can be used to reset `TEXTSIZE` to its original value.

```
Command: (setvar "TEXTSIZE" V1)
```

```
0.125
```

This returns the value of `TEXTSIZE` to the value of the variable `V1`, which is the original system variable setting.
Exercise 28-1

1. For this exercise, enable the autosnap aperture box for use with the object snap modes. Use the proper AutoLISP syntax to write expressions for the functions given below. Write out the expressions and then test them by entering them into the computer.

A. Display the current APERTURE system variable setting and save it to the new variable APER.
B. Create the new variable APER4 and use it to set the aperture size to four pixels.
C. Use the LINE command to draw two lines on screen. Next, use the endpoint object snap mode to connect the endpoints of the two lines.
D. Reset the APERTURE system variable to the original setting.

2. Add the following capabilities to the EX27_07.lsp file that you created in Exercise 27-7.

A. Create a variable named V1 to store the current text size. Insert a line for this function after the (defun) function line.
B. At the end of the program, insert a function that resets the text size to its original value.

Working with Lists

In AutoLISP, a list is defined as a stored set of values that are enclosed in parentheses. Lists are commonly used to provide point locations and other data for use in functions. A list is created, for example, when you pick a point on screen in response to the (getpoint) function. The list is composed of three numbers—the X, Y, and Z coordinate values. You can tell it is a list because AutoLISP returns the numbers enclosed in parentheses. On the other hand, a number entered in response to the (getreal) function is returned as a real number (it is not enclosed in parentheses). A single number is not a list. The following expression returns a list.

Command: (setq P1 (getpoint "Enter a point:"))

Enter a point: (pick a point)
(2.0 2.75 0.0)

The individual values in a list are called atoms and can be used in an AutoLISP program to create new points. The (car) function retrieves the first atom in a list. The variable P1 in the example above is composed of the list (2.0 2.75 0.0). Thus, using the (car) function with the P1 variable returns a value of 2.0.

Command: (car P1)
2.0

The second atom in a list is retrieved with the (cadr) function. Find the second atom of the list stored in the variable P1 by entering the following.

Command: (cadr P1)
2.75

The (list) function allows you to create a new list of two coordinates by extracting values from existing points using the (car) and (cadr) functions and placing these values as arguments to the (list) function. Values returned by the (list) function are placed inside of parentheses. The X coordinate of the first variable, P1, can be combined with the Y coordinate of a second point variable, named P2, to form a third point variable, named P3. Study the following example and Figure 28-1. The coordinates stored in the variable P1 are (2.0 2.75).
Command: 
\texttt{(setq P2 (getcorner P1 "Enter second point: "))})
\textbackslash n
Enter second point: 6,4.5
\textbackslash n
(6.0 4.5 0.0)

Command: 
\texttt{(setq P3 (list (car P1)(cadr P2)))})
\textbackslash n
(2.0 4.5)

As discussed in the previous chapter, a function is almost always followed by an argument. An \textit{argument} consists of data on or with which a function operates. An expression must be composed of only one function and any required arguments. Therefore, the functions \texttt{(car)} and \texttt{(cadr)} must be separated because they are two different expressions combined to make a list. The \texttt{(car)} value of the list stored in \texttt{P1} is to be the X value of \texttt{P3}, so it is given first. The \texttt{(cadr)} value of the list stored in \texttt{P2} is placed second because it is to be the Y value of \texttt{P3}. Notice the number of closing parentheses at the end of the expression.

Now, with three points defined, there are many things you can do. For example, you can draw lines through the points to form a right triangle, Figure 28-2. To do so, use the \texttt{(command)} function as follows.

Command: 
\texttt{(command "LINE" P1 P2 P3 "C")}
\textbackslash n

The \texttt{(car)} and \texttt{(cadr)} functions allow you to work with 2D coordinates. The \texttt{(caddr)} function allows you to use the third atom of a list. This can be the Z coordinate of a point. Enter the following at your keyboard.

Command: 
\texttt{(setq B (list 3 4 6))}
\textbackslash n
(3 4 6)

You have created a list of three atoms, or values, and assigned it to the variable \texttt{B}. The third value is retrieved with the \texttt{(caddr)} function.

Command: 
\texttt{(caddr B)}
\textbackslash n
6

Since 6 is a single value, not a list, it is not enclosed in parentheses. Now, use the \texttt{(car)} and \texttt{(cadr)} functions to find the other two atoms of the list.

Command: 
\texttt{(car B)}
\textbackslash n
3

Command: 
\texttt{(cadr B)}
\textbackslash n
4

\textbf{Figure 28-1.}
A third point identified as \texttt{P3} has been created using the \texttt{(car)} value of the list \texttt{P1} and the \texttt{(cadr)} value of the list \texttt{P2}.

\texttt{P3}
\begin{align*}
X &= 2.0 \texttt{(car P1)} \\
Y &= 4.5 \texttt{(cadr P2)}
\end{align*}

\texttt{P2}
\begin{align*}
X &= 6.0 \texttt{(car P2)} \\
Y &= 4.5 \texttt{(cadr P2)}
\end{align*}

\texttt{P1}
\begin{align*}
X &= 2.0 \texttt{(car P1)} \\
Y &= 2.75 \texttt{(cadr P1)}
\end{align*}

\textbf{Figure 28-2.}
Once the third point, \texttt{P3}, is created, a triangle can be drawn.
The following is a short AutoLISP program that uses the (car) and (cadr) retrieval functions to place an X at the midpoint of two selected points.

```
(defun C:MDPNT ()
  (setq PT1 (getpoint "Enter the first point: "))
  (setq PT2 (getpoint "Enter the second point: "))
  (setq PT3 (list (/ (+ (car PT1) (car PT2)) 2) (/ (+ (cadr PT1) (cadr PT2)) 2)))
  (setvar "PDMODE" 3)
  (command "POINT" PT3)
)
```

The (cdr) function is also used to work with lists. It allows you to retrieve the second and remaining values of a list. Earlier in this discussion, the list (3 4 6) was assigned to variable B. In the following example, the (cdr) function is used to return the list (4 6).

```
Command: (cdr B).
(4 6)
```

This returns a list of two values, or coordinates, that can be further manipulated with the (car) and (cadr) functions. Study Figure 28-3 and the following examples.

```
Command: (car (cdr B)).
4
Command: (cadr (cdr B)).
6
```

The first example is asking for the first atom—(car)—of the list generated from the last two atoms—(cdr)—of variable B. In the second example, the second atom—(cadr)—of the list generated from the last two atoms of variable B is returned.

The four functions used to manipulate lists—(car), (cadr), (caddr), and (cdr)—may seem confusing at first. Practice using them to see how they work. Practice with a list of numbers, coordinate values, or text strings. Remember, text strings must be enclosed in quotation marks. Try the following examples to see what happens. Enter the expressions on the command line exactly as shown and press [Enter] at the end of each line.

```
(setq NOTES (list "DO" "RE" "MI"))
(car NOTES)
(cadr NOTES)
(caddr NOTES)
(cdr NOTES)
(setq LASTNOTES (cdr NOTES))
(car (cdr NOTES))
(cadr (cdr NOTES))
(car LASTNOTES)
(cadr LASTNOTES)
```

---

**Figure 28-3.**
The (cdr) function creates a list containing the second and remaining atoms of a list. The new list can be manipulated as necessary with the (car) and (cadr) functions.
As you continue to work in AutoLISP, you will find many uses for the functions that allow you to work with lists. Remember the following review.

- **(car)**. Returns the first atom in a list.
- **(cadr)**. Returns the second atom in a list.
- **(caddr)**. Returns the third atom in a list.
- **(cdr)**. Returns a list comprised of the second and remaining atoms of a list. The returned values are placed in a list. If the original list contains two atoms, only the second atom is returned and it is placed in a list.
- **(list)**. Creates a list of all values entered as arguments to the function name.

---

**Exercise 28-2**

1. Write an AutoLISP program that draws a right triangle using points provided by the user. The 90° angle can be on the left or right side.
2. Use the following guidelines to write the program.
   A. Define a function named **TRIANGLE**.
   B. Set the variable **P1** as the first point of the triangle. Provide a user prompt.
   C. Set the variable **P2** as the endpoint of the hypotenuse using the **(getcorner)** function. Provide a user prompt and be sure it is on the next line.
   D. Set the variable **P3** to the X coordinate of **P2** and the Y coordinate of **P1**.
   E. Write an expression that draws a triangle through all three points.
3. Write the program in proper AutoLISP syntax. Use the Visual LISP Editor and save the file as **EX28_02.lsp**.
4. Before saving the file, check for matching parentheses and quotation marks. Save the program and then test it.

---

**Using Polar Coordinates and Angles**

The ability to work with angles is vital if you plan to do much AutoLISP programming. Four functions—**(angle)**, **(polar)**, **(getangle)**, and **(getorient)**—allow you to use angles when writing program files. AutoLISP works with these functions using the radian system of angle measurement. This system of measurement is explained in the next section.

**Measuring Angles in Radians**

The **(angle)** function is used to calculate the angle in the XY plane of a straight line between two given points. The value of the angle is given in radians. Radian angle measurement is a system in which 180° equals “pi” (π). Pi is approximately equal to 3.14159.
AutoLISP functions use radians for angular measurement, but AutoCAD commands typically use degrees. Therefore, to use a radian angle in an AutoCAD command, it must first be converted to degrees. Conversely, a degree angle to be used by AutoLISP must be converted to radians. The following AutoLISP formulas are used for those conversions.

- To convert degrees to radians, use the formula:
  \[ (* \pi \left(\frac{ad}{180.0}\right)) \]
  where \( ad \) = angle in degrees.
- To convert radians to degrees, use the formula:
  \[ \left(\frac{ar}{\pi}\right) * 180.0 \]
  where \( ar \) = angle in radians.

The following table gives common angles measured in degrees, the AutoLISP expressions used to convert the angular values to radian values, and the resulting values in radians rounded to four decimal places.

<table>
<thead>
<tr>
<th>Angle (degrees)</th>
<th>AutoLISP Expression</th>
<th>Angle (radians)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>(/ pi 6)</td>
<td>0.5236</td>
</tr>
<tr>
<td>45</td>
<td>(/ pi 4)</td>
<td>0.7854</td>
</tr>
<tr>
<td>60</td>
<td>(/ pi 3)</td>
<td>1.0472</td>
</tr>
<tr>
<td>90</td>
<td>(/ pi 2)</td>
<td>1.5708</td>
</tr>
<tr>
<td>135</td>
<td>(/ (* pi 3) 4)</td>
<td>2.3562</td>
</tr>
<tr>
<td>180</td>
<td>pi</td>
<td>3.1416</td>
</tr>
<tr>
<td>270</td>
<td>(/ (* pi 3) 2)</td>
<td>4.7124</td>
</tr>
<tr>
<td>360</td>
<td>(* pi 2)</td>
<td>6.2832</td>
</tr>
</tbody>
</table>

The following example illustrates how the angle between two points can be set to a variable, and then converted to degrees.

Command: (setq P1 (getpoint "Enter first point:"))
Enter first point: 1.75,5.25
(1.75 5.25 0.0)
Command: (setq P2 (getpoint "Enter second point:"))
Enter second point: 6.75,7.25
(6.75 7.25 0.0)
Command: (setq A1-R (angle P1 P2))
0.380506

The angle represented by the variable \( A1-R \) is measured in radians (0.380506). To convert this value to degrees, use the following expression.

Command: (/ (* A1-R 180.0) pi)
21.8014
Command: !A1-R
0.380506
The value 21.8014 is the angle in degrees between the coordinates in variables \( P1 \) and \( P2 \). However, notice that this conversion does not set the variable \( A1-R \) to the value in degrees. Make the degree value permanent by assigning it to a variable using the following expression.

\[
\text{Command: } (\text{setq } A1-D (/ (* A1-R 180.0) \pi))
\]

\[
21.8014
\]
\[
\text{Command: } !A1-D
\]

The variable \( A1-D \) now has a value of 21.8014.

**Exercise 28-3**

1. Using AutoLISP expressions, locate the endpoints of a line and store each endpoint as a variable.
2. Use the \( \text{angle} \) function to find the angle of the line. Set the value to the variable \( A \).
3. Use the proper formula to convert the radian value to degrees.
4. Use the proper expression to assign the degree value to the variable \( A \).

**Providing for Angular Input by the User**

The \( \text{getangle} \) function allows the user to input an angular value for use in an application. This function is often nested in a \( \text{setq} \) function and used to set a variable that can be used by another function. The \( \text{getangle} \) function is similar to the \( \text{getdist} \) function covered in the previous chapter in that the user can input two points to describe the angle or the angle can be input as degrees from the keyboard. If points are being picked to describe the angle, the \( \text{getangle} \) function automatically issues a Specify second point: prompt. The following example illustrates how you can set a variable to an angular value that is input by the user by picking two points:

\[
\text{Command: } (\text{setq } A1 (\text{getangle } \text{"Pick first point: "}))
\]

Pick first point: (pick the first point)
Specify second point: (pick the second point)
angle (in radians)

The following example illustrates how you can set a variable to an angular value by entering an angle from the keyboard:

\[
\text{Command: } (\text{setq } A2 (\text{getangle } \text{"Enter an angle: "}))
\]

Enter an angle: 120
2.0944

The angular value is always returned in radians. To convert it to degrees, use the formula presented in the previous section.

The \( \text{getangle} \) function uses the current \( \text{ANGBASE} \) (angle 0 direction) and \( \text{ANGDIR} \) (clockwise or counterclockwise) system variables. Therefore, if you have angles set to be measured from north (where \( \text{ANGBASE} = 90^\circ \)), angles picked with the \( \text{getangle} \) function will be measured from north. If the \( \text{ANGDIR} \) system variable is set to measure angles clockwise, the \( \text{getangle} \) function will accept input of clockwise values, but returns counterclockwise values.

A companion function to \( \text{getangle} \) is \( \text{getorient} \). It is used in exactly the same manner as the \( \text{getangle} \) function. However, \( \text{getorient} \) always measures angles counterclockwise from east \( (0^\circ) \), regardless of the current \( \text{ANGBASE} \) and \( \text{ANGDIR} \) system variable settings.
Exercise 28-4

1. Use the (getangle) function to assign an angular value to the variable ANG1R.
   A. Convert the radian value to degrees using the proper AutoLISP expression. Assign the degree value to ANG1D.
   B. Convert the degree value back to radians using the proper AutoLISP expression. Assign the radian value to the variable ANG2R.
   C. Compare the value stored in ANG1R to that of ANG2R to be sure they match.

2. Set the ANGBASE system variable to 0. Be sure to turn off dynamic input.
   A. Use the (getangle) function and the point 1,1 for the first point and point 1,5 as the second point. Note the returned value is in radians, 1.5708.
   B. Do the same as above, but this time use the (getorient) function. Note that the returned result is the same.
   C. Set the ANGBASE system variable to 90.
   D. Use the (getangle) function and the same point values as in Step A again. Note that the returned result is 0.0 since the direction for angle 0 is now 90 degrees and (getangle) always works in respect to the ANGBASE system variable.
   E. Use the (getorient) function and the same point values as in Step A again. Note that the returned result is 1.5708 since the (getorient) function always works in respect to 0 degrees being toward 3 o’clock, regardless of the ANGBASE setting.
   F. Note that both (getangle) and (getorient) always return values in radians.

Using Polar Coordinates

The (polar) function allows you to specify a new point based on the angle and distance from an existing point. Three arguments are required for the (polar) function. The first argument must contain the coordinates of the base point from which you are locating the new point. The second argument is the angular direction (in radians) to go from the base point argument. The third argument is the distance value from the base point argument to the new point. The syntax for the (polar) function is:

(polar base_point angle distance)

For example, suppose you want to specify a point as P1 and locate another point, P2, at a specific distance (three units) and angle (60°) from P1. Enter the following expressions.

Command: (setq P1 (getpoint "Enter point: "))
Enter point: 4.0,4.5
(4.0 4.5 0.0)
Command: (setq D (getdist P1 "Enter distance: "))
Enter distance: 3.0
3.0
Command: (setq A (/ pi 3))
1.0472
In this example, the desired angle is 60°. However, AutoLISP uses radians for angular values. Therefore, the degree value is being expressed as a relationship to pi. The resulting value of the (/ pi 3) expression, 1.0472, is saved as the variable A. Next, use the (polar) function to locate the second point relative to P1 at the specified angle and distance. A line can then be drawn from P1 to P2 using the (command) function. The sequence is:

Command: (setq P2 (polar P1 A D)).
(5.5 7.09808 0.0)
Command: (command "LINE" P1 P2 "").

Exercise 28-5

1. Write an AutoLISP program that draws a right triangle. Refer to Exercise 28-2. However, use the (polar) function instead of the list functions. Write the program in proper AutoLISP format using the Visual LISP Editor.
2. Use the following guidelines for writing the program.
   A. Define a function named POLARTRI.
   B. Set the variable P1 as the first corner of the triangle. Provide a user prompt.
   C. Set the variable D as the length of one side.
   D. Set the variable P2 as the second corner located 0° from P1 at a distance of D.
   E. Set the variable P3 as the third corner located 90° from P2 at a distance of D.
   F. Write an expression that draws a line through all three points and closes the triangle.
3. Save the file as EX28_05.lsp and load the program into AutoCAD. Test the program.

Locating AutoCAD’s AutoLISP Files

One of the best ways to become familiar with AutoLISP is to enter expressions and programs into your computer. Look for programs in the books, magazines, newsgroups, and blogs that you read. Get a feel for how the functions and arguments go together and how they work in AutoCAD. Make a habit of reading through one of the AutoCAD journals and experiment with the AutoLISP routines printed in them. Also, refer to the online documentation and developer help for other samples.

AutoLISP files are typically saved with the .lsp extension. A variety of AutoLISP programs are supplied with AutoCAD. These are saved in the AutoCAD folder structure. You can use Windows Explorer to search the AutoCAD folder structure and list the .lsp files.

The AutoLISP files found in the \Support folder are standard files that support many of AutoCAD’s built-in features. When the command that starts the function is entered, the associated program file is automatically loaded.

PROFESSIONAL TIP

For easier access to any AutoLISP program file, add its folder in the Support File Search Path listing located in the Files tab of the Options dialog box.
Sample AutoLISP Programs

The following programs are provided for you to copy and add to your acaddoc.lsp file or to your custom menus. Practice using the routines for a few minutes a couple of times a week. This will help you begin to better understand and use AutoLISP. Train yourself to learn a new function every week. Before long, you will be writing your own useful programs.

**Erasing the Entire Screen**

This program sets two variables to the minimum and maximum extents of the drawn objects. It then erases everything within those extents and redraws the screen. Name this program zap.lsp.

```
;;; ERASES ENTIRE EXTENTS.
(defun C:ZAP ()
  (setq EMIN (getvar "EXTMIN"))
  (setq EMAX (getvar "EXTMAX"))
  (command "ERASE" "C" EMIN EMAX ""
  (command "REDRAW")
  (princ)
)
```

**Setting the Current Layer**

Similar to the built-in Make Object’s Layer Current tool on the Layers panel in the Home tab of the ribbon, this program asks the user to pick an object on the layer to be set current. The program finds the layer of the object picked and sets it current. Name this program lp.lsp.

```
;;; AUTHOR ROD RAWLS
(defun C:LP (/ E)
  (while (not (setq E (entsel "Select object on target layer...")))
    (alert "No object selected!")
  )
  (setq LN (cdr (assoc 8 (entget (car E))))
  (command 
  (princ)
)
```

**Cleaning Overlapping Corners**

This program allows you to trim the overlapping ends of intersecting lines. You are requested to pick the two lines that intersect and overlap. The points you pick are on the portion to keep. The program does the rest. Name the program cleanc.lsp. Note: The value returned at the end of the program is the original OSMODE setting.

```
;;; AUTHOR: GEORGE HEAD
;;; PRINTED IN CADENCE MAGAZINE
(defun C:CLEANC (/ O1 P1 P2)
  (setq O1 (getvar "OSMODE"))
  (setvar "OSMODE" 512)
  (command "FILLET" "R" 0)
  (setq P1 (getpoint "\nPick a line:"))
  (setq P2 (getpoint "\nPick other line:"))
  (command "FILLET" P1 P2)
  (setvar "OSMODE" O1)
  (princ)
)
```
Calculating the Length of Lines

This program calculates the length of all lines on a specified layer. It can be used for estimating and material takeoffs. This program only works with lines, not with polylines. Name the program linear.lsp. After loading it into AutoCAD, respond to the first prompt by entering the name of the layer that contains the lines you wish to total. The calculation is given in current drawing units. Also, the layer name is case sensitive.

;;; AUTHOR: JOE PUCILOWSKI
;;; COMPANY: JOSEPH & ASSOCIATES
;;; REVISED BY CRAIG BLACK
;;; NOTE: THIS PROGRAM FIGURES THE TOTAL NUMBER OF LINEAR
;;; UNITS (FEET, INCHES, ETC.) OF LINES ON A SPECIFIC LAYER.
;;; 
;;; (defun C:LINEAR ()
; (setq TOTAL 0
; E (entnext)
; NUMLIN 0
; LAYPIK (getstring T "\nAdd up lines on layer: ")
;)
;(if (tblsearch "LAYER" LAYPIK)
; (progn
; (while E
; (setq ENTTYP (cdr (assoc 0 (setq EG (entget E))))
; LAYNAM (cdr (assoc 8 EG))
;)
;)
;)
; )
;)
;)
;)
;(setq E (entnext E))
;)
;)
;(princ (strcat "\nFound "
; (itoa NUMLIN)
; " lines on layer <>" LAYPIK
; " > with a total of ")
; (rtos TOTAL)
; " linear units." 
;)
;)
;)
;(princ "\nLayer does not exist.")
;)
;)}

Changing the Grid Rotation

The first routine, titled S, rotates the grid to the angle between the X axis and any picked line. The second routine, SS, returns the grid to zero rotation. These functions are achieved by rotating the snap. Save the file as rotgrid.lsp.

;;; AUTHOR: EBEN KUNZ
;;; COMPANY: KUNA ASSOCIATES ARCHITECTS
;;; REVISED BY CRAIG BLACK
;;
(defun C:S (/ PT1 PT2)
  (setq |OSMODE| (getvar "OSMODE"))
  (setq "ORTHOMODE" 0)
  (setq PT1 (osnap (getpoint "Pick line to match new Grid angle: ") "NEA"))
  (setq PT2 (osnap PT1 "END"))
  (command "SNAP" "R" PT1 PT2)
  (setq "SNAPMODE" 0)
  (setq "OSMODE" |OSMODE|)
  (princ)
)
(defun C:SS ()
  (prompt "Return Grid to zero.")
  (command "SNAP" "R" "0.0")
  (setvar "SNAPMODE" 0)
)

Moving Objects to a Selected Layer

This routine, named oblay.lsp, allows you to move objects to a layer by picking an object on the destination layer. After you select an object on the destination layer, you can select multiple objects using any AutoCAD selection method. When you press [Enter] to end the command, the objects are moved to the destination layer.

;;; AUTHOR: SHELDON MCCARTHY
;;; COMPANY: EPCM SERVICES LTD.
;;
(defun C:OBLAY ()
  (setq 1A
    (cdr (assoc 8 (entget (car (entsel "Entity on destination layer: ")))))
  (prompt "Objects to change...")
  (ssget)
  (command "CHANGE" "P" "P" "LA" 1A)
  (princ)
)
Moving Objects to the Current Layer

This simple program, titled cl.lsp, quickly changes selected objects to the current layer. You can select multiple objects using any AutoCAD selection method. When you press [Enter] to end the command, the objects are moved to the current layer.

;;; AUTHOR: BILL FANE
;;; COMPANY: WISER, INC.
;;;
(defun C:CL (/ THINGS)
  (prompt "nSelect objects to move to current layer...")
  (setq THINGS (ssget))
  (command "CHANGE"
    THINGS
    "" "P"
    "LA"
    (getvar "CLAYER")
    ""
  )
  (princ)
)

Chapter Review

Chapter Review

Answer the following questions. Write your answers on a separate sheet of paper or complete the electronic chapter review on the companion website.

www.g-wlearning.com/CAD

1. Name the function that allows you to return a real number and use it as a variable value.
2. Which two functions allow you to work with system variables?
3. Define the following AutoLISP functions.
   A. (car)
   B. (cadr)
   C. (cdr)
   D. (caddr)
   E. (list)
4. Write the proper AutoLISP notation to return the last two atoms of the list (4 7 3) as a list.
5. Write an expression to set a variable named A to the result of Question 4.
6. Write an expression to return the second atom of the list created in Question 5.
7. Compare and contrast the (getangle) and (getorient) functions.
8. Write an expression to set the angle between points P3 and P4 to the variable A.
9. Which system of angular measurement does AutoLISP use?
10. Explain the purpose of the (polar) function.
Drawing Problems

Write AutoLISP programs for the following problems. Use the **Visual LISP Editor**. Save the files as P28_(problem number) with the .lsp extension.

1. Add the following capabilities to the right triangle function developed in Exercise 28-2.
   A. Use the `(getdist)` function instead of `(getcorner)).
   B. Allow the angle of the hypotenuse to be picked.
   C. Allow the length of a side or the hypotenuse length to be picked.

2. Create an AutoLISP program similar to that in Problem 1, but write it so that it draws an equilateral triangle (with equal angles and equal sides). Use the `(polar)` function.

3. Revise the program in Problem 1 in Chapter 27 to draw a rectangle using the `(getcorner)` function to find the second corner. Also, revise the program so that the `LINE` command is used instead of the `RECTANG` command.

4. Add a **Fillet 0** command to your **Modify** menu. Use menu macros and AutoLISP expressions to create the command. Follow these guidelines:
   A. Retrieve the current fillet radius setting and assign it to an AutoLISP variable.
   B. Set the fillet radius to 0.
   C. Allow the user to select two lines and automatically enter a 0 radius fillet.
   D. Reset the fillet radius to the original value.
   E. Assign an appropriate mnemonic key to the new menu command.

5. Write an AutoLISP program that allows the user to measure the distance between two points using the `DIST` command. Use AutoLISP expressions to do the following:
   A. Assign the current unit precision for read-only linear units to an AutoLISP variable.
   B. Prompt for the desired unit precision from the user and store the value as a variable.
   C. Set the unit precision with the user-defined variable value.
   D. Allow the user to measure the distance between two selected points with the `DIST` command.
   E. Reset the unit precision to the original value.

6. Write an AutoLISP program to draw a rectangle and place a circle having a user-specified diameter in the center of the rectangle.
   A. Incorporate the rectangle program from Problem 3.
   B. Use the `(angle)`, `(polar)`, and `(distance)` functions to find the center point of the rectangle.
   C. Prompt the user to enter the diameter of the circle.
   D. Use the `CIRCLE` command to draw the circle at the center point of the rectangle.
7. Write a program to draw a leader with a diameter dimension having plus and minus tolerances.
   A. Issue prompts that allow the user to set the **DIMTP** and **DIMTM** system variables and save the specified values to AutoLISP variables.
   B. Set the new values to the **DIMTP** and **DIMTM** system variables.
   C. Turn the **DIMTOL** system variable on.
   D. Activate the **DIMDIAMETER** command. Use the (**entsel**) function to set the selection specification to a variable as follows.
      
      (setq SC (**entsel**))
      
   E. Using the (**getpoint**) function, issue a prompt that allows the user to pick a location for the leader line and the default dimension text.
   F. Turn the **DIMTOL** system variable off.

8. Write an AutoLISP program to draw a leader with a bubble attached to the end.
   A. Prompt the user for the start point of the leader and set it to the variable **P1**.
   B. Prompt the user for the endpoint of the leader and set it to the variable **P2**.
   C. Prompt the user for the text height and set it to a variable.
   D. Issue a prompt that asks for the text string (specify a maximum of two characters) and set the resulting text to a variable.
   E. Calculate the circle diameter at three times the text height and set it to a variable.
   F. Set the center point of the circle to a point relative to **P2** using the (**polar**) function. Set the relative distance as the radius of the circle. Assign the center point to the variable **P3**.
   G. Use the **LEADER** command to draw a leader from **P1** to **P2**.
   H. Draw the leader line with no shoulder and no annotation text.
   I. Draw a circle with the center point at **P3**.
   J. Draw text in the center of the circle using the appropriate justification option of the **TEXT** command.

9. Develop a program that draws a line of text and places a box around it.
   A. Prompt the user for the text height and set it to the variable **TXHT**.
   B. Prompt the user for a point representing the lower-left corner of the box and set it to a variable.
   C. Prompt for the text string from the user.
   D. Set the text string length to the variable **LG1**. Use the (**strlen**) function. The following is an example of using this function.
      
      (setq TEXT (**getstring** T "Enter Text: "))
      (setq LG1 (**strlen** TEXT))
      
   E. Set the X length of the box to a variable using the expression:
      
      (^ LG1 TXHT)
      
   F. Set the Y length of the box to a variable using the expression:
      
      (^ 3 TXHT)
      
   G. Draw the box using the variables set in E and F.
   H. Calculate the center point of the box and set it to the variable **CEN1**.
   I. Draw the text string inside the box. Use the **MC** text justification option for point **CEN1**.