

# Chapter 33

---

## Working with AutoLISP

### Learning Objectives

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

- *Perform mathematical operations using AutoLISP.*
- *Use trigonometrical functions in AutoLISP.*
- *Understand the basic AutoLISP functions and their applications.*
- *Load and run AutoLISP programs.*
- *Use the Load/Unload Applications dialog box.*
- *Use flowcharts to analyze problems.*
- *Test a condition using conditional functions.*

## ABOUT AutoLISP

Developed by Autodesk, Inc., **AutoLISP** is an implementation of the **LISP** programming language (LISP is an acronym for **LISt Processor**.) The first reference to LISP was made by John McCarthy in the April 1960 issue of *The Communications of the ACM*.

Except for **FORTRAN** and **COBOL**, most of the languages developed in the early 1960s have become obsolete. But LISP has survived and has become a leading programming language for artificial intelligence (AI). Some of the dialects of the LISP programming language are Common LISP, BYSCO LISP, ExperLISP, GCLISP, IQLISP, LISP/80, LISP/88, MuLISP, TLCLISP, UO-LISP, Waltz LISP, and XLISP. XLISP is a public-domain LISP interpreter. The LISP dialect that resembles AutoLISP is Common LISP. The AutoLISP interpreter is embedded within the AutoCAD software package. However, AutoCAD LT and AutoCAD Versions 2.17 and lower lack the AutoLISP interpreter; therefore, you can use the AutoLISP programming language only with AutoCAD Release 2.18 and up.

AutoCAD contains most of the commands used to generate a drawing. However, some commands are not provided in AutoCAD. For example, AutoCAD has no command to make global changes in the drawing text objects. With AutoLISP you can write a program in the AutoLISP programming language that will make global or selective changes in the drawing text objects. You can use AutoLISP to write any program or embed it in the menu and thus customize your system to make it more efficient.

The AutoLISP programming language has been used by hundreds of third-party software developers to write software packages for various applications. For example, the author of this text has developed a software package, **SMLayout**, that generates flat layouts of various geometrical shapes like transitions, intersection of pipes and cylinders, elbows, cones, and tank heads. There is a huge demand for AutoLISP programmers as consultants for developing application software and custom menus.

This chapter assumes that you are familiar with AutoCAD commands and AutoCAD system variables. However, you need not be an AutoCAD or programming expert to begin learning AutoLISP. This chapter also assumes that you have no prior programming knowledge. If you are familiar with any other programming language, learning AutoLISP may be easy. A thorough discussion of various functions and a step-by-step explanation of the examples should make it fun to learn. This chapter discusses the most frequently used AutoLISP functions and their application in writing a program. For the functions that are not discussed in this chapter, refer to the **AutoLISP Programmers Reference Manual** from Autodesk. AutoLISP does not require any special hardware. If your system runs AutoCAD, it will also run AutoLISP. To write AutoLISP programs you can use any text editor.

## MATHEMATICAL OPERATIONS

A mathematical function constitutes an important feature of any programming language. Most of the mathematical functions commonly used in programming and mathematical calculations are available in AutoLISP. You can use AutoLISP to add, subtract, multiply, and divide numbers. You can also use it to find the sine, cosine, and arctangent of

angles expressed in radians. There is a host of other calculations you can do with AutoLISP. This section discusses the most frequently used mathematical functions supported by the AutoLISP programming language.

## Addition

Format (+ num1 num2 num3 - - -)

This function (+) calculates the sum of all the numbers to the right of the plus (+) sign (num1 + num2 + num3 + . . .). The numbers can be integers or real. If the numbers are integers, then the sum is an integer. If the numbers are real, the sum is real. However, if some numbers are real and some are integers, the sum is real. In the following display, all numbers in the first two examples are integers, so the result is an integer. In the third example, one number is a real number (50.0), so the sum is a real number.

### Examples

Command: (+ 2 5)	returns 7
Command: (+ 2 30 4 50)	returns 86
Command: (+ 2 30 4 50.0)	returns 86.0

## Subtraction

Format (- num1 num2 num3 - - -)

This function (-) subtracts the second number from the first number (num1 - num2). If there are more than two numbers, the second and subsequent numbers are added and the sum is subtracted from the first number [num1 - (num2 + num3 + . . .)]. In the first of the following examples, 14 is subtracted from 28 and returns 14. Since both numbers are integers, the result is an integer. In the third example, 20 and 10.0 are added, and the sum of these two numbers (30.0) is subtracted from 50, returning a real number, 20.0.

### Examples

Command: (- 28 14)	returns 14
Command: (- 25 7 11)	returns 7
Command: (- 50 20 10.0)	returns 20.0
Command: (- 20 30)	returns -10
Command: (- 20.0 30.0)	returns -10.0

## Multiplication

Format (\* num1 num2 num3 - - -)

This function (\*) calculates the product of the numbers to the right of the asterisk (num1 x num2 x num3 x . . .). If the numbers are integers, the product of these numbers is an integer. If one of the numbers is a real number, the product is a real number.

### Examples

Command: (* 2 5)	returns 10
------------------	------------

Command: (* 2 5 3)	returns 30
Command: (* 2 5 3 2.0)	returns 60.0
Command: (* 2 -5.5)	returns -11.0
Command: (* 2.0 -5.5 -2)	returns 22.0

## Division

Format (/ **num1 num2 num3 - -**)

This function (/) divides the first number by the second number (num1/num2). If there are more than two numbers, the first number is divided by the product of the second and subsequent numbers [num1 / (num2 x num3 x . . .)]. In the fourth of the following examples, 200 is divided by the product of 5.0 and 4 [200 / (5.0 \* 4)].

### Examples

Command: (/ 30)	returns 30
Command: (/ 3 2)	returns 1
Command: (/ 3.0 2)	returns 1.5
Command: (/ 200.0 5.0 4)	returns 10.0
Command: (/ 200 -5)	returns -40
Command: (/ -200 -5.0)	returns 40.0

## INCREMENTED, DECREMENTED, AND ABSOLUTE NUMBERS

### Incremented Number

Format (**1+ number**)

This function (**1+**) adds 1 (integer) to **number** and returns a number that is incremented by 1. In the second example below, 1 is added to -10.5 and returns -9.5.

### Examples

(1+ 20)	returns 21
(1+ -10.5)	returns -9.5

### Decrement Number

Format (**1- number**)

This function (**1-**) subtracts 1 (integer) from the **number** and returns a number that is decremented by 1. In the second example below, 1 is subtracted from -10.5 and returns -11.5

### Examples

(1- 10)	returns 9
(1- -10.5)	returns -11.5

## Absolute Number

Format (**abs num**)

The **abs** function returns the absolute value of a number. The number may be an integer number or a real number. In the second example below, the function returns 20 because the absolute value of -20 is 20.

### Examples

(abs 20)	returns 20
(abs -20)	returns 20
(abs -20.5)	returns 20.5

## TRIGONOMETRIC FUNCTIONS

### sin

Format (**sin angle**)

The **sin** function calculates the sine of an angle, where the angle is expressed in radians. In the second example, the **sin** function calculates the sine of pi (180-degree) and returns 0.

### Examples

Command: (sin 0)	returns 0.0
Command: (sin pi)	returns 1.22461e-016
Command: (sin 1.0472)	returns 0.866027

### cos

Format (**cos angle**)

The **cos** function calculates the cosine of an angle, where the angle is expressed in radians. In the third of the following examples, the **cos** function calculates the cosine of pi (180-degree) and returns -1.0.

### Examples

Command: (cos 0)	returns 1.0
Command: (cos 0.0)	returns 1.0
Command: (cos pi)	returns -1.0
Command: (cos 1.0)	returns 0.540302

### atan

Format (**atan num1**)

The **atan** function calculates the arctangent of **num1**, and the calculated angle is expressed in radians. In the second example, the **atan** function calculates the arctangent of 1.0 and returns 0.785398 (radians).

**Examples**

Command: (atan 0.5)	returns 0.463648
Command: (atan 1.0)	returns 0.785398
Command: (atan -1.0)	returns -0.785398

You can also specify a second number in the atan function:

Format (**atan num1 num2**)

If the second number is specified, the function returns the arctangent of (num1/num2) in radians. In the first example, the first number (0.5) is divided by the second number (1.0), and the **atan** function calculates the arctangent of the dividend ( $0.5/1.0 = 0.5$ ).

**Examples**

Command: (atan 0.5 1.0)	returns 0.463648 radians
Command: (atan 2.0 3.0)	returns 0.588003 radians
Command: (atan 2.0 -3.0)	returns 2.55359 radians
Command: (atan -2.0 3.00)	returns -0.588003 radians
Command: (atan -2.0 -3.0)	returns -2.55359 radians
Command: (atan 1.0 0.0)	returns 1.5708 radians
Command: (atan -0.5 0.0)	returns -1.5708 radians

**angtos**

Format (**angtos angle [mode [precision]]**)

The **angtos** function returns the angle expressed in radians in a string format. The format of the string is controlled by the **mode** and **precision** settings.

**Examples**

(angtos 0.588003 0 4)	returns "33.6901"
(angtos 2.55359 0 4)	returns "146.3099"
(angtos 1.5708 0 4)	returns "90.0002"
(angtos -1.5708 0 2)	returns "270.00"



### Note

In *(angtos angle [mode [precision]])*

*angle* is angle in radians

*mode* is the angtos mode that corresponds to the AutoCAD system variable AUNITS

The following modes are available in AutoCAD:

<u>ANGTOS MODE</u>	<u>EDITING FORMAT</u>
0	Decimal degrees
1	Degrees/minutes/seconds
2	Grads
3	Radians
4	Surveyor's units

**Precision** is an integer that controls the number of decimal places. Precision corresponds to the AutoCAD system variable **AUPREC**. The minimum value of **precision** is zero and the maximum is four.

In the first example mentioned above, angle is 0.588003 radians, mode is 0 (angle in degrees), and precision is 4 (four places after decimal). The function will return 33.6901.

## RELATIONAL STATEMENTS

Programs generally involve features that test a particular condition. If the condition is true, the program performs certain functions, and if the condition is not true, then the program performs other functions. For example, the relational statement (if (< x 5)) tests true if the value of the variable x is less than 5. This type of test condition is frequently used in programming. The following section discusses various relational statements used in AutoLISP programming.

### Equal to

Format (= atom1 atom2 - - - )

This function (=) checks whether the two atoms are equal. If they are equal, the condition is true and the function will return **T**. If the specified atoms are not equal, the condition is false and the function will return **nil**.

#### Examples

(= 5 5)	returns T
(= 5 4.9)	returns nil
(= 5.5 5.5 5.5)	returns T
(= "yes" "yes")	returns T
(= "yes" "yes" "no")	returns nil

### Not equal to

Format (/= atom1 atom2 - - - )

This function (/=) checks whether the two atoms are not equal. If they are not equal, the condition is true and the function will return **T**. If the specified atoms are equal, the condition is false and the function will return **nil**.

#### Examples

(/= 50 4)	returns T
(/= 50 50)	returns nil
(/= 50 -50)	returns T
(/= "yes" "no")	returns T

### Less than

Format (**< atom1 atom2 - - -**)

This function (<) checks whether the first atom (**atom1**) is less than the second atom (**atom2**). If it is true, then the function will return **T**. If it is not, the function will return **nil**.

#### Examples

(< 3 5)	returns T
(< 5 3 4 2)	returns nil
(< "x" "y")	returns T

### Less than or equal to

Format (**<= atom1 atom2 - - -**)

This function (<=) checks whether the first atom (**atom1**) is less than or equal to the second atom (**atom2**). If it is, the function will return **T**. If it is not, the function will return **nil**.

#### Examples

(<= 10 15)	returns T
(<= "c" "b")	returns nil
(<= -2.0 0)	returns T

### Greater than

Format (**> atom1 atom2 - - -**)

This function (>) checks whether the first atom (**atom1**) is greater than the second atom (**atom2**). If it is, the function will return **T**. If it is not, then the function will return **nil**. In the first example below, 15 is greater than 10. Therefore, this relational function is true and the function will return **T**. In the second example, 10 is greater than 9, but this number is not greater than the second 9; therefore, this function will return **nil**.

#### Examples

(> 15 10)	returns T
(> 10 9 9)	returns nil
(> "c" "b")	returns T



## Greater than or equal to

Format (**>= atom1 atom2 - - -**)

This function (**>=**) checks whether the first atom (**atom1**) is greater than or equal to the second atom (**atom2**). If it is, the function returns **T**; otherwise, it will return **nil**. In the first example below, 78 is greater than 50, even though 78 is not equal to 50; therefore, it will return **T**.

### Examples

( <b>&gt;=</b> 78 50)	returns T
( <b>&gt;=</b> "x" "y")	returns T
( <b>&gt;=</b> "78" "80")	returns nil

## defun,setq, getpoint, AND Command FUNCTIONS

### defun

The **defun** function is used to define a function in an AutoLISP program. The format of the **defun** function is:

```
(defun name [argument])
      Where  Name ----- Name of the function
            Argument----- Argument list
```

### Examples

(**defun** ADNUM ( )

Defines a function ADNUM with no arguments or local symbols. This means all the variables used in the program are global variables. A global variable does not lose its value after the programs ends.

(**defun** ADNUM (a b c)

Defines a function ADNUM that has three arguments: **a**, **b**, and **c**. The variables **a**, **b**, and **c** receive their value from outside the program.

(**defun** ADNUM (/ a b)

Defines a function ADNUM that has two local variables: **a** and **b**. A local variable is one that retains its value during program execution and can be used within that program only.

(**defun** C:ADNUM ( )

With **C:** in front of the function name, the function can be executed by entering the name of the function at the AutoCAD Command prompt. If **C:** is not used, the function name has to be enclosed in parentheses.



**Note**

*AutoLISP contains some built-in functions. Do not use any of those names for function or variable names. The following is a list of some of the names reserved for AutoLISP built-in functions. (Refer to the AutoLISP Programmer's Reference manual for a complete list of AutoLISP built-in functions.)*

abs	ads	alloc
and	angle	angtos
append	apply	atom
ascii	assoc	atan
atof	atoi	distance
equal	fix	float
if	length	list
load	member	nil
not	nth	null
open	or	pi
read	repeat	reverse
set	type	while

**setq**

The **setq** function is used to assign a value to a variable. The format of the **setq** function is:

**(setq Name Value [Name Value].....)**

Where **Name** ----- Name of variable

**Value** ----- Value assigned to variable

The value assigned to a variable can be any expression (numeric, string, or alphanumeric).

Command: **(setq X 12)**

Command: **(setq X 6.5)**

Command: **(setq X 8.5 Y 12)**

In this last expression, the number 8.5 is assigned to the variable **X**, and the number 12 is assigned to the variable **Y**.

Command: **(setq answer "YES")**

In this expression the string value "YES" is assigned to the variable **answer**.

The **setq** function can also be used in conjunction with other expressions to assign a value to a variable. In the following examples the **setq** function has been used to assign values to different variables.

**(setq pt1 (getpoint "Enter start point: "))**

**(setq ang1 (getangle "Enter included angle:"))**

**(setq answer (getstring "Enter YES or NO: "))**

**Note**

*AutoLISP uses some built-in function names and symbols. Do not assign values to any of those functions. The following functions are valid ones, but must never be used because the **pi** and **angle** functions that are reserved functions will be redefined.*

```
(setq pi 3.0)
(setq angle (. . .))
```

**getpoint**

The **getpoint** function pauses to enable you to enter the X, Y coordinates or X, Y, Z coordinates of a point. The coordinates of the point can be entered from the keyboard or by using the screen cursor. The format of the **getpoint** function is:

**(getpoint [point] [prompt])**

Where **Point** ----- Enter a point, or select a point

**Prompt** ----- Prompt to be displayed on screen

**Example**

```
(setq pt1 (getpoint))
(setq pt1 (getpoint "Enter starting point"))
```

**Note**

*You cannot enter the name of another AutoLISP routine in response to the **getpoint** function.*

*A 2D or a 3D point is always defined with respect to the current user coordinate system (UCS).*

**Command**

The **Command** function is used to execute standard AutoCAD commands from within an AutoLISP program. The AutoCAD command name and the command options have to be enclosed in double quotation marks. The format of the **Command** function is:

**(Command "commandname")**

Where **Command** ----- AutoLISP function

**commandname** ----- AutoCAD command

**Example**

**(Command "line" pt1 pt2 "")**

Where **"Line"** ----- AutoCAD LINE Command

**Pt1** ----- First point

**Pt2** ----- Second point

**" "** ----- for Return (two double quotes with no space between)

**Note**

Prior to AutoCAD Release 12, the **Command** function **could not be used** to execute the AutoCAD PLOT command. For example: (Command "plot" . . .) was not a valid statement. In AutoCAD Release 13 and later, you can use plot with the Command function (Command "plot" . . .).

The **Command** function can also be used to enter data with the **TEXT** command. For example, enter (command "text" 4.0,4 "" "AutoCAD Text") at the Command prompt. You will notice that the text is automatically written.

You cannot use the input functions of AutoLISP with the **Command** function. The input functions are **getpoint**, **getangle**, **getstring**, and **getint**. For example, (Command "getpoint" . . .) or (Command "getangle" . . .) are not valid functions. If the program contains such a function, an error message is displayed when the program is loaded. However, you can enclose them in parentheses (Command "text" (getpoint) (getdist) (getangle) "hello, sailer")

### Example 1

Write a program that will prompt you to select three points of a triangle and then draw lines through those points to generate the triangle shown in Figure 33-1.

#### Step 1: Understanding the LISP program

Most programs consist of essentially three parts: **input**, **output**, and **process**. **Process** includes what is involved in generating the desired output from the given input (Figure 33-2). Before writing a program, you must identify these three parts. In this example, the **input** to the program is the coordinates of the three points. The desired **output** is a triangle. The **process** needed to generate a triangle is to draw three lines from P1 to P2, P2 to P3, and P3 to P1. Identifying these three sections makes the programming process less confusing.

The process section of the program is vital to the success of the program. Sometimes it is simple, but sometimes it involves complicated calculations. If the program involves many calculations, divide the program into sections (and perhaps subsections) that are laid out in a logical and systematic order. Also, remember that programs need to be edited from time to

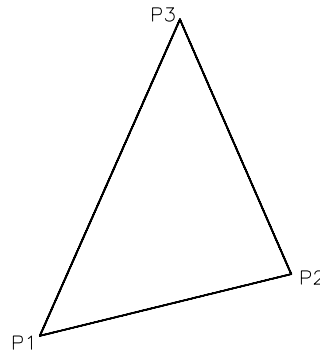


Figure 33-1 Triangle P1, P2, P3

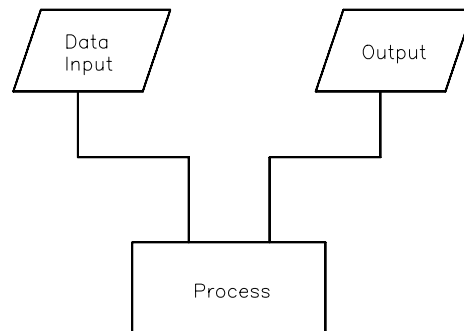


Figure 33-2 Three elements of a Program

time. This is the reason, it is wise to document the programs as clearly and unambiguously as possible so that other programmers can understand what the program is doing at different stages of its execution. Give sketches, and identify points where possible.

**Input**

Location of point P1  
Location of point P2  
Location of point P3

**Output**

Triangle P1, P2, P3

**Process**

Line from P1 to P2  
Line from P2 to P3  
Line from P3 to P1

**Step 2: Writing the LISP program**

Use any text editor to write the following LISP program. The following file is a listing of the AutoLISP program for Example 1. **The line numbers at the right are not a part of the program; they are shown here for reference only.**

```

;This program will prompt you to enter three points          1
;of a triangle from the keyboard, or select three points    2
;by using the screen cursor. P1, P2, P3 are triangle corners. 3
                                                              4
(defun c:TRIANG1()                                          5
  (setq P1 (getpoint "\n Enter first point of Triangle: ")) 6
  (setq P2 (getpoint "\n Enter second point of Triangle: ")) 7
  (setq P3 (getpoint "\n Enter third point of Triangle: ")) 8
  (Command "LINE" P1 P2 P3 "C")                             9
)                                                            10

```

**Explanation**

Lines 1-3

The first three lines are comment lines describing the function of the program. These lines are important because they make it easier to edit a program. Comments should be used when needed. All comment lines must start with a semicolon (;). These lines are ignored when the program is loaded.

Line 4

This is a blank line that separates the comment section from the program. Blank lines can be used to separate different modules of a program. This makes it easier to identify different sections that constitute a program. The blank lines have no effect on the program.

Line 5

**(defun c:TRIANG1()**

In this line, **defun** is an AutoLISP function that defines the function **TRIANG1**. **TRIANG1** is the name of the function. The **c:** in front of the function name **TRIANG1** enables it to be

executed like an AutoCAD command. If the *c:* is missing, the **TRIANG1** command can be executed only by enclosing it in parentheses (**TRIANG1**). The **TRIANG1** function has three global variables (**P1**, **P2**, and **P3**). When you first write an AutoLISP program, it is a good practice to keep the variables global, because after you load and run the program, you can check the values of these variables by entering an exclamation point (!) followed by the variable name at the Command prompt (Command: !P1). Once the program is tested and it works, you should make the variable local, (**defun c:TRIANG1(/ P1 P2 P3)**).

Line 6

**(setq P1 (getpoint "\n Enter first point of Triangle: "))**

In this line, the **getpoint** function pauses for you to enter the first point of the triangle. The prompt, **Enter first point of Triangle**, is displayed in the prompt area of the screen. You can enter the coordinates of this point at the keyboard or select a point by using the screen cursor. The **setq** function then assigns these coordinates to the variable **P1**. **\n** is used for the carriage return so that the statement that follows **\n** is printed on the next line ("**n**" stands for "newline".)

Lines 7 and 8

**(setq P2 (getpoint "\n Enter second point of Triangle: "))**

**(setq P3 (getpoint "\n Enter third point of Triangle: "))**

These two lines prompt you to enter the second and third corners of the triangle. These coordinates are then assigned to the variables **P2** and **P3**. **\n** causes a carriage return so that the input prompts are displayed on the next line.

Line 9

**(Command "LINE" P1 P2 P3 "C")**

In this line, the **Command** function is used to enter the **LINE** command and then draw a line from **P1** to **P2** and from **P2** to **P3**. "**C**" (for "close" option) joins the last point, **P3**, with the first point, **P1**. All AutoCAD commands and options, when used in an AutoLISP program, have to be enclosed in double quotation marks. The variables **P1**, **P2**, **P3** are separated by a blank space.

Line 10

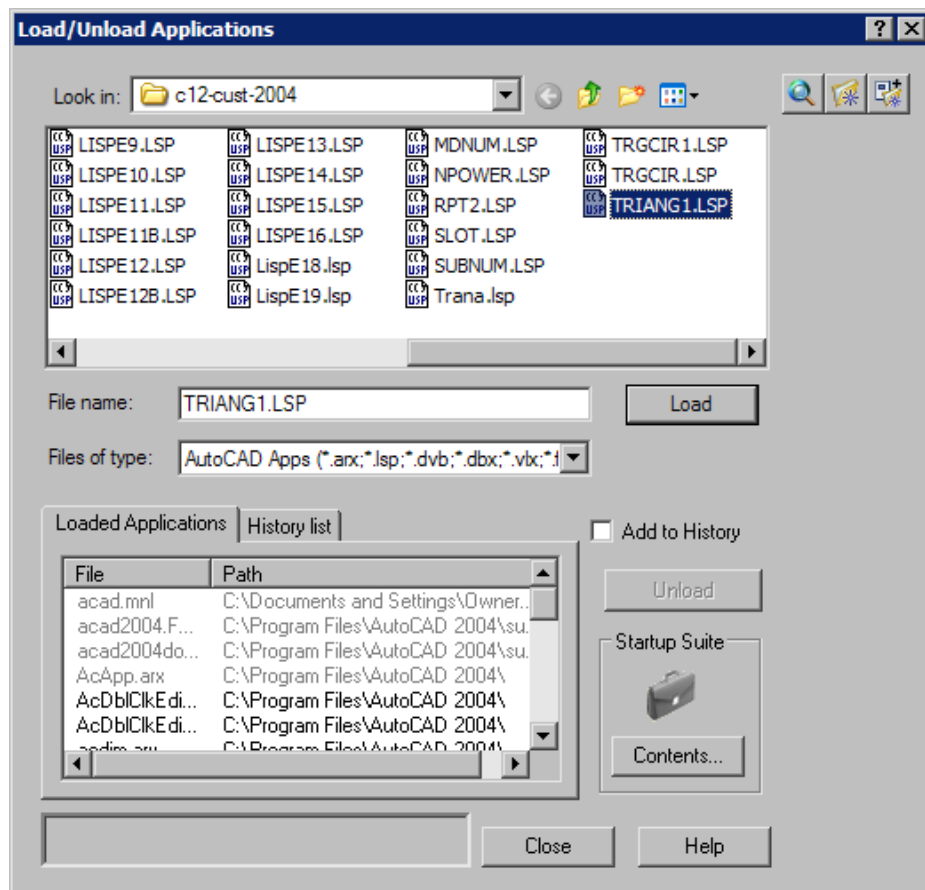
This line consists of a closing parenthesis that completes the definition of the function, **TRIANG1**. This parenthesis could have been combined with the previous line. It is good practice to keep it on a separate line so that the end of a definition can be easily identified. In this program there is only one function defined, so it is easy to locate the end of a definition. But in some programs a number of definitions or modules within the same program might need to be clearly identified. The parentheses and blank lines help to identify the start and end of a definition or a section in the program.

## LOADING AN AUTOLISP PROGRAM

Save the file with the extension *.lsp*. There are generally two names associated with an AutoLISP program: the program file name and the function name. For example, *triang1.lsp* is the name of the file, not a function name. All AutoLISP file names have the extension *.lsp*. An AutoLISP

file can have one or several functions defined within the same file. For example, TRIANG1 in Example 1 is the name of a function. To execute a function, the AutoLISP program file that defines that function must be loaded. Use the following procedure to load an AutoLISP file when you are in the drawing editor.

Loading an AutoLISP program can be achieved either through the dialog box or by entering a command at the Command prompt. Choose the **Tools > AutoLISP > Load** menu or choose **Load Application** from the **Tools** menu to display the **Load/Unload Application** dialog box (Figure 33-3). This dialog box can be used to load LSP, VLX, FAS, VBA, DBX, and ObjectARX applications. VBA, DBX, and ObjectARX files are loaded immediately when you select a file. LSP, VLX, and FAS files are queued and loaded when you close the **Load/Unload Application** dialog box. The top portion of the dialog box lists the files in the selected directory. The file type can be changed by entering the file type (\*.lsp) in the **Files of type:** edit box or by selecting the file type in the pop-down list. A file can be loaded by selecting the file and then choosing the **Load** button or by double-clicking on the file. The following is the description of other features of the **Load/Unload Application** dialog box:



**Figure 33-3** Loading AutoLISP files using the **Load/Unload Applications** dialog box

**Load**

The **Load** button can be used to load and reload the selected files. The files can be selected from the file list box, **Load Application** tab, or **History List** tab. The Object ARX files cannot be reloaded. You must first unload the ObjectARX file and then reload it.

**Load Application Tab**

When you choose the **Load Application** tab, AutoCAD displays the applications that are currently loaded. You can add files to this list by dragging the file names from the file list box and then dropping them in the **Load Application** list.

**History List Tab**

When you choose the **History List** tab, AutoCAD displays the list of files that have been previously loaded with **Add to History** check box selected. If the **Add to History** check box is not selected and you drag and drop the files in the History list, the files are loaded but not added to the **History List**. The users can directly reload a file by selecting it from the history list in this tab.

**Add to History**

When **Add to History** is selected, it adds the files to the **History List** when you drag and drop the files in the **History List**.

**Unload**

The **Unload** button appears when you choose the **Loaded Application** tab. To unload an application, select the file name in the **Loaded Application** list and then choose the **Unload** button. LISP files and the ObjectARX files that are not registered for unloading cannot be unloaded.

**Remove**

The **Remove** button appears when you choose the **History List** tab. To remove a file from the History List, select the file in the History List and then choose the **Remove** button.

**Startup Suit**

The files in the **Startup Suit** are automatically loaded each time you start AutoCAD. When you choose **Startup Suit**, AutoCAD displays the **Startup Suit** dialog box that contains a list of files. You can add files to the list by choosing the **Add** button. You can also drag the files from the file list box and drop them in the **Startup Suit**. To add files from the **History List**, right-click on the file.

The AutoLISP program can also be loaded using the **LOAD** command in the following format.

Command: (load "[path]file name")

Where **Command** ----- AutoCAD command prompt

**load** ----- Loads an AutoLISP program file

**file name** ----- Path and name of AutoLISP program file



The AutoLISP file name and the optional path name must be enclosed in double quotes. The **load** and **file name** must be enclosed in parentheses. If the parentheses are missing, AutoCAD will try to load a shape or a text font file, not an AutoLISP file. The space between **load** and **file name** is not required. If AutoCAD is successful in loading the file, it will display the name of the function in the Command prompt area of the screen.

To run the program, type the name of the function at the AutoCAD Command prompt, and press ENTER (Command: **TRIANG1**). If the function name does not contain **C:** in the program, you can run the program by enclosing the function name in parentheses:

Command: **TRIANG1** or Command: (**TRIANG1**)



#### Note

Use a forward slash when defining the path for loading an AutoLISP program. For example, if the AutoLISP file **TRIANG** is in the **LISP** subdirectory on the **C** drive, use the following command to load the file. You can also use a double backslash (\\) in place of the forward slash.

Command: (**load "c:/lisp/triang"**) or Command: (**load "c:\\lisp\\triang"**)



#### Tip

You can also load an application by using the standard Windows drag and drop technique. To load a LISP program, select the file in the Windows Explorer and then drag and drop it in the graphics window of AutoCAD. The selected program will be automatically loaded.

## Exercise 1

General

Write an AutoLISP program that will draw a line between two points (Figure 33-4). The program must prompt the user to enter the **X** and **Y** coordinates of the points.

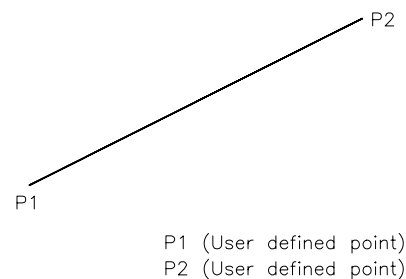


Figure 33-4 Draw line from point P1 to P2

## getcorner, getdist, AND setvar FUNCTIONS

### getcorner

The **getcorner** function pauses for you to enter the coordinates of a point. The coordinates of the point can be entered or the screen crosshairs can be used to specify its location. This function requires a base point and displays a rectangle with respect to the base point as you move the screen crosshairs around the screen. The format of the **getcorner** function is given next.

(getcorner point [prompt])

Where **point** ----- Base point  
**prompt** ----- Prompt displayed on screen

### Examples

```
(getcorner pt1)
(setq pt2 (getcorner pt1))
(setq pt2 (getcorner pt1 "Enter second point: "))
```



### Note

*The base point and the point that you select in response to the getcorner function are located with respect to the current UCS.*

*If the point you select is a 3D point with X, Y, and Z coordinates, the Z coordinate is ignored. The point assumes current elevation as its Z coordinate.*

## getdist

The **getdist** function pauses for you to enter distance, and it then returns the distance as a real number. The format of the **getdist** function is:

(getdist [point] [prompt])

Where **Point** ----- First point for distance  
**Prompt** ----- Any prompt that needs to be displayed on screen

### Examples

```
(getdist)
(setq dist (getdist))
(setq dist (getdist pt1))
(setq dist (getdist "Enter distance"))
(setq dist (getdist pt1 "Enter second point for distance"))
```

The distance can be entered by selecting two points on the screen. For example, if the assignment is (**setq dist (getdist)**), you can enter a number or select two points. If the assignment is (**setq dist (getdist pt1)**), where the first point (pt1) is already defined, you need to select the second point only. The getdist function will always return the distance as a real number. For example, if the current setting is architectural and the distance is entered in architectural units, the getdist function will return the distance as a real number.

## setvar

The **setvar** function assigns a value to an AutoCAD system variable. The name of the system variable must be enclosed in double quotes. The format of the **setvar** function is:

(setvar "variable-name" value)

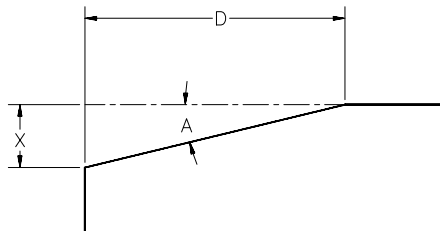
Where **Variable-name** AutoCAD system variable  
**Value** ----- Value to be assigned to the system variable

**Examples**

```
(setvar "cmdecho" 0)
(setvar "dimscale" 1.5)
(setvar "ltscale" 0.5)
(setvar "dimcen" -0.25)
```

**Example 2**

Write an AutoLISP program that will generate a chamfer between two given lines by entering the chamfer angle and the chamfer distance. To generate a chamfer, AutoCAD uses the values assigned to system variables **CHAMFERA** and **CHAMFERB**. When you select the **CHAMFER** command, the first and second chamfer distances are automatically assigned to the system variables **CHAMFERA** and **CHAMFERB**. The **CHAMFER** command then uses these assigned values to generate a chamfer. However, in most engineering drawings, the preferred way to generate the chamfer is to enter the chamfer length and the chamfer angle, as shown in Figure 33-5.



*Figure 33-5 Chamfer with angle A and distance D*

**Input**

First chamfer distance (D)  
Chamfer angle (A)

**Output**

Chamfer between any two  
selected lines

**Step 1: Understanding the program algorithm****Process**

1. Calculate second chamfer distance.
2. Assign these values to the system variables **CHAMFERA** and **CHAMFERB**.
3. Use **CHAMFER** command.  
to generate chamfer,

**Calculations**

$$\begin{aligned}
 x/d &= \tan a \\
 x &= d * (\tan a) \\
 &= d * [(\sin a) / (\cos a)]
 \end{aligned}$$

**Step 2: Writing the LISP program**

Use any text editor to write down the LISP program. The following file is a listing of the program for Example 3. **The line numbers on the right are not a part of the file; they are for reference only.**

```

;This program generates a chamfer by entering
;the chamfer angle and the chamfer distance
;
(defun c:chamf (/ d a)
  (setvar "cmdecho" 0)
  (graphscr)
  (setq d (getdist "\n Enter chamfer distance: "))
  (setq a (getangle "\n Enter chamfer angle: "))
  (setvar "chamfera" d)
  (setvar "chamferb" (* d (/ (sin a) (cos a))))
  (command "chamfer")
  (setvar "cmdecho" 1)
  (princ)
)

```

**Explanation**

Line 7

**(setq d (getdist "\n Enter chamfer distance: "))**

The **getdist** function pauses for you to enter the chamfer distance, then the **setq** function assigns that value to variable d.

Line 8

**(setq a (getangle "\n Enter chamfer angle: "))**

The **getangle** pauses for you to enter the chamfer angle, then the **setq** function assigns that value to variable a.

Line 9

**(setvar "chamfera" d)**

The **setvar** function assigns the value of variable d to AutoCAD system variable **chamfera**.

Line 10

**(setvar "chamferb" (\* d (/ (sin a) (cos a))))**

The **setvar** function assigns the value obtained from the expression **(\* d (/ (sin a) (cos a)))** to the AutoCAD system variable **chamferb**.

Line 11

**(command "chamfer")**

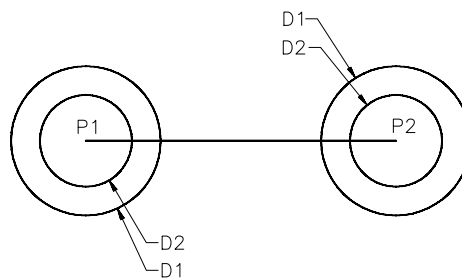
The **Command** function uses the **CHAMFER** command to generate a chamfer.

**Step 3: Loading the LISP program**

Save the file with the extension *.lsp* and then follow the procedure described in Example 1 to load the lisp file.

**Exercise 2***General*

Write an AutoLISP program that will generate the drawing shown in Figure 33-6. The program should prompt the user to enter points P1 and P2 and diameters D1 and D2.



*Figure 33-6 Concentric circles with connecting line*

**list FUNCTION**

In AutoLISP the **list** function is used to define a 2D or 3D point. The list function can also be designated by using the single quote character (*'*), if the expression does not contain any variables or undefined items.

**Examples**

(Setq x (list 2.5 3 56))	returns (2.5 3 56)
(Setq x '(2.5 3 56))	returns (2.5 3 56)

**Tip**

*A list when included with integers or real numbers can be enclosed within a single parentheses.  
A list when represented in the form of a single quote can be given outside the parentheses.*

**car, cdr, and cadr FUNCTIONS****car**

The **car** function returns the first element of a list. If the list does not contain any elements, the function will return **nil**. Remember that before using this function, you need to list the elements of the function using the **list** function. The format of the **car** function is:

(car list)

Where **car** ----- Returns the first element  
**list** ----- List of elements

**Examples**

(car '(2.5 3 56))	returns 2.5
(car '(x y z))	returns X
(car '((15 20) 56))	returns (15 20)
(car '())	returns nil
(car (list 2 3.0 4))	returns 2
(car '(A B C))	

The single quotation mark signifies a list.

**cdr**

The **cdr** function returns a list with the first element removed from the list. The format of the **cdr** function is:

(cdr list)

Where **cdr** ----- Returns a list with the first element removed  
**list** ----- List of elements

**Examples**

(cdr '(2.5 3 56))	returns (3 56)
(cdr '(x y z))	returns (Y Z)
(cdr '((15 20) 56))	returns (56)
(cdr '())	returns nil

**cadr**

The **cadr** function performs two operations, **cdr** and **car**, to return the second element of the list. The **cdr** function removes the first element, and the **car** function returns the first element of the new list. The format of the **cadr** function is:

(cadr list)

Where **cadr** ----- Performs two operations (car (cdr '(x y z)))  
**list** ----- List of elements

**Examples**

(cadr '(2 3))	returns 3
(cadr '(2 3 56))	returns 3
(cadr '(x y z))	returns y
(cadr '((15 20) 56 24))	returns 56

In these examples, **cadr** performs two functions:

(cadr '(x y z))	= (car (cdr '(x y z)))
	= (car (y z)) returns y

**Note**

*In addition to the functions `car`, `cdr`, and `cadr`, several other functions can be used to extract different elements of a list. The following is a list of some of the functions where the function `f` consists of a list `'(x y z w)`.*

<code>(setq f '(x y z w))</code>	
<code>(caar f)=(car (car f))</code>	returns <code>x</code>
<code>(cdar f)=(cdr (car f))</code>	returns <code>(y)</code>
<code>(cadar)=(car (cdr (car f)))</code>	returns <code>y</code>
<code>(cddr f)=(cdr (cdr f))</code>	returns <code>(w)</code>
<code>(caddr f)=(car (cdr (cdr f)))</code>	returns <code>w</code>
<code>(last f)</code>	returns <code>w</code>

## graphscr, textscr, princ, AND terpri FUNCTIONS

### graphscr

The **graphscr** function switches from the text window to the graphics window, provided the system has only one screen. If the system has two screens, this function is ignored.

### textscr

The **textscr** function switches from the graphics window to the text window, provided the system has only one screen. If the system has two screens, this function is ignored.

### princ

The **princ** function prints (or displays) the value of the variable. If the variable is enclosed in double quotes, the function prints (or displays) the expression that is enclosed in the quotes. The format of the **princ** function is:

```
(princ [variable or expression])
```

#### Examples

<code>(princ)</code>	prints a blank on screen
<code>(princ a)</code>	prints the value of variable <code>a</code> on screen
<code>(princ "Welcome")</code>	prints <code>Welcome</code> on screen

### terpri

The **terpri** function prints a new line on the screen, just as `\n` does. This function is used to print the line that follows the **terpri** function.

#### Examples

```
(setq p1 (getpoint "Enter first point: "))(terpri)
(setq p2 (getpoint "Enter second point: "))
```

The first line (`Enter first point:` ) will be displayed in the screen's Command prompt area. The **terpri** function causes a carriage return; therefore, the second line (`Enter second point:`)

will be displayed on a new line, just below the first line. If the `terpri` function is missing, the two lines will be displayed on the same line (Enter first point: Enter second point:).

### Example 3

Write a program that will prompt you to enter two opposite corners of a rectangle and then draw the rectangle on the screen as shown in Figure 33-7.

#### Step 1: Understanding the program algorithm

##### Input

Coordinates of point P1  
Coordinates of point P3

##### Output

Rectangle

##### Process

1. Calculate the coordinates of the points P2 and P4.
2. Draw the following lines.  
Line from P1 to P2  
Line from P2 to P3  
Line from P3 to P4  
Line from P4 to P1

The X and Y coordinates of points P2 and P4 can be calculated using the `car` and `cadr` functions. The `car` function extracts the X coordinate of a given list, and the `cadr` function extracts the Y coordinate.

##### **X coordinate of point p2**

```
x2 = x3
x2 = car (x3 y3)
x2 = car p3
```

##### **Y coordinate of point p2**

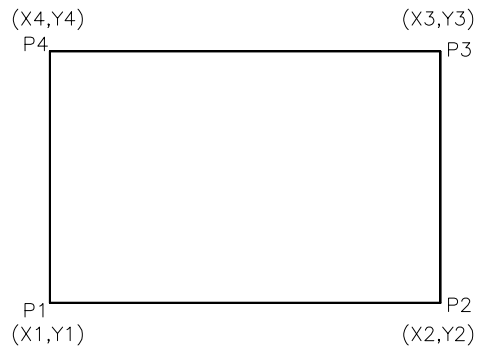
```
y2 = y1
y2 = cadr (x1 y1)
y2 = cadr p1
```

##### **X coordinate of point p4**

```
x4 = x1
x4 = car(x1 y1)
x4 = car p1
```

##### **Y coordinate of point p4**

```
y4 = y3
y4 = cadr (x3 y3)
```



**Figure 33-7** Rectangle P1 P2 P3 P4



y4 = cadr p3

**Therefore, points p2 and p4 are:**

p2 = (list (car p3) (cadr p1))

p4 = (list (car p1) (cadr p3))

### Step 2: Writing the LISP program

Use any text editor to write the LISP program. The following file is a listing of the program for Example 3. **The line numbers at the right are for reference only; they are not a part of the program.**

```

;This program will draw a rectangle. User will      1
;be prompted to enter the two opposite corners      2
;                                                    3
(defun c:RECT1(/ p1 p2 p3 p4)                       4
  (graphscr)                                         5
  (setvar "cmdecho" 0)                               6
  (prompt "RECT1 command draws a rectangle")(terpri) 7
  (setq p1 (getpoint "Enter first corner"))(terpri)  8
  (setq p3 (getpoint "Enter opposite corner"))(terpri) 9
  (setq p2 (list (car p3) (cadr p1)))                10
  (setq p4 (list (car p1) (cadr p3)))                11
  (command "line" p1 p2 p3 p4 "c")                  12
  (setvar "cmdecho" 1)                               13
  (princ)                                             14
)                                                     15

```

### Explanation

Lines 1-3

The first three lines are comment lines that describe the function of the program. All comment lines that start with a semicolon are ignored when the program is loaded.

Line 4

**(defun c:RECT1(/ p1 p2 p3 p4)**

The **defun** function defines the function **RECT1**.

Line 5

**(graphscr)**

This function switches the text screen to the graphics screen, if the current screen happens to be a text screen. Otherwise, this function has no effect on the display screen.

Line 6

**(setvar "cmdecho" 0)**

The **setvar** function assigns the value 0 to the **CMDECHO** system variable, which turns the echo off. When **CMDECHO** is off, Command prompts are not displayed in the Command prompt area of the screen.

Line 7

**(prompt “RECT1 command draws a rectangle”)(terpri)**

The **prompt** function will display the information in double quotes (“RECT1 command draws a rectangle”). The function **terpri** causes a carriage return so that the next text is printed on a separate line.

Line 8

**(setq p1 (getpoint “Enter first corner”))(terpri)**

The **getpoint** function pauses for you to enter a point (the first corner of the rectangle), and the **setq** function assigns that value to variable p1.

Line 9

**(setq p3 (getpoint “Enter opposite corner”))(terpri)**

The **getpoint** function pauses for you to enter a point (the opposite corner of the rectangle), and the **setq** function assigns that value to variable p3.

Line 10

**(setq p2 (list (car p3) (cadr p1)))**

The **cadr** function extracts the Y coordinate of point p1, and the **car** function extracts the X coordinate of point p3. These two values form a list and the **setq** function assigns that value to variable p2.

Line 11

**(setq p4 (list (car p1) (cadr p3)))**

The **cadr** function extracts the Y coordinate of point p3, and the **car** function extracts the X coordinate of point p1. These two values form a list and the **setq** function assigns that value to variable p4.

Line 12

**(command “line” p1 p2 p3 p4 “c”)**

The **command** function uses the AutoCAD **LINE** command to draw lines between points p1, p2, p3, and p4. The c (close) joins the last point, p4, with the first point, p1.

Line 13

**(setvar “cmdecho” 1)**

The **setvar** function assigns a value of 1 to the AutoCAD system variable **CMDECHO**, which turns the echo on.

Line 14

**(princ)**

The **princ** function prints a blank on the screen. If this line is missing, AutoCAD will print the value of the last expression. This value does not affect the program in any way. However, it might be confusing at times. The **princ** function is used to prevent display of the last expression in the command prompt area.

Line 15

The closing parenthesis completes the definition of the function **RECT1** and ends the program.



#### Note

*In this program the rectangle is generated after you define the two corners of the rectangle. The rectangle is not dragged as you move the screen crosshairs to enter the second corner. However, the rectangle can be dragged by using the **getcorner** function, as shown in the following program listing:*

```
;This program will draw a rectangle with the
;drag mode on and using getcorner function
;
(defun c:RECT2(/ p1 p2 p3 p4)
  (graphscr)
  (setvar "cmdecho" 0)
  (prompt "RECT2 command draws a rectangle")(terpri)
  (setq p1 (getpoint "Enter first corner"))(terpri)
  (setq p3 (getcorner p1 "Enter opposite corner"))(terpri)
  (setq p2 (list (car p3) (cadr p1)))
  (setq p4 (list (car p1) (cadr p3)))
  (command "line" p1 p2 p3 p4 "c")
  (setvar "cmdecho" 1)
  (princ)
)
```

#### Step 3: Loading the LISP program

Save the file with the extension \*.lsp and then load the file using the **APPLOAD** command as described in Example 1.

## getangle and getorient FUNCTIONS

### getangle

The **getangle** function pauses for you to enter the angle. Then it returns the value of that angle in radians. The format of the **getangle** function is:

```
(getangle [point] [prompt])
      Where point ----- First point of the angle
           prompt ----- Any prompt that needs to be displayed on screen
```

#### Examples

```
(getangle)
(setq ang (getangle))
(setq ang (getangle pt1))----- pt1 is a predefined point
(setq ang (getangle "Enter taper angle"))
(setq ang (getangle pt1 "Enter second point of angle"))
```

The angle you enter is affected by the angle setting. The angle settings can be changed using the **UNITS** command or by changing the value of the **ANGBASE** and **ANGDIR** system variables. Following are the default settings for measuring an angle:

The angle is measured with respect to the positive *X* axis (3 o'clock position). The value of this setting is saved in the **ANGBASE** system variable.

The angle is positive if it is measured in the counterclockwise direction and negative if it is measured in the clockwise direction. The value of this setting is saved in the **ANGDIR** system variable.

If the angle has a default setting [Figure 33-8(a)], the **getangle** function will return 2.35619 radians for an angle of 135.

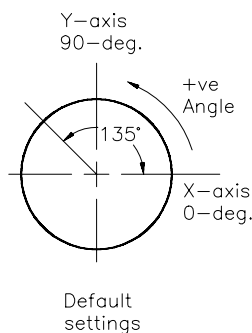
### Examples

(setq ang (getangle "Enter angle")) returns 2.35619 for an angle of 135-degree

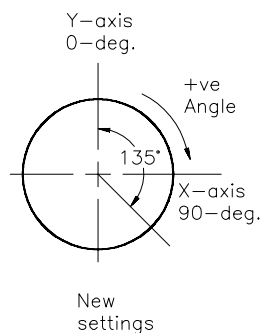
Figure 33-8(b) shows the new settings of the angle, where the *Y* axis is 0-degree and the angles measured clockwise are positive. The **getangle** function will return 3.92699 for an angle of 135-degree. The **getangle** function calculates the angle in the counterclockwise direction, **ignoring the direction set in the **ANGDIR** system variable**, with respect to the angle base as set in the system variable **ANGBASE** [Figure 33-9(b)].

### Examples

(setq ang (getangle "Enter angle")) returns 3.92699



*Figure 33-8(a)*



*Figure 33-8(b)*

## getorient

The **getorient** function pauses for you to enter the **angle**, then it returns the value of that angle in radians. The format of the **getorient** function is:

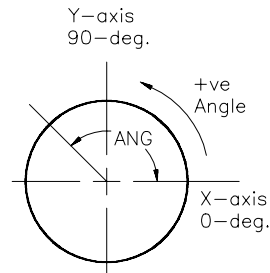


Figure 33-9(a)

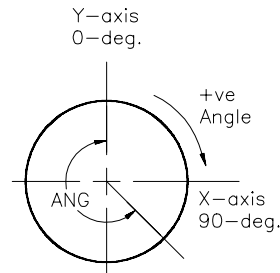


Figure 33-9(b)

(getorient [point] [prompt])

Where **point** ----- First point of the angle

**prompt** ----- Any prompt that needs to be displayed on the screen

### Examples

(getorient)

(setq ang (getorient))

(setq ang (getorient pt1))

(setq ang (getorient "Enter taper angle"))

(setq ang (getorient pt1 "Enter second point of angle"))

The **getorient** function is just like the **getangle** function. Both return the value of the angle in radians. However, the **getorient** function always measures the angle with a positive X axis (3 o'clock position) and in a counterclockwise direction. **It ignores the ANGBASE and ANGDIR settings.** If the settings have not been changed, as shown in Figure 33-10(a) (default settings for **ANGDIR** and **ANGBASE**), for an angle of 135-degree the **getorient** function will return 2.35619 radians. If the settings are changed, as shown in Figure 33-10(b), for an angle of 135-degree the **getorient** function will return 5.49778 radians. Although the settings have been changed where the angle is measured with the positive Y axis and in a clockwise direction, the **getorient** function ignores the new settings and measures the angle from positive X axis and in a counterclockwise direction.



### Note

For the **getangle** and **getorient** functions you can enter the angle by typing the angle or by selecting two points on the screen. If the assignment is (setq ang (getorient pt1)), where the first point *pt1* is already defined, you will be prompted to enter the second point. You can enter this point by selecting a point on the screen or by entering the coordinates of the second point.

180-degree is equal to  $\pi$  (3.14159) radians. To calculate an angle in radians, use the following relation:

$$\text{Angle in radians} = (\pi \times \text{angle})/180$$

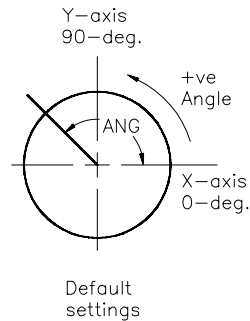


Figure 33-10(a)

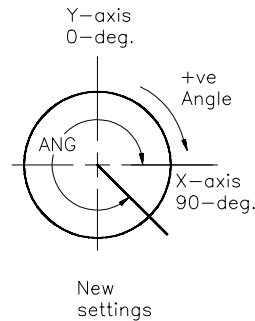


Figure 33-10(b)

## Getint, getreal, getstring, AND getvar FUNCTIONS

### getint

The **getint** function pauses for you to enter an integer. The function always returns an integer. If you enter a real number, it will inform you that it requires an integer value. The format of the **getint** function is:

**(getint [prompt])**

Where **prompt** ----- Optional prompt that you want to display on screen

#### Examples

```
(getint)
(setq numx (getint))
(setq numx (getint "Enter number of rows: "))
(setq numx (getint "\n Enter number of rows: "))
```

### getreal

The **getreal** function pauses for you to enter a real number and it returns a real number. If you happen to enter an integer, it returns a real number. The format of the **getreal** function is:

**(getreal [prompt])**

Where **prompt** ----- Optional prompt that is displayed on screen

#### Examples

```
(getreal)
(setq realnumx (getreal))
(setq realnumx (getreal "Enter num1: "))
(setq realnumx (getreal "\n Enter num2: "))
```

### getstring

The **getstring** function pauses for you to enter a string value, and it always returns a string, even if the string you enter contains numbers only. The format of the **getstring** function is:

**(getstring [cr] [prompt])**

Where **cr** ----- May be T or Nil. The default value, if not specified, is Nil. If the value is T, the input string can have blanks.

Also note that the string must be terminated by ENTER.

**prompt** ----- Optional prompt that is displayed on screen

**Examples**

```
(getstring)
```

```
(setq answer (getstring))
```

```
(setq answer (getstring "Enter Y for yes, N for no: ))
```

```
(setq answer (getstring "\n Enter Y for yes, N for no: ))
```

**Note**

*The maximum length of the string is 256 characters. If the string exceeds 256 characters, the exceeding characters are ignored.*

**getvar**

The **getvar** function lets you retrieve the value of an AutoCAD system variable. The format of the **getvar** function is:

**(getvar "variable")**

Where **variable** ----- AutoCAD system variable name

**Examples**

```
(getvar)
```

```
(getvar "dimcen") returns 0.09
```

```
(getvar "ltscale") returns 1.0
```

```
(getvar "limmax") returns 12.00,9.00
```

```
(getvar "limmin") returns 0.00,0.00
```

**Note**

*The system variable name should always be enclosed in double quotes.*

*You can retrieve only one variable value in one assignment. To retrieve the values of several system variables, use a separate assignment for each variable.*

**polar AND sqrt FUNCTIONS****Polar**

The **polar** function defines a point at a given angle and distance from the given point (Figure 33-11). The angle is expressed in radians, measured positive in the counterclockwise direction (assuming default settings for **ANGBASE** and **ANGDIR**). The format of the **polar** function is:

**(polar point angle distance)**

Where **point** ----- Reference point  
**angle** ----- Angle the point makes with the referenced point  
**distance** ----- Distance of the point from the referenced point

**Examples**

```
(polar pt1 ang dis)
(setq pt2 (polar pt1 ang dis))
(setq pt2 (polar '(2.0 3.25) ang dis))
```

**sqrt**

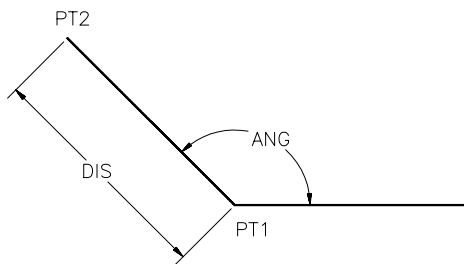
The **sqrt** function calculates the square root of a number, and the value this function returns is always a real number, Figure 33-12. The format of the **sqrt** function is:

**(sqrt number)**

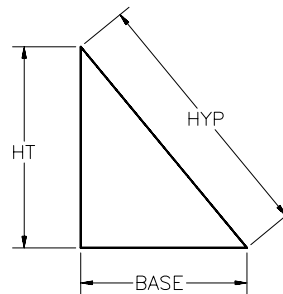
Where **number** ----- Number you want to find the square root of  
 (real or integer)

**Examples**

```
(sqrt 144)           returns 12.0
(sqrt 144.0)         returns 12.0
(setq x (sqrt 57.25)) returns 7.566373
(setq x (sqrt (* 25 36.5))) returns 30.207615
(setq x (sqrt (/ 7.5 (cos 0.75)))) returns 3.2016035
(setq hyp (sqrt (+ (* base base) (* ht ht))))
```



**Figure 33-11** Using the **polar** function to define a point



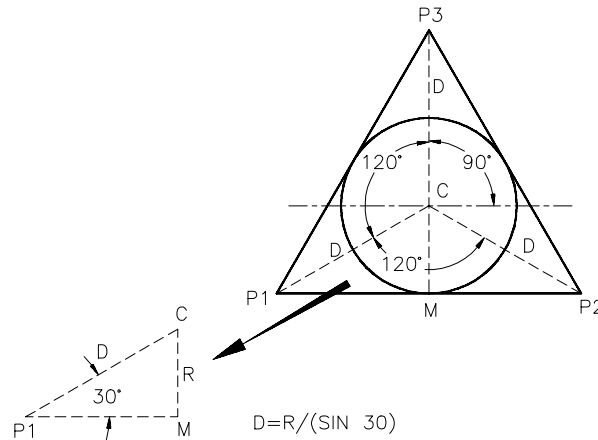
```
(setq hyp (sqrt (+(*base base) (*ht ht))))
```

**Figure 33-12** Application of the **sqrt** function

**Example 4**

Write an AutoLISP program that will draw an equilateral triangle outside a circle (Figure 33-13). The sides of the triangle are tangent to the circle. The program should prompt you to enter the radius and the center point of the circle.





**Figure 33-13** Equilateral triangle outside a circle

### Step 1: Writing the LISP program

Use any text editor to write the LISP program. The following file is the listing of the AutoLISP program for Example 4.

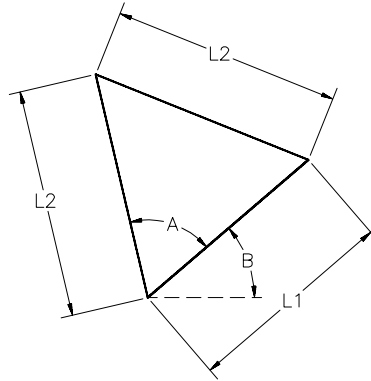
```
;This program will draw a triangle outside
;the circle with the lines tangent to circle
;
(defun dtr (a)
  (* a (/ pi 180.0))
)
(defun c:trgcir(/ r c d p1 p2 p3)
  (setvar "cmdecho" 0)
  (graphscr)
  (setq r(getdist "\n Enter circle radius: "))
  (setq c(getpoint "\n Enter center of circle: "))
  (setq d(/ r (sin(dtr 30))))
  (setq p1(polar c (dtr 210) d))
  (setq p2(polar c (dtr 330) d))
  (setq p3(polar c (dtr 90) d))
  (command "circle" c r)
  (command "line" p1 p2 p3 "c")
  (setvar "cmdecho" 1)
  (princ)
)
```

### Step 2: Loading the LISP program

Save the LISP file and then load it using the **APPLOAD** command as described in Example 1.

**Exercise 3***General*

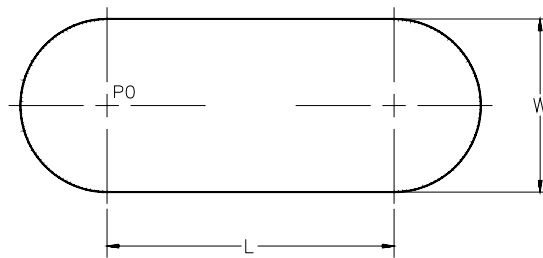
Write an AutoLISP program that will draw an isosceles triangle P1,P2,P3. The base of the triangle (P1,P2) makes an angle B with the positive X axis (Figure 33-14). The program should prompt you to enter the starting point, P1, length L1, and angles A and B.



**Figure 33-14** Isosceles triangle at an angle

**Exercise 4***General*

Write a program that will draw a slot with centerlines. The program should prompt you to enter slot length, slot width, and the layer name for the centerlines (Figure 33-15).



**Figure 33-15** Slot of length  $L$  and width  $W$

## itoa, rtos, strcase, AND prompt FUNCTIONS

### itoa

The **itoa** function changes an integer into a string and returns the integer as a string. The format of the **itoa** function is:

**(itoa number)**

Where **Number** ----- The integer number that you want to convert into a string

#### Examples

```
(itoa 89)           returns "89"
(itoa -356)         returns "-356"

(setq intnum 7)
(itoa intnum)       returns "7"

(setq intnum 345)
(setq intstrg (itoa intnum)) returns "345"
```

### rtos

The **rtos** function changes a real number into a string and the function returns the real number as a string. The format of the **rtos** function is:

**(rtos realnum)**

Where **Realnum** ----- The real number that you want to convert into a string

#### Examples

```
(rtos 50.6) returns "50.6"
(rtos -30.0) returns "-30.0"
(setq realstrg (rtos 5.25)) returns "5.25"

(setq realnum 75.25)
(setq realstrg (rtos realnum)) returns "75.25"
```

The **rtos** function can also include mode and precision. If mode and precision are not provided, then the current AutoCAD settings are used. The format of the **rtos** function with mode and precision is:

**(rtos realnum [mode [precision]])**

Where **realnum** ----- Real number  
**mode** ----- Unit mode, like decimal, scientific  
**precision** ----- Number of decimal places or denominator of fractional units

## strcase

The **strcase** function converts the characters of a string into uppercase or lowercase. The format of the **strcase** function is:

**(strcase string [true])**

Where **String** ----- String that needs to be converted to uppercase or lowercase

**True** ----- If it is not nil, all characters are converted to lowercase

The **true** is optional. If it is missing or if the value of **true** is nil, the string is converted to uppercase. If the value of true is not nil, the string is converted to lowercase.

### Examples

(strcase "Welcome Home") returns "WELCOME HOME"

(setq t 0)

(strcase "Welcome Home" t) returns "welcome home"

(strcase "Welcome Home" a) returns "WELCOME HOME"

(setq answer (strcase (getstring "Enter Yes or No: ")))

## prompt

The **prompt** function is used to display a message on the screen in the Command prompt area. The contents of the message must be enclosed in double quotes. The format of the **prompt** function is:

**(prompt message)**

Where **message** ----- Message that you want to display on the screen

### Examples

(prompt "Enter circle diameter: ")

(setq d (getdist (prompt "Enter circle diameter: ")))

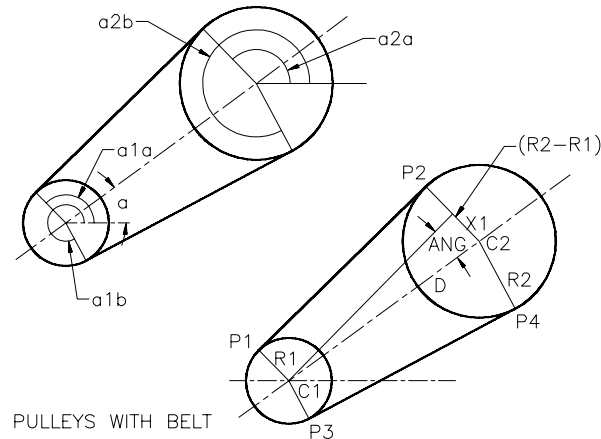


### Note

*On a two-screen system, the prompt function displays the message on both screens.*

## Example 5

Write a program that will draw two circles of radii  $r_1$  and  $r_2$ , representing two pulleys that are separated by a distance  $d$ . The line joining the centers of the two circles makes an angle  $\alpha$  with the  $X$  axis, as shown in Figure 33-16.



**Figure 33-16** Two circles with tangent lines

### Step 1: Understanding the program algorithm

### Input

Radius of small circle - r1  
Radius of large circle - r2  
Distance between circles - d  
Angle of center line - a  
Center of small circle - c1

### Output

Small circle of radius -  $r_1$   
Large circle of radius -  $r_2$   
Lines tangent to circles

## Process

1. Calculate distance  $x1$ ,  $x2$ .
2. Calculate angle  $ang$ .
3. Locate point  $c2$  with respect to point  $c1$ .
4. Locate points  $p1$ ,  $p2$ ,  $p3$ ,  $p4$ .
5. Draw small circle with radius  $r1$  and center  $c1$ .
6. Draw large circle with radius  $r2$  and center  $c2$ .
7. Draw lines  $p1$  to  $p2$  and  $p3$  to  $p4$ .

## Calculations

$$\mathbf{x}_1 = \mathbf{r}_2 - \mathbf{r}_1$$

$$x2 = \text{SQRT} [ d^2 - (r2 - r1)^2 ]$$

$$\tan \text{ ang} = x1 / x2$$

$$\text{ang} = \text{atan} (x1 / x2)$$

$$a1a = 90 + a + \text{ang}$$

$$a1b = 270 + a - \text{ang}$$

$$a2a = 90 + a + \text{ang}$$

$$a2b = 270 + a - \text{ang}$$

### Step 2: Writing the LISP program

The following file is a listing of the AutoLISP program for Example 5. **The line numbers on the right are not a part of the file. These numbers are for reference only.** Use any text editor to write down the following LISP file.

```

;This program draws a tangent (belt) over two          1
;pulleys that are separated by a given distance.      2
;                                                      3
;This function changes degrees into radians           4
(defun dtr (a)                                         5
  (* a (/ pi 180.0))                                  6
)                                                       7
;End of dtr function                                  8
;The belt function draws lines that are tangent to circles 9
(defun c:belt(/ r1 r2 d a c1 x1 x2 c2 p1 p2 p3 p4)    10
  (setvar "cmdecho" 0)                                11
  (graphscr)                                           12
  (setq r1(getdist "\n Enter radius of small pulley: ")) 13
  (setq r2(getdist "\n Enter radius of larger pulley: ")) 14
  (setq d(getdist "\n Enter distance between pulleys: ")) 15
  (setq a(getangle "\n Enter angle of pulleys: "))      16
  (setq c1(getpoint "\n Enter center of small pulley: ")) 17
  (setq x1 (- r2 r1))                                  18
  (setq x2 (sqrt (- (* d d) (* (- r2 r1) (- r2 r1))))) 19
  (setq ang (atan (/ x1 x2)))                           20
  (setq c2 (polar c1 a d))                              21
  (setq p1 (polar c1 (+ ang a (dtr 90)) r1))           22
  (setq p3 (polar c1 (- (+ a (dtr 270)) ang) r1))      23
  (setq p2 (polar c2 (+ ang a (dtr 90)) r2))           24
  (setq p4 (polar c2 (- (+ a (dtr 270)) ang) r2))      25
  ;                                                    26
;The following lines draw circles and lines           27
(command "circle" c1 p3)                               28
(command "circle" c2 p2)                               29
(command "line" p1 p2 "")                              30
(command "line" p3 p4 "")                              31
(setvar "cmdecho" 1)                                   32
(princ)                                                 33

```

**Explanation**

Line 5

**(defun dtr (a)**

In this line, the **defun** function defines a function, **dtr (a)**, that converts degrees into radians.

Line 6

**(\* a (/ pi 180.0))**

**(/ pi 180)** divides the value of **pi** by 180, and the product is then multiplied by angle **a** (180-degree is equal to **pi** radians).

Line 10

**(defun c:belt(/ r1 r2 d a c1 x1 x2 c2 p1 p2 p3 p4)**

In this line, the function **defun** defines a function, **c:belt**, that generates two circles with tangent lines.

Line 18

**(setq x1 (- r2 r1))**

In this line, the function **setq** assigns a value of **r2 - r1** to variable **x1**.

Line 19

**(setq x2 (sqrt (- (\* d d) (\* (- r2 r1) (- r2 r1))))))**

In this line, **(- r2 r1)** subtracts the value of **r1** from **r2** and **(\* (- r2 r1) (- r2 r1))** calculates the square of **(- r2 r1)**. **(sqrt (- (\* d d) (\* (- r2 r1) (- r2 r1))))** calculates the square root of the difference, and **setq x2** assigns the product of this expression to variable **x2**.

Line 20

**(setq ang (atan (/ x1 x2)))**

In this line, **(atan (/ x1 x2))** calculates the arctangent of the product of **(/ x1 x2)**. The function **setq ang** assigns the value of the angle in radians to variable **ang**.

Line 21

**(setq c2 (polar c1 a d))**

In this line, **(polar c1 a d)** uses the **polar** function to locate point **c2** with respect to **c1** at a distance **d** and making an angle **a** with the positive **X** axis.

Line 22

**(setq p1 (polar c1 (+ ang a (dtr 90)) r1))**

In this line, **(polar c1 (+ ang a (dtr 90)) r1)** locates point **p1** with respect to **c1** at a distance **r1** and making an angle **(+ ang a (dtr 90))** with the positive **X** axis.

Line 28

**(command "circle" c1 p3)**

In this line, the **Command** function uses the **CIRCLE** command to draw a circle with center **c1** and a radius defined by the point **p3**.

Line 30

(command "line" p1 p2 "")

In this line, the **Command** function uses the **LINE** command to draw a line from p1 to p2. The pair of double quotes (") at the end introduces a Return, which terminates the **LINE** command.

### Step 3: Loading the LISP program

Save the program and then load the program as described in the Example 1.

## Exercise 5

General

Write an AutoLISP program that will draw two lines tangent to two circles, as shown in Figure 33-17. The program should prompt you to enter the circle diameters and the center distance between the circles.

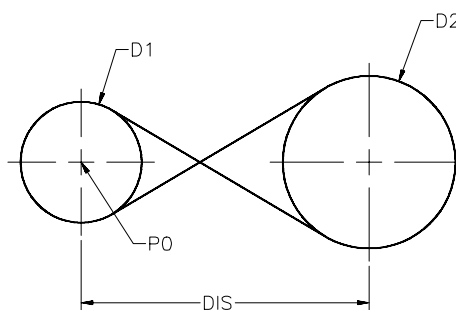


Figure 33-17 Circles with two tangent lines

## FLOWCHARTS

A **flowchart** is a graphical representation of an algorithm. It is used to analyze a problem systematically. It gives a better understanding of the problem, especially if the problem involves some conditional statements. It consists of standard symbols that represent a certain function in the program. For example, a rectangle is used to represent a process that takes place when the program is executed. The blocks are connected by lines indicating the sequence of operations. Figure 33-18 gives the standard symbols that can be used in a flowchart.

## CONDITIONAL FUNCTIONS

The relational functions discussed earlier in the chapter establish a relationship between two atoms. For example, ( $< x y$ ) describes a test condition for an operation. To use such functions in a meaningful way a conditional function is required. For example, (if ( $< x y$ ) (setq z (- y x)) (setq z (- x y))) describes the action to be taken when the condition is true (T) and when it is false (nil). If the condition is true, then  $z = y - x$ . If the condition is not true, then  $z = x - y$ . Therefore, conditional functions are very important for any programming language, including AutoLISP.



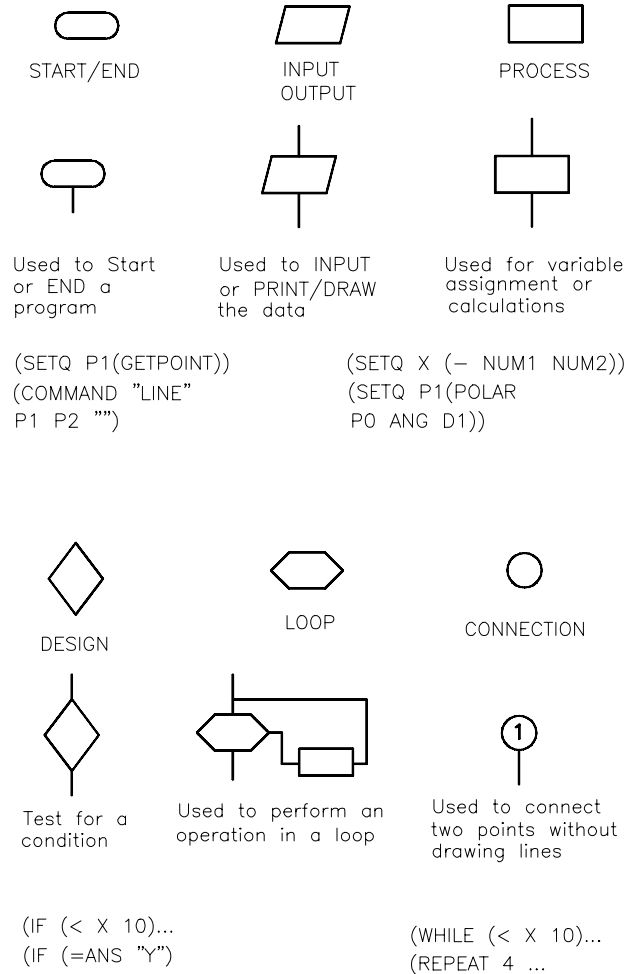


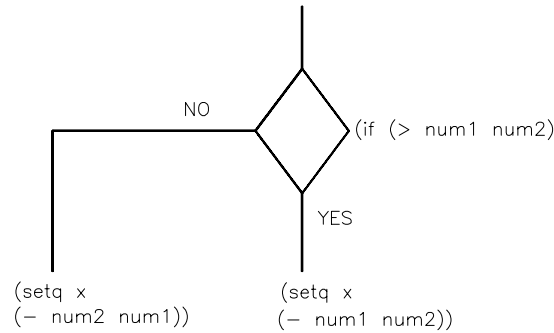
Figure 33-18 Flowchart symbols

**if**

The **if** function (Figure 33-19) evaluates the first expression (then) if the specified condition returns "true," it evaluates the second expression (else) if the specified condition returns "nil." The format of the **if** function is:

**(if condition then [else])**

Where **condition** ----- Specified conditional statement  
**then** ----- Expression evaluated if the condition returns T  
**else** ----- Expression evaluated if the condition returns nil



**Figure 33-19** *if function*

### Examples

```

(if (= 7 7) ("true"))           returns "true"
(if (= 5 7) ("true") ("false")) returns "false"
(setq ans "yes")
(if (= ans "yes") ("Yes") ("No")) returns "Yes"
(setq num1 8)
(setq num2 10)
(if (> num1 num2)
  (setq x (- num1 num2))
  (setq x (- num2 num1))
)                                returns 2
  
```

## Example 6

Write an AutoLISP program that will subtract a smaller number from a larger number. The program should also prompt you to enter two numbers.

### Step 1: Understanding program algorithm and flowchart

#### Input

Number (num1)  
Number (num2)

#### Output

x = num1 - num2  
or  
x = num2 - num1

#### Process

If num1 > num2 then x = num1 - num2  
If num1 < num2 then x = num2 - num1

The flowchart in Figure 33-20 describes the process involved in writing the program using standard flowchart symbols.

### Step 2: Writing the LISP program

Use any text editor to write the LISP file. The following file is a listing of the program for

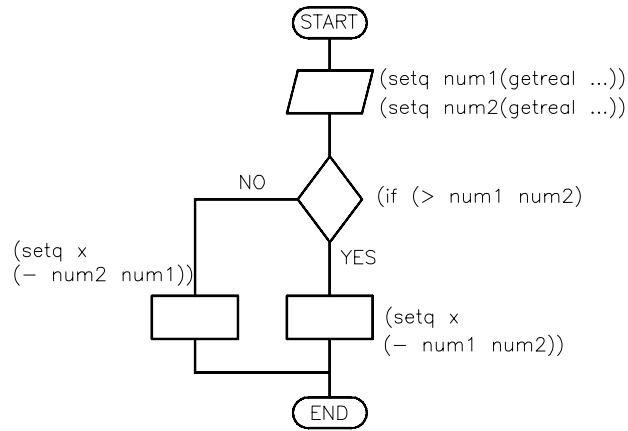


Figure 33-20 Flowchart for Example 6

Example 6. The line numbers are not a part of the file; they are for reference only.

```

;This program subtracts smaller number          1
;from larger number                             2
;                                                 3
(defun c:subnum( )                             4
  (setvar "cmdecho" 0)                         5
  (setq num1 (getreal "\n Enter first number: ")) 6
  (setq num2 (getreal "\n Enter second number: ")) 7
  (if (> num1 num2)                             8
    (setq x (- num1 num2))                      9
    (setq x (- num2 num1)))                    10
  )                                           11
  (princ x)                                   12
  (setvar "cmdecho" 1)                       13
  (princ)                                    14
)                                           15

```

### Explanation

Line 8

**(if (> num1 num2))**

In this line, the **if** function evaluates the test expression **(> num1 num2)**. If the condition is true, it returns **T**. If the condition is not true, it returns **nil**.

Line 9

**(setq x (- num1 num2))**

This expression is evaluated if the test expression **(if (> num1 num2))** returns **T**. The value of variable **num2** is subtracted from **num1**, and the resulting value is assigned to variable **x**.

Line 10

```
(setq x (- num2 num1))
```

This expression is evaluated if the test expression (**if** (**>** num1 num2) returns nil. The value of variable num1 is subtracted from num2, and the resulting value is assigned to variable x.

Line 11

```
)
```

The closing parenthesis completes the definition of the **if** function.

### Step 3: Loading the LISP program

Save the file and then load the file using the **APPLOAD** command as described in Example 1.

## Example 7

Write an AutoLISP program that will enable you to multiply or divide two numbers (Figure 33-21). The program should prompt you to enter the choice of multiplication or division. The program should also display an appropriate message if you do not enter the right choice.

### Step 1: Making the flowchart

A flowchart can be designed as shown in Figure 33-21.

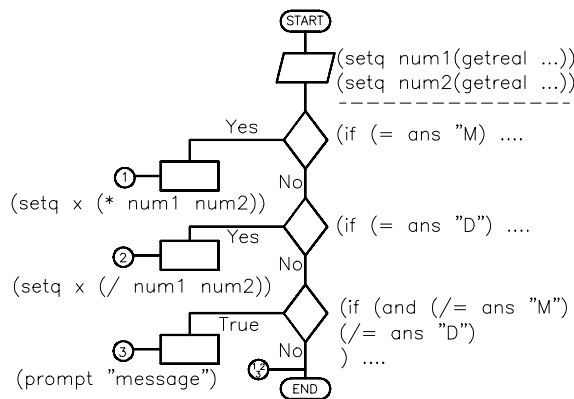


Figure 33-21 Flow diagram for Example 7

### Step 2: Writing the lisp file

The following file is a listing of the AutoLISP program for Example 7:

```

;This program multiplies or divides two given numbers
(defun c:mdnum()
  (setvar "cmdecho" 0)
  (setq num1 (getreal "\n Enter first number: "))
  (setq num2 (getreal "\n Enter second number: "))
  (prompt "Do you want to multiply or divide. Enter M or D: ")

```

```

    (setq ans (strcase (getstring)))
    (if (= ans "M")
        (setq x (* num1 num2))
    )
    (if (= ans "D")
        (setq x (/ num1 num2))
    )
    (if (and (/= ans "D")(/= ans "M"))
        (prompt "Sorry! Wrong entry, Try again")
        (princ x)
    )
    (setvar "cmdecho" 1)
    (princ)
)

```

### Step 3: Loading the lisp file

Save the file and then load the file using the **APPLOAD** command as described in Example 1.

### progn

The **progn** function can be used with the **if** function to evaluate several expressions. The format of the **progn** function is:

**(progn expression expression . . .)**

The **if** function evaluates only one expression if the test condition returns "true." The **progn** function can be used in conjunction with the **if** function to evaluate several expressions. Here is an example to demonstrate the use of the **progn** function with the **if** function.

### Example

```

(defun c:IFPRGN()
    (setq p1(getint "Enter the integer"))
    (if (>= p1 5)
        (progn (command "line" "2,2" "3,3" "")
                (command "rec" "3,3" "6,6") )

        (progn (command "circle" "3,3" 1)
                (command "line" "3,3" "5,5" ""))
    )
    ; End of program
)

```

**while**

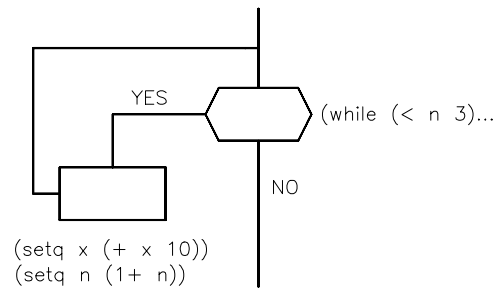
The **while** function (Figure 33-22) evaluates a test condition. If the condition is true (expression does not return nil), the operations that follow the while statement are repeated until the test expression returns nil. The format of the **while** function is:

**(while testexpression operations)**

Where **Testexpression**

Expression that tests a condition

**Operations** ----- Operations to be performed until the test expression returns nil



**Figure 33-22** While function

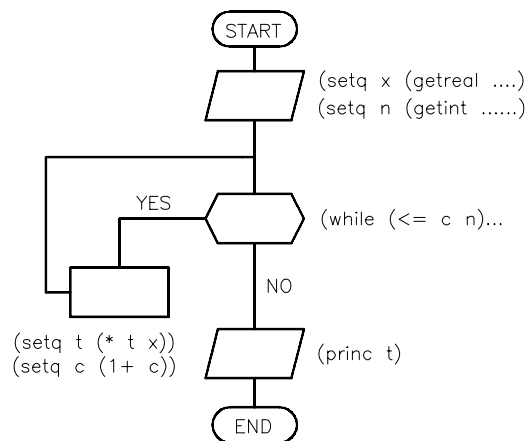
**Example**

```

(while (= ans "yes")
  (setq x (+ x 1))
  (setq ans (getstring "Enter yes or no: "))
)
(while (< n 3)
  (setq x (+ x 10))
  (setq n (1+ n))
)
  
```

**Example 8**

Write an AutoLISP program that will find the nth power of a given number. The power is an integer. The program should prompt you to enter the number and the nth power (Figure 33-23).



**Figure 33-23** Flowchart for Example 8

**Step 1: Understanding the program algorithm and flowchart****Input**

Number x  
nth power n

**Output**

product  $x^n$

**Process**

1. Set the value of  $t = 1$  and  $c = 1$ .
2. Multiply  $t * x$  and assign that value to the variable  $t$ .
3. Repeat the process until the counter  $c$  is less than or equal to  $n$ .

**Step 2: Writing the lisp program**

The following file is a listing of the AutoLISP program for Example 8.

```
;This program calculates the nth
;power of a given number
(defun c:npower()
  (setvar "cmdecho" 0)
  (setq x(getreal "\n Enter a number: "))
  (setq n(getint "\n Enter Nth power-integer number: "))
  (setq t 1) (setq c 1)
  (while (<= c n)
    (setq t (* t x))
    (setq c (1+ c))
  )
  (princ t)
  (setvar "cmdecho" 1)
  (princ)
)
```

**Step 3: Loading the LISP file**

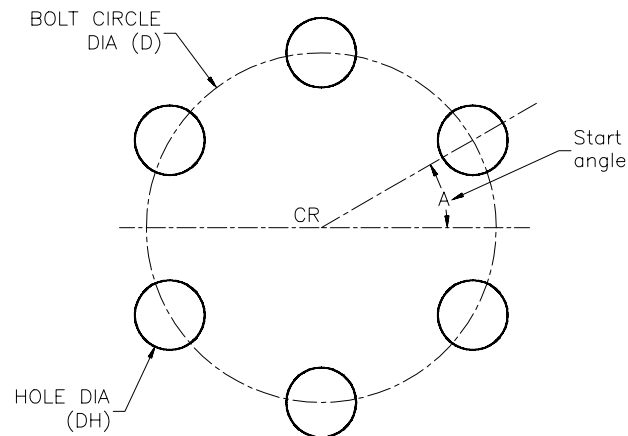
Save the file and then load the file with the help of the **APPLOAD** command.

**Example 9**

Write an AutoLISP program that will generate the holes of a bolt circle (Figure 33-24). The program should prompt you to enter the center point of the bolt circle, the bolt circle diameter, the bolt circle hole diameter, the number of holes, and the start angle of the bolt circles.

**Step 1: Writing the LISP program**

```
;This program generates the bolt circles
;
(defun c:bc1( )
  (graphscr)
```



**Figure 33-24** Bolt circle with six holes

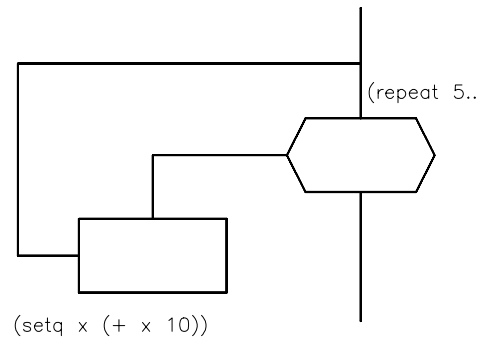
```
(setvar "cmdecho" 0)
(setq cr(getpoint "\n Enter center of Bolt-Circle: "))
(setq d(getdist "\n Dia of Bolt-Circle: "))
(setq n(getint "\n Number of holes in Bolt-Circle: "))
(setq a(getangle "\n Enter start angle: "))
(setq dh(getdist "\n Enter diameter of hole: "))
(setq inc(/ (* 2 pi) n))
(setq ang 0)
(setq r (/ dh 2))
(while (< ang (* 2 pi))
  (setq p1 (polar cr (+ a inc) (/ d 2)))
  (command "circle" p1 r)
  (setq a (+ a inc))
  (setq ang (+ ang inc))
)
(setvar "cmdecho" 1)
(princ)
)
```

### Step 2: Loading the LISP file

Save the file and then load it using the **APPLOAD** command.

### repeat

The **repeat** function evaluates the expressions **n** number of times as specified in the **repeat** function (Figure 33-25). The variable **n** must be an integer. The format of the **repeat** function is:



**Figure 33-25** Repeat function



**repeat n**

Where ----- n is an integer that defines the number of times the expressions are to be evaluated

**Example**

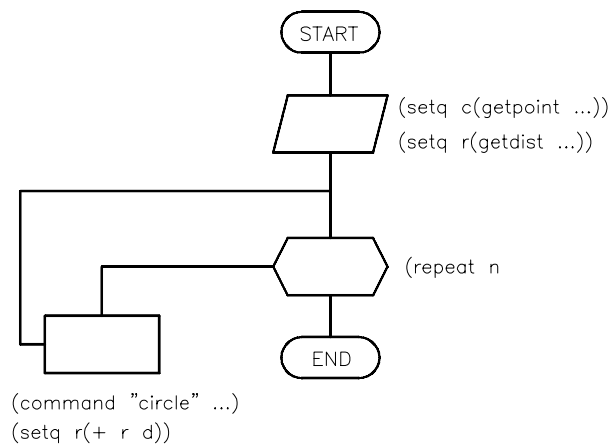
```
(repeat 5
  (setq x (+ x 10))
)
```

**Note**

*AutoCAD allows you to load the specified AutoLISP programs automatically, each time you start AutoCAD. For example, if you are working on a project and you have loaded an AutoLISP program, the program will autoload if you start another drawing. You can enable this feature by adding the name of the file to the Startup Suite of the **Load/Unload Application** dialog box. For details, see the Load/Unload Application discussed earlier in this chapter.*

**Example 10**

Write an AutoLISP program that will generate a given number of concentric circles. The program should prompt you to enter the center point of the circles, the start radius, and the radius increment (Figure 33-26).



**Figure 33-26 Flowchart for Example 10**

The following file is a listing of the AutoLISP program for Example 10.

```
;This program uses the repeat function to draw
;a given number of concentric circles.
(defun c:concir( )
  (graphscr)
  (setvar "cmdecho" 0)
```

```
(setq c (getpoint "\n Enter center point of circles: "))
(setq n (getint "\n Enter number of circles: "))
(setq r (getdist "\n Enter radius of first circle: "))
(setq d (getdist "\n Enter radius increment: "))
(repeat n
  (command "circle" c r)
  (setq r (+ r d))
)
(setvar "cmdecho" 1)
(princ)
)
```

### Self-Evaluation Test

Answer the following questions and then match your answers to the answers given at the end of the chapter.

1. LISP is a leading programming language for \_\_\_\_\_ .
2. The AutoLISP interpreter is embedded within the \_\_\_\_\_ software package.
3. The function \_\_\_\_\_ checks whether the two atoms are not equal.
4. AutoLISP contains some \_\_\_\_\_ functions.
5. If some built-in functions are used with **setq** then there is every possibility that reserved functions will be \_\_\_\_\_ .
6. When using the **LOAD** function, the AutoLISP file name and the optional path name must be enclosed in \_\_\_\_\_ .
7. The command \_\_\_\_\_ is used to load a LISP file.
8. The **strcase** function converts the characters of a string into \_\_\_\_\_ .
9. The **Progn** function can be used in conjunction with the \_\_\_\_\_ .
10. The **setq** function is used to assign a value to \_\_\_\_\_ .

### Review Questions

1. Fill in the blanks:

Command: (+ 2 30 5 50)

returns \_\_\_\_\_

Command: (+ 2 30 4 55.0)	returns _____
(- 20 40)	returns _____
(- 30.0 40.0)	returns _____
(* 72 5 3 2.0)	returns _____
(* 7 -5.5)	returns _____
(/ 299 -5)	returns _____
(/ -200 -9.0)	returns _____
(1- 99)	returns _____
(1- -18.5)	returns _____
(abs -90)	returns _____
(abs -27.5)	returns _____
(sin pi)	returns _____
(sin 1.5)	returns _____
(cos pi)	returns _____
(cos 1.2)	returns _____
(atan 1.1 0.0)	returns _____ radians
(atan -0.4 0.0)	returns _____ radians
(angtos 1.5708 0 5)	returns _____
(angtos -1.5708 0 3)	returns _____
(< "x" "y")	returns _____
(>= 80 90 79)	returns _____

2. The \_\_\_\_\_ function pauses to enable you to enter the X, Y coordinates or X, Y, Z coordinates of a point.
3. The \_\_\_\_\_ function is used to execute standard AutoCAD commands from within an AutoLISP program.
4. In an AutoLISP expression, the AutoCAD command name and the command options have to be enclosed in double quotation marks. (T/F).
5. The **getdist** function pauses for you to enter a \_\_\_\_\_ and it then returns the distance as a real number.
6. The \_\_\_\_\_ function assigns a value to an AutoCAD system variable. The name of the system variable must be enclosed in \_\_\_\_\_.
7. The **cadr** function performs two operations, \_\_\_\_\_ and \_\_\_\_\_, to return the second element of the list.
8. The \_\_\_\_\_ function prints a new line on the screen just as \n.

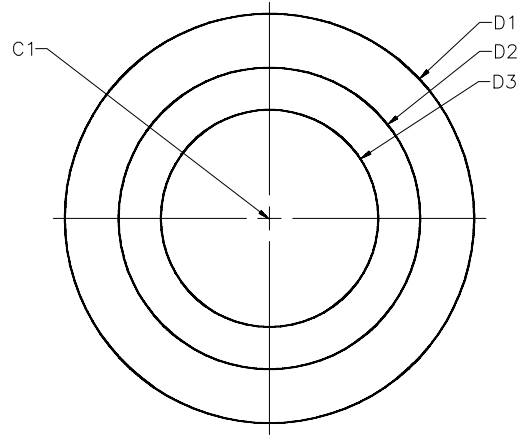
9. The \_\_\_\_\_ function pauses for you to enter the angle, then it returns the value of that angle in radians.
10. The \_\_\_\_\_ function always measures the angle with a positive X axis and in a counterclockwise direction.
11. The \_\_\_\_\_ function pauses for you to enter an integer. The function always returns an integer, even if the number that you enter is a real number.
12. The \_\_\_\_\_ function lets you retrieve the value of an AutoCAD system variable.
13. The \_\_\_\_\_ function defines a point at a given angle and distance from the given point.
14. The \_\_\_\_\_ function calculates the square root of a number and the value this function returns is always a real number.
15. The \_\_\_\_\_ function changes a real number into a string and the function returns the real number as a string.
16. The **if** function evaluates the test expression (**> num1 num2**). If the condition is true, it returns \_\_\_\_\_; if the condition is not true, it returns \_\_\_\_\_.
17. The \_\_\_\_\_ function can be used with the **if** function to evaluate several expressions.
18. The **while** function evaluates the test condition. If the condition is true (expression does not return nil) the operations that follow the while statement are \_\_\_\_\_ until the test expression returns \_\_\_\_\_.
19. The **repeat** function evaluates the expressions n number of times as specified in the **repeat** function. The variable n must be a real number. (T/F).

## Exercises

### Exercise 6

*General*

Write an AutoLISP program that will draw three concentric circles with center C1 and diameters D1, D2, D3 (Figure 33-27). The program should prompt you to enter the coordinates of center point C1 and the circle diameters D1, D2, D3.

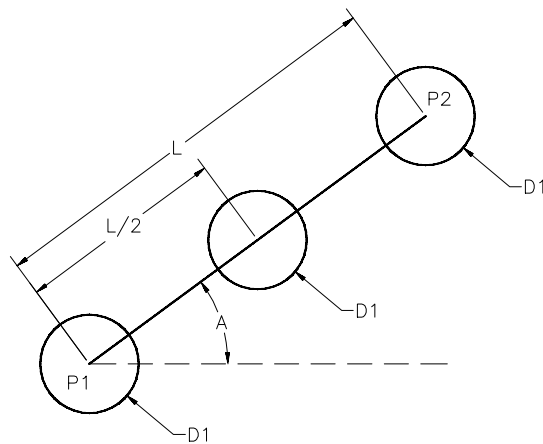


**Figure 33-27** Three concentric circles with diameters  $D1$ ,  $D2$ ,  $D3$

### Exercise 7

General

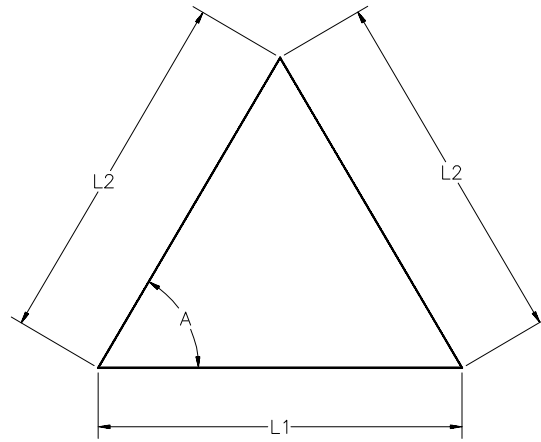
Write an AutoLISP program that will draw a line from point  $P1$  to point  $P2$  (Figure 33-28). Line  $P1, P2$  makes an angle  $A$  with the positive  $X$  axis. Distance between the points  $P1$  and  $P2$  is  $L$ . The diameter of the circles is  $D1$  ( $D1 = L/4$ ).



**Figure 33-28** Circles and line making an angle  $A$  with  $X$ -axis

**Exercise 8***General*

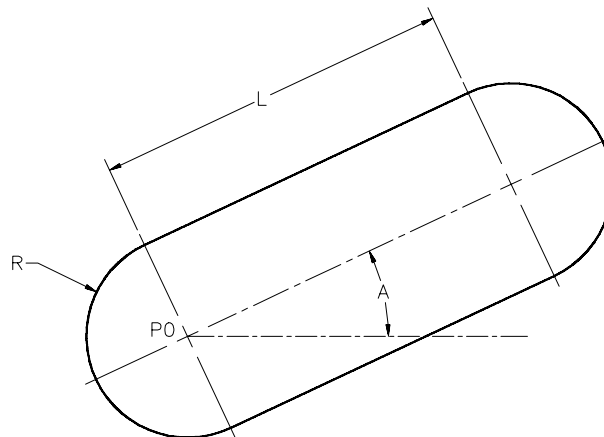
Write an AutoLISP program that will draw an isosceles triangle P1, P2, P3 (Figure 33-29). The program should prompt you to enter the starting point P1, length L1, and the included angle A.



**Figure 33-29** Isosceles triangle

**Exercise 9***General*

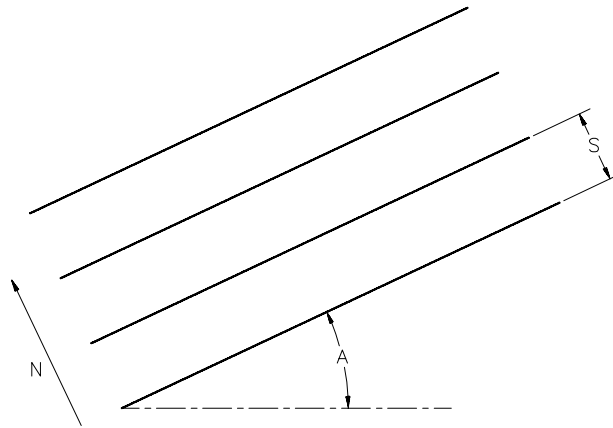
Write a program that will draw a slot with center lines. The program should prompt you to enter slot length, slot width, and the layer name for center lines (Figure 33-30).



**Figure 33-30** Slot of length L and radius R

**Exercise 10***General*

Write an AutoLISP program that will draw a line and then generate a given number of lines (N), parallel to the first line (Figure 33-31).



**Figure 33-31** *N* number of lines offset at a distance *S*

**Answers to the Self-Evaluation Test**

1. Artificial Intelligence, 2. AutoCAD, 3. /=, 4. Built- in, 5. Redefined, 6. double quotes, 7. APPLOAD, 8. Uppercase or Lowercase, 9. If, 10. Variable.

