ATP164
First Step in AutoLISP Programming

Segment 1

Date: January 8, 2007

Instructor: Mohmed Zuber Shaikh
Level: Beginning
Category: AutoLISP

Web: www.AUGI.com

Introduction

PC based CAD systems are replacing the drawing board concept of drafting and have changed the thinking of designers, engineers and draftsman's towards design and drafting problem. AutoCAD is among the most popular CAD softwares available. AutoCAD with its wide range of command tools, is capable of navigating you through the most complicated drafting tasks with considerable accuracy. New release announcements for AutoCAD reflect the commitment of Autodesk to creating more efficient environment for users. Fortunately, through different programming interfaces even users can enhance their own version of AutoCAD. AutoLISP is among the best programming interfaces available.

Ease of customization is a key factor in the popularity of AutoCAD, and AutoLISP programming has become a significant customization option in AutoCAD. LISP is the chosen language for research and development of artificial intelligence and expert systems. With its simple syntax, LISP is among the easiest of all programming languages to learn. AutoCAD has a built-in LISP interpreter to response the programming code entered at command line or the code loaded from external files. As AutoCAD reads LISP code directly, a separate compiler is not required.

An AutoCAD user who is in very good touch with commands can easily identify the scope of further development of some commands to enhance the productivity of AutoCAD. With proficiency in AutoLISP one can easily modify the predefined commands to suit his/her needs or can write new functions to build a customized version of AutoCAD. Other programming interface like ActiveX Automation & VBA can also be used to augment the standard commands of AutoCAD. Visual LISP environment is pointing towards the future trend in LISP. The wonderful VLISP editor with unique features suitable to LISP environment is now available by default. Specific color-coding, parenthesis matching and VLX conversion make VLISP environment more near and dear to AutoLISP programmer.

AutoLISP can be used effectively to develop the specific task oriented software in an AutoCAD environment. More specifically, a mechanical engineer with knowledge of AutoCAD, AutoLISP and design aspect of gear can write a program to produce a gear design and drawing with very little end user input. Once such software is tested, the end user has to input only parameters required for design and tested software could produce design draft and complete the drawing for him. A Civil engineer can write a AutoLISP program for culvert bridge, which can read drawing parameters obtained from actual work site and produce design and drawing.

A good bunch of AutoLISP programs are supplied with AutoCAD software, AutoCAD reference books, bonus CD-ROM, AUGI Forums, and Magazines. There are numerous Internet sites for accessing AutoLISP codes. Learning how to access and load AutoLISP files will inject creativity among users and results in enhancement of productivity.

This course material is targeted for AutoCAD users who wish to begin with the basics of programming to file handling in LISP. Course material covers easy explanations of pre-defined functions of AutoLISP along with examples, writing of user defined functions, logical expressions, branching and looping in program and file handling. *The entity and device access functions are required at advance level of programming and hence are not placed in this course.* If your company is engaged in repetitive drafting task of structures, machines, PCB's or piping, AutoLISP could be a best friend at your service. Everybody is doing repetitive tasks at one time. The appended literature on AutoLISP would certainly pave a path for good programming. Just be prepared to spend lots of time with your computer - not because programming in AutoLISP is all that difficult, but because it is so addicting! And once you've created your own first program, you'll be hooked.

1.1 AutoLISP

LISP is a programming language that was developed in 1959 by John McCarthy for use in programming artificial intelligence. There are numerous dialects of LISP - MacLISP, InterLISP, ZetaLISP, Common LISP etc. and AutoLISP is one of them. LISP is among the easiest of all programming languages to learn and master. AutoLISP is a high level programming language suited to graphics applications. AutoLISP closely resembles Common LISP, an artificial intelligence programming language, which itself is sufficient to announce the versatility of language.

What is programming? It is nothing but a set of instructions. Those who has never written a single line code of computer programming must have passed many instructions to many people in any work environment (A real life programmer?) and must have received many instructions for him in life school (A real life computer?). Programming is something very similar to it. Unknowingly you have acted like programmer and computer in life. As life goes on with certain set of rules and regulations, programming too have certain rules and regulations of writing code. AutoLISP with very simple syntax and functions is among the simplest of the programming languages. LISP is an acronym for ***LISt Processing***, which indicates that LISP works with lists. Do you think "Auto", a prefix before Lisp has some special meaning? AutoLISP programming is significant customization option in AutoCAD. Repetitive and difficult tasks can be automated by creating small but powerful AutoLISP programs. AutoLISP is an interpreter embedded in AutoCAD software. AutoLISP is an evaluator and not compiler which makes it little bit slower than compiler based languages. But its simplicity of syntax overpowers the slowness of execution. While the thought of using AutoLISP may seem a little intimidating, it doesn't really require a lot of computer knowledge to use it. AutoLISP code is nothing but a nice blending of AutoCAD commands and AutoLISP functions arranged in logical way.

"I don't do repetitive design & drawing. My projects are always unique". Many people think that way and are thereby keeping themselves away from programming. Even in unique projects there are many things which can be termed as repetitive. Every living organism is actually programmed for ONE DAY by a super programmer. Every life is in a loop of days. Your entire life is in a loop condition with two sub routines called Day & Night. Everyday input parameters and variables are slightly changed and human output varies accordingly. The loop continues for lifetime, who says "I don't do repetitive task". Suddenly on some odd day your program encounters an "exit" condition and you are out of the loop and world.

You are a born computer and a programmer. So far in your life cycle, you must have programmed many occasions of your life and others life. Now start programming on a computer. There is a little difference in computer programming; you must strictly adhere to the syntax of a programming language. Let's assume you got a nice intelligent mechanically ready robot sitting in your drawing room. You need to program it for journey from your home to Railway Station. Following would be the probable sample set of instruction you must pass on to Robot.

- Take certain steps straight from drawing room
- Beware of down steps and dog before moving to courtyard
- Move some steps left, some right and you will be on road.
- Beware of traffic on road – this is a list of traffic rules in this city – "think" about kids on wheels and don't forget… Oh! Endless list…
- … now you are on Railway Station.

Fine! Your program (set of instructions) is over and ready for loading into Robot. You are excited about this event and loaded program after inviting some friends for a nice show. BUT nothing happens… Your dear Robot is still sitting tight in drawing room. All eyes are towards you, what went wrong? You are again going through listing of your program, may be running in hundreds of line. Each and every instruction seems to be perfect. Even you don't forget to mention that beware of open manhole near Narmada Project Colony. This is a routine problem with every programmer. The process is called debugging – finding errors in a code. Your Robot is still not responding for a simple reason. You forgot to write, "Stand Up" instruction for Robot in the beginning of program and hence it is still sitting in the drawing room. The AutoLISP or for that matter, any programming is like above scenario.

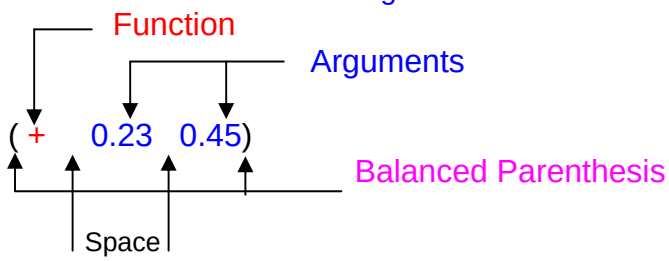
What are the ingredients you are going to use for delicious cuisine called AutoLISP program? Many parentheses, some AutoCAD commands, some AutoLISP functions and pinch of logic.

1.2 Evaluation

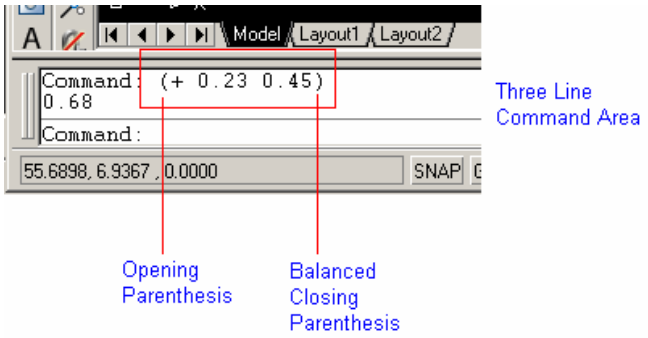
AutoLISP is accessed through the interpreter embedded in AutoCAD software. Anything intended for AutoLISP interpreter, from the simplest expression to the most complex program, must be written with opening and closing balanced parenthesis, which includes function and arguments within it.

A function is an instruction telling the AutoLISP interpreter what to do.

An argument is a value or variable on which function has to act.



There is no specific switch for starting AutoLISP. Just write above expression on AutoCAD three line command area and result is ready. AutoLISP Interpreter is always at your service.



Evaluation of AutoLISP Expression directly at command prompt is the easiest mock drill before actual code writing.

AutoLISP expression can be bunched as program in ASCII file and can be evaluated by writing **(load "Program_name")** at AutoCAD command prompt. AutoLISP expressions can also be written in Visual LISP environment by invoking VLIDE command on AutoCAD prompt. Continuous line AutoLISP expression can also be evaluated at AutoCAD prompt. When you enter information at the command prompt, the interpreter evaluates it, and then returns an answer.

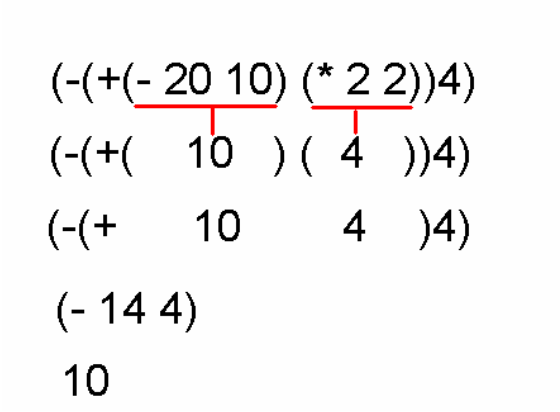
for example :
Command : (+ 6 3)
9
Command : (+ 5 4 3 2)
14

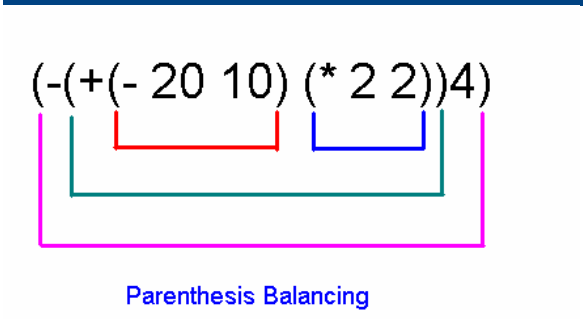
Well, as you must have observed, at least **one space** is require between operator and argument. But why operator first?

Try with this...
Command : (6 + 3)
; error: bad function: 6

That means you must adhere to the syntax of language to get the desired result. Don't you think LISP follows the spoken English syntax? We don't say "six add three", it would be "add six to three" and that's AutoLISP is all about, a handy calculator.....

Command : (-(+(- 20 10) (* 2 2))4)
10





Parenthesis Balancing

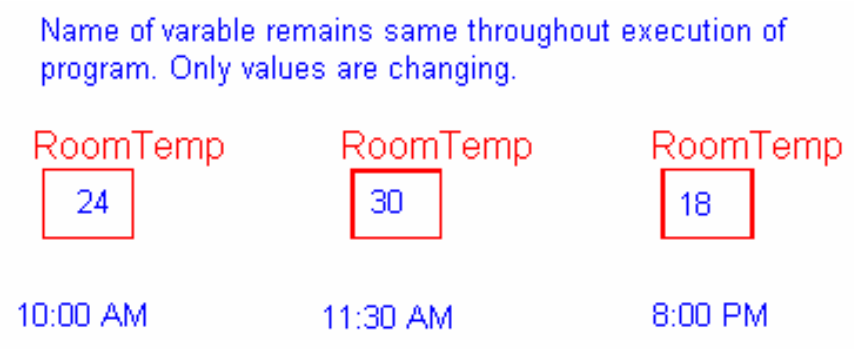
Try out and study the balancing of parenthesis and set of evaluation.

1.3 Variable

Memory variables are the tools by which a computer can organize, store and recall information at later stage. Variables are the basics to computer programming, all programming languages depend on memory variables to decide the logical flow of program, AutoLISP is no exception to this rule. You must be aware with system variables of AutoCAD invoked by command SETVAR. These are memory boxes used by AutoCAD to store some preset read only values and some variables are open to user for modifications. For example: ACADVER memory variable is Read Only and displays version of AutoCAD you are using. Users can not change the VALUE of this variable. If you invoke Dimension Style dialogue box for changing height of dimension text, internally the text height is changed in memory variable called DIMTXT. There are hundreds of such variables defined in AutoCAD, the value of which can be called in your program using GETVAR function of AutoLISP.

AutoLISP variables can be of four types: integer (bardia=10), real (bridgeheight=4.5), points (stpt1=2,3) and string (city=Bharuch). A variable's type is automatically attached to it based on the type of value assigned. Variables are sometimes termed as Symbol names. Variables are not case sensitive and except characters like ( ) . ‘ “ ; it can contain any combination of alphanumeric and notation characters. Name of variable consisting of only numeric characters are not allowed.

A variable is like a box that holds a value. Any data name can be given to variable to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times, during execution. In AutoLISP, you can assign a value to a variable name and use it in a command or with in expressions to perform calculations and branching of logical flow of program.



A practical example: PM is a variable holding values like Indira Gandhi, I. K. Gujaral or Dr. Manmohan Singh at different time span. Programmer should always choose the variable name in meaningful way to reflect its nature and function in the program. A variable name should be self-explanatory like...

Stpt1	Starting point 1
Newlimmin	Dummy variable to store limmin
Total	Total of any real or integer values
Slb_th	Slab thickness
Pm	Prime Minister

1.3.1 Setq

AutoLISP predefined function to attach value with variable. This function tells AutoLISP to assign a value to a memory variable. This function requires first and second arguments for variable name and value respectively. Variable assumes the data type of the value assigned to it. Variable retains its value until new value is assigned.

The Syntax of function is ...

*(***Setq** *<variable1> <value1> [<variable2> <value2>]..)*

The Setq function tells AutoLISP to assign a value to a variable.

```
(setq gold 4000)
```

Value [Integer in this case] 4000 is assigned to the variable 'gold'. The variable gold has occupied some memory and gold is pointing to that memory. You can use gold variable whenever operation-involving value 4000 is required.

For example type (* gold 2) at AutoCAD command prompt, AutoLISP evaluator return the answer as 8000. If you wish to see the value attached with any variable, simply write '!' sign (For Exclamation sign press Shift + 1) before variable name at AutoCAD prompt.

```
Command : !gold
 4000
Command : (setq  gold (/ (*  gold  2) 8))
1000
Command : !gold
1000
```

As you must have observed in syntax, optionally you can assign different values to different symbols at a time, but setq function returns the result of the evaluation of the last expression. The following line

will define three variables in memory, slbth (Slab thickness) housing value 150, beam housing value 200 and column housing value of 450.

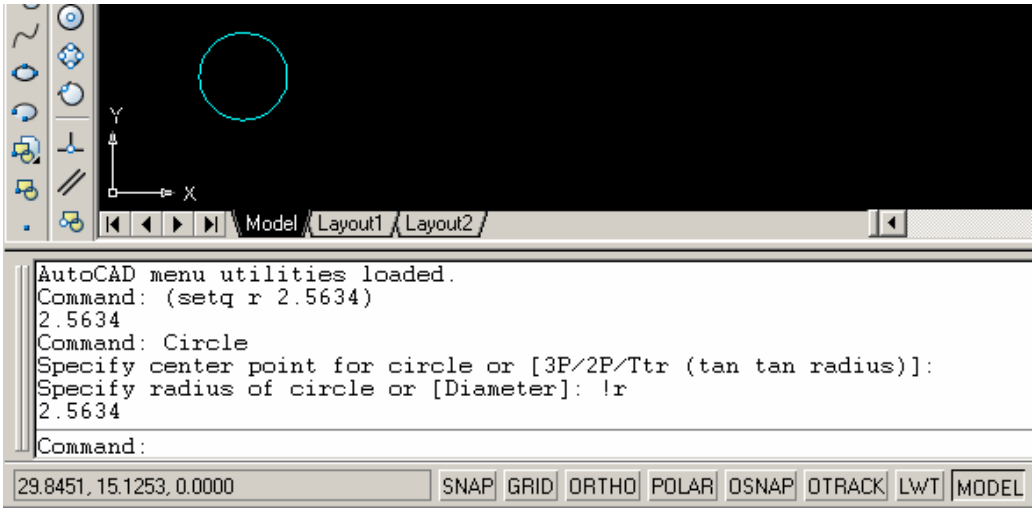
```
Command : (setq slbth  150  beam 200  column 450)
450
```

Once value is assigned to the variable, it can be used in response to any AutoCAD command prompts. If you are handling the drafting job with very complex numbers and string data, it can be stored in a memory variable and can be reused at command prompt as shown below.

```
Command : (setq  r  2.5634)
 2.5634
```

```
Command : Circle
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: Pick center point for circle
Specify radius of circle or [Diameter]: !r
```

The above sequence of command shall produce circle of radius 2.5634.



Memory variables defined in AutoCAD can be extracted for use in your program. GETVAR is the function for extracting value from any memory variable defined in AutoCAD.

```
Command: (setq  mydimht (getvar "dimtxt"))
0.18
```

Whatever value available in dimtxt memory variable at the time of executing this line will be stored in user defined variable mydimht. [My dimension height]. The returning value 0.18 shows that dimtxt variable was storing 0.18 value at the time of executing this line on command prompt. I should specify that now mydimht will have 0.18 value stored in it.

Command: (setq mydimht (* mydimht 2))
0.36

Now value of mydimht variable has changed to 0.36. That's why it is called variable. Unless you change it, this value will remain in effect for Current drawing file.

1.4 Data types in AutoLISP

The variety of data types available allows you to choose the data type suitable to your application and computer, which also helps AutoLISP for efficient memory management. Variables are provided into several categories called data types. Computer stores different types of data differently, so the use of data types helps AutoLISP communicate with the computer more efficiently.

DATA TYPE

1. **Integer**
[Number without any decimal fraction]

Example: 383, -83

Variable Assignment :

Command : (setq states 25)
25
Command : (setq two 2)
2
Command : (/ states two)
12
[Why 12? Why not 12.5, because it results in integer]

2. **Real**
[Real number includes decimal values.]

Example : 383.45, 0.38

Variable assignment :

Command : (setq Brh 3.82)
3.82
Command : (+ .382 2)
error: misplaced dot on input
Command : (+ 0.382 2)
2.382
Why so? values between 0.1 and -0.1 must begin with zero.

AutoLISP rule, can't help …

3. **String**
[string means text containing alphanumeric characters and punctuation marks]

Example :
"Tel. 91-98981-64055"
"Bharuch - 392 002"

Variable assignment :

You can store a whole string in a variable.

Command : (setq cnm "Augi HotNews")
"Augi HotNews"
Command : !cnm
"Augi HotNews"

The text supplied as argument must be in double quote.

Command : (setq city London)
nil

4. **Lists**
[A list is set of elements enclosed by parenthesis]

Example : '(1 3 8 9 2)
(list “Bharuch” “London” “Mumbai”)

The individual elements of list must be separated by atleast one space.

You can store the data to be processed as list in AutoLISP. For example: you can store a full month of rainfall data in a list to draw graphs or other comparison or manipulation. A quality control engineer can store a set of standard values in a list to compare with results processed. Civil Engineer can store initial ground levels for canal or road. Can you recall my sentences in the starting of this course material? **LISP is an acronym for *LISt Processing*, which indicates that LISP works with lists.**

If you assign three values in list, you may treat it as x, y & z coordinates.

(setq pt1 '(3 3 4))

Above statement shall assign the values 3, 3 & 4 to x, y & z coordinates respectively for pt1. Just try with following command sequence.

Command: Line
Specify first point: !pt1
(3 3 4)
Specify next point or [Undo]: *@2,0,0*

The value stored in pt1 is utilized to start line. What you see on screen is, a horizontal line of 2 units from point 3,3,4

Example :

Command : (setq a4 '(1 2 3 4 5 6 7 8 9))
(1 2 3 4 5 6 7 8 9)

Command : (setq a5 (1 2 3 4 5 6 7 8 9))
error: bad function: 1

Command : (setq a6 (list 1 2 3 4 5 6 7 8 9))
(1 2 3 4 5 6 7 8 9)

Command : !a4
(1 2 3 4 5 6 7 8 9)

Command : !a5
nil

Command : !a6
(1 2 3 4 5 6 7 8 9)

Command : (setq a5 a6)
(1 2 3 4 5 6 7 8 9)

Command : !a5
(1 2 3 4 5 6 7 8 9)

Command : (setq a7 '(1 2 (3 4) 5))
(1 2 (3 4) 5)

5. **File descriptor :**

Like all programming languages AutoLISP has functions to open files for read - write operations. File descriptors are used to keep track of files. We will discuss it at length in Segment_3.

6. **Entity Names :**

AutoCAD assigns an alphanumeric code-name to every object, which is not defined by user and entity names change from one drawing session to another.

7**. Selection sets** :

Selection set can be created and can be stored in a variable to be acted upon as a group. *The entity and device access functions are required at advance level of programming and hence are not placed in this course.*

8**. Symbols [Variables]** :

Symbols or variables are the key of programming. We have already discussed the concept of variables.

9**. Subrs** :

Subrs are predefined subroutines or functions embedded in AutoLISP. For example, setq is predefined function to assign a value to variable, like setq there are many mathematical and logical functions to carry out various task of programming.

```
Command : !setq
  <subr : #3C50>

Command : !car
  <subr : #3796>
```

1.5 List manipulation

LISP is nothing but *LISt Processing* and manipulation of list is one of the prime task in AutoLISP programming.

As you know AutoCAD keeps track of your drawing through the database prepared for every objects, all points of drawing are stored as coordinates (x,y,z) and because it is a group of values it is handled as a list in AutoLISP.

Example :

```
(setq  pt1  '( 3  8))              [ 2D point ]
(setq  pt2  '( 3  8  2))           [ 3D point ]

(setq   gl  (list  21.3  22.5  23.5  23.8  22.6  23.0  24.5 22.5  23.5 26.5))
```

gl variable could be a list of ground reduced levels to draw contour...

```
(setq l_values (list  100 108 112 118 123 128 132 139 142))
```

l_values could be a testing loading's to be applied on any testing specimen for quality control.

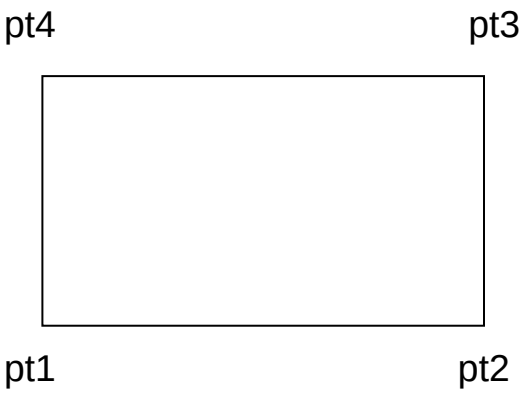
Manipulation of list can be in any shape described below..

- Accessing members of list
- Constructing new list from existing list.
- Construction of new variable using members of list.

Let's start with simple manipulation of LIST…

Fig. 01

back to AutoCAD for a while...



Suppose you have two points: pt1 and pt3 on graphics editor and you wish to find out the pt2 and pt4 framing other corner points of assumed rectangle, simply you will use **.x of pt3** and **.y of pt1** to extract coordinates of pt2. Similarly **.x of pt1** and **.y of pt3** would result in pt4. You can also use the same technique of extracting the coordinates in AutoLISP with different syntax but same logic.

Two locations 3,4 and 7,7 are stored as variables in pt1 and pt3. List function of AutoLISP can be employed to construct a list of such coordinates.

```
(setq  pt1  (list  3  4))
(setq  pt3  (list  7  7))
```

in plain vanilla way new variable pt2 can be constructed as below ..

(setq pt2 (list .x of pt3 .y of pt1))

but, above statement is not a valid statement for AutoLISP. The programming in AutoLISP has its own syntax and functions to do the same job. AutoLISP provides predefined functions like **car, cadr, cdr, caddr** and many more to access the members of list.

A valid statement for assigning desired coordinates to pt2 would be ...

```
(setq  pt2  (list  (car pt3 )   (cadr pt1)))
                    ↑            ↑
equivalent to     .x of pt3      .y of pt1
```

So, let's forget about your favorite BMW and move to grand garage of LISP for studying family of Car..(s)...

Car : In order to access the first member of a list, we use the function car.

```
Command : (setq  pt1  '( 3  4 ))
(3 4)
Command : (car pt1)
3
```

Cadr : Second member of a list can be accessed using cadr function.

Caddr: to access third member.

Cadddr : to access fourth member.

(car (cddddr)) : to access fifth member.

(cadr (cddddr)) : to access sixth member.... and so on ...

Try with this example on AutoCAD prompt...

```
Command : (setq boltdia '(10 12 16 20 24 28 32))
(10 12 16 20 24 28 32)

Command : (car boltdia)
10

Command : (cadr boltdia)
12

Command : (caddr boltdia)
16
```

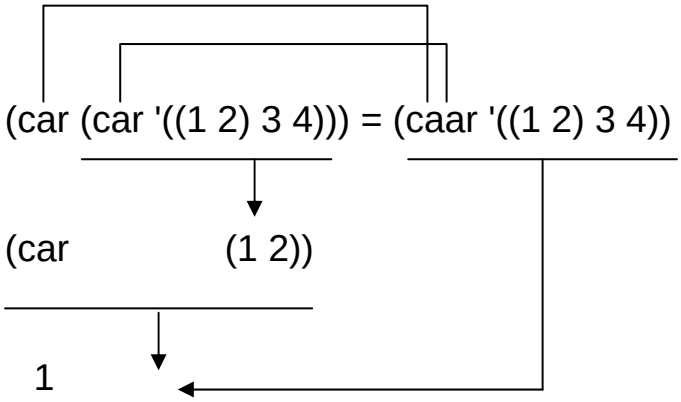
.... and so on with other members.

CAR and CDR are basic functions, AutoLISP support concatenation of above two functions up to four level deep.

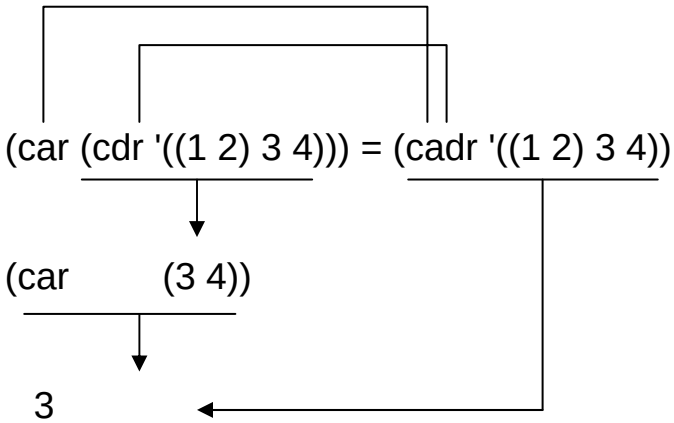
CDR : This function returns a list containing list other than first element of list.

```
Command : (cdr  '( A E I O U))
(E I O U)
Command : (cdr  '( (A E) I O U))
(I O U)
```

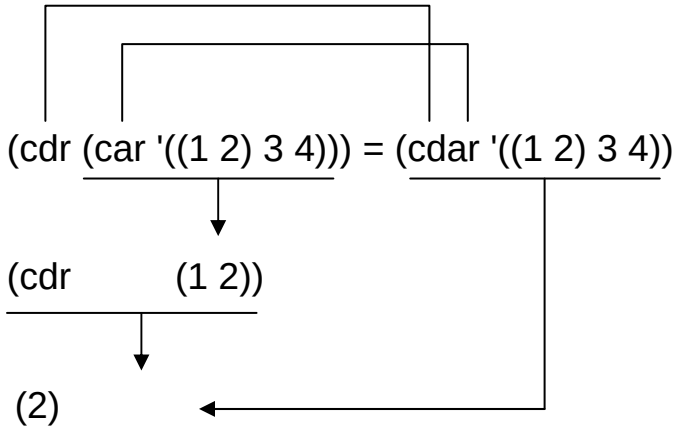
Caar : Car and Car combine to form Caar.



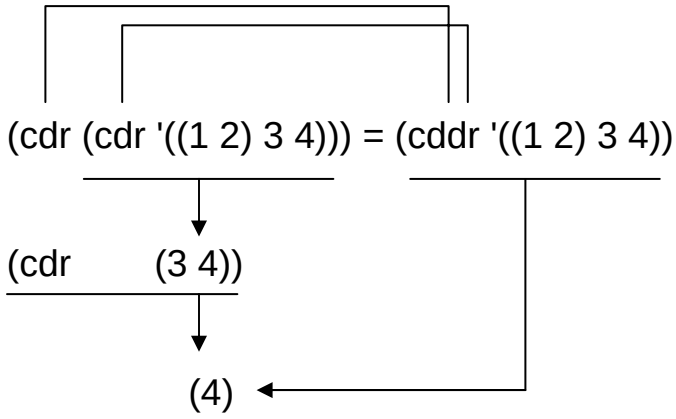
Cadr : Car and Cdr combine to form cadr.



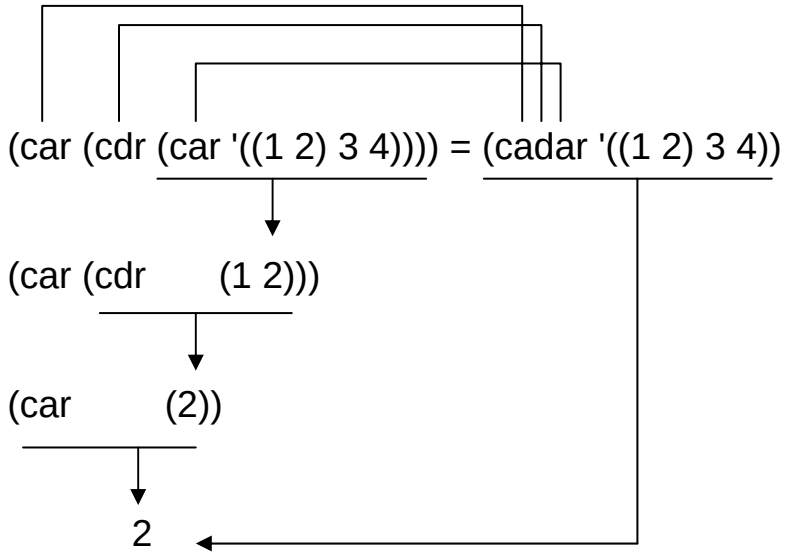
Cdar : cdr and car combine to form Cdar.



Cddr : Cdr and Cdr combine to form Cddr.



Cadar : Car, Cdr and Car combine to form Cadar.



In AutoLISP we are going to use Car, Cadr and Caddr frequently for obtaining x, y and z coordinates of point respectively. The predefined function ***nth*** would be a better choice to access the members of a list. It is recommended to access first three elements with Car, Cadr and Caddr and for accessing fourth member onwards, use nth function.(Discussed in Segment_2). **SO, JUST THINK ABOUT car, cadr & caddr AT PRESENT.**

```
(setq pt2  '(5.25  1.0))          (a 2D point)
(setq pt3  '(5.25  1.0  3.0))     (a 3D point)
```

Then:

```
(car pt2)       returns     5.25
(cadr pt2)      returns     1.0
(caddr pt2)     returns     nil
(car pt3)       returns     5.25
(cadr pt3)      returns     1.0
(caddr pt3)     returns     3.0
```

If :

```
(setq  pt1  (list  1  2))
(setq  pt2  (list  5  8))
```

Then :

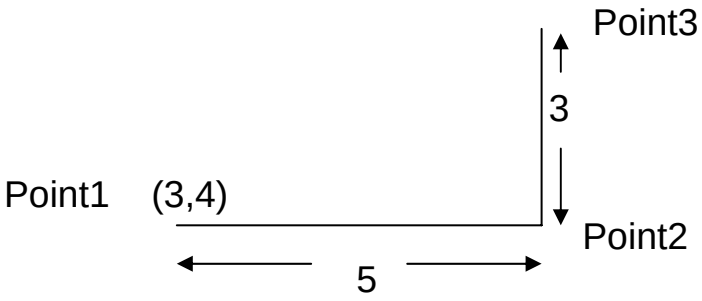
(car pt1)	returns	1
(cadr pt1)	returns	2
(car pt2)	returns	5
(cadr pt2)	returns	8

Using the list function, you can construct a point variable using x and y components of other variables.

(list (car pt2) (cadr pt1))

result ?
try it, and store the list in new variable ptz.

The main task of AutoLISP programming is to find out the locations [coordinates] of different points and apply the AutoCAD command to frame the desired drawing. Following steps shall clarify the use of car and cadr.



A. (setq point1 '(3 4))

Suppose point1 is a variable having value of (3 4) as list, then the coordinates of the point2 shall be ...

.x of point1 + distance between point1 & point2 , .y of point1

3 + 5 , 4

(car point1) + 5 , (cadr point1)

AutoLISP statement for the point2 would be..

B. (setq point2 (list (+(car point1) 5)(cadr point1)))

AutoLISP statement for the point3 would be..

C. (setq point3 (list (car point2) (+(cadr point2) 3)))

Write A,B & C statements on AutoCAD prompt followed by the command written below and observe the result.

```
Command  : line
From point : !point1
To point   : !point2
To point   : !point3
To point   :
```

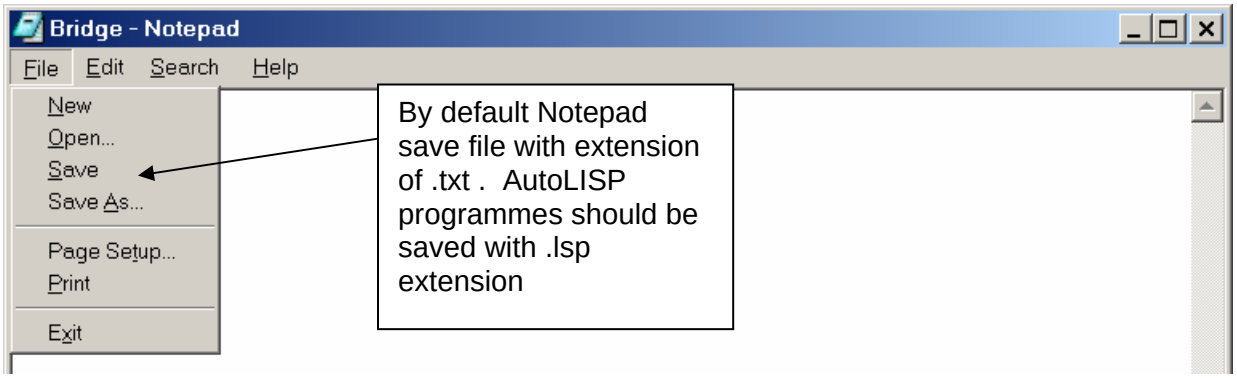
Programming in AutoLISP

A full feature AutoLISP program with the facility of reading the data, process on data and branching of logical flow may take more than hundred lines, which is practically impossible to enter at the AutoCAD prompt. Hence the whole program is written to a standard ASCII file using a text editor, which can be loaded on AutoCAD prompt to 'run' the program. AutoLISP programs are written in file with extension of '.LSP'. A good hand on your text editor would minimize the probable syntax error while writing the program. You can try your friendly default editor Notepad to enter the AutoLISP program. However, I recommend Visual LISP editor for writing programs.

2.1 Editor - Notepad

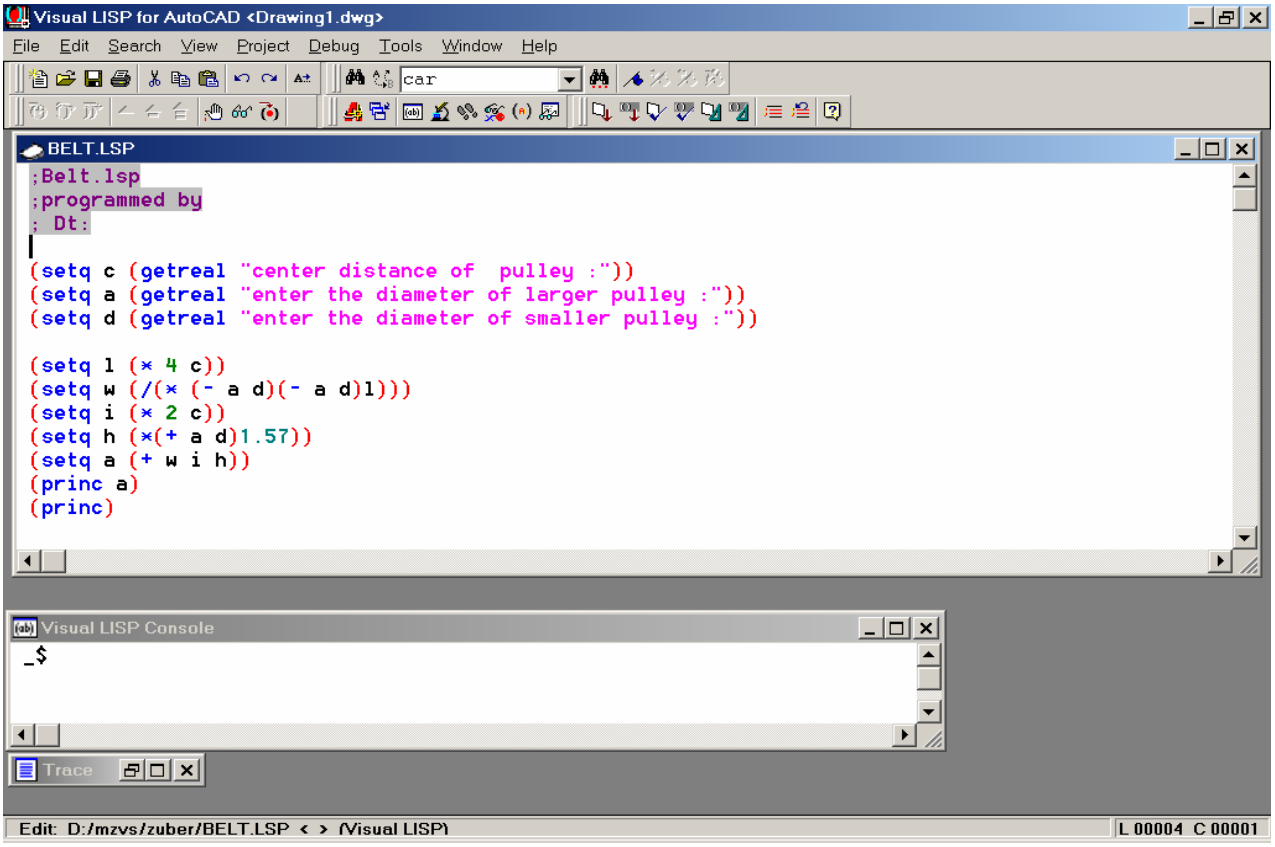
AutoLISP programs are stored on disk in ASCII [American Standard Code for Information Interchange] format. These files are created using a text editor. Master the text editor before you can get down to business of AutoLISP programming. Notepad is the default text editor, which can be invoked in AutoCAD environment by writing **Notepad** at AutoCAD command prompt. Though it is simple in use, author presumes that ready to use help on editor shall be productive for novice.

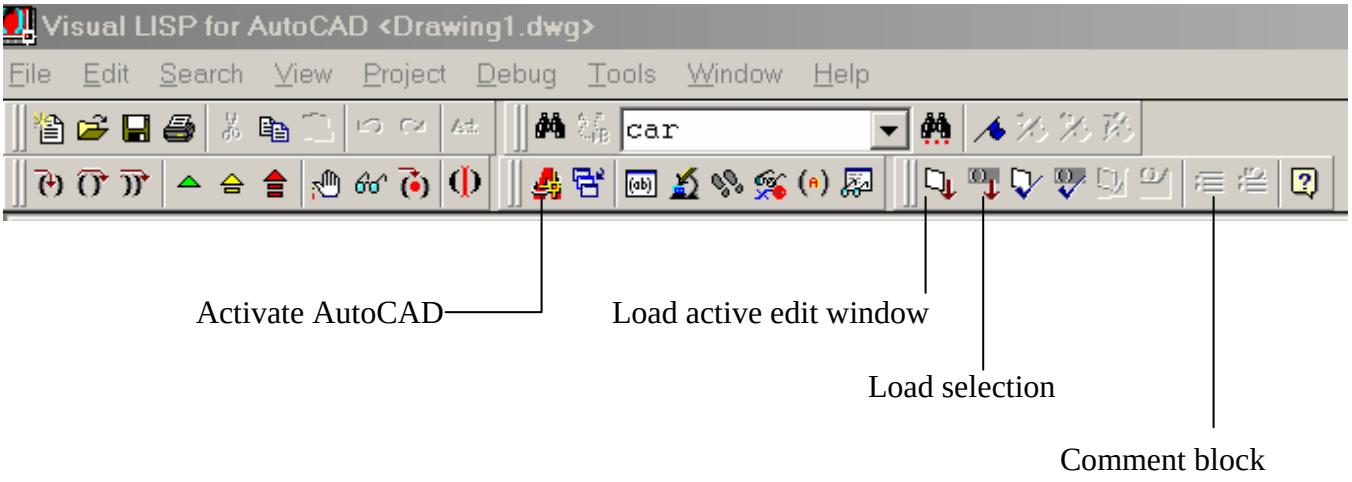
Fig. 02



Visual LISP Editor

Visual Lisp Editor can be invoked by writing ***VLIDE or VLISP*** at AutoCAD command prompt.





The VLISP text editor is much more than a writing tool; it's a central component of the VLISP programming environment. To appreciate the versatility and value of the VLISP text editor, you need to be familiar with the AutoLISP language. THE IMPORTANT FEATURES OF VISUAL LISP EDITOR ARE DOCUMENTED AT THE END OF THIS SEGMENT.

2.2 Program

A set of instructions could be fed into the computer, that cause the computer to perform calculations based on this set of instructions, called programs. AutoLISP program is nothing but a combination of AutoCAD commands written in LISP way, AutoLISP data types, predefined AutoLISP functions and user-defined functions.

Predefined functions are functions that come with AutoCAD software written by Autodesk Inc. USA. The predefined functions are compiled and stored in machine language. Setq is one such predefined function we have already discussed. The user-defined function written in AutoLISP remains in ASCII format and cannot be converted in machine language, hence they are comparatively slow. Using Visual LISP, programs can be converted to ARX / VLX format, which runs faster and converts to unreadable code. AutoLISP predefined functions can be grouped as arithmetic, string handling, equality and conditional, list manipulation, symbol handling, function-handling, query and command, display control, user input, geometric, conversion and device access functions.

Here, we shall start with simple predefined AutoLISP functions. Throughout the course material the following conventions are used for syntax of function.

1. Function name in small letter.	setq
2. Arguments between	< >
3. Optional Arguments	[ ]
4. More arguments possible	...

2.3 Some useful Math Functions

1. (1+ <number>)

This function increment the number by 1 and returns the integer as a real incremented number. As we have discussed, an operator is written before passing arguments in AutoLISP but (1+ <number>) and (1-<number>) are the exception to the rule. Please note that there is no space between 1 and + or 1 and -.

2. (1- <number>)

Function decrement the number by 1.

Example:

```
(1+ 8)          returns     9
(1+ 5.5)        returns     6.5
(1- 8)          returns     7
(1- 5.5)        returns     4.5
(setq nm 30)
(1+ nm)         returns     31
```

3. (abs <number>)

This function returns the absolute value of integer or real number.

Example:

```
(abs -200)          returns     200
(abs -33.3)         returns     33.3
(setq absno -11)
(abs absno)         returns     11
```

4. (atan <number1> [<number2>])

<number2> is optional in this function, if only number1 is supplied, this function returns the arctangent of number1 in radians, ranging from - n/2 to + n/2 radians. If both number1 and number2 are supplied, the arctangent of number1/number2 is returned, depending on the sign of number1.

Example:

```
(atan 2.0)          returns     1.10715
(atan 2.0 3.0)      returns     0.58800
```

5. (cos <angle>)

This function returns the cosine of angle expressed in *radians.*

Example:

(cos 1.0)	returns	0.540302
(cos pi)	returns	-1.0

6. (expt <base> <power>)

Function returns base raised to power. If both arguments are integers the result will be an integer, for both real arguments the result shall be real.

Example:

(expt 3 4)	returns	81
(expt 3.3 4.4)	returns	191.188

7. (fix <number>)

Function converts a number from real to integer.

Example:

(fix 8.3)	returns	8
(fix 8)	returns	8

8. (float <number>)

Function converts number from integer to real.

Example:

(float 8)	returns	8.0
(float 8.3)	returns	8.3

9. (gcd <number1> <number2>)

Function finds greatest common denominator of *integer* used as arguments.

Example:

(gcd 36 54) returns 18
(gcd 100 10) returns 10

10. (log <number>)

This function returns the natural log of number in *real.*

Example:

(log 4) returns 1.38629
(log 1.0) returns 0.0

11. (max <number> <number>...)

This function returns the largest of the numbers passed as parameters. Arguments may be *integer or real*.

Example:

(max 9 3 6) returns 9
(max 6.5 3.5 2) returns 6.5

12. (min <number> <number>...)

This function returns the lowest of the numbers passed as parameters. Arguments may be *integer or real*.

Example:

(min 9 3 6) returns 3
(min 6.5 3.5 2) returns 2

13. (rem <number> <number2>...)

rem function divides number1 by number2 and returns the remainder. You can select real or integer as arguments.

Example:

(rem 50 5) returns 0
(rem 50 6) returns 2

14. (sin <angle>)

This function returns sine of angle expressed in *radians*.

Example:

(sin 1.0)	returns	0.841471
(sin 2.0)	returns	0.909297

15. (sqrt <number>)

This function returns the square root of its single parameter. Result shall be in *real*.

Example:

(sqrt 36)	returns	6.0

16. (+ <number> <number>...)

returns the sum of all <number>s.

17. (- <number> <number>...)

subtracts the second <number> from the first and returns the difference.

(- 50 40)	returns	10
(- 50 40.0 2.5)	returns	7.5

18. (* <number> <number>...)

return the product of all <number>s.

19. (/ <number> <number>...)

This function divides the first <number> by the second and returns the quotient.

(/ 100 2)	returns	50
(/ 100 20.0 2)	returns	2.5
(/ 100 20 2)	returns	2

20. (= <atom> <atom>...)

"equal to" relational function.

(= 4 4.0)	returns	T
(= 345 344 345)	returns	Nil
(= "abc" "abc")	returns	T
(= "abc" "abz")	returns	Nil

21. (/= <atom> <atom>...)

"not equal to" relational function.

(/= 10 20)	returns	T
(/= "abc" "abc")	returns	Nil

22. (< <atom> <atom>...)

"less than" relational function.

23. (<= <atom> <atom>...)

"less than or equal to" relational function.

24. (> <atom> <atom>...)

"greater than" relational function.

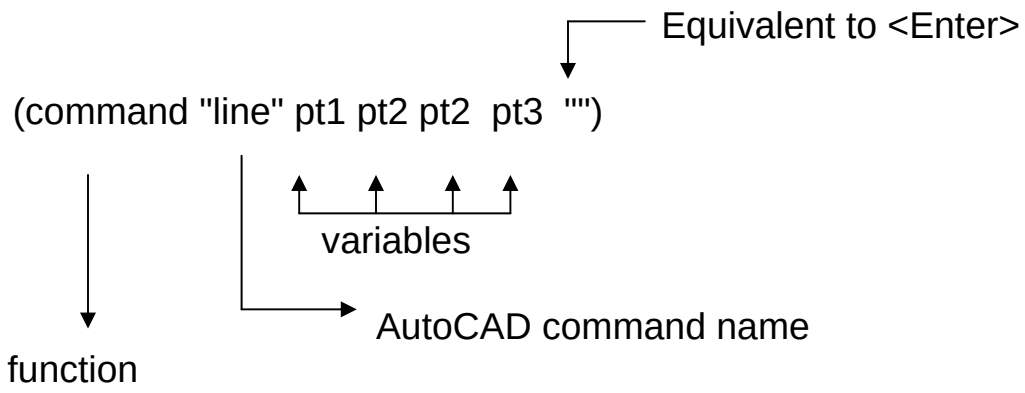
2.4 First program in AutoLISP

```
; Program in AutoLISP
; First try with programming on Dt: 06-11-2006
; Programmed By

(setq  pt1  '(3 4))              ; 2D point pt1
(setq  length  4)                ; integer value 4 assigned to
                                 ; the variable 'length'.
(setq  height  3)                ; integer value 3 assigned to
                                 ; the variable 'height'.

(setq pt2 (list (+(car pt1)length) (cadr pt1)))
(setq pt3 (list (car pt2) (+(cadr pt2)height)))
(setq pt4 (list (car pt1) (cadr pt3)))
(command "line" pt1 pt2 pt3 pt4 "c")

; End of program
```

Write above ten lines as it is in file ***Rect.lsp*** in any text editor of your choice and save it.

In the process of writing the first program you are exposed to TWO important aspect of programming. FIRSTLY, semicolon - ; - at the beginning of line indicates that, this line is comment and AutoLISP interpreter shall skip this line without evaluating it. Comments in the program are for programmers use, he may wish to include it in the program to record the purpose of program, name of firm for whom program is developed, last revised date of program, name of developer etc. etc... You may include the comments at any location in program. The SECOND aspect is writing of AutoCAD command in AutoLISP programming. Lets discuss the AutoLISP function 'command'.

(COMMAND <arguments…>)



The command function executes standard AutoCAD commands from with in AutoLISP. The <arguments..> followed by command in above syntax, represent AutoCAD commands and subcommands with all probable parameters, which must be enclosed in quotation marks. The variables of program should not be enclosed in quotation marks.

```
(command "line" pt1 pt2 pt3 pt4 "c")
```

In the above statement "line" and "c" are enclosed in quotation marks, treating them as keyboard input. This line shall join the pt1, pt2, pt3 and pt4 by the line entity, on execution. All the required entries should be provided in a command expression. If a programmer wishes to have user entry in the command expression, then the word pause must be placed at the location where entries are expected.

```
(command "line" pt1 pt2 pause "")
```

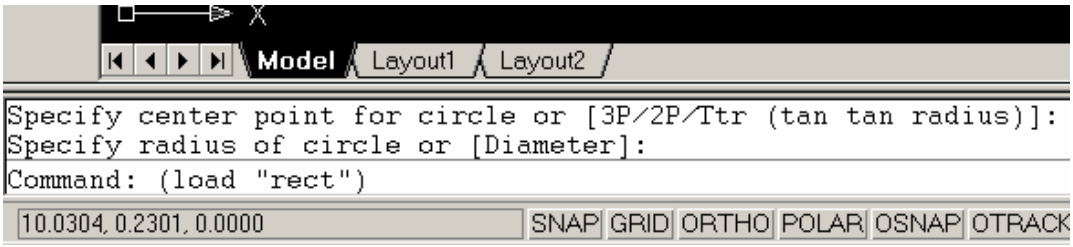
In the above example AutoCAD places the first point pt1, second point pt2 and user provides the third point.

```
(command "erase" "w" pause pause "")
```

The above line lets the user to place a window with a first and second corner to erase out entities enclosed by window.

How to execute the program?

Write **(load "rect")** on AutoCAD prompt and press <Enter> to observe the result of Rect.lsp program on graphics screen.



As soon as you press <Enter>, AutoLISP interpreter shall start its work.
Or if you have entered the code in VLISP environment, press *Load active edit window* button.

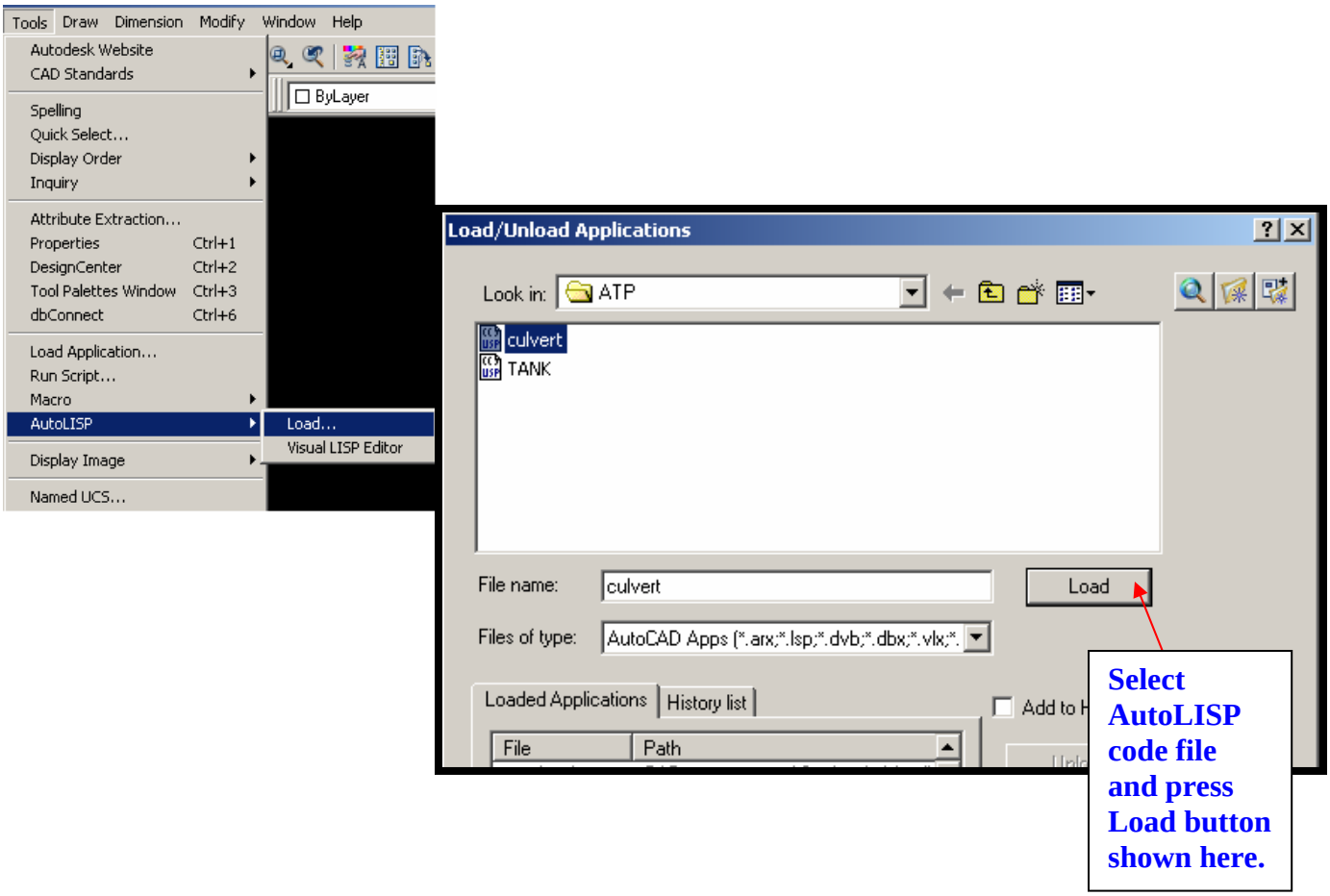
- 2D point assigned as list 3,4 to pt1
- integer 4 assigned to length
- integer 3 assigned to height
- 2D point calculated and stored in pt2
- 2D point calculated and stored in pt3

- 2D point calculated and stored in pt4
- Line command joins pt1, pt2, pt3 & pt4

You can observe a rectangle of 4 unit width and 3 unit height on screen....Well, this is not a user friendly program. We don't need a rectangle of fixed width and height at certain location. In short there must be some way to get width, height and location as user response. Get family of functions in AutoLISP is primarily for accepting the user response.

The program RECT.LSP must be available in AutoCAD search path for loading it as mentioned above. If program is saved at some other location, you can load it by calling Load/Unload Applications dialogue box available under Tools group in Main Menu. Alternatively you can use APPLOAD command to activate same dialogue box.

The same program can also be loaded by calling Load Application from Tools pull down menu.



2.5 Getxxxx functions

Get family of functions are used in programming for accepting user input. Depending on the requirement of application the form of data input is changing and hence Autodesk has devised self-explanatory name for each type of data. For example ***Getint*** clearly indicates that function is useful for accepting only integer data.

What is user input?

When you enter CIRCLE command on AutoCAD prompt, software responds with following line

CIRCLE Specify center point for circle or [3P/2P/Ttr (tan tan radius)]:

In response to this prompt…

You may select a point on screen – it is called user input
You may type 3P or other options – it is called user input

1. Getint :

(getint [<prompt>])

This function pauses for user input of an integer and returns the same integer. Limitation of integer range [- 32768 to 32767] is applicable for this function. Prompt is optional. Prompt is a string to be displayed by AutoCAD when AutoLISP interprets the line.

In Rect.lsp program , when we write (setq length 4), we are restricting the use of program. Instead of this, Getint can be employed to get the length of users choice in integer only. The length input will depends on the choice of user and hence Rectangle of any length can be created by such program. What will happen if you omit prompt "Enter length: "? The AutoLISP interpreter will stop because of getint call for accepting user input but it will not be clear to user what programmer is expecting from user. In CIRCLE command of AutoCAD, the prompt is "Specify center point for circle or [3P/2P/Ttr (tan tan radius)]:" It is the responsibility of programmer to decide the most appropriate prompt which clearly specify what input he is expecting from user.

```
(setq length (getint "Enter length : "))
```

If a user commits a mistake by writing Real value in above call, AutoCAD rejects that entry and displays error message "Requires an integer value." Unless integer value is provided or blank enter pressed, Enter Length: prompt will appear.

2. Getreal :

(getreal [<prompt>])

Function pauses for user input of a real number and returns that number in real. This function will accept integer value but will return it as real.

Example:

(setq height (getreal "Enter height of rectangle : "))

2. Getpoint :

(getpoint [<point>] [<prompt>])

Getpoint function pauses for user input of a point. If point is selected on drawing area, it returns x,y,z points of selected point. <point> and <prompt> are optional. Prompt is string to be displayed by AutoCAD when AutoLISP interprets the line.

In Rect.lsp program, instead of assigning fix point to pt1 as '(3 4), you can use getpoint to left the choice of point on user.

(setq pt1 (getpoint "Pick lower left corner of rectangle :"))

If the optional <point> is supplied, a rubber-band line from that point shall appear on graphics screen.

(setq pt2 (getpoint '(3 4) "Pick second point :"))

User may specify a point by showing it on graphics screen using any pointing device or by coordinate entry.

4. Getcorner :

(Getcorner <point> [<prompt>])

This function allows a selection of a point, as opposite corner point of window, with reference to base point as other corner of window. This function use base point as corner of window, hence when you move the cross hair on screen window will be formed.

In Rect.lsp program instead of calculating the pt2 using pt1, length & height. You can left the choice of pt2 on user using getcorner function.

(setq pt2 (getcorner pt1 "Pick opposite corner :"))

5. Getdist :

(Getdist [<point>] [<prompt>])

This function always returns the distance in real. When function pause for user input of a distance, user may enter it in three ways...

- by typing a number [very similar to Getint & Getreal call]

- by 'SHOWing' two points on screen. You may also use AutoCAD 'OSNAP' method for designating points.

- If optional <point> argument is specified, AutoCAD consider it as first of these two points.

```
(setq length (getdist "Enter distance or show by two points"))
(setq height (getdist '(1.0 1.0) "Second point : "))
```

As Getdist has flexibility of showing distance using existing objects, it is more preferable way of accepting values.

Rect.lsp with user response :

Now we shall rewrite the Rect.lsp code in light of discussion on get family of functions. We will call it Bx.lsp.

```
; Bx.lsp
 ; New version of Rect.lsp with user input.
 ;| Multiline comments can be inserted by prefixing  semicolon
 and pipe at the starting of comment paragraph and ending  the
 paragraph with pipe and semicolon. |;

 (setq pt1 (getpoint "Pick lower left corner of box : "))
(setq length (getreal "Enter length : "))
(setq height (getreal "Enter height : "))
(setq pt2 (list (+(car pt1)length) (cadr pt1)))
(setq pt3 (list (car pt2) (+(cadr pt2)height)))
(setq pt4 (list (car pt1) (cadr pt3)))

(command "line" pt1 pt2 pt3 pt4 "c")
;End of program
```

Save and load the box program. The appearance of box on graphics screen is now a matter of user's choice. Don't you think use of getdist function in line 2 & 3 makes your program more flexible? Setq the length and height using getdist function...save it.... run it...

```
(setq length (getdist "Enter length : "))
(setq height (getdist "Enter height : "))
```

6. Getangle :

(getangle [<point>] [<prompt>])

This function pauses for an angle input from the user. The angle input may be in degrees, radian or other unit format but AutoCAD always return the value in radians. The Prompt is an optional string to be displayed by AutoCAD when AutoLISP interprets the line.

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

The user can also input the angle by showing two points on graphics screen. AutoCAD shall calculate the angle between the "SHOWn' two points and shall return the same in radian. <point> in this function is optional, if it is supplied as argument, AutoCAD shall assume it as first of the two points to be supplied to calculate angle.

(setq ang2 (getangle '(0 0) "Enter angle :"))

Reference point 0,0 shall be assumed as first point and AutoCAD shall wait for other point to calculate the angle.

Enter the following at AutoCAD prompt :

(setq ang3 (getangle "Enter angle : "))

- if you respond with <90, the returning value shall be in radian i.e 1.5708

- if you respond by showing 2,2 and 4,4 points on screen, AutoCAD calculates the angle between 2,2 and 4,4 coordinates and returns 0.785398 i.e radian equivalent of 45 degrees.

7. Getstring :

(getstring [<cr>] [<prompt>])

Text input is possible using Getstring function. Getstring will accept almost anything as input, so it can be used to input words or sentences of up to 132 characters long.

```
(setq nm     (getstring "Your name please"))          Accept one word.
(setq name (getstring T "Your name please"))
```

Getstring followed by T allow the user to enter whole sentences or phrases. The T argument in the above example could be replaced by an integer, a real value, or any expression that will not evaluate to nil. You can enter numbers in response to getstring, but they will be treated as strings and you won't be able

to apply math functions to them. You can, however, convert a number that is a string variable in to a real or integer, using the Atof, Atoi and Ascii functions.

The getstring and the read-line function gets a string from the user. The read-line function is also capable of reading from a file. Strings are important, especially while dealing with Input/Output strings.

2.6 How do you use F2 in AutoLISP ?

(textscr)	This function flips to text screen , if you are on graphics screen, just like pressing of F2 function key in AutoCAD.
(graphscr)	This function flips to graphics screen , if you are on text screen, just like pressing of F2 function key in AutoCAD.

2.7 Screen Display Functions in AutoLISP :

In general any commercial program starts with announcement of its utility, version number, year, developers name etc. etc. To do this job in AutoLISP,you need text screen functions. Text screen functions in AutoLISP are plain vanilla functions without color and text type graphics.

1. (princ <expr> [<file-desc>])

(princ "Hello") prints Hello and returns "Hello"

Example:

Example below shows the framing of opening screen using screen display function. \n is instruction for printing next text on new line.

```
(textscr)
(princ " _____________________________________________\n")
(princ "|                                             |\n")
(princ "|  PROGRAMME FOR DESIGN & DRAWING OF GEAR     |\n")
(princ "|                WRITTEN BY                   |\n")
(princ "|                                             |\n")
(princ "|                  X Y Z                      |\n")
(princ "|_____________________________________________|\n")
(graphscr)
```

2. (print <expr> [<file-desc>])

This function is similar to the prin1 function except that it prints the result of the evaluation on a new line. (print (+ 3 8) prints 11 on a new line followed by a space and again 11 as returning value. As print always print on new line , do not include \n for line feed, other wise it will be considered as a part of the string.

3. (prompt <msg>)

This function displays <msg> on your screen's prompt area.

(prompt "Program for ABC Ltd. ")

2.8 Exercise for Participants

These exercise programs are not for real engineering use, but they are good enough to provide clues to those who are new to programming. You can study the assignment of variables, AutoLISP functions, method of deriving list from other lists, use of AutoCAD commands and how the results are obtained on drawing area as well as command prompt.

Intentionally, I am not supplying this code in actual .lsp file. I want you to write this program yourself, so that you can learn the techniques of debugging the code. **BEWARE, A SINGLE MISSING PARENTHESIS OR DOUBLE QUOTE [“] CAN PUT YOU IN HOT WATER.**

Exercise 1

```
;tank.lsp
;Programme for tank
;06-11-2006

;First obtain height of tank from user

(setq ht (getreal "enter height of tank :"))

;Obtining screen point for placing tank

(setq pt1 (getpoint "pick lower left corner of Tank :"))

;ht4 variable is 40 percent of ht, which is given by user, ht2 is half of ht, ht5
is half of ht, ht25 is one fourth of ht and htb2 is 20 percent of ht

(setq ht4 (* 0.4 ht))
(setq ht2 (/ ht 2.0))
(setq ht5 (* 0.5 ht))
(setq ht25 (* 0.25 ht))
(setq htb2 (* 0.2 ht))

(setq pt2 (list (+(car pt1)ht4)(cadr pt1)))
(setq pt3 (list (car pt2) (+(cadr pt2)ht2)))
(setq pt4 (list (+(car pt2)htb2)(cadr pt3)))
```

```
(setq pt5 (list (car pt4) (+(cadr pt4)ht25)))
(setq pt6 (list (car pt4) (-(cadr pt4)ht25)))
(setq pt7 (list (car pt2) (+(cadr pt2) ht)))

(setq pt8 (list (car pt1) (cadr pt7)))
(command "color" "red")
(command "line" pt1 pt2 pt3 "")
(command "line" pt3 pt4 pt5 pt6 "")
(command "line" pt3 pt7 pt8 pt1 "")
(command "color" "green")
(command "arc" pt1 "e" pt2 "a" 180)
(command "arc" pt7 "e" pt8 "a" 180)
```

Exercise 2

```
;culvert.lsp
;Simple road culvert
;06-11-2006

;First six lines of the programme is an attempt at setting some fixed values
;in variables. I have given variable names, which are not so meaningful.
;First you try with programme as written below and
;then observe output sketch drawn by programme. Later modify variable names with
;more meaningful names of your own choice. Run the program with modified
;variable names and compare both outputs.

(setq c 0.1)
(setq bt 0.3)
(setq tt 0.15)
(setq tw 0.45)
(setq bc 0.15)
(setq ad 0.6)

(setq ht2 (+ ad tt))

(setq ht3 (+ tt bt))
(setq w1 0.03)
(setq ht  (getreal "\n Enter height of culvert :"))
(setq len (getreal "\n Enter length of pipe :"))
(setq dia (getreal "\n Enter diameter pipe :"))
(setq nht2 (- ht ht2))
(setq bht(* 0.4 ht))
(setq stpt(getpoint "\n Pick bottom left corner of culvert :"))
(setq tap (/ len 60))
(setq w (- tw (* 2 c)))
(setq bht1(+ w c))
(setq ht1 (- ht ht3))
(setq ht4 (- ht (+ bt tt ad)))
(setq pt1 (list (+(car stpt) bht1 bht) (cadr stpt)))
(setq pt2 (list (car pt1) (+(cadr pt1) bt)))
```

```
(setq pt3 (list (-(car pt2) tt) (cadr pt2)))
(setq pt4 (list (-(car pt1) bht) (+ (cadr stpt) nht2)))
(setq pt5 (list (car pt4) (+ (cadr pt4) ht2)))
(setq pt6 (list (+(car pt5) c) (cadr pt5)))
(setq pt7 (list (car pt6) (+(cadr pt6) bc)))
(setq pt8 (list (- (car pt7) tw) (cadr pt7)))
(setq pt9 (list (car pt8) (- (cadr pt8) bc)))
(setq pt10 (list (+(car pt9) c) (cadr pt9)))
(setq pt11 (list (car pt10) (cadr pt3)))
(setq pt12 (list (car stpt) (cadr pt11)))

(setq len1 (/ len 2))

(command "color" "y")
(command "line" stpt pt1 pt2 pt3 pt4 pt5 pt6 pt7 pt8 pt9 pt10 pt11 pt12 stpt"")
(command "color" "r")
(setq pt13 (list (car pt11) (+ (cadr pt11) 0.6)))
(setq pt14 (list (car pt13) (+ (cadr pt13) dia)))
(setq pt15 (list (+(car pt14) len1) (cadr pt14)))
(setq pt16 (list (car pt15) (cadr pt13)))
(setq pt17 (list (car pt16) (cadr pt4)))
(setq pt18 (list (car pt17) (+(cadr pt17) tap 0.65)))
(command "line" pt5 pt18 "")
(command "line" pt4 pt17 "")
(command "pline" pt13 "w" w1 w1 pt16 "")
(command "pline" pt14 pt15 "w" 0 0 "")
(setq m (list (car pt16) (+ (cadr pt16) 5)))
(setq wi1 (list (+(car pt16) 3) (-(cadr pt16) 3)))
(setq wi2 (list (-(car pt8) 2) (+(cadr pt8) 2)))
(command "mirror" "w" wi1 wi2 "" pt16 m "")
(setq dpt1 (list (-(car pt14 )1) (cadr pt14)))
(setq dpt2 (list (car stpt) (-(cadr stpt) 1)))
(setq dpt3 (list (car dpt2) (-(cadr dpt2) 0.5)))
(command "dim" "ver" pt9 pt12 dpt1 "" "exit")
(command "dim" "hor" stpt pt1 dpt2 "" "exit")
(setq w2 (+ bht1 bht) )
(setq hi1(- ht ht3))
(setq ca1 (* tw tt))
(setq ca2 (* w2 bt))
(setq ba1 (* hi1 tw))
(setq ba2 (* 0.5 bht ht4))
(setq carea (+ ca1 ca2))
(setq barea (+ ba1 ba2))
;end of program
```

I am sure, when you finish with these two exercises, your confidence of searching points and manipulation of lists through car and cadr will be good enough to boost your thinking about actual problem solving.

VISUAL LISP EDITOR
SOME USEFUL FEATURES

Color Coding of Files

The text editor identifies different parts of an AutoLISP program and assigns distinct colors to them. This allows you to find program components easily such as function calls and variable names, and helps you find typographical errors.

Formatting of Text

The text editor can format AutoLISP code for you, making the code easier to read. You can choose from a number of different formatting styles.

Parenthesis Matching

AutoLISP code contains many parentheses, and the editor helps you detect missing parentheses by finding the close parenthesis that goes with an open parenthesis.

Execution of AutoLISP Expressions

You can test expressions and lines of code without leaving the text editor.

Multiple File Searching

The text editor can search for a word or expression in several files with a single command.

Syntax Checking of AutoLISP Code

The text editor can evaluate AutoLISP code and highlight syntax errors.

A. Color Coding:

As soon as you enter text in the VLISP editor, it tries to determine if the entered code is a built-in AutoLISP function, a number, a string, or some other language element. Visual LISP has designated specific colors to every type of element possible in program coding. This helps you detect missing quotes or misspelled function names. The default color scheme is below.

1. Pre-defined functions and protected variables	Blue
2. Integers	Green
3. Real Numbers	Teal
4. Strings	Magenta
5. Parenthesis	Red
6. Comments	Magenta on gray background
7. User symbols – variables	Black
8. Unrecognized items	Black

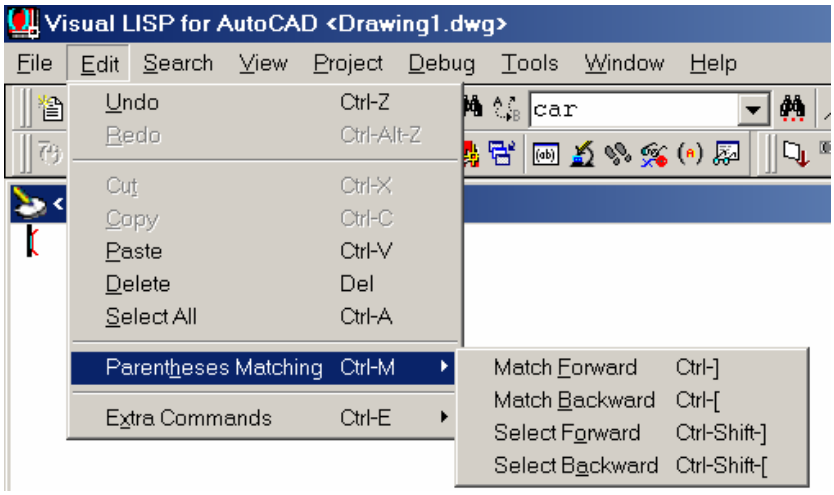
B. Parenthesis Matching

VLISP provides a parenthesis-matching feature to help you find the close parenthesis that corresponds to an open parenthesis. Earlier, matching of parenthesis must be a difficult task, resulting in calling LISP as Lost In Stupid Parenthesis.

To match an open parenthesis with corresponding close parenthesis…

1. Place your cursor in front of the opening parenthesis (For example on page 23, parenthesis precedes the setq function call)

2. Press CTRL+SHIFT+]. (Double-clicking also does the trick. The area between opening and closing parenthesis are selected)



C. Completing the word by Apropos

Sometime it becomes difficult for programmer to memorize all AutoLISP functions. If you are working on more than one language, there is a chance of confusion regarding use of appropriate function. VLisp editor helps you through *Complete Word by Apropos* facility.

Type in an expression similar to the one shown below:

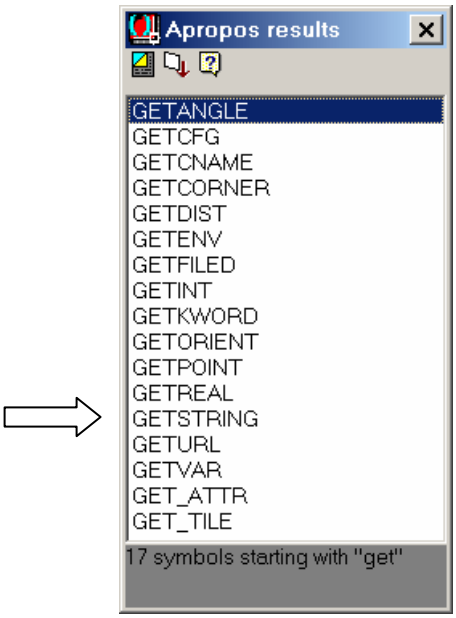
(setq name (getstring "\n Enter name of company : "))

Numbers of AutoLISP functions starts with ***get*** word. In above statement if you forget the exact name of function *getstring* , Visual LISP Complete Word by Apropos feature shall come to your rescue.

1. Enter the following on a blank line in your current AutoLISP session:

   (setq name (get

3. Press CTRL+SHIFT+SPACEBAR.



VLISP displays a list of all AutoLISP symbols that begin with the letters *get*. [As shown in figure].

D. Completing a Word by Matching

Using Complete Word by Match, completes a partially entered word by matching the part you have typed with another word in the same window. For example, suppose the following shows the history of your Console window:

```
(setq  d1    (getdist   "\n Enter distance between pulley : "))
(setq  Ldia  (getreal   "\n Enter left pulley diameter : "))
(setq  Rdia  (getreal   "\n Enter right pulley diameter : "))
```

to complete the word by matching….

1. Type the following as fourth line of code..

   ```
   (setq  ans  (g
   ```

2. Press CTRL+SPACEBAR to invoke Complete Word by Match. VLISP finds the last word you entered that began with letter "s" and completes the word you started to type:

   ```
   (setq  ans  (getreal
   ```

3. If that is not the word you are looking for, press CTRL+SPACEBAR again. VLISP searches back through the Console history for the previous occurrence of a word beginning with the letter "s":

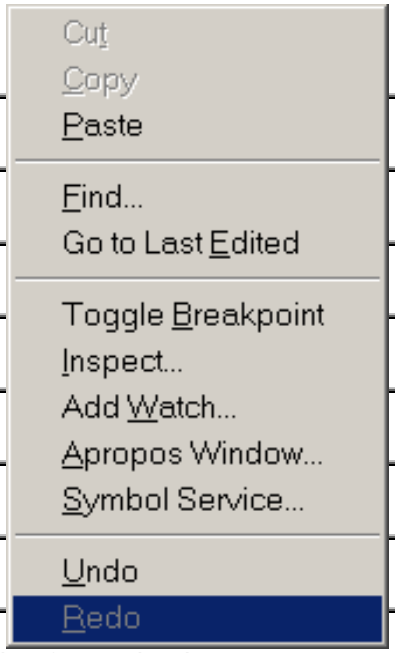
   (setq ans (getdist

E. Shortcut menu

The most important functions needed when working with the VLISP Console window are combined into a shortcut menu for fast access. **Right-click** anywhere in the Console window or press SHIFT+F10 to display the shortcut menu.

Depending on whether there is text selected in the Console window and depending on the cursor position, some commands may not be appropriate at the moment and cannot be activated from the shortcut menu. The following table summarizes the commands that may be available from the Console window shortcut menu.

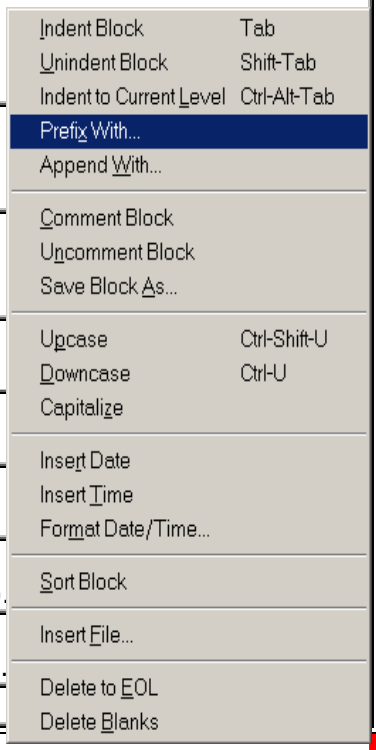
Console window shortcut menu commands	
Command	**Action**
Cut	Removes the selected text from the Console window and moves it to the Windows Clipboard
Copy	Copies the selected text to the Clipboard
Paste	Pastes the Clipboard contents to the cursor location
Clear Console window	Empties the Console window
Find	Finds specified text in the Console window
Inspect	Opens the Inspect dialog box
Add Watch	Opens the Watch window
Apropos window	Opens the Apropos window
Symbol Service	Opens the Symbol Service dialog box
Undo	Reverses the last operation
Redo	Reverses the effects of the previous Undo
AutoCAD Mode	Transfers all input to the AutoCAD command line for evaluation
Toggle Console Log	Copies Console window output to the log file



F. Formatting commands

Press **CTRL+E** while in an active VLISP text editor window to display a list containing the following editor options.

Text editor code formatting commands	
Option	**Effect**
Indent Block	Indents the selected block of text by adding a tab to the beginning of each line.
Unindent	Unindents the selected block of text by removing a tab.
Indent to Current Level	Indents the current line to the same level as the previous line of program code.
Prefix With	Adds a text string to the beginning of the current line, or to each line in a block of selected lines, after prompting you for the string.
Append With	Appends a text string to selected lines of text, after prompting you for the string.
Comment Block	Converts a block of code to comments.
Uncomment Block	Changes a block of comments to active text.
Save Block As	Copies selected text to a new file.
Upcase	Converts the selected text to all uppercase.
Downcase	Converts the selected text to all lowercase.
Capitalize	Capitalizes the first letter of word in the selected text.
Insert date	Inserts the current date (default format is MM/DD/YY).
Insert time	Inserts the current time (default format is HH:MM:SS).



Format Date/Time	Changes the date and time format.
Sort Block	Sorts the selected block of code in alphabetical order.
Insert File	Inserts the contents of a text file into the current editor window at the cursor position.
Delete to EOL	Erases everything from the cursor position to the end of the current line.
Delete Blanks	Deletes all blank spaces from the cursor position to the first non-blank character in the line.

My Dear participants, in next segment we will try to explore String handling functions, Data type conversion functions, functions useful for controlling user input, user defined functions and branching and decision making functions.

I know – When I am teaching, I am learning more.

Remember that this material is only a portion of the class, support is always available online in the private course forum. I encourage you to visit the course forum and ask any questions that you may have about this segment or simply join in the discussion. The ATP Mantra is: the only stupid question is the one you don't ask. *Thanks again for attending this course!*