DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

CURSO: AUTOLISP LENGUAJE DE PROGRAMACION PARA AMBIENTE AUTOCAD
FECHA: 10 al 26 DE NOVIEMBRE, 1997

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
ING. OSCAR MARTIN DEL CAMPO				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF.
ORGANIZACIÓN Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACIÓN DEL CURSO	
APLICACIÓN PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

CONCEPTO	CALIF.
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDÁCTICO UTILIZADO	

Promedio _____

Evaluación total del curso_____

Continúa...2

1. ¿Le agradó su estancia en la División de Educación Continua?

SI ☐ NO ☐

Si indica que "NO" diga porqué: _____

2. Medio a través del cual se enteró del curso:

Periódico *Excélsior*	
Periódico *La Jornada*	
Folleto anual	
Folleto del curso	
Gaceta UNAM	
Revistas técnicas	
Otro medio (Indique cuál)	

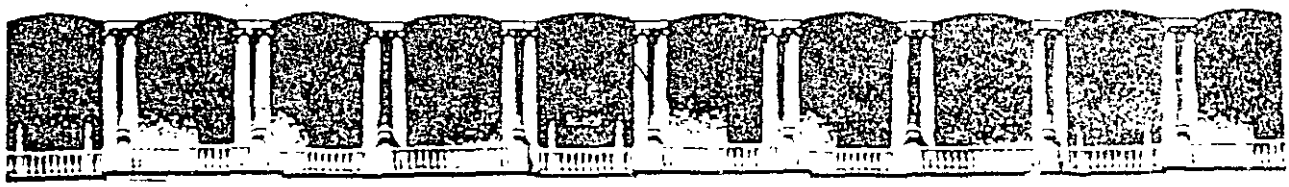
3. ¿Qué cambios sugeriría al curso para mejorarlo?

4. ¿Recomendaría el curso a otra(s) persona(s) ?

SI ☐ NO ☐

5.¿Qué cursos sugiere que imparta la División de Educación Continua?

6. Otras sugerencias:



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

MATERIAL DIDACTICO DEL CURSO

AUTOLISP
(LENGUAJE DE PROGRAMACION PARA AMBIENTE AUTOCAD)

NOVIEMBRE, 1997

Palacio de Mineria Calle de Tacuba 5 Primer piso Deleg. Cuauhtemoc 06000 Mexico, D.F. APDO. Postal M-2265
Telefonos 512-8955 512-5121 521-7335 521-1537 Fax 510-0573 521-4020 AL 26

ÍNDICE

LISTAS

características de las listas

Asignación de valores a listas

FUNCIONES

Estructura de una función

Tipos de argumentos de una función

Como evalúa las funciones AutoLISP

Asignación de valores a variables

ENTIDADES

Características de las entidades

Manipulación de entidades

GLOSARIO DE FUNCIONES

Funciones matemáticas

- funciones aritméticas
- funciones logarítmicas y trigonometricas

Funciones de manipulación de listas

Funciones de asignación

Funciones procedimentales

Funciones condicionales

Funciones relaciónales

INTRODUCCIÓN

AutoLISP es un lenguaje de programación derivado del lenguaje para aplicaciones con inteligencia artificial conocido como Commom LISP, el cual es un lenguaje orientado a objetos, los que maneja o trabaja con listas de símbolos. Esto contrasta con otros lenguajes de programación que solo procesan instrucciones y datos numéricos. Además, AutoLISP es un lenguaje de programación que es "evaluado" en vez de ser interpretado o compilado como los demás.

El nombre de LISP proviene de List Processing (procesamiento de listas). Fue desarrollado en los años 50, es uno de los lenguajes de programación mas antiguos aún en uso. Muchas características nuevas le han sido añadidas con eso transcurso de los años para convertirlo en un lenguaje de programación sumamente útil para muchas aplicaciones.

En particular en los programas de diseño asistido por computadora CAD, AutoLISP se vale de ciertas funciones especiales del **Commom LISP** que han sido específicamente adaptadas para la manipulación de los dibujos creados con AutoCAD. Debido a ello, AutoLISP es un verdadero lenguaje de programación y no un simple lenguaje de macros.

ESPACIO DE TRABAJO Y MEMORIA USADA POR AUTOLISP

El comando **SET** del sistema operativo DOS permite ajustar la cantidad de memoria **RAM** que AutoCAD emplea como aérea de trabajo, así como la cantidad de memoria RAM que se aparta para las funciones de AutoLISP. También puede controlar la cantidad y la ubicación de la memoria extendida y(o) expandida que usa AutoCAD para trabajar con los archivos temporales.

La memoria **RAM** se puede ajustar con la ayuda del comando **SET** agregando las

siguiente línea en el archivo de **AUTOEXEC.BAT**

SET = ACADFREERAM = [Numero de Kb]

Los valores máximo que acepta **ACADFREERAM** son de 30k (20K a 24K para las versiones 2.6 a las 9 de AutoCAD). El valor por default es de 24K.

En contraposición especificar un valor mas bajo para **ACADEFREERAM** aumentara la capacidad de memoria convencional o **RAM**, lo cual permite que AutoCAD trabaje mas fijo. Por tal motivo puede que le convenga reducir este valor hasta el mínimo, Esto mediante la expresión:

SET ACADFREERAM = 5

AJUSTE DE LA MEMORIA DISPONIBLE PARA LAS FUNCIONES DE AUTOLISP

AutoCAD utiliza dos tipos de memoria para utilizar AutoLISP: La memoria **HEAD** y la memoria **STACK**. En términos muy generales, las memoria **HEAD** es la que se usa para almacenar funciones y variables y también se conoce como "espacios nodos"; mientras que la memoria **STACK** almacena argumentos y resultados parciales durante la evaluación de expresiones.

El valor por default para la memoria **LISPHEAD** es de 40000 bytes, mientras que para la **LISPSTACK** es de 3000 bytes (o 5000 bytes cada uno hasta la versión 9 de AutoCAD). Por lo general estos valores son suficientes, no obstante, en caso de visualizar el mensaje:

INSUFFICIENT NODE SPACE (Espacio de nodos insuficiente).

Mientras se este ejecutando un programa de AutoLISP aumente estos valores

introduciendo en el archivo de **AUTOEXEC.BAT** Las siguientes líneas:

SET LISPHEAP = 42000

SET LISPSTACK = 3000

La restricción que se tiene es que la sume de las dos memorias no debe sobrepasar el valor de 45000 bytes.

MEMORIA EXTENDIDA PARA AUTOLISP

Si posee una computadora con un procesador 80286 en adelante y la versión 10 de AutoCAD, usted puede aprovechar la memoria extendida para ejecutar programas mas extensos y mas complejos de AutoLISP.

Para que se pueda usar la memoria expandida, la computadora debe funcionar bajo una versión de DOS superior a la 2.0, disponer por lo menos de 64K de memoria convencional y contar con 512K de memoria extendida disponible.

Para usar el AutoLISP extendido debe contarse con los siguientes archivos en el subdirectorio de Acad:

ACADLX.OVL

EXTLISP.EXE

REMLISP.EXE

ACADLX.OVL Es un archivo que contiene un programa especial de superposición [overlay] que AutoCAD usa para el AutoLISP extendido, mientras que **EXTLISP.EXE** es un programa residente en memoria que debe ser cargado a la computadora antes de AutoCAD. Por su parte, **REMLISP.EXE** es un programa que elimina de la memoria el programa

EXTENLISP.EXE, cuando usted necesite mas cantidad de memoria para las otras aplicaciones.

Con estos archivos ubicados en su directorio de acad y trabajando desde ese mismo directorio, teclee **EXTENLISP** y pulse [enter] y a continuación proceda con el arranque de AutoCAD. Seleccione la opción 5 del menú principal[main menú], para pasar al menú de configuración. Visualizara un listado con la configuración actual que tiene la computadora.

Seleccione la opción 8 para pasar al submenú de configuración de los parámetros operacionales de AutoCAD. Ahora, elija la opción 7 para seleccionar las características que le permitieran trabajar con AutoLISP.

AutoCAD le preguntara si quiere activar AutoLISP conteste que si. A continuación el programa le preguntara si desea usar el programa AutoLISP extendido conteste que si y regrese al menú principal. Al completar este procedimiento AutoCAD estará listo para usar la memoria extendida para AutoLISP. A menos que cambie nuevamente la configuración, no olvide cargar de aquí en adelante el programa **EXTLISP.EXE** antes de cargar AutoCAD.

MEMORIA EXTENDIDA PARA LOS ARCHIVOS TEMPORALES Y PARA TRABAJO CON EXTLISP

Para especificar la cantidad de memoria extendida que AutoCAD necesita para almacenar los archivos temporales, del dibujo que estamos editando, se usa el comando **SET** , para ello antes de arrancar AutoCAD, a partir del sistema operativo teclee la siguiente línea:

SET ACADXMEM=[UBICACIÓN INCIAL, TAMAñO]

El valor de la ubicación inicial puede estar entre 1024K y 16384K, siendo 1024K el comienzo de la memoria extendida.

Por ejemplo si nosotros tenemos una computadora con 512K de memoria extendida, de los cuales queremos asignar a AutoCAD los 256K mas altos. Para esto se pondría la siguiente línea:

SET ACADXMEN=1280K,256K

Debe cuidar no dejar espacios en blanco entre las especificaciones de **ACADXMEN**. La ubicación inicial en este ejemplo es de 1280K, El resultado de sumar 256K a 1024K. Usted puede omitir el segundo valor, en caso que desee dedicar toda la memoria extendida excedente a AutoCAD. Igualmente puede omitir la ubicación inicial, en caso que desee que AutoCAD la seleccione automáticamente; para ello, proceda el seguido valor con una coma. Finalmente, puede forzar a AutoCAD a ignorar la memoria extendida, introduciendo la palabra none en lugar de los valores.

La ubicación de memoria de **EXTLISP**, trabaja de la misma forma que para los archivos temporales. Solo que en lugar de usar **ACAXMEN**, se usa **LISPXMEN**. Las líneas para la ubicación de memoria de archivos temporales y de AutoLISP se pueden incluir el archivo autoexec.bat si es necesario.

ALMACENAMIENTO Y CARGA DE PROGRAMAS EN AUTOLISP

Algunas veces solo necesitamos de pequeños funciones para hacer una tarea muy pequeña que simplemente tecleando las líneas desde la línea de comandos, esto puede ser por ejemplo si queremos sumar dos números, es una tarea muy simple que no requiere hacer un programa, pero en la mayoría de las tareas que queremos realizar es necesario hacer programas relativamente largos que sería muy latoso escribirlos cada vez que que necesitemos usarlas, por lo algo mas eficiente sería guardarlos en archivos. Estos archivos los podemos hacer desde un editor que maneje los caracteres en ascii, y no en un editor de

palabras que tenga caracteres especiales como centrado de pagina, sangría, etc. Estos archivos deben tener la extensión lsp, que es la extensión que AutoLISP reconoce para los archivos ejecutables.

Cuando ya se almaceno el archivo con extensión lsp, lo siguiente que hay que hacer es entrar al entorno de AutoCAD para cargar estos archivos en la memoria y poder utilizarlos posteriormente. La forma de cargar un archivo en AutoCAD es utilizando la función **Load**, la cual se explicara mas adelante.

Al utilizar la función load los archivos se cargan uno por uno, esto puede ser muy laborioso si queremos utilizar algunos programas muy comunes para nosotros. Existe la posibilidad de indicarle a AutoCAD que cargue todos los archivos automáticamente, Esto se puede hacerse cargando los archivos que utilizamos mas frecuentemente en el archivo acad.lsp que es un archivo que AutoCAD, carga automáticamente al iniciar. Los nombres de los archivos que agreguemos a acad.lsp deben ser los mas usados, porque al cargar estos archivos los almacena en la memoria y esto hará que el funcionamiento de AutoCAD sea mas lento.

COMO TRABAJA AUTOLISP

AutoLISP es un lenguaje orientado a la manipulación de símbolos, por ello se encuentra que la forma de programar con él es totalmente diferente a la utilizada con los otros lenguajes de programación convencionales como el **BASIC**, **PASCAL** o **C**.

AutoLISP procesa listas de símbolos en vez de datos numéricos. La mayoría de los otros lenguajes de programación están diseñados para procesar solo datos numéricos. Los lenguajes de programación como el **FORTRAN**, **BASIC** y el **C** están orientados matemáticamente o numéricamente, esto es, son utilizados para manipular números.

AutoLISP es un lenguaje que es evaluado en vez de ser interpretado o compilado. Los lenguajes de programación normales leen el texto del programa línea por línea y lo convierten en instrucciones de maquina, generando un archivo de código de maquina, el cual es ejecutado rápidamente. Mientras que AutoLISP se encuentra entre un lenguaje interpretado y compilado. Cuando se encuentra por primera vez un bloque de código, este se convierte en código compacto; si dicho bloque se encuentra de nuevo mientras se ejecuta el programa, el evaluador detecta que ya ha sido evaluado lo ejecuta, sin la necesidad de volver a procesar el bloque, con lo que la velocidad del programa se aumenta.

Resumiendo existen tres características de AutoLISP que lo distinguen de la mayoría de los otros lenguajes de programación:

AutoLISP manipula símbolos en vez de números

Es un lenguaje orientado al objeto en vez de ser lenguaje procedimental.

Es un lenguaje que evalúa en vez de interpretar o compilar.

ADICIÓN DE NUEVAS ORDENES A AUTOCAD

AutoLISP le permite crear sus propias ordenes personales para AutoCAD que pueden ser ejecutadas en la línea de comandos como cualquier otra orden de AutoCAD. Puede incluso redefinir una orden para hacer cosas completamente diferentes de las que originalmente hacia. Por ejemplo, supóngase que se hizo una aplicación donde buscaba que el único tipo de línea valida en un dibujo fuera una polilínea. La gente que utiliza el símbolo podría olvidar y utilizar a veces la orden línea. Para evitar que ocurra este problema podría redefinir la orden línea para que significa polilínea.

FUNCIONAMIENTO DE AUTOLISP EN AUTOCAD

Todo lo que usted teclea es la línea de comandos es enviado a la rutina **interprete de comandos**. El interprete de comandos es una función de AutoCAD que lee las ordenes que usted introduce y las compara con todas las ordenes que pueden ser ejecutadas. Si la orden que ha introducido coincide con alguna de las ordenes, AutoCAD sabe que hacer a continuación. Por el ejemplo si teclea el comando line el interprete de comandos enviara un mensaje a la rutina Line de AutoCAD y la orden line comenzara a ejecutarse.

Si el interprete de comandos encuentra un paréntesis abierto, supone que todo lo que viene a continuación es una expresión de AutoLISP hasta que encuentra el correspondiente paréntesis de cierre. Envía esta expresión el evaluador de AutoLISP para su evaluación. Además si el interprete de comandos encuentra el símbolo de exclamación, seguido inmediatamente por una palabra, comenzara a hacer una búsqueda para ver si la palabra esta definida como un símbolo de AutoLISP. Imprimiría entonces el valor que le ha sido asignado a ese símbolo en la siguiente línea de la pantalla, si no ha sido asignado ningún valor a la variable de AutoLISP se escribirá la palabra nil, esta palabra la utiliza AutoLISP

para indicar que el contenido del símbolo esta vacío, que es diferente a un valor de cero. AutoLISP siempre retornada el valor de nil aunque el símbolo no este definido.

TIPOS DE DATOS EN AUTOLISP

Hay diez tipos de datos en AutoLISP:

Enteros **(INT)**

Números en coma flotante**(REAL)**

Cadenas de caracteres **(STR)**

Símbolos **(SYM)**

Listas **(LIST)**

Archivos **(FILE)**

Subrutinas **(SUBR)**

Conjuntos designados **(PICKSET)**

Tablas de asignación **(PAGETB)**

Nombre de entidades **(ENAME)**

El LISP esta diseñado para utilizar un rango ilimitado de datos. El AutoLISP es solo un ejemplo de un sistema LISP. Los tipos de datos soportados por un sistema especifico LISP deben ser creados y modificados en el lenguaje por los programadores que lo crearon. Usted no podrá crear normalmente sus propios tipos de datos.

En AutoLISP no se le asigna un tipo de datos a un símbolo, si no que el símbolo puede recibir datos de cualquier tipo, puede incluso cambiar el tipo de datos solo cambiando el valor asignado al símbolo. Por ejemplo no se le tiene que asignar a un dato que es de tipo entero, y si le queremos dar una cadena, marcaría error, si no que simplemente le asignamos tipos de datos diferentes sin importarnos que valor tenía antes.

La posibilidad de reutilizar símbolos con diferentes tipos de datos puede ser muy útil

por que permite al programador una gran flexibilidad al escribir el código. La parte negativa es que se tiene que poner mucho cuidado en no perder el rastro del tipo de datos que ha asignado a los símbolos ya que podría acabar pasando un tipo erróneo a una función.

ENTEROS

Son números sin fracción decimal. Estos números pueden ser positivos o negativos, Cualquier operación que se realice solo con números enteros dará un resultado de un número entero. Esto puede causar muchos problemas si se quieren los resultados de operaciones como la división que puede dar un número con fracción decimal, la cual no será tomara en cuenta en el resultado, provocando errores.

REALES

Son todos los números con punto y fracción decimal . Las operaciones que se realicen con números reales darán números reales. Cuando se realiza una operación con reales y enteros el resultado dará real.

CADENAS DE CARACTERES

Las cadenas de caracteres son todos aquellos valores que no son números y que se manejan como caracteres. Todas las cadenas de caracteres van delimitadas por comillas "ejemplo", de no poner las comillas AutoLISP pensara que se trata de una variable y la tratara de evaluar produciéndose un error.

DESCRIPTOR DE ARCHIVOS

Son referencias indirectas que se establecen entre la maquina y el usuario para poder usar un archivo ya sea para lectura o para escritura. Estas referencias varían dependiendo de. la posición en que se encuentre el archivo en la memoria, así un descriptor de archivo que manejemos hoy será diferente al descriptor de archivos que manejemos mañana, esto por que la posición de al archivo en memoria cambio.

LISTAS

Son todos los tipos de datos fundamentales de AutoLISP. Las listas se delimitan por los paréntesis (lista). Todo lo que se encuentre dentro de los paréntesis será tomado como parte de la lista, la cual incluso puede estar formada por sublistas.

SÍMBOLOS

Los símbolos para AutoLISP son las variables que reciben las funciones. Un símbolo es simplemente una cosa que se refiere a otra cosa. El dibujo de una silla, por ejemplo, es un símbolo que representa una silla de la misma forma que lo hace la palabra silla.

La representación visual puede ser un medio diferente al de la palabra impresa, pero el objetivo de los dos medios es el mismo. El propósito es el de comunicar algún significado de una persona a otra.

AutoCAD utiliza los símbolos, los cuales deben ser creados y estar asociados a algún significado. Y cuando se quieran utilizar solo se llamaran e insertaran en el dibujo.

EN AutoLISP la definición de un símbolo se representaría por ejemplo:

(setq x 5)

El símbolo x recibirá el valor 5, por lo que el símbolo x será manejado como si fuera un entero con valor de 5.

RESTRICCIONES EN EL USO DE LOS SÍMBOLOS

Cuando usted asigna un valor a un símbolo, esta ligando este valor al símbolo. El concepto de ligadura en LISP es diferente de la asignación de un valor a un nombre de variable encontrado en otros lenguajes de programación. En AutoLISP los símbolos pueden recibir cualquier tipo de datos con solo el hecho de cambiarles el valor, esto difiere de los demás lenguajes en que si se le asigna un tipo de dato a una variable, esta variable solo

puede recibir valores de ese tipo. La variable representa un recipiente para un cierto tipo de valores, en cambio para AutoLISP el símbolo solo representa el valor. El valor, no el símbolo, determina el tipo de información que esta representada.

Un símbolo puede estar ligado a una función o a un valor pero no a ambos a la vez. El símbolo es la el bloque básico en la construcción de la lista. Un símbolo puede estar ligado a los valores de cualquier tipo de datos disponibles incluso a otros símbolos.

En AutoLISP deberá evitar un nombre de símbolo con mas de seis caracteres. Puede utilizar tantos caracteres como desee, pero los nombres de símbolos que tengan mas de seis letras utilizan mas memoria. Por otra parte, con los nombres de símbolos pequeños, el código es mas rápido al ejecutarlo.

LISTAS

Para comenzar a trabajar con AutoLISP es necesario comprender, manejar y saber como construir listas. Las listas siempre están expresadas como elementos mostrados entre paréntesis y separados por espacios. Los símbolos tienen una utilización mas general que las listas.

En AutoLISP, hay solo una clase de estructura, la lista. Una lista esta formada por símbolos (a veces referidos como palabras o átomos), números, cadenas o cualquiera de los otros tipos de datos validos. Una lista puede incluso estar formada por otras listas. Las listas son de dos tipos básicos:

La lista estándar (normalmente llamada lista)

la lista sin evaluar . Una lista sin evaluar es una lista que esta precedida por el símbolo de no evaluación (normalmente le apóstrofe). El apóstrofe le indica a AutoLISP que no debe evaluar esta lista.

La regla básica sobre las listas es que a menos que sea una lista sin evaluar, su primer elemento debe ser un nombre de función valido de AutoLISP, seguido del numero correcto de argumentos que corresponde a esta función.

(función argumento1 argumento2 ... argumentoN)

A continuación se dan algunos ejemplos:

(* 5 5)

la lista anterior regresara el valor de 25

(+ (* 2 2) (* 3 2))

Esta lista regresara el valor de 10 puesto que primero se evalúan los datos contenidos en los paréntesis interiores y por ultimo el valor del paréntesis exterior.

Una lista sin evaluar seria

(setq a'(+ 4 4))

al llamar a el símbolo a AutoLISP regresaría

(+ 4 4)

puesto que el símbolo (') indico a AutoLISP no evaluar lo que estaba contenido en a.

ASIGNACIÓN DE VALORES A LISTAS

El mayor uso que se le puede dar a asignar a un símbolo una lista, es que el valor de esa lista sea una coordenada, que nos sirva para poder realizar alguna tarea posteriormente. Existen dos formas de crear una lista:

Con la función **list** o anteponiendo a la lista el apóstrofe con lo que se considera una lista sin evaluar, esto se hace para que no se provoque un error, puesto que AutoLISP tomaría el primer elemento de la lista y al no ser una función provocaría un error.

Ejemplo:

(setq punto (list 1 2 3))

(setq punto '(1 2 3))

Esto hace que al símbolo punto se le asigne la lista **(1 2 3)**

Otras funciones que permiten agregar elementos a una lista son: la función **cond** que agrega un elemento al principio de una lista y la función append que junta varias listas en otra lista.

Otras funciones importantes para el manejo de listas son las que me permiten extraer elementos de una lista para su análisis individual estas funciones son **CAR** y la **CDR**. La función car extrae el primer elemento de una lista y la función cdr devuelve la lista sin su primer elemento. Se pueden hacer combinaciones con estas dos funciones las cuales mas adelante se explicaran.

FUNCIONES

Una función es un conjunto de instrucciones en LISP que dice a la computadora que hacer específicamente con algo. Es el bloque básico de un programa LISP Hay dos tipos de funciones en AutoLISP; un tipo correspondiente a las funciones integradas, que son parte de AutoLISP y el otro a las funciones que define usted mismo.

Las funciones integradas incluyen las funciones matemáticas:

+ - * /

También están las funciones de manipulación de listas:

cons car Cdr list

Hay funciones únicas de AutoCAD tales como:

getpoint entget tblsearch grread

Estas funciones integradas son llamadas primitivas de AutoLISP, usted puede definir sus propias funciones, además de las funciones integradas si utiliza la palabra **defun**.

No Hay operadores solo funciones. La mayoría de los lenguajes de programación tiene operadores, procedimientos, funciones y ordenes. Cada una de estas estructuras debe ser utilizada de forma diferente cuando se escribe el programa. AutoLISP solo tiene funciones, que son manejadas siempre en forma idéntica. Esto es lo que hace a AutoLISP fácil de usar en algunas ocasiones y que sea confuso en otras.

Una función en AutoLISP tiene cuatro cualidades:

* La mayoría de las funciones requieren argumentos o parámetros. Un argumento o parámetro es una variable independiente que determina la salida de la función. La función + , por ejemplo, requiere al menos de dos variables que son sumadas para obtener el resultado. El resultado de + depende del valor de argumentos. por ejemplo. la función (+ 4 8) retorna un valor diferente que (+ 9 8)

* Una función siempre retorna un valor. Muchas funciones retornan un valor nil. La función + , por ejemplo, devuelve la suma de argumentos. Algunas funciones son utilizadas por el efecto "lateral" que producen y retornan un valor que normalmente es ignorado; en tales casos, el valor retornado es nulo. la función "(terpri)" es un ejemplo de una función que no utiliza argumentos, tiene un efecto lateral y devuelve un nulo. "terpri" (TErminal PRInt) dice a AutoCAD que imprima una línea de texto en blanco en la pantalla.

* Cualquier símbolo declarado dentro de la lista de parámetros de una función esta ligado a los valores solo mientras la función esta en uso. Esto se conoce como una ligadura local. Si un símbolo esta ligado a un valor fuera de una función (global) y este mismo símbolo es declarado dentro de la lista de parámetros en una función, entonces este símbolo esta ligado a un nuevo valor mientras la función se esta ejecutando. Volverá al valor de ligadura cuando termine la función.

Una función devuelve siempre el valor del ultimo símbolo lista evaluada dentro de esta función. por ejemplo:

```
(defun cubo ( x )
(type x )
( * x x x )
)
```

el valor retornado por esta expresión es **INT**. vera que el valor retornado por cubo fue el resultado de la evaluación de la ultima función y el valor de type es ignorado.

ESTRUCTURA DE UNA FUNCIÓN

Una función tiene la siguiente estructura general:

(Nombre_de_la_función arg1 arg2 arg3 .... argN)

Para que AutoCAD reconozca que se trata de una función de AutoLISP es necesario que la función este encerrada entre paréntesis.

El primer elemento de cualquier función es siempre el nombre de la función. Si AutoLISP encuentra el primer elemento y no es ningún elemento definido marcara error.

Después del nombre de la función van los argumentos. Como se puede ver los argumentos van separados por espacios en blanco y pueden ser desde cero hasta N. El numero y tipo de argumentos dependerá de la función que se este ejecutando así para cero argumentos se tiene la función (terpri) la cual imprime una línea en blanco, para N argumentos se tiene por ejemplo la función de suma.

TIPOS DE ARGUMENTOS DE UNA FUNCIÓN

Los argumentos son todos los valores que se le pasan a las funciones para que ellas puedan realizar las tareas para las que fueron diseñadas . Los principales y mas usuales tipos de argumentos que maneja AutoLISP y la forma en que se manejaran mas adelante para explicar las funciones son:

<lista-asociación> Listas de asociaciones

<Ángulo> Que se miden en radianes

<AtomoN> Puede ser un nombre de entidad valido (line, circle, arc, etc.).

<ExprN> Puede ser cualquier expresión valida en AutoLISP

<descriptor-archivo> Un descriptor de archivo es un tipo de dato devuelto por la función open, este descriptor es el nombre de un archivo pero en formato que entienda que pueda entender la maquina.

<Función> Puede ser cualquier nombre valido de función definida

<argN> Cualquier tipo de dato valido

<lista> Cualquier lista valida

<numeroN> Puede ser un entero o un real

<indicadorN> Debe ser una cadena, pero no necesariamente una literal

<punto> debe ser una lista de dos o tres reales que describe una coordenada

<cadenaN> Debe ser una cadena

<sim> Debe ser un nombre de símbolo

<Conjunto-selección> Un conjunto de selección

<Nombrevar> Un nombre de variable de dibujo de AutoCAD como una cadena

COMO EVALÚA LAS FUNCIONES AUTOLISP

Cuando el evaluador de AutoLISP encuentra una lista, lo primero que hace es mirar su primer símbolo. Si encuentra que el primer en la lista es el nombre de una función valida, pasa el resto de la lista a la función para ejecutar el código de esta función. Cuando la función termina de evaluarse devuelve el resultado de la misma. por ejemplo:

orden: **(+ 4 8)**

12

es lo que sucede. El evaluador de AutoLISP coge la lista, busca el primer símbolo (el símbolo es +) y encuentra una función llamada +, El código para la función + dice al evaluador que tome el segundo símbolo de la lista (4) sume otro símbolo (80 a él y retorne el resultado (12). Si quiere utilizar el resultado de esta función tiene que asociar este valor a otro símbolo con una expresión, o bien pasarlo a una función de mayor nivel en una expresión anidada.

En LISP como en el álgebra, se evalúa siempre la expresión desde dentro hacia afuera.

ASIGNACIÓN DE VALORES A VARIABLES

A las variables se les asignan valores con la función setq. Una función puede ser un simple operador, como el de suma, pero también puede consistir de un conjunto de instrucciones más complejas igual que un programa.

Ejemplo:

(setq área 10)

Para obtener el valor de una variable, dentro del ambiente de AutoCAD, se teclea un signo de admiración ! y el nombre de la variable:

!área

Devuelve 10

MANIPULACIÓN DE ENTIDADES

En AutoCAD se le da el nombre de entidad a cualquier objeto dibujado en la pantalla, así por ejemplo una línea, un circulo o un texto son entidades. Cada entidad tiene un nombre que es reconocido por AutoCAD, este nombre cambia cada vez que se elabora un dibujo, por lo que sería no adecuado intentar recordar cada uno de ellos.

AutoLISP usa funciones de entidades para preguntar por el nombre de entidades de la base de datos del dibujo. El nombre que regresa esta en forma de código el cual entiende AutoCAD, por ejemplo si queremos el nombre de una línea AutoCAD nos devolverá algo parecido a lo siguiente:

<Entity name: 6000006a>

El conocer el nombre de un objeto nos puede servir para poder traer de la base de datos los datos relacionados con la línea que estamos manejando. Los datos son devueltos en forma de una lista la cual esta compuesta de sublistas, cada una de ellas indicando cierta característica de la entidad, así por ejemplo los datos que puede traer son parecidos a lo siguiente:

(-1 . <Entity name:600006a>) (0 . "LINE") (8 . "0") (10 1.000 1.000) (11 2.000 2.000))

Se puede observar que la lista esta formada por sublistas las cuales tienen como primer elemento un número, este número es el código DXF que maneja AutoCAD para indicar que tipo de propiedad esta manejando así por ejemplo el código -1 nos esta indicando el nombre que tiene la entidad, el 0 el tipo de entidad, etc. El segundo elemento es el valor de la propiedad: Si la sublista tiene dos elementos se representara como una lista de asociación la cual separa los elementos por un punto. Si tiene más elementos se guardara como una lista normal. Principalmente este tipo de listas nos regresan las coordenadas de los puntos que definen la entidad.

La sintaxis que se manejan para las propiedades es:
Si la lista es de dos elementos
(código . propiedad)
Si la lista es de más de dos elementos
(código elemento1 elemento2 ... elementoN)

A continuación se muestran los principales códigos DXF que se manejan en AutoCAD:

-1	Nombre de la entidad
0	Tipo de entidad
6	Tipo de línea
7	Estilo de texto
8	Nombre del layer donde esta la entidad

Para una línea

10	Punto inicial
11	Punto final Para un circulo
10	Centro
40	Radio

Para un arco

10	Centro

40	Radio
50	Ángulo inicial
51	Ángulo final

Para bloques insertados

2	Nombre del bloque
10	Punto de inserción
41	Escala en x
42	Escala en y
43	Escala en z
50	Ángulo de inclinación (en radianes)

Las funciones manejan a los entidades son la que comienzan con el prefijo ENT. Así por ejemplo para traer el nombre de una entidad existen las funciones:

(Entnext) la cual trae la primera entidad de la base de datos. Si a la función entget se le pone como argumento el nombre de una entidad (entnext entidad5) el nombre de entidad que traerá será la siguiente de la que fue puesta como argumento (traería la entidad 6).

(Entlast) la cual trae la ultima entidad dibujada en el dibujo.

(Entsel) la cual nos permite seleccionar la entidad que nosotros queremos.

(entdel) borra una entidad de la base de datos

(entget) trae de la base de datos las propiedades de la entidad.

GLOSARIO DE FUNCIONES

A continuación se explicaran en forma detallada las funciones que maneja AutoLISP, se explicara lo que puede hacerse con cada función y se pondrá un ejemplo para el mejor entendimiento.

FUNCIONES MATEMÁTICAS

Las funciones matemáticas se dividen en dos partes: funciones aritméticas básicas y funciones logarítmicas y trigonométricas. Las funciones siempre devolverán un tipo de dato (número). El que el resultado devuelto sea un entero o un real depende del tipo de los datos de la lista de argumentos. Si todos los argumentos de una lista son enteros en resultado será entero, si al menos uno de los argumentos es real, el resultado será un real.

FUNCIONES ARITMÉTICAS.

AutoLISP contiene todas las funciones aritméticas básicas. AutoLISP utiliza la notación prefija, esto significa que el operador esta situado antes del operando, como por ejemplo **(+ 4 8)**. Normalmente, si usted quiere sumar 4 y 8 diría 4 mas 8. Si mira en el ejemplo y ve que + es una función que suma argumentos presentados a continuación y retorna un resultado, diría sumar 4 a 8.

Todas las funciones aritméticas pueden tener uno o mas argumentos. AutoLISP aplica la misma operación a todos los elementos esto es no es necesario indicar a AutoLISP que vuelva a sumar, por ejemplo

(+ 3 4 5 10)

da como resultado 22

(/ 16 4 2)

retorna 2

en la ultima función AutoLISP toma el 8 y lo divide entre 4, el resultado 4 lo divide entre el 2 dando como resultado 2.

es importante hacer notar que AutoLISP despliega el resultado de la ultima función evaluada y no los intermedios.

Otro aspecto importante lo vamos a notar en el siguiente ejemplo:

(/ 190 8 4 3)

da como resultado 1

como puede verse rápidamente en su calculadora el resultado correcto es 1.9779167. Pero al meter a la función solo números enteros AutoLISP trabajo solo con los enteros y no tomo en cuenta las fracciones, para que no subiera dado el resultado correcto es necesario meter en la función al menos un numero real así por

ejemplo:

(/ 190 8 4 3.0)

da como resultado 1.979167

Otra cosa que debe resaltar un numero real siempre debe empezar con un entero incluso si es cero, así por ejemplo .45 marcaría error lo valido es 0.45.

descripción de las funciones aritméticas

FUNCIÓN (+ numero1 numero2 ... numeroN)

La función **(+)** suma la lista de números y devuelve el resultado de dicha suma.

ejemplo:

(+ 5 6 7)

devuelve 18

FUNCIÓN (- numero1 numero2 ... numeroN)

La función **(-)** sustrae el segundo numero del primero y devuelve el resultado. Si hay mas de dos números en la lista el tercer numero y los sucesivos se substraen del resultado de la sustracción anterior.

(- 4 3 3)

devuelve -2

FUNCIÓN (* numero1 numero2 ... numeroN)

La función **(*)** multiplicara todos los números de la lista y devolverá el producto de dicha lista:

ejemplo:

(* 3 4 3)

devuelve 36

FUNCIÓN (/ numero1 numero2 ... numeroN)

La función **(/)** dividirá el primer numero entre el segundo y devolverá el resultado como el cociente. Si hay mas de dos números en la lista de argumentos, se dividirá el primer número entre el producto de los demás números.

por ejemplo:

(/96 4 3 2)

es lo mismo que

(/ 96 (* 4 3 2))

o en álgebra

96/(4*3*2)

Si todos los números de la lista de argumentos son enteros el resultado será un entero. Esto puede ser un problema si los números se dividen exactamente, y el resultado puede ser impredecible. Si siempre se usa al menos un numero real en la lista de argumentos (incluso si el lado derecho del decimal es cero), todos los enteros de la lista se convertirán en reales antes de evaluar la lista, y el resultado será un real.

por ejemplo

(/ 6 72)

devuelve 0

(/ 6 72.0)

devuelve 0.083333

FUNCIÓN (1+ numero) Y (1- numero)

Las dos funciones **(1+)** y **(1-)** están agrupadas aquí ya que principalmente amabas se usan como contadores. 1+ acepta un numero que se le pase y lo incrementara en 1, lo mismo que (+ numero 1). 1- aceptara un numero y lo decrementara en 1, es equivalente a (- numero 1). Las funciones se recomiendan para incrementar o decrementar un numero en un bucle, puesto que harán mas sencillo y claro el código.

ejemplo:

(1+ 2)

devuelve 3

(1- 1)

devuelve 0

FUNCIÓN (abs numero)

La función **(abs)** devolverá el valor absoluto de un numero. Esto es, elimina el signo menos de un numero negativo.

ejemplo:

(abs -7) y (abs 7)

devolverán 7

Esta función es útil para filtra la salida de valores ángulos negativos cuando se comparan ángulos. Por ejemplo para todos los propósitos prácticos, el ángulo 3.14 es idéntico a -3.14, sin embargo si se comparan los dos ángulos para ver si son iguales el resultado indicara que son diferentes. por ejemplo

(equal 3.14 -3.14)

devolverá nil

Con nil AutoLISP indica que no se cumple la igualdad. Para que fueran iguales se debe hacer lo siguiente:

(equal (abs 3.14) (abs -3.14))

devolverá T

FUNCIÓN (exp numero)

La función **(exp)** devolverá e (la base del logaritmo natural 2.718282) elevado a la potencia número.

Ejemplo:

(exp 2)

devuelve 7.389056

que es lo mismo a:

(* 2.718282 2)

que devuelve 7.389057

las diferencias de dan por el redondeo

FUNCIÓN (expt base potencia)

La función **(expt)** elevara el argumento base al argumento potencia.

ejemplo

(expt 4 2)

devolverá 16

FUNCIÓN (fix numero)

La función **(fix)** devolverá el valor del numero como entero. Esto lo realiza truncando el numero en vez de redondearlo.

ejemplo

(fix 47.6) y (fix 47)

devolverán 47

FUNCIÓN (float numero)

Es la función inversa de fix. **(Float)** acepta un numero y lo convierte en real.

Ejemplos:

(float 9)

Devolverá 9.0000000

FUNCIÓN (gcd entero1 entero2)

La función **(gcd) (máximo común divisor)** devolverá el máximo común divisor de entero1 y entero2. Ambos argumentos deben ser enteros no negativos dentro del rango de 0 a 32,767; en otro caso, se devolverá un mensaje de error.

Ejemplo:

(gcd 11 33)

devolverá 11

FUNCIONES (max numero1 numero2 ..... numeroN)

(min numero1 numero2 ..... numeroN)

Las funciones **(max)** y **(min)** devolverán respectivamente los valores máximo y mínimo de los números de una lista. Estos números pueden ser reales o enteros. Si aparece un real en la lista de argumentos, el resultado será un real. La lista de números debe ser parte de la lista de argumentos y no debe aparecer como una lista anidada. Sin embargo se puede usar la función Apply para pasar una lista a max o min.

Ejemplo:

(max 27 34 77.7)

Devolverá 77.700000

(min -7 5.4)

Devolverá -7

FUNCIÓN (sqrt numero)

La función **(sqrt) (raíz cuadrada)** devolverá la raíz cuadrada de un numero como un real.

Ejemplos:

(sqrt 37)

Devolverá 6.082763

FUNCIONES LOGARÍTMICAS Y TRIGONOMETRICAS

FUNCIÓN (atan numero1 numero2)

La función **(atan) (ángulo cuya tangente es).** Funciona de dos formas diferentes si solo se suministra un numero devolverá el arco cuya tangente es numero1 en radianes. Si se suministran los dos argumentos, se calculara el ángulo cuya tangente es numero1/numero2.

FUNCIÓN (cos ángulo)

La función **(cos)** devolverá el coseno de la argumento ángulo. Cos espera que el ángulo sea expresado en radianes.

FUNCIÓN (log numero)

La función **(Log)** devuelve el logaritmo natural de la argumento como un real.

FUNCIÓN (sin ángulo)

La función **(sin)** devolverá el seno del argumento ángulo. El argumento ángulo se debe expresar en radianes.

FUNCIONES DE MANIPULACIÓN DE LISTAS

Muchas de las funciones de AutoLISP, tales como las funciones matemáticas y de cadena, son comunes a la mayoría de los lenguajes de programación. Sin embargo, las funciones de manipulación de listas son exclusivas de AutoLISP. Las siguientes funciones son el corazón de AutoLISP y suministran poderosas herramientas para la manipulación de símbolos.

Una lista puede estar formada por símbolos, números, cadenas u otras listas. Un símbolo se crea solamente cuando se declara como el segundo elemento de una lista que empieza con las funciones de AutoLISP defun, setq o quote. Todo símbolo que se crea es añadido a la lista de átomos y cada vez que se evalúan dicho símbolo AutoLISP debe, en primer lugar buscar en la lista de átomos el nombre del símbolo.

La memoria de AutoLISP esta formada por una serie de nodos. Cada nodo tiene un tamaño de 10 bytes (12 bytes en AutoLISP ampliado) y tienen una dirección especifica.

Cuando se crea un símbolo y se añade a la lista de átomos, se incluye en dicha lista, junto al nombre del símbolo, una dirección de memoria que apunta al nodo que contiene al símbolo. Si el símbolo esta asociado a un valor, el nodo que contiene el símbolo almacenara una dirección de memoria que apunta a un nodo que contiene el valor asociado al símbolo.

Si un símbolo esta asociado a una lista, el nodo que contiene el símbolo apuntara al nodo que contiene el primer elemento de una lista y ese nodo a su vez apuntara al un nodo que contiene el próximo elemento de la lista y así sucesivamente.

En las listas mas complejas tal como una lista que contiene otras listas, por ejemplo ((a b)(c d)), la primera lista tendrá un nodo que apunta a la segunda lista y el primer símbolo de la

primera lista.

Entender como AutoLISP crea y almacena símbolos y listas en memoria es importante si se pretende desarrollar un método eficiente para escribir un programa en AutoLISP. Al escribir el código en AutoLISP, hay que tener en cuenta que los símbolos y las listas que se almacenan en memoria se ejecutaran mas rápidamente que si se escribe el código de una manera convencional.

FUNCIÓN (append lista1 lista2 .... listasN)

La función **(append)** juntara una serie de listas en una solo lista. Esta función es particularmente útil cuando se quiere combinar varias listas en una sola para una manipulación mas fácil. Los argumentos a añadir debe ser listas.

Append es una delas tres funciones de construcción de listas (las otras dos son cons y list). Cada función tiene un propósito especifico y cada una construirá una lista de una forma diferente.

Ejemplos:

(append '(a b c) ' (d e f))

Devolverá (a b c d e f)

A continuación de da un ejemplo de como se añade un símbolo a una lista.

(append '(x y)(list 'a)

Devolverá (x y a)

FUNCIÓN (assoc argumento lista-asociación)

La lista de asociación (algunas veces llamada lista-a) es una lista que contiene sublistas. El primer elemento de cada una de estas sublistas se llama clave. La función assoc buscara en una lista de asociación dad una clave especifica y devolverá la primera sublista que contenga la clave. Cada registro de entidad de AutoCAD se formatea como una lista de

asociación. Assoc es tan rápida que puede buscar en una lista de asociación que contenga varios cientos de sublistas, en menos de un segundo.

Cada vez que se quiera buscar en una base de datos, se deberá formatear esta como una lista de asociación.

Ejemplos

(set animal '((gatos persas siameses)(perros pastores pequineses)

(caballos percherones árabes)))

(setq persas '((pelo largo)(cara lisa)))

A la primera lista se accede con:

(assoc 'cara persas)

Devolverá (cara lisa)

Es importante notar que el símbolo asociado a la segunda lista Persas se encuentra también en la primera lista-a con la clave de gatos. Una vez que se evalúa la segunda lista, se convierte en una sublista de la primera lista-a. A la segunda lista se puede acceder a través de la primera lista, de la siguiente forma:

(assoc 'cara (eval (cadr (assoc 'gatos animales))))

Devolverá (cara lisa)

FUNCIONES (car lista)

(cdr lista)

Las funciones **(car)** y **(cdr)** están diseñadas para separar una lista. Car devolverá el primer elemento de la lista que se le pasa como argumento y Cdr devolverá la lista menos el primer elemento. Si la lista es vacía, ambas funciones devolverá nil. Esto es muy simple y potente.

Ejemplos:

(car '(a b c))

Devolverá a

(cdr '(a b c))

Devolverá (b c)

Si la lista de un par de elementos separados por un punto, cdr devolverá el ultimo elemento de la lista.

(cdr '(0."line"))

Devolverá "line"

Car y Cdr son particularmente convenientes para separar una lista símbolo a símbolo de forma que se pueda operar cada vez sobre cada símbolo de la lista.

Car y Cdr se pueden combinar para devolver elementos que estén profundamente anidados dentro de una lista.

Ejemplos:

(car (cdr'(a b c)))

Devolverá b

(car (cdr(cdr (a b c))))

Devolverá C

Con Car y Cdr combinados se pueden encontrar cualquier elemento de una lista pero llega un momento que son demasiado complicados o confusas las declaraciones.

Hay un modo ligeramente mas fácil para realizarlo. Hay 28 funciones que son equivalentes del uso de las diferentes combinaciones de car y cdr para obtener elementos de una lista. Se conocen estas funciones como las concatenaciones de car y cdr.

LAS CONCATENACIONES DE CAR Y CDR

A continuación se listan las 28 funciones diferentes que representan las diferentes combinaciones de car y cdr:

(caar)(caaar)(cadar)(caaaaar)(caaadr)

(caadar)(caaddr)(caadr)(cadaar)(cadadr)

(caddar)(cadddr)(caddr)(cadr)(cdaar)

(cdadr)(cdar)(cddar)(cddr)(cdaar)

(cdaadr)(cdadar)(cdaddr)(cddaar)(cddadr)

(cdddar)(cddddr)(cdddr)

Las 28 concatenaciones se pueden dividir en cuatro grupos, basándose la división en la profundidad con que pueden obtenerse elementos de una lista anidada.

El primer grupo solo tendrá elementos del nivel superior de un a lista. Dada la siguiente lista:

(setq ls ' a b c d e))

Entonces:

FUNCIÓN	DEVUELVE	EQUIVALENTE
(car ls)	a	**(car ls)**
(cadr ls)	b	**((car(cdr ls))**
(caddr ls)	c	**(car(cdr(cdr ls)))**
(caddr ls)	d	**(car(cdr(cdr(cdr ls))))**
(cdr ls)	(b c d e)	**(cdr ls)**
(cddr ls)	(c d e)	**(cdr(cdr ls))**
(cdddr ls)	(d e)	**(cdr(cdr(cdr ls))))**
(cddddr ls)	(e)	**(cdr(cdr(cdr(cdr ls)))))**

El segundo grupo de funciones obtendrá elementos del segundo nivel de anidamiento en la lista. Con la siguiente lista de ejemplo:

(setq ls '(s b c d)(e f g h)(i j k l)))

Entonces:

FUNCIÓN	DEVUELVE	EQUIVALENTE
(caar ls)	a	**(car ls)**
(cadar ls)	b	**(car(cdr(car ls)))**
(caddr ls)	e	**(car(car(cdr ls))))**
(cadadr ls)	f	**(car(cdr(car(cdr ls))))**
(caaddr ls)	i	**(car(car(cdr(cdr ls))))**
(cdar ls)	(b c d)	**(cdr(car ls))**
(cddar ls)	(c d)	**(cdr(cdr(car ls)))**
(cdddar ls)	(d)	**(cdr(cdr(cdr(car ls))))**
(cdadr ls)	(f g h)	**(cdr(car(cdr ls)))**
(cddadr ls)	(g h)	**(cdr(cdr(car(cdr ls))))**
(cdaddr ls)	(j k l)	**(cdr(car(cdr(cdr ls))))**

El tercer grupo de funciones obtendrá elementos anidados a tres niveles de profundidad en la lista. Con la siguiente lista:

(setq ls '(((a b c)(d e f))((g h i)(j k l))))

FUNCIÓN	DEVUELVE	EQUIVALENTE
(caaar ls)	a	**(car(car(car ls)))**
(cadaar ls)	b	**(car(cdr(car(car ls))))**
(caadar ls)	d	**(car(car(cdr(car ls))))**
(caaadr ls)	g	**(car(car(car(cdr ls))))**

(cddaar ls)	(c)	**(cdr(cdr(car(car ls))))**
(cdaar ls)	(b c)	**(cdr(car(car ls)))**
(cdadar ls)	(e f)	**(cdr(car(cdr(car ls))))**
(cdaadr ls)	(h i)	**(cdr(car(car(cdr ls))))**

Finalmente, las dos funciones del cuarto grupo obtendrán elementos sepultados a cuatro niveles de profundidad en la lista. Dada la siguiente lista:

(setq ls '((((a b c))))

Entonces:

FUNCIÓN	**DEVUELVE**	**EQUIVALENTE**
(caaaar ls)	a	**(car(car(car(car ls))))**
(cdaaar ls)	(b c)	**(cdr(car(car(car ls))))**

Por supuesto que se pueden combinar estas funciones e incluso ir a mas profundidad.

FUNCIÓN (cons Nuevo-primer-elemento lista)

(Cons) es una de las tres funciones para construir una lista. Cons agregará un primer elemento nuevo como el primer argumento, o un átomo o una list, y una lista como el segundo argumento, y devolverá esa lista con el primer elemento nuevo añadido al principio de la lista.

Cons también aceptará un átomo como el segundo elemento de la lista y devolverá lo que se conoce como una lista de un par de elementos separados por un punto. Una lista de un par de elementos separados por un punto es una lista de tres elementos donde el segundo elemento es un punto y el tercer elemento es siempre un átomo. Una lista de este tipo ocupa menos memoria que una lista de dos elementos. El ultimo elemento se puede obtener con cdr. La única forma de construir una lista de un par de elementos separados por un punto es

con Cons. Cons también puede crear listas de cadenas.

ejemplo:

(cons 'a '(b c))

devuelve (a b c)

(cons 'a ())

Devuelve (a)

(cons (a b) ())

devuelve ((a b))

(cons 'a 'b)

Devuelve (a . b).

FUNCIÓN (last lista)

La función **(last)** devolverá el ultimo elemento de una lista. Hay que ser precavidos con esta función cuando se manipulan listas de puntos. Muchas veces, el ultimo elemento de una lista no es el que nosotros queríamos.

Ejemplo:

(last (a b c))

Devuelve c

FUNCIÓN (length lista)

La función **(length)** devolverá un entero que informa del número de elementos de una lista.

Ejemplo:

(length (a b c))

Devuelve 3

FUNCIÓN (list expr1 .... exprN)

La función **list** aceptara cualquier número de expresiones LISP validas (incluyendo las

cadenas) y las concatenara en una lista. List es una de las tres funciones de AutoLISP que crean listas.

Ejemplo:

(list 'a 'b 'c)

Devuelve (a b c)

(list (x y z) (a b c))

Devuelve (x y z a b c)

List es muy útil para crear listas que manipulan coordenadas bi y tridimensionales.

FUNCIÓN (member expr lista)

(Member) buscara en lista la primera ocurrencia de expr y devolverá el resto de la lista desde expr hasta el final. Si no se encuentra expr member devuelve nil.

Member se puede utilizar para devolver algo de una valor de una lista o para comprobar simplemente si expr esta en la lista.

Ejemplo:

(member 'a (a b c d e))

Devuelve (b c d e)

Cuando se buscan cadenas en listas con member y con assoc, se debe conocer el caso de las cadenas. Mientras que AutoLISP no es sensible al caso de símbolos, si lo es con las cadenas.

FUNCIÓN (nth numero_de_posicion lista)

(Nth) devolverá el elemento n-esimo de la lista. Las posiciones en una lista siempre se numeran de derecha a izquierda, empezando desde cero. Si el numero de posición es mayor que el número de elementos de una lista menos uno, se devuelve nil.

Ejemplo:

(nth 3 '(a b c d e))

devuelve c

FUNCIÓN (reverse lista)

(Reverse) devuelve la lista de argumentos con todos sus elementos en orden inverso. Puesto que las listas normalmente se crean añadiendo elementos al principio de la lista, reverse se usa a menudo para colocar la lista a la que se aplica en el orden correcto.

Ejemplo:

(reverse '(c b a))

Devuelve (a b c)

FUNCIÓN (substr nuevo_argumento antiguo_argumento lista)

La función **(substr)** buscará en la lista las ocurrencias del antiguo argumento y luego devolverá una copia de esa lista con cada caso del antiguo argumento reemplazado por el nuevo argumento. Si la lisa no encuentra el antiguo argumento, la lista se devuelve sin ser modificada. Substr es una de las funciones de LISP que puede modificar cualquier cosa de la base de datos del dibujo e incluso alterar el propio código de AutoLISP.

Ejemplo:

(substr a z (a b c))

Devuelve (z a c)

FUNCIONES DE ASIGNACIÓN

Estas funciones se utilizan para asignar y leer valores en AutoCAD y AutoLISP.

FUNCIÓN (getenv nombre_variable)

El sistema operativo tiene un vector de almacenamiento del entorno que contiene los nombres de las variables del entorno por ejemplo la variable PATH de DOS. Se puede utilizar la función **(getenv)** para obtener los contenidos de estas variables.

Ejemplo:

(getenv "ACAD")

Devuelve "c/acad"

FUNCIÓN (getvar nombre_de_variable)

La función **(getvar)** devuelve el valor actual de una variable del sistema de AutoCAD. El nombre de variable del argumento debe ser una variable del sistema valida y el nombre de variable se debe encerrar entre comillas (puesto que es una cadena). Si el nombre de variable no es una variable del sistema valida getvar devolverá nil.

Ejemplo:

(getvar "orthomode")

Devuelve 0

FUNCIÓN (quote expr)

La función **(quote)** devolverá una expresión sin evaluación. La versión abreviada de quote consiste en poner un apóstrofe delante de una expresión:

Ejemplo:

(quote (+ 8 9))

Devuelve (+ 8 9)

Y es lo mismo que:

('(+ 8 9))

FUNCIÓN (set sim expr)

La función la función **(set)** establecerá el valor del sim, si el sim es un símbolo precedido de un apóstrofe, igual al valor de la expr y devolverá dicho valor.

ejemplo:

(set 'a 6)

Devuelve 6

Si se quiere asociar el valor de una expresión a un símbold es más directo usar la función setq. Sin embargo set tiene la posibilidad única de reasignar un valor de un símbolo indirectamente.

Ejemplo (set 'x 4.0)

Devuelve 4.0000000

(set 'z 'x)

En este caso set suministra un enlace entre el sim z y el sim x. A continuación si se asigna un valor a Z (sin comillas)

(set z 100)

Se encontrara que el sim x también tiene asociado el valor 100

FUNCIÓN (setq sim exp1 ... exprN)

La función (**setq**) asignará o asociará el valor de la expr al símbolo. Setq puede asignar un numero indefinido de expresiones a un número parecido de símbolos.

Setq es la principal función de asignación de AutoLISP. Setq junto con set y defun son las únicas funciones que añadirán símbolos a la lista de átomos.

Si un símbolo no existe, se creara con set. Setq y set se pueden usar para definir localmente un símbolo si el nombre del símbolo aparece en la lista de argumentos de una definición de función.

Ejemplo:

(setq x 47)

asigna el valor de 47 a x

Para verificar si se asigno bien el valor al símbolo se puede utilizar el símbolo ! antes del símbolo, con esto se desplegara el valor asignado a la símbolo, si un símbolo no existe o no tiene asignados valores se despliega nil.

Ejemplo:

!x

Devuelve 47

FUNCIÓN (setvar nombre_de_variable valor)

(Setvar) se usa para establecer el valor de una variable del sistema de AutoCAD (nombre de variable) y devuelve el valor. El nombre de variable debe ser una cadena encerrada entre comillas y debe ser un nombre valido de variable del sistema de AutoCAD.

El valor debe ser un tipo de dato que esta dentro del rango de valores valido para la variable que se esta analizando. Por ejemplo. la variable cmdecho requiere un entero entre le rango de 0 y 1.

Ejemplo:

(setvar "cmdecho" 1)

FUNCIONES PROCEDIMENTALES

Las funciones procedimentales se usan para describir una serie de instrucciones pasa a paso para que la computadora las ejecute. Las funciones procedimentales son comunes a todos los lenguajes de programación.

FUNCIÓN (apply función lista)

(Apply) hará que la función especificada se le pase la lista como los argumentos de la función y devolverá el resultado de la ejecución de esa función. La función puede ser una función incorporada, una función definida por el usuario, o una función anónima hecha con lambda.

La lista debe contener los mismos argumentos que serán usados en la lista de argumentos de la función.

Apply es útil cuando se quiere operar sobre listas de datos que normalmente no serian evaluadas.

(apply a '(1 2 3 4 5 6 7 8))

Devuelve (1 2 3 4 5 6 7 8)

Si se desea encontrar el valor máximo contenido en la lista, se puede intentar lo siguiente:

(max a)

Pero puesto que la lista esta precedida por un apóstrofe, esta función fracasara, ya que AutoLISP intentara evalúa el propio símbolo en vez de la lista asociada al símbolo. Sin embargo apply funcionar en esta situación, pues ya espera una lista como argumento:

(apply 'max a)

Devuelve 8

FUNCIÓN(command orden arg1_de_orden .....argN_de_orden)

La función **(command)** llama al procesador de ordenes de AutoCAD, pasa la orden a la línea de ordenes, y continuación, pasa los argumentos de la orden a la orden de AutoCAD que fue llamada. Los datos contenidos en la lista de argumentos de la orden son tratados como si hubieran introducido en el indicador de ordenes desde el teclado. Command siempre devuelve nil.

Command es la única función de AutoLISP que añadirá registros de entidades a la base de datos del dibujo. La mayoría de las otras funciones solo alteraran los registros existentes en un grado limitado. Si se quiere añadir una línea de dibujo con AutoLISP, se usara la función command para llamar a la función line.

Command llamara a cualquier orden valida de AutoCAD incluyendo las ordenes

definidas por el usuario creadas con la opción defun.

FUNCIÓN(defun nombre_funcion lista_argumentos expr1....exprN)

(Defun) es la función que se utilizara cuando creemos nuestra propias funciones en AutoLISP. De hecho, no podemos hacer nada en AutoLISP hasta que definamos lo que queremos hacer utilizando una función.

Defun define una función denominada nombre función. Lista argumentos contiene una lista de argumentos (si hay alguno) pasados a la definición y puede contener opcionalmente una lista de símbolos declarados a nivel local, seguidos de una expr que contiene la definición de la función en si. Defun devuelve nombre función si ha sido cargada satisfactoriamente por AutoLISP.

El primer elemento de una lista de definición de función es la palabra defun que le dice a AutoLISP que lo que sigue es una definición de una función, adapta esta función al símbolo siguiente a nombre de función y coloca nombre función en la lista de átomos, para que las restantes funciones tengan acceso a esta. Por esto la impresión de un archivo en LISP tarda mucho en cargarse.

después del nombre de la función, viene la lista de argumentos, llamada también lista de parámetros. algunas funciones no tienen argumentos, mientras que otras pueden tener muchos.

La lista de argumentos puede contener también símbolos declarados localmente en la función. Los argumentos y los símbolos declarados a nivel local tiene que estar separados por un barra (/) y un espacio dentro de la lista de argumentos.

Nosotros podemos crear funciones que funcionen como comandos de AutoCAD. Para hacer esto necesitamos anteponer al nombre de la función C: que indican que evalúa la

función como si fuera un comando. Las reglas que se tiene que seguir para hacer una función de este tipo es que la función no tiene que tener argumentos. Pero si puede tener variables locales.

A continuación tenemos varios ejemplos de listas de argumentos:

(defun fx() expr)	Sin argumentos, ni símbolos locales
(defun fx(x) expr)	Un argumento, ningún símbolo local
(defun fx(x y) expr)	dos argumentos, ningún símbolo local
(defun fx(x y / z) expr)	Dos argumentos un símbolo local

Expr contiene la definición la definición de la función .

Ejemplo

(defun fx(x) (* x 2)

FUNCIÓN(eval expr)

La función **(eval)** fuerza la evaluación de expr y devuelve el resultado de la evaluación de dicha expresión. Eval es útil por dos razones. Añadiendo eval a una función podemos agilizar en ocasiones su ejecución y, esta es la razón mas importante y eval puede evaluar una expresión con apóstrofe.

Ejemplo:

(setq b '(+ 3 4))

(eval b)

Devuelve 7

FUNCIÓN(foreach nombre lista expr1 ... exprN)

La función **(foreach)** asigna cada elemento contenido en lista a nombre. Para cada elemento asignado de esta manera, se evalúa cad expresión expr. Al final, devuelve el valor

de la ultima expresión evaluada con el ultimo miembro de la lista.

Ejemplo:

(foreach tornillo '(placa brazo base)(print tornillo)

Lo anterior imprime primero placa, luego brazo y por ultimo base, para esto primero toma el elemento placa, aplica la función print, después toma los otros elementos y vuelve a aplicar la función print.

FUNCIÓN (lambda argumentos expr1 ... exprN)

Si se desea evitar la tarea de definir una nueva función, se puede utilizar la función **(lambda)**. La función aparece donde se utiliza, en vez de tener que definiria en cualquier sitio. Como es habitual, el valor devuelto es el valor de la ultima expresión.

Ejemplo:

(apply'(lambda(a b c)(- a (+ b c)))'(3 6 9))

Devuelve 12

FUNCIÓN (mapcar función lista1 .... listaN).

(Mapcar) ejecutar una función utilizando listas, elemento por elemento, las listas serán leídas de izquierda a derecha. Para cada lista, los elementos se pasaran ordenadamente a la función.

Ejemplo:

(setq q 10 r 5 s 20 t 4)

(mapcar '/ (list q s)(list r t)

Devuelve (2 5)

FUNCIÓN (cenucmd cadena)

Un menú de AutoCAD contiene submenus. Con **(menucmd),** podemos pasar de uno a otro. De esta forma, podemos utilizar AutoLISP para controlar un archivo de menús. Hay que utilizar una cadena de la forma seccion = submenú. La sección tiene que ser uno de los

nombres validos de sección, tales como S, B , P1- p10, A1 o ti-t4.

Ejemplo:

(menucmd "s = line")

FUNCIÓN (progn expr1 ... exprN)

la función **(progn)** evalúa una lista de expresiones. Cad expresión de la lista es evaluada por separado, devolviendo solamente el valor de la ultima expresión. Podemos forzar la evaluación de mas de una expresión para una función que solo espera una expresión.

Ejemplo:

(if /= x y)(progn (setq x (* y 3)(setq y (* x 4))))

FUNCIÓN (repeat numero expr1 .... exprN)

Si queremos repetir una expresión un numero finito de veces, podemos utilizar la función **(repeat).** Las expresiones serán repetidas, las veces que indique el numero. El valor devuelto será el de la ultima expresión.

Ejemplo:

(setq x 1)

(repeat 10 (setq x (1+ x)))

Devuelve 11

FUNCIONES CONDICIONALES

Las funciones condicionales son aquellas que comprueban el resultado de expresiones y realizan las operaciones especificas dependiendo de los resultados de sus comprobaciones. Por ejemplo, la expresión " Si quiere ver una película, vaya al cine" es una expresión condicional, ya que comprueba el resultado de la expresión "quiere ver una película". Dependiendo de si la quiere o no, el resultado de la prueba será verdadero o falso.

FUNCIÓN (cond Exprn_prueba1 expr_resultado1.. expr_ResultadoN)

La función **(cond)** evalúa listas en las que el primer miembro de la lista es una expresión a verificar. Podemos introducir cualquier numero de listas, conteniendo cada una de ellas las expresiones a evaluar en el caso de que la expresión a verificar no de como resultado nil. La función cond evaluara el primer elemento de la lista hasta que encuentre uno cuyo valor no sea nil. Cuando se encuentra una lista de ese tipo, evalúa todas las expresiones del resto de esa lista, devolviendo el valor de la ultima expresión evaluada.

Ejemplo:

(setq a 12)

(cond ((=a 12) 1) ((= a 13)0)

Devuelve 1

FUNCIÓN (if exp_prueba expr_afirmativa expr_negativa)

La función condicional (if) toma una expresión a verificar, la evalúa y bifurca a una de las dos expresiones siguientes dependiendo del resultado. Si es distinto de nil, se evalúa la primera expresión, De lo contrario se evalúa la segunda expresión.

Ejemplo:

(setq x 5)

(if (= x 5) print "5" (Print "Incorrecto"))

Devuelve 5

FUNCIÓN (while expr_prueba expr1 .... exprN)

La función condicional **(while)** continua evaluando la expresión prueba hasta que devuelve nil. Mientras el resultado de expresión prueba será distinto de ni, se evalúan las expresiones que le siguen.

Ejemplo:

(setq a 12)

(setq b 1)

(while (1-a) (1+b))

Devuelve 12

FUNCIÓN (= Atomo1 Atomo2 ... ATomoN)

Esta función compara los átomos de la lista y si todos los átomos de la lista son iguales devuelve una T (True) Los átomos pueden ser de cualquier tipo incluyendo números y cadenas. Hay que tomar en cuenta que en AutoLISP los enteros son iguales a los mismos números reales (7 es igual que 7.000).

Ejemplo:

(setq a 1)

(setq b 1)

(setq c 1)

(= a b c)

Devuelve T

FUNCIÓN (/= Atomo1 Atomo2)

Esta función compara los átomos de la lista y si los dos átomos de la lista son diferentes devuelve una T (True). En esta función solo se permite comparar dos átomos.

Ejemplo

(setq a 1)

(setq b 1)

(/= a b)

Devuelve T

FUNCIÓN (< Atomo1 Atomo2 ... ATomoN)

Esta función compara los átomos de la lista y si todos los átomos de la lista ordenados de izquierda a derecha son mayores que el anterior devuelve True.

Ejemplo:

(setq a 1)

(setq b 2)

(setq c 3)

(= a b c)

Devuelve T

FUNCIÓN (<= Atomo1 Atomo2 ... ATomoN)

Esta función compara los átomos de la lista y si todos los átomos de la lista ordenados de izquierda a derecha son mayores o iguales que el anterior devuelve True.

Ejemplo:

(<= 1 2 2 3 4 4)

Devuelve T

FUNCIÓN (<= Atomo1 Atomo2 ... ATomoN)

Esta función compara los átomos de la lista y si todos los átomos de la lista ordenados de derecha a izquierda son mayores que el anterior devuelve True.

Ejemplo

(> 3 2 1)

Devuelve T

FUNCIÓN (>= Atomo1 Atomo2 ... ATomoN)

Esta función compara los átomos de la lista y si todos los átomos de la lista ordenados de derecha izquierda son mayores o iguales que el anterior devuelve True.

Ejemplo:

(<= 4 3 3 2 2 1 1)

Devuelve T

FUNCIÓN (eq exp1 exp2)

Esta función es diferente a la función = o equal. La función eq no solo verifica la igualdad de las dos expresiones, sino también si comparten o no el mismo objeto relacionado con ellas. Se diferencia de la función = en que compara expresiones en lugar de átomos.

Ejemplo

(setq a '(c d e f))

(setq b '(c d e f))

(setq g b)

(eq g b)

Devuelve T

(eq a g)

Devuelve nil

FUNCIONES PREDICATIVAS

Las funciones predicativas son aquellas que producen un efecto basado en las comprobación de una o más expresiones. El sujeto es la expresión a comprobar y el predicado es el resultado de la comprobación.

FUNCIÓN (And expr1 expr2 ... exprN)

La función **(And)** combina Las distintas expresiones. El resultado será la función Y lógica de todas las expresiones. Si cualquiera de las expresiones se evalúa como nil, de detiene la evaluación, y toda la función Y será considerada como nil.

Ejemplo

(setq x 1)

(setq y 2)

(setq z (equal 1 2))

Devuelve nil

(setq r "abc")

(and 2 x y r)

Devuelve T

(and x y z r)

Devuelve nil

FUNCIÓN (atom argumento)

La función (**atom**) se utiliza para ver si una argumento es o no un átomo. Si argumento no es un átomo, la función devuelve T.

Ejemplo

(setq x 'a)

(setq y '(a b c))

(atom x) y (atom 'x)

Devuelven T

(atom 'y)

Devuelve T

(atom y)

Devuelve nil

FUNCIÓN (boundp atomo1)

Esta función nos permite determinar si un átomo esta asociado o no a un argumento. Si el átomo esta relacionado con alguno, la función boundp devuelve T.

Ejemplo:

(setq x nil)

(boundp x)

Devuelve nil

(setq x 1)

(boundp x)

Devuelve T

FUNCIÓN (listp argumento)

La función **(listp)** devuelve T si el argumento es una lista y nil si es cualquier otra cosa.

Ejemplo:

Setq (a 1)

(listp a)

Devuelve nil

(setq a '(1 2 3))

Devuelve T

FUNCIÓN (minusp argumento)

La función **(minusp)** nos sirve para cuando queremos ver si un argumento es un número negativo, no importándonos si es entero o real.

Ejemplo:

(minusp 1)

Devuelve nil

(minusp -2)

Devuelve T

FUNCIÓN (not argumento)

La función **(not)** devuelve el valor complementario de un argumento. Si al evaluar el argumento el resultado es T, la función not devuelve nil. Si al evaluar una función se obtiene nil, la función not devuelve T.

Ejemplo:

(setq x 1)

(not x)

Devuelve nil

(setq x nil)

(not x)

Devuelve T

FUNCIÓN (null argumento)

la función **(null)** la utilizamos cuando queremos comprobar si el argumento es o no nil. La función null devuelve un valor de T si el argumento esta asociado a nil y nil si no lo esta.

Ejemplo:

(setq x nil)

(null x)

Devuelve T

(setq x 1)

(null x)

Devuelve Nil

FUNCIÓN (numberp argumento)

La función **(numberp)** la utilizamos para ver si un argumento es un número entero o real, si el argumento es entero o real numberp devuelve T.

Ejemplo:

(numberp 3)

Devuelve T

(numberp a)

Devuelve Nil

FUNCIÓN (or expr1 expr2 ... exprN)

La función **(or)** relaciona todas las expresiones de la lista utilizando el operador OR lógico. Si cualquiera de las expresiones es distinta de nil, la evaluación se detiene en esta expresión y devuelve T.

Ejemplo:

(setq a nil)

(setq b nil)

(or a b)

Devuelve nil

(setq c 1)

(or a b c)

Devuelve T

FUNCIÓN type argumento)

La función **(type)** la utilizamos cuando queremos conocer el tipo de dato de un argumento.

Ejemplo:

(type 3.1416)

Devuelve Real

(type "letra")

Devuelva STR

FUNCIÓN (zerop argumento)

La función **(zerop)** sirve para determinar si un argumento es o no un número, no importando que sea entero o real, cuyo valor al evaluarlo es cero. La función devuelve T si el argumento es un entero o real y vale cero.

Ejemplo:

(zerop 1)

Devuelve nil

(zerop 0)

Devuelve T

(zerop (- 1 1))

Devuelve T

FUNCIONES DE CADENAS

AutoLISP permite manipuiar cadenas de caracteres. Las funciones de manipulación de cadenas permiten visualizar cadenas, verificarlas, compararlas e incluso modificar sus características.

FUNCIONES DE MANIPULACIÓN DE CADENAS

FUNCIÓN (strcase cadena1 [upper/lower])

La función **(strcase)** cogerá una cadena y convertirá todos sus caracteres en mayúsculas o minúsculas, dependiendo del valor del
argumento [upper/lower]. Si se utiliza [upper/lower] y es distinto de nil, todos los caracteres serán convertidos a minúsculas. Si no hay argumento [upper/lower] o es evaluado como nil, la cadena será convertida a mayúsculas.

Ejemplo:

(setq s "mi cadena")

(srtcase s nil)

Devuelve "MI CADENA"

(strcase s 1)

Devuelve "mi cadena"

(srtcase s)

Devuelve "MI CADENA"

FUNCIÓN (strcat cadena1 cadena2 ... cadenaN)

La función **(strcat)** permite concatenar cadenas. La concatenación consiste en combinar las cadenas para formar una que conste de todas las cadenas de la lista unidas. La concatenación se realizará según el orden en que las cadenas aparecen en la lista.

Ejemplo:

(strcat "Me" "gusta" "AutoLISP" .)

Devuelve Me gusta AutoLISP.

FUNCIÓN (strlen cadena)

La función **(strlen)** nos permite conocer la longitud de una cadena. Esta función devuelve la longitud en forma de numero entero.

Ejemplo:

(strlen "hola")

Devuelve 3

FUNCIÓN (substr cadena1 primer_caracter número_caracteres)

La función **(substr)** permite extraer una subcadena de una cadena. Obtendremos la subcadena a partir de primer_caracter, y consistirá en el numero de caracteres indicados. Si se omite el número de caracteres, obtendremos el resto de la cadena a partir de el primer_caracter.

Ejemplo:

(substr " Guadalajara" 3)

Devuelve dalajara

(substr "Hola como estas", 6 4)

Devuelve Como

FUNCIONES DE CONVERSIÓN DE CADENAS

FUNCIÓN (angtos angulo1 método precisión)

La función **(angtos)** convertirá un ángulo en una cadena. El argumento angulo1 tiene que ser un número real expresado en radianes. El método se basa en la variable del sistema **ACOCP**. Si **ACOCP** es cero, la cadena será expresada con los símbolos de grados. Si es uno, el formato será de grados/minutos/segundos.

Si es 2, la cadena será expresara en grados centesimales. Si es 3 se utilizaran radianes. Finalmente, si el método es 4 se utilizaran unidades de topografía.

Si se utiliza precisión tiene que ser un entero que exprese el numero de posiciones decimales deseadas. Los argumentos método y posición realizan las mismas funciones que las variables del sistema **AUNITS** y **AUPREC**. Los argumentos método y precisión son operacionales. Se utilizaran los valores activos de las variables del sistema.

Ejemplo:

(angtos (/pi 2) 0 2)

Devuelve 90

(angtos (/ pi 2) 3 3)

Devuelve 1.57r

FUNCIÓN (ascii cadena)

La función **(ascii)** nos da el código ascii de el primer elemento de la cadena. El código ascii de un carácter es un numero que sigue una norma de la industria de las computadoras. Los códigos ascii de los caracteres imprimibles van del 32 al 127.

Ejemplo:

(ascii "Tres")

Devuelve 84

(ascii "T")

Devuelve 84

(ascii "U")

Devuelve 85

(ascii "u")

Devuelve 117

FUNCIÓN (atof cadena)

La función **(atof)** convierte cadenas que tienen expresiones numéricas en números reales.

Ejemplo:

(atof "4.5")

Devuelve 4.5

(atof "numero 4.5")

Devuelve 0,0

FUNCIÓN (atoi cadena)

La función **(atoi)** convierte cadenas que tienen expresiones numéricas en números enteros.

Ejemplo:

(atoi "4.5")

Devuelve 4

(atof "numero 4.5")

Devuelve 0

FUNCION (chr numero)

La función **(chr)** permite convertir el código ascii de un carácter en dicho carácter ascii. Para el proceso inverso, podemos utilizar la función ascii.

Ejemplo:

(chr 84)

Devuelve T

FUNCIÓN (itoa entero)

La función **(itoa)** realiza la conversión de un entero en una cadena en una cadena en ascii.

Ejemplo:

(itoa -25)

Devuelve "-25"

FUNCIÓN (read cadena)

La función **(read)** devuelve el primer átomo o lista de una cadena. La cadena no puede contener espacios en blanco.

Ejemplo:

(setq a "AutoCAD")

(read a)

Devuelve AutoCAD

FUNCIÓN (rtos numero método precisión)

La función **(rtos)** convierte un real en una cadena, de forma similar a atoi, pero el numero puede ser formateado al convertirse en una cadena. Si se especifica un método, tiene que ser uno de los siguientes. El método 1 crea una cadena en notación científica. El método 2 crea una cadena con decimales. El método 3 utiliza notación de ingeniería (pies y pulgadas). El método 4 general notación arquitectónica (pies y fracciones de pulgadas). El

método 5 genera fracciones simples. El método precisión corresponde al numero de posiciones decimales almacenado en la variable del sistema LUPREC.

Ejemplo:

(rtos pi 5 8)

Devuelve "3 1/8"

(rtos pi 1 4)

Devuelve "3.1416+00"

FUNCIONES GRÁFICAS

Las funciones gráficas de AutoLISP se dividen en dos grupos principales. Uno de ellos permite añadir entidades gráficas a nuestra base de datos y la otra permite manipular gráficos en la pantalla de computadora.

FUNCIONES DE MANIPULACIÓN DE GRÁFICOS

AutoLISP permite manipular gráficos de varias formas potentes. Puede realizar prácticamente todas las funciones implementadas por las ordenes de AutoCAD. Además debido a que AutoLISP es un autentico lenguaje de programación, las posibilidades de mejorar y ampliar las ordenes de AutoCAD son prácticamente ilimitadas.

FUNCIÓN (angle punto1 punto2)

La función **(angle)** devuelve el o de una línea en el sistema de coordenadas del usuario. Las coordenadas de los puntos que delimitan la línea se pasan en forma de dos listas y el valor de ángulo es devuelto por la función en radianes.

Ejemplo:

(angle '(2.0 2.0) '(5.0 5.0))

Devuelve 0.3927

FUNCIÓN (distance punto1 punto2)

La función (distance) devuelve la distancia entre dos puntos. Los puntos pueden estar en un espacio tridimensional. se ignorará el valor del eje Z.

Ejemplo:

(distance '(1.0 1.0) ' (2.0 1.0)

Devuelve 1

FUNCIÓN (inters punto1 punto2 punto3 punto4 [finito])

La función **(inters)** permite localizar el punto de intersección de dos líneas. La función devuelve nil si las líneas no se cortan. Di no se incluye el argumento [finito], se considera que las líneas tiene su longitud finita, determinada por sus puntos terminales. Mejor dicho son dicho argumento está presente y es nil, se considera que las líneas tiene un longitud infinita. si esta presente y es distinto de nil, tendrá el mismo efecto que si no estuviera presente.

Ejemplo:

(setq L1i '(0 0) Lff '(1 1))

(setq L2i '(0 1) Lff '(1 0))

(inters L1i L1f L2i L2f)

Devuelve (0.5 0.5)

FUNCIÓN (osnap punto1 método)

La función **(osnap)** fija los objetos. Si se selecciona un punto, éste será devuelto por la función. Sin no se alcanza, la función devuelve nil. El método utilizado es midpoint (punto medio) center (centro), endp (punto final), correspondiente a los modos de fijación de AutoCAD.

Ejemplo:

(osnap '(1 1) "midp")

Devuelve (0.75 1.0)

FUNCIÓN (polar punto ángulo distancia)

La función **(polar)** permite obtener un punto a partir de un ángulo respecto del eje y una distancia desde un punto inicial.

Ejemplo:

(polar '(0 0) 45.0 1)

Devuelve (0.7071 0.7071)

FUNCIÓN (redraw nombre_entidad método)

La función **(redraw)** vuelve a dibujar todo el área de visualización activa si no le dan argumentos. Si se utiliza nombre_entidad, solo se dibujaran de nuevo las entidades del tipo especificado. Si da un método, se utilizara uno de los siguientes métodos: Si es 1, la entidad volverá a ser dibujada en la pantalla. Si es 2, la entidad será borrada. Si el método es 3, la entidad será destacada si la pantalla lo permite. Finalmente, si el método es 4, la entidad dejará de estar destacada si lo estaba.

FUNCIÓN (trans punto inicio fin desplazamiento)

La función **(trans)** permite trasladar un punto de un sistema de coordenadas a otro. El punto puede estar en el espacio tridimensional. Los códigos inicio y fin son necesarios y se refieren a los sistemas de coordenadas a utilizar como origen y destino. Si el inicio o fin son 0, el sistema de coordenadas es el sistema de coordenadas universal. Si es 1, el sistema será el sistema de coordenada del usuario. Si es 2, será el sistema de coordenadas de la pantalla. También podemos introducir una entidad como inicio o fin y se utilizara el sistema de coordenadas de entidad de dicha entidad.

FUNCIONES GRÁFICAS DE PANTALLA

AutoLISP contiene funciones que controlan el estado de la pantalla gráfica. Las siguiente funciones nos permitirán borrar, dibujar leer y realizar otras muchas tareas

relacionadas con la pantalla.

FUNCIÓN (graphscr)

Esta función nos sirve para cambiarse al modo gráfico.

FUNCIÓN (grclear)

La función **(grclear)** borra el área de visualización activa cuando la función es llamada. El resto de la pantalla permanece igual.

FUNCIÓN (grdraw inicio fin color destacar)

Cuando llamamos a la función **(grdraw),** se dibuja una línea entre dos puntos, inicio y fin. Podemos usar cualquiera de los colores permitidos por AutoCAD. Si utilizamos el color -1 se hará un O exclusivo de la línea con la pantalla, complementando los colores de las zonas por las que pasa y neutralizándose al dibujar sobre el mismo. i su sistema permite destacar en la pantalla, si se introduce el argumento destacar y es distinto de cera la línea será destacada.

FUNCIÓN (grtext caja texto destacar)

La función **(grtext)** permite dibujar texto en partes no gráficas de la pantalla. El texto introducido será escrito en una caja numérica (desde 0 hasta el tipo de menú más elevado menos 1). Igual que en la función grdraw, podemos destacar el texto si el sistema lo permite.

FUNCIÓN (grread coordenadas)

La función **(grread)** lee el dispositivo apuntador de AutoCAD y devuelve una lista. El primer elemento de esta lista es un numero que especifica el tipo de dispositivo que envía la información que va a continuación. La lista de códigos y los dispositivos a los que representan es la siguiente:

2	(caracter-teclado codigo-ascii)
3	(punto-seleccionado lista-de-coordenadas)
4	(celda-menú-pantalla número-caja)
5	(modo-arrastre coordenadas)
6	(elemento-menú-botones numero-boton)
7	(opción-menu-tablero1 número-caja)
8	(opción-menu-tablero2 número-caja)
9	(opción-menu-tablero3 número-caja)
10	(opción-menu-tablero4 número-caja)
11	(opción-menú-aux1 número-caja)
12	(coordenadas-boton-apuntador boton-apuntador)
13	(opción-menú-pantalla número-caja)

FUNCIÓN (textscr)

En AutoCAD existen dos modos de funcionamiento, texto y gráficos. La función textscr permite configurar la pantalla en modo texto.

FUNCIÓN (vports)

Cada ventana de visualización tiene una lista de descriptores que determina su tamaño y su posición. Si utilizamos la función **(vports),** obtendremos los descriptores de las ventanas de visualización que se encuentren activas.

La descripción de una ventana de visualización consta del número de la ventana seguido de dos pares de números reales. Estos números no son coordenadas. En vez de esto, se refieren a la parte de la pantalla ocupada por una ventana determinada. El primer par de números reales se refiere a la esquina inferior izquierda, mientras que el segundo se refiere a la esquina superior derecha. Si solo hay una ventana de visualización, y ocupa toda la pantalla, y las coordenadas de la esquina inferior izquierda serán (0 0) y las de la esquina

superior derecha serán (1 1). Al aumentar el numero de ventanas de visualización, la parte del total de la pantalla disminuye. Así, una ventana de visualización que ocupe la mitad de la pantalla podría tener su esquina inferior izquierda en el punto (0 0.5) y su esquina superior derecha en (1 1)

Ejemplo:

(viewports)

Devuelve ((1 (0 0)(0.5 1))(2(0.5 0)(1 1)))

FUNCIONES DE ENTRADA SALIDA

Para comunicarnos con AutoLISP y, a través de este, con AutoCAD, tenemos que poder leer valores introducidos por el usuario. Existen muchas formas de comunicarnos con AutoCAD. Podemos escribir respuestas desde el teclado, seleccionar valores con un digitalizador o utilizar un ratón. AutoLISP ofrece un conjunto de funciones básicas para permitir la comunicación con AutoCAD.

FUNCIONES DE ENTRADA DE LA PANTALLA

FUNCIÓN (getangle punto mensaje)

La función **(getangle)** lee una expresión angular introducida por el usuario. Podemos introducir un punto, que será utilizado como punto base para obtener en ángulo. También podemos incluir un mensaje indicando que se introduzca el ángulo. El ángulo será devuelto en radianes, aunque puede ser introducido por el usuario en las unidades activas en cada momento. la respuesta del usuario puede ser seleccionar el ángulo con una línea móvil.

Ejemplo:

(getangle)

Pide un ángulo: [damos un ángulo] 1.25

(setq a '(12.34 45.76))

(getangle a "Escribe el ángulo que quieras:")

Devuelve Escribe el ángulo que quieras: 2.75

FUNCIÓN (getcorner punto mensaje)

La función **(getcorner)** nos sirve cuando queremos pedirle al usuario que introduzca la segunda esquina de una caja que puede seleccionar con un rectángulo móvil. La función getcorner pasa las coordenadas de un punto como argumento y opcionalmente se puede enviar un mensaje. Después de comenzar la función, el rectángulo empezara en el punto especificado y puede ser extendido hasta seleccionar la posición definitiva de la otra esquina.
Ejemplo:

(getcorner '(10 11) " Seleccione la esquina opuesta")

Devuelve "Seleccione la esquina opuesta:

FUNCIÓN (getdist punto mensaje)

La función **(getdist)** permite que el usuario introduzca una distancia. La distancia puede ser introducida en respuesta a un mensaje que especifiquemos. Si especificamos un punto, será utilizado como punto inicial para la distancia; si no es así, el usuario puede seleccionar un punto inicial
Ejemplo:

(getdist)

Pide una distancia: 25

(getdist '(12 30) " De una distancia, por favor:")

Devuelve De una distancia, por favor: 50

FUNCIÓN (getint mensaje)

La función **(getint)** la utilizamos cuando queremos que el usuario teclee un número entero. Opcionalmente, podemos pasarle como argumento un mensaje que aparecerá cuando la función empiece a ejecutarse.
Ejemplo:

(getint "Introduzca un número entero:")

Devuelve Introduzca un número entero: 1

FUNCIÓN (getkword mensaje)

La función **(getkword)** permite pedirle al usuario una respuesta que tiene que estar dentro de una lista de posibles opciones. Para crear la lista, hay que utilizar la función initget,. Si la palabra introducida no esta en la lista de opciones válidas, AutoCAD indicará al usuario que repita la entrada.

Ejemplo:

(initget 3 " Ángulo Base")

(getkword "Introduzca ángulo o base:")

Devuelve Introduzca ángulo o base: ángulo

FUNCIÓN (getorient punto mensaje)

Tenemos dos opciones a la hora de leer ángulos. Generalmente en AutoLISP los ángulos se especifican en relación a la parte positiva del eje X como cero grados, incrementándose en sentido antihorario. Por supuesto, es posible establecer otra base y dirección con las variables del sistema **ANGBASE** y **ANGDIR**. El usuario puede utilizar también la orden UNITS para modificar el ángulo base y la dirección de rotación . La función **(getorient)** devuelve la medida angular en términos del ángulo base y la dirección de incremento, activos en cada momento.

Ejemplo:

(getorient) _.

Pide Un ángulo: 0.75

(getorient)

Pide la orientación: 2.75

FUNCION (getpoint punto mensaje)

La función **(getpoint)** permite que el usuario introduzca un punto. Podemos utilizar opcionalmente un punto base o un mensaje. Si introducimos un punto base, este originara el uso de una línea móvil que apuntara a la posición de las líneas indicadoras.

Ejemplo:

(getpoint '(3 6) "introduzca un punto:")

Devuelve Introduzca un punto: (5 5)

FUNCIÓN (getreal mensaje)

La función **(getreal)** es similar a la getint, con la diferencia de permite leer un numero real.

Ejemplo:

(getreal "Introduzca un número real:")

Devuelve Introduzca un numero real: 4.5

FUNCIÓN (getstring retorno-carro mensaje)

La función (**getstring**) permite leer una cadena introducida por el usuario. Si se utiliza el argumento opcional retorno-carro y es distinto de nil, la cadena podrá contener espacios; de lo contrario, si aparece cualquier espacio en blanco se detendrá la función. Como en la mayoría de las funciones get, se puede utilizar un mensaje.

Ejemplo:

(getstring 1 "Introduzca el nombre de este objeto:")

Devuelve Introduzca el nombre de este objeto: "hola"

FUNCIÓN (initget bitfield cadena)

La función **(initget)** inicializa toda la familia de funciones get. Existen muchas formas posibles en las que un usuario puede responder a los mensajes. Es posible que queramos

mostrar las respuestas invalidas inadecuadas, tema que entra dentro de las tareas de la función initget.

Para seleccionar los estados que queremos utilizar conjuntamente con initget, hay que sumar los números. Los números tienen los siguientes significados:

1	No permitir entradas nulas
2	No permitir valores iguales a cero
4	No permitir valores negativos
8	No verificar los limites, independientemente de LIMCHECK
16	Se obtendrán puntos en tres dimensiones en vez de dos
32	Representar los rectángulos y líneas móviles con líneas discontinuas

FUNCIONES DE SALIDA DE LA PANTALLA

Igual que podemos leer la información introducida por el usuario, podemos enviar información de AutoLISP a la pantalla. Con estas funciones se envía texto a la pantalla sin esperar una respuesta del usuario.

FUNCIÓN (prin1 expr descriptor-archivo)

La función (**prin1**) tiene como argumentos una expresión y, opcionalmente, un descriptor de archivo un descriptor de archivo es una operación que realizamos con un archivo como W de escribir. La expresión puede ser cualquier expresión valida en AutoLISP. Dentro de los tipos soportados, podemos imprimir cadenas, enteras y reales. Si se utiliza un descriptor de archivo, se supone que el archivo está abierto y que el descriptor es valido.

Ejemplo:

(setq x 25)

(prin1 x)

Devuelve 25

FUNCIÓN (princ exp descriptor-archivo)

La función **(princ)** es prácticamente idéntica a la print1. La única diferencia consiste en que los caracteres de control que aparecen en la expresión no son expandidos. la función princ es apropiada para utilizarla con la función read-line. Print1 se utilizaría con load.

Ejemplo:

(setq x "\nFuncion a imprimir")

(princ x)

Devuelve '\nFuncion a imprimir')

Si hubiéramos utilizado print1, el salto de línea habría añadido una línea antes de imprimir.

FUNCIÓN (print expr descriptor-archivo)

La función **(print)** es esencialmente similar a prin1. La única diferencia consiste en que se avanza una línea antes de imprimir la cadena y se le añade un espacio al final. Así, una cadena como "Cadena", se imprimirá como si fuera "\nCadena.\n".

Ejemplo:

(print "Cadena de texto".)

Devuelve "Cadena de texto"

FUNCIÓN (prompt mensaje)

La función **(prompt)** imprime un mensaje y deja el cursor detenido al final del mensaje. La ventaja de esta función sobre el grupo de funciones prin consiste en que este

mensaje aparecerá en ambos monitores en un sistema con dos monitores.

Ejemplo:

(prompt "Mensaje")

Devuelve Mensaje

FUNCIÓN (terpri mensaje)

La función (terpri) realiza la labor de imprimir una línea en blanco en la pantalla.

Ejemplo:

(terpri)

Devuelve una línea en blanco

FUNCIÓN DE ENTRADA SALIDA PARA ARCHIVOS

AutoLISP es capaz de escribir, completar y leer los archivos. Para hacer esto, AutoLISP nos da acceso a las llamadas a las funciones habituales del sistema operativo a través de sus propios mecanismos. Si conoce el lenguaje C, encontrara el mecanismo de AutoLISP no es muy diferente de los métodos de acceso de C.

FUNCIÓN (open nombre-archivo)

Si abrimos un archivo, habrá que cerrarlo de nuevo con la función close para evitar que se acumulen demasiados archivos abiertos y para asegurarse de que todos los datos enviados al archivo han sido escritos en éste. Hay que utilizar el nombre de archivo con el que se abrió. Si el archivo ya esta cerrado, se producida un error.

Ejemplo:

(close archivo)

FUNCIÓN (findfile nombre-archivo)

La función **(findfile)** permite determinar si existe un archivo, o si se encuentra en un

directorio especifico. Especificando el nombre del archivo, se realiza una búsqueda de la forma en que lo hace el sistema operativo. Por ejemplo si estamos utilizando el dos, la búsqueda se realizara comenzando por el directorio activo y continuara por los caminos que pueden estar especificados en el entorno.

Ejemplo:

(findfile "command.com")

Devuelve "/command.com"

(findfile "/acad/acad.exe")

Devuelve "/acad/acad.exe"

FUNCIÓN (load nombre-archivo caso-error)

La función **(load)** permite cargar expresiones de AutoLISP y evaluarlas inmediatamente. Si se utiliza el argumento opcional caso-error , en el caso de que la función falle por cualquier razón, se devolverá el valor suministrado.

Ejemplo:

(load "función" -1)

Si devuelve -1 es que la función ha fallado.

FUNCIÓN (open nombre-archivo método)

Para leer o escribir en un archivo, hay que abrirlo previamente. La función **(open)** utiliza como argumentos el nombre del archivo y el método de acceso. El nombre del archivo puede ser cualquier nombre permitido por el sistema operativo, incluyendo una vía de acceso. El método tiene que ser una de las tres letras minúsculas siguientes:

w El archivo se abre para escritura.

r El archivo se abre para lectura.

a El archivo se abre para lectura pero Si el archivo no existe lo crea.

Ejemplo:

(setq archivo1 (open "archivo1" "r")

Si devuelve nil es que el archivo no existe.

(setq archivo1 (open "archivo1" "w")

Devuelve <file#001>

(setq archivo2 (open "archivo2" "a")

Devuelve <file#002>

FUNCIÓN (read-char descriptor-archivo)

La función **(read-char)** permite leer un carácter de un archivo. SI se le pasa un descriptor de archivo valido, se utilizara el archivo que ha sido abierto para lectura con ese descriptor. Si no se utiliza descriptor, se leerá desde el teclado. Si no hay ningún carácter en la memoria intermedia del teclado, la función esperará hasta que el usuario pulse una tecla.

Ejemplo:

(read-char)

Si tiene un carácter almacenado en la memoria devolverá tomara este carácter como el leído

(read-char)

Teclee 97

El 97 será tomado como un carácter en ascii, el cual es a.

FUNCIÓN (read-line descriptor de archivo)

La función **(read-line)** permite leer una línea de un archivo hasta el próximo carácter de avance de línea. La línea puede estar en un archivo o ser introducida desde el teclado (si no utiliza descriptor). Si se encuentra el final de archivo, la función devolverá nil.

Ejemplo:

(read-line)

El usuario escribe una línea: Línea a escribir

(read-line leer1)

AutoCAD lee la primera línea del archivo leer1

FUNCIÓN (write-char numero descriptor-archivo)

La función **(write-char)** permite escribir un carácter en la pantalla si omitimos el argumento correspondiente al descriptor de archivo. Si utilizamos un DESCRIPTOR de archivo valido, el carácter será escrito en un archivo. El numero corresponde al código Ascii del carácter a escribir.

Ejemplo:

(write-char (ascii "N"))

Escribe N en la pantalla

(write-char (74) escribir1)

Escribe la letra J en el archivo escribir1

FUNCIÓN (write-line cadena descriptor-archivo)

La función **(write-line)** permite escribir una cadena en un archivo. Podemos omitir el DESCRIPTOR de archivo para que se imprima la línea en la pantalla.

Ejemplo:

(write-line "línea a imprimir".)

La línea se imprimirá en la pantalla: línea a imprimir

(write-line "línea a imprimir". enviar)

La línea se imprimirá en el archivo enviar: línea a imprimir

FUNCIONES DE MANIPULACIÓN DE ENTIDADES

AutoCAD permite crear entidades de muchos tipos. AutoLISP posee un grupo de funciones que nos ayudan a manipular estas entidades.

FUNCIONES DE SELECCIÓN DE CONJUNTOS

Las entidades las podemos seleccionar de acuerdo a ciertas características por ejemplo el tipo de layer donde se localizan, el tipo de entidad, el nombre de la entidad, ó haciendo una selección manual con el Mouse. Para poder hacer estas selecciones utilizamos

las funciones de selección de conjuntos. Ya seleccionadas las entidades podemos realizar las tareas que queramos con ellas.

FUNCIÓN (ssadd nombre-entidad conjunto-seleccion)

La función **(ssadd)** funciona de tres formas si no le pasamos argumentos nos regresara una conjunto de selección vacía. Si llamamos a la función pasándole exclusivamente un nombre de entidad, la función crea un nuevo conjunto de selección y le añade el nombre de la entidad. Si se utilizan un nombre de entidad y un conjunto de selección ya existente, el nombre de entidad se añade al conjunto de selección determinado.

Ejemplo:

(setq nuevo (ssadd))

(ssadd entidad nuevo)

FUNCIÓN (ssdel nombre-entidad conjunto-seleccion)

La función (ssdel) permite suprimir un nombre de entidad de un conjunto de selecciones. Para que el nombre de entidad sea eliminado, hay que especificar un nombre de entidad y un nombre de conjunto de selecciones.

Ejemplo:

(ssdel "Bloque1" conjunto)

FUNCIÓN (ssget "X" filtros)

La función **(ssget)** filtra las entidades de un dibujo utilizando varios criterios de filtro. Los filtros utilizados consisten en una lista de asociación. Por ejemplo, podríamos obtener un conjunto de selección de todas las líneas verdes de una capa del dibujo. La lista de filtro tiene que contener parejas de códigos de grupos seguidos por nombres. Los códigos de grupo que identifican los nombres utilizados en la lista de asociación son los siguientes:

0 Tipo de entidad
2 Nombre del bloque
6 Nombre del tipo de línea
7 Nombre del estilo de texto
8 Nombre de la capa
39 Espesor (real)
62 Numero de color
66 Atributos
210 Definición de dirección de extrusión con 3 reales

La lista de asociación utiliza los códigos de grupo para identificar el tipo de nombre que sigue a cada código.

Ejemplo:

(setq set (ssget "X" (list (cons 2 "bloque1"))))

Esto devolverá todos los bloques llamados bloque1

FUNCIÓN (sslength conjunto-seleccion)

Si utilizamos **(ssget)**, podemos conocer la longitud del conjunto de selección devuelto por la función. Para ello, hay que darle el nombre del conjunto como argumento a la función sslength.

Ejemplo:

(setq set (ssget "X" (list (cons 2 "bloque1"))))

Esto devolverá todos los bloques llamados bloque1

(sslength set)

25

El numero 25 indica que se han encontrado 25 ocurrencias del bloque Bloque1, dentro del dibujo.

FUNCIÓN (ssname conjunto-selección índice)

Si ha utilizado ssget para obtener un conjunto de selección de entidades utilizadas en un dibujo, La función **(ssname)** permite obtener el nombre de cualquiera de las entidades del conjunto mediante un numero índice. Si sabemos cuantas entidades hay en el conjunto de selección (supongamos que hemos utilizado sslength para conocerlo), podemos examinar el conjunto por posiciones. La primera posición se toma como cero.

Ejemplo:

(setq set (ssget "X" (list cons 2 "bloque1"))))

Esto devolverá todos los bloque llamados bloque1

(sslength set)

25

Esto indica que hay 25 entidades en el dibujo.

(ssname set 24)

Esta llamada devuelve el numero de la ultima entidad.

FUNCIONES SOBRE NOMBRES DE ENTIDADES

Existe un conjunto de funciones de AutoLISP que permite manipular nombres de entidades. Estas funciones operan en toda la base de datos de AutoCAD. Por ejemplo, podemos conocer la siguiente entidad, la ultima entidad, o pedirle al usuario que seleccione un objeto.

FUNCIÓN (entnext nombre-entidad)

Si queremos conocer el nombre de la primera entidad de la base de datos, sólo tenemos que utilizar el valor de retorno de la función entnext sin argumentos, Si queremos conocer el nombre de la entidad que sigue a una dada, tan sólo hay que pasarle como argumento el nombre de dicha entidad. La función entnext nos permite recorrer la base de datos, utilizando el nombre de la entidad siguiente como argumento de entnext. Si se han

suprimido algunas entidades después de la entidad introducida, la función devolverá nil.

Ejemplo:

(setq entidad1 (entnext))

(setq entidad2 (entnext entidad 1))

FUNCIÓN (entlast)

La función **(entlast)** devuelve el nombre de la última entidad del dibujo. Como en entnext, si se han suprimido algunas entidades, no serán devueltas.

Ejemplo:

(setq entidad1 (entlast))

FUNCIÓN (entsel mensaje)

La función **(entsel)** permite que el usuario seleccione una entidad. Opcionalmente, podemos incluir un mensaje. Obtendremos una lista cuyo primer elemento es el nombre de la entidad y el segundo el par de coordenadas que identifica el punto utilizado para seleccionar la entidad.

Ejemplo:

(setq entidad1 (entsel "seleccione un objeto:"))

El usuario selecciona un objeto y obtendremos el nombre de la entidad.

FUNCIÓN (handent punto-entidad)

Si queremos conocer el punto de referencia de una entidad y queremos obtener el nombre actual de dicha entidad, podemos utilizar la función **(handent).** Los puntos de referencia son eternos, mientras que una entidad cambia de nombre al cargar el dibujo.

Ejemplo:

(handent "1B75D")

Devuelve <Entity name: 60000023>

FUNCIONES DE DATOS PARA ENTIDADES

Podemos borrar entidades, obtener información sobre una de ellas, modificarlas y actualizarlas. Las funciones de datos para entidades son el método ofrecido por AutoLISP para modificar la información almacenada en dichas entidades.

FUNCIÓN (entdel nombre-entidad)

La función **(entdel)** permite borrar una entidad a partir de su nombre. esta función actúa como un conmutador, de manera que si la entidad ha sido borrada, la recupera.

Ejemplo:

(entdel entlast))

Con esto la ultima entidad ha sido borrada.

FUNCIÓN (entget nombre-entidad)

La función **(entget)** devuelve una lista de datos sobre la definición de la entidad a partir de su nombre. La lista está en el mismo formato que las listas de la función ssget, que consiste en un código de grupo seguido de un nombre. Se permiten todos los grupos de códigos posibles.

Ejemplo:

(setq lista1 (entget entnext)))

((-1 . < Entidad name: 60000023 >)

(0 . "Line")

(8 . "Pline")

(10 3,4 10.9 3.0)

(11 12.0 12.1 3.0)

FUNCIÓN (entmod lista-entidad)

La función **(entmod)** realiza la función opuesta a la función entget. Mientras que entget devuelve una lista de los datos que configuran la entidad deseada, la función entmod coge la lista de una entidad y sustituye los datos de la entidad deseada con la lista introducida en la función.

Ejemplo:

Se tiene que lista1 contiene los datos:

```
(setq lista1
      (substr ( cons 10 0.0 0.0 0.0 )
          (assoc 10 lista1 )
              lista1 ))
(entmod lista1 )
```

FUNCIÓN (entupd nombre-entidad)

Al utilizar la función **(entmod)** conjuntamente con objetos complejos, tales como polilíneas, no veremos la modificación de la entidad sobre la pantalla. Para visualizar un cambio después de efectuarlo , podemos llamar a la función entupd, dándole el nombre de la entidad que ha sido modificada.

Ejemplo:

```
(entupd lista1 )
```

FUNCIONES DE ACCESO A LAS TABLAS DE SÍMBOLOS

AutoCAD utiliza tablas para almacenar información sobre un dibujo y la forma en que esta siendo visualizado. También guarda información sobre capas, tipos de líneas etc. que podemos consultar con las tablas de símbolos.

FUNCIÓN (tblnext nombre-tabla inicializar)

Podemos examinar las tablas LAYER, LTYPE, UCS, BLOCK y VPORT. Cada vez que utilizamos **(tblnext)**, obtendremos el siguiente valor de la tabla que deseamos. Si se especifica el argumento inicializar y es distinto de nil, la posición de la función será establecida al principio de la tabla. Al final de la tabla, la función devolverá nil. La información suministrada por la función estará organizada como las listas definidas para la función entget.
Ejemplo:

(tabnext "LAYER")

FUNCIÓN (tblsearch nombre-tabla nombre-entrada siguiente)

La función **(tblsearch)** busca un elemento en una tabla. si se especifica el argumento siguiente y es distinto de nil, se obtendrá el elemento de la tabla que sigue al que hemos introducido.
Ejemplo:

(tblsearch "LAYER" "CAPA1")

FUNCIONES VARIAS

Algunas funciones por sus funciones no caen específicamente en alguna de las clasificaciones que hemos mencionado anteriormente, pero son necesarias para el desarrollo de programas en AutoLISP.

FUNCIÓN (ver)

La función **(ver)** devuelve el numero de la versión de AutoLISP que se esta utilizando. Podemos utilizarlo en los programas para asegurarnos de que ciertas prestaciones están soportadas por la versión que se esta ejecutando. De esta forma, podemos mostrar los correspondientes mensajes en la pantalla, o programar soluciones alternas.

Ejemplo:

(ver)

Devuelve "AutoLISP Release 10.0"

FUNCIÓN (alloc numero)

La función **(alloc)** establece la cantidad de memoria solicitada cuando se utiliza la función expand o cuando se necesitan mas nodos para realizar el proceso de evaluación. En este caso, utilizamos el espacio necesario para almacenar 512 nodos, que será la unidad de incremento de espacio para futuras peticiones.

Ejemplo:

(alloc 600)

512

FUNCIÓN (expand numero)

En AutoCAD podemos utilizar mas segmentos de memoria para AutoLISP antes de que sean necesarios. Para ello la función **(expand),** cuyo parámetro es un numero que será multiplicado por el valor indicado por alloc.

Ejemplo:

(expand 5)

Devuelve 10240 que es el número de segmentos solicitados para el espacio de nodos.

FUNCIÓN (gc)

La función (**gc**) llamada así por ser la abreviación de garbage colletion o recogida de desperdicios es la función por la cual AutoLISP elimina nodos que ya no se utilizan.

FUNCIÓN (mem)

La función (**mem**) permite conocer el estado de la memoria. Muestra la cantidad de memoria para nodos, nodos libres, segmentos, espacio libre y recuperable. El valor devuelto es nil.

FUNCIÓN (vmon)

La función (**vmon**) activa la paginación virtual de AutoLISP. Cuando esta activa, permite al programa superar los limites del espacio disponible para nodos. Antes de utilizar funciones defun, si queremos que sean pagínales, tenemos que ejecutar vmnon. El proceso de paginación incrementa ampliamente la capacidad del sistema, ya que intercambia paginas de código con la unidad o la memoria extendida según sea necesario.

FUNCIÓN (trace funcion1 funcion2 ... funciónN)

La función (**trace**) permite observar el funcionamiento de las funciones que se introducen como argumentos. Cuando se evalúa cualquiera de las funciones de la lista, se ve el principio y final de la función.

Ejemplo:

(trace martillos clavos)

Devuelve CLAVOS

FUNCIÓN (untrace funcion1 funcion2 ... funciónN)

La función (**untrace**) permite desactivar el seguimiento de las funciones elegidas con trace que se le pasan como parámetros.

FUNCIÓN (*error* cadena)

La función (***error***) permite controlar el proceso de salida en caso de error. Hay que declararla como función.
Ejemplo:

```
(defun *error* (mensaje)
        (princ "Error:")
        (princ mensaje)
        (terpri))
```

FUNCIÓN (pi)

El valor de pi es de 3.141592654..... En AutoLISP, el símbolo pi es una constante que será evaluada como dicho numero.

FUNCIÓN (nil)

El valor de nil es muy especial. No significa cero. Significa "nada ha sido definido o asociado".

FUNCIONES DE OPERACIONES BOOLEANAS DE BITS

Una operación lógica con bits es aquella en que se utiliza un entero combinado con un operador lógico.

FUNCIÓN (~ numero)

Puede que deseemos verificar un numero para encontrar su función NOT bit a bit. En este caso, se invierten todos los bits. El valor devuelto por la función (~) será un numero que tenga todos los bits 0 del argumento a 1 y, viceversa, todos los bits del argumento que estaban a 1 estarán a 0.
Ejemplo:

Se tiene que el numero 9 es en binario 0111

(~ 9)

Devuelve (-10)

El valor devuelto es 0110 que es el numero binario de -10.

FUNCIÓN (boole operador entero1 entero2)

La función **(boole)** permite obtener el resultado de aplicar un operador determinado a un par de enteros. El operador es un numero de cuatro bits del 0 al 15. Existen 16 funciones posibles, que pueden aplicarse sobre 2 variables, entero1 y entero2. Algunos valores útiles del operador son 6 (XOR), 7 (OR), y 8 (NOT).

Ejemplo:

Si el operador es 7 la función se comportara de la siguiente forma:

(boole 7 4 2)

Devuelve 6

FUNCIÓN (logand numero1 numero2 ... numeroN)

La función **(logan)** permite realizar operaciones cuando la función boole da resultados confusos, que nos pueden ocasionar equivocaciones.

Ejemplo:

(logand 10 5)

Devuelve 0

FUNCIÓN (logior numero1 numero2 ... numeroN)

La función **(logior)** realiza la operación lógica OR entre una lista de enteros.

Ejemplo:

(logior 10 4)

Devuelve 14

FUNCIÓN (lsh numero bits)

La función **(lsh)** permite desplazar bits en un números enteros. Si el argumento bits es positivo, los bits serán desplazados a la izquierda. Si es negativo, serán desplazados a la derecha. El desplazamiento de bits desécha los bits que salen del tamaño de la palabra de memoria. Si una palabra es desplazada hacia la derecha, se pierden los bits del extremo derecho. Estos caen a la derecha de la palabra. Si un bit coincide con la posición del extremo izquierdo de una palabra, su digno pasa a ser negativo. Si dicha posición entre un bit a 0, el signo del numero será positivo.

Ejemplo:

Tenemos el numero 10 decimal (en binario es 00001010)

(lsh 10 + 1)

Devuelve 20 (Por que es el decimal de 00010100)

BIBLIOGRAFÍA

AUTOCAD MANUAL DE REFERENCIA

NELSON JOHNSON

EDITORIAL MCGRAW HILL 1990

MEXICO, 1992

INSIDE AUTOCAD

D. RAKER AND H. RICE

EDITORIAL NEW RIDER PUBLISHING

OREGON, 1990