Contenido

Programación Gráfica

Introducción

Dentro del desarrollo de sistemas informaticos existen aquellos que utilizan tecnología combinada entre graficos e información lo que los convierte en sistemas de analisis, existen otros sistemas de información que son graficos con objetos elementales tales como el punto, la linea, el arco, y el texto que van conformando una base de información visual la misma que de igual forma sirve para realizar analisis de esos objetos ya que estos pueden representar por ejemplo: ciudades, localidades, tendido de lineas telefonicas, vias, trazado de tuberías de agua, alcantarillado, gas y muchisimos elementos mas de acuerdo a la conformacion de esa información en composiciones arquitectonicas, de ingenieria civil, agronomia, catastros y mas ramas que pueden valerse de este tipo de herramientas para sus representaciones. Al referirse a herramientas se reconoce que en el mercado existen tales, sin ebmargo las mismas pueden ser optimizadas a traves del lenguaje fuente con las que fueron creadas y ese es uno de los objetivos del presente curso, ademas de establecer la base de cómo funcionan dichas herramientas en el hambito del tratamiento de la información grafica.

Una de estas herramientas y por su difusión y uso a nivel mundial de la mismas es el CAD (Diseño Asistido por Computadora), que es la base de generacion de información que se utiliza en diferentes ambitos, ademas de ser la base generadora de información georeferenciada para los sistemas GIS (Sistema de Información Geográfico).

Es por tanto que una de las palicaciones CAD, fue desarrollada por la empresa AutoDesk, con su producto estrella AutoCAD, desde su creación con fines militares hasta llegar ahora con fines comerciales en diferentes areas de aplicación.

Cunado se desarrollo el AutoCAD, su lenguaje de estructuracion fue el LISP, podero lenguaje de bajo nivel de programación el mismo que por su versatilidad puudo implementar un enorme potencial grafico y de desarrollo al AutoCAD, de alli nace una combinación para denominar a un lenguaje de desarrollo denominado AutoLISP, el mismo que combina comandos propios del CAD con funciones del LISP.

Sin mas preámbulos se pasara a conocer este maravilloso mundo de programación grafica dentro de un entorno grafico, que permitira establecer una directriz a un nuevo concepto de sistemas informaticos, de desarrollo, optimizacion y analisis.

1 Capitulo 1: Conceptos previos y definiciones

1.1 Lenguaje de programación LISP

El **LISP** es un lenguaje de programación relativamente antiguo, desarrollado en los años cincuenta para la investigación en Inteligencia Artificia.

Su nombre proviene de **LISt Processing** (Procesado de Listas), puesto que la base de su funcionamiento es el manejo de listas en vez de datos numéricos.

Una lista es un conjunto de símbolos que pueden ser nombres de variables, datos concretos numéricos o textuales, funciones definidas por el propio usuario, etc. El símbolo es, pues, la unidad básica con un contenido o un significado para el programa en **LISP**.

La lista es el mecanismo que junta una serie de símbolos y los evalúa, es decir, los procesa, obteniendo un resultado. El lenguaje LISP procesa directamente las listas en cuanto se encuentran formadas y obtiene o "devuelve" el resultado de ese proceso.

Esta característica del manejo de listas otorga al LISP una gran versatilidad y le distingue de otros lenguajes de programación orientados a la manipulación de números.

Las ventajas que supone la utilización de un lenguaje basado en LISP para programar desde AutoCAD se podrían resumir en los siguientes puntos:

- Facilidad para manejar objetos heterogéneos: números, caracteres, funciones, entidades de dibujo, etcétera Para LISP basta representar cualquiera de esos objetos con un "símbolo", y no hay necesidad de definir previamente qué tipo de datos va a contener ese símbolo.
- Facilidad para la interacción en un proceso de dibujo.
- Sencillez de aprendizaje y comprensión.
- El hecho de que el LISP sea un lenguaje muy utilizado en investigación y desarrollo de Inteligencia Artificial y Sistemas Expertos.
- Sencillez de sintaxis, por lo que un intérprete de LISP es fácil de realizar y resulta convenientemente pequeño.

El LISP es un lenguaje que es evaluado en vez de compilado o interpretado.

Los lenguajes interpretados por la computadora y cada palabra es convertida a lenguaje maquina. Esto hace que sean muy lentos.

Los lenguajes compilados son mucho más rápidos, porque en el proceso de compilación todo el programa se convierte en instrucciones de máquina.

Un lenguaje evaluado como el LISP es un paso intermedio entre los

interpretados y los compilados. No es tan rápido como estos últimos pero resulta mucho más flexible e interactivo. Es posible, por ejemplo, escribir un programa en LISP que sea capaz de modificarse a sí mismo bajo ciertos aspectos: esta es la base de los llamados Sistemas Expertos.

El mecanismo evaluadar de LISP es la propia lista. Una lista es un conjunto de símbolos separados entre sí por al menos un espacio en blanco y encerrados entre paréntesis. Desde el momento en que existe una expresión encerrada entre paréntesis, el LISP la considera como una lista y la evalúa intentando ofrecer un resultado.

El LISP no es un lenguaje de programación único, sino que existen muchas versiones de LISP: MacLISP, InterLISP, ZetaLISP, Common LISP.

1.2 AutoLISP, una versión especifica de LISP

AutoLISP es un subconjunto del lenguaje de programación Common LISP. Puesto que AutoLISP está diseñado para funcionar desde un dibujo de AutoCAD, se han seleccionado las características de LISP mas adecuadas para este fin y además se han añadido otras nuevas, sobre todo en lo relativo a !a manipulación de entidades de dibujo, acceso a la Base de Datos de AutoCAD e interacción gráfica en general.

Los programas en AutoLISP son simples archivos de texto, con la extensión obligatoria **.LSP**, donde el usuario escribe uno o varios programas contenidos en ese archivo. Una vez hecho esto, basta cargar el archivo desde el la ***Línea de Comandos*** **– Command:** - de AutoCAD. para poder acceder directamente a lodos los programas contenidos en el.

Además se pueden escribir directamente instrucciones en AutoLISP desde la línea de comandos del dibujo en AutoCAD, es decir, escribir conjuntos de símbolos encerrados entre paréntesis. AutoLISP evalúa inmediatamente esa lista y ofrece un resultado. Y si la expresión contiene definiciones de funciones o variables, quedan cargadas en la memoria para su utilización posterior.

Una de las más importantes potencialidades de AutoLISP es el acceso directo a la Base de Datos de un dibujo en AutoCAD. Toda la información del dibujo se encuentra en ella: layers (capas), style (estilos de texto), type line (tipos de líneas), USC (sistemas de coordenadas) o VIEW (vistas almacenadas), atributos con sus diferentes valores en cada inserción, etc., así como todas las entidades contenidas en el dibujo.

Se puede utilizar AutoLISP para modificar esa Base de Datos o para extraer la información que interese de ella con objeto de importarla, por ejemplo, en una base de datos externa. La forma más habitual de extraer información es mediante los datos de atributos asociados a bloques.

1.3 Tipos de objetos en AutoLISP

Los objetos en AutoLISP representan todos los tipos de componentes de un programa. En esencia son dos, como ya se ha dicho: listas y símbolos. Además es posible incluir como elementos de listas valores concretos ("constantes"), ya sean numéricos o textuales.

Atendiendo a sus características se pueden definir varios tipos de objetos en AutoLISP:

Lista: es un objeto compuesto de:

- Paréntesis de apertura.
- Uno o más elementos separados por al menos un espado en blanco
- Paréntesis de cierre.

Los elementos de la lista pueden ser símbolos, valores concretos (“constantes" numéricas o textuales), o también otras listas incluidas.

Elemento: cualquiera de los componentes de una lista

Átomo: representa una información indivisible; Un valor concreto, o un símbolo de variable que contiene un valor son átomos. Una lista o una función delmida por el usuario no son átomos.

Símbolo: cualquier elemento de lista que no sea un valor concreto. El símbolo puede ser un nombre de variable, un nombre de función definida por el usuario o un nombre de comando de AutoLISP.

Enteros: valores numéricos enteros, ya sean explícitos o contenidos en variables.

Reales: valores numéricos con precisión de coma flotante.

Cadenas: valores textuales que contienen cadenas de caracteres (código ASCII).

- Deben ir entre comillas

Descriptores de fichero o archivo: valores que representan un archivo abierto para lectura o escritura desde un programa en AutoLISP.

Por ejemplo:

<File: #5c9Ø>.

"Nombres" de entidades de AutoCAD: valores que representan el "nombre" de una entidad en la Base de Dalos.

Por ejemplo:

<Entity name: 6ØØØØA56>.

"Conjuntos designados" de AutoCAD: valores que representan una designación de entidades de dibujo.

Por ejemplo:
<Sclection set: 3>.

Función de usuario: es un símbolo que representa el nombre de una función definida por el usuario.

Función inherente o Comando: es un símbolo con el nombre de un comando predefinido de AutoLISP, A veces se les llama Subrutinas o Funciones predefinidas.

Funciones ADOS o Subrutinas externas: son símbolos con aplicaciones de ADS en Lenguaje C, cargadas desde AutoLISP. (véase en el Capitulo 17).

En sistemas MS-DOS los enteros son números de 16 bits con signo, es decir, comprendidos entre -32768 y +32767. Los números reales se representan con separación de coma flotante de doble precisión, con un mínimo de 14 cifras significativas.

Las cadenas pueden tener cualquier longitud. No obstante, para valores explícitos ("constantes" de tipo cadena) existe una limitación de 100 caracteres.

1.4 Procedimientos de evaluación en AutoLISP

La base de todo intérprete de LISP es su algoritmo evaluador. Este analiza cada línea del programa y devuelve un valor como resultado. La evaluación en AutoLISP se realiza de acuerdo con las siguientes reglas:

- Los valores enteros, reales, cadenas de texto, descriptores de archivos, así como los nombres de subrutinas o comandos de AutoLISP, devuelven su propio valor como resultado.
- Los símbolos de variables participan con el valor que contienen (que les está asociado) en el momento de la evaluación
- Las listas se evalúan de acuerdo con el primer elemento. Si éste es un nombre de función inherente o comando, los elementos restantes de la lista son considerados como los argumentos de ese comando. En caso contrario, se considera como un nombre de función definida por el usuario, también con el resto de elementos como argumentos.

Cuando los elementos de una lista son a su vez otras listas, éstas se van evaluando desde el nivel de anidación inferior (las listas más “interiores"). Puesto que cada lista contiene un paréntesis de apertura y otro de cierre, cuando existen listas incluidas en otras todos los paréntesis que se vayan abriendo deben tener sus correspondientes de cierre.

Si se introduce desde el teclado una expresión en AutoLISP a la que falta por cerrar algún paréntesis, se visualiza un mensaje:

n>

donde "n" es un número que indica cuántos paréntesis fallan por cerrar. Se pueden teclear esos paréntesis y subsanar el error.

También es posible evaluar directamente un símbolo (extraer, por ejemplo, el valor actual contenido en una variable) tecleando el signo de cerrar admiración seguido del nombre del símbolo.

Por ejemplo:

! P1

devolvería el valor actual contenido en la variable "P1".

1.5 Conveciones de AutoLlSP

Las expresiones contenidas en un programa de AutoLlSP pueden introducirse directamente desde el teclado durante la edición de un dibujo de AutoCAD, escribirse en un archivo de texto ASCII o ser suministradas por una variable del tipo cadena.

Existen una serie de convenciones para la interpretación por AutoLISP de esas expresiones que pueden resumir en los siguientes puntos:

- Los nombres de símbolos pueden utilizar todos los caracteres imprimibles (letras, números, signos de puntuación, etc.), salvo los prohibidos:
  ( ) . ' " ;
- Los caracteres que terminan un nombre de símbolo o un valor explícito (una constante numérica o de texto) son:
  ( ) ' " ; (espacio en blanco) (fin de línea)
- Una expresión puede ser tan larga como se quiera. Puede ocupar varias líneas del archivo de texto
- Los espacios en blanco de separación entre símbolos son interpretados como un solo espacio en cada par de símbolos.
- Los nombres de símbolos no pueden empezar por una cifra. Las mayúsculas y minúsculas son diferentes para AutoLISP.
- Los valores explícitos (constantes) de números pueden empezar con el carácter **+** o , que es interpretado como el signo del número.
- Los valores de constantes de número reales deben empezar con una cifra significativa. El carácter se interpreta como el punto decimal. También se admite **+** o **-** para el signo y "**e**" o "**E**" para notación exponencial (científica).
- No es válida la coma decimal ni tampoco se puede abreviar, como en "6" (hay que poner Ø.6)
- Los valores de constantes con cadenas de texto son caracteres que empiezan y terminan por comillas. Dentro de las cadenas se pueden

incluir caracteres de control mediante la contrabarra \. códigos permitidos son:

****	significa el carácter contrabarra \.
\e	significa "cambio de código/ESC".
\n	significa nueva línea, "Retorno de carro".
\r	significa RETURN.
\t	significa Tabulador
\nnn	significa el carácter con número de código octal (no ASCII que es decimal) “nnn”
\"	significa el carácter ".

Los códigos deben ir en minúsculas. Para incluir en una cadena un código ASCII hay que calcular su valor octal. Por ejemplo, el carácter dólar, “**$**”, es ASCII 36; su valor octal será 44 y en la cadena habrá que indicar el código de control "**\44**".

- El apóstrofo se puede utilizar como abreviatura del comando QUOTE.

  ' lin equivale a (**QUOTE** lin)

  El comando QUOTE devuelve el literal del símbolo. Es decir, cuando en una expresión un símbolo aparece precedido por apóstrofo o se le aplica el comando QUOTE, no se evalúa con el valor que contiene en ese momento, sino que devuelve el propio nombre literal del símbolo.
- Se pueden incluir comentarios en un archivo de texto con programas en AutoLISP, comenzando con un punto y coma, "**;**". A partir de donde encuentre un punto y coma hasta el final de la línea, AutoLISP considera que son comentarios y no los tiene en cuenta.

Se establecen unas convenciones para explicar el formato en que debe ir escrito cada comando de AutoLISP y los argumentos que admite:

- El nombre del comando o función inherente va en mayúsculas para distinguirlo de los argumentos.
- El número de argumentos y el tipo se muestran entre paréntesis quebrados. Los puntos suspensivos indican la posibilidad de especificar más argumentos.
- Los argumentos opcionales, que pueden omitirse, van encerrados entre corchetes.

  (ANGTOS <ang> **[** <modo> **[** <prec>**]]**)

  Indica que el nombre del comando en AutoLISP es ANGTOS, que tiene un **argumento <ang> obligatorio** que no puede faltar, pues en caso contrario produciría un error. Además tiene un argumento optativo que es <modo> y otro argumento optativo que es <prec>, pero que depende de <modo>. Es decir, se puede indicar <modo> y faltar <prec>, pero, si se indica <prec>, debe haberse indicado también <modo>.

2 Capitulo 2: Primeros pasos

2.1 Creación de un programa en AutoLISP

Los programas en AutoLISP son archivos de texto con la extensión **.LSP**. Por tanto, se crean directa te con un Editor, se recomienda el Block de Notas.

Nombre de archivo: Saludo.lsp

Contenido del archivo:

```
(defun c:SALUDO()
  (ALERT “Hola mundo”)
)
```

La función creada SALUDO, al ser cargada dentro de AutoCAD, y ejecutada provocara un resulta de mostrar un cuadro de dialogo con el contenido de “Hola mundo”, como se muestra en la siguiente figura.

2.2 (LOAD <nombre dé archivo> [<reserror>] Cargar programas

Esta función carga las expresiones en AutoLISP contenidas en **<nombre de archivo>**. Este nombre se da entre comillas, puesto que es una cadena de texto, y sin extensión, pues sólo se admiten los archivos con extensión .LSP. El archivo se localiza en el directorio actual.

Si se encuentra en otro directorio, hay que indicar el camino. En Sistema Operativo MS-DOS, como los caminos se indican mediante contrabarras, "\", AutoLISP no la admite directamente como carácter de texto y es necesario indicarla con **dos contrabarras**, "\\".

Por ejemplo, se tiene un programa en un archivo llamado "dblin.lsp", que se encuentra en el directorio actual. Para cargarlo basta con tipear directamente en la Línea de comandos:

```
(LOAD"dblin")
```

Si el archivo se encontrara en un directorio **c:\programs**, la forma de cargarlo seria;

(LOAD"c:****programs****dblin")

Una vez cargado el archivo, todas las variables y funciones definidas en él, así como los resultados parciales de las evaluaciones, quedan almacenadas en las áreas de memoria "heap" y "stack". Si se quieren efectuar modificaciones en el archivo, es preciso volverlo a cargar, pues en caso contrario permanecerían en la memoria las definiciones y evaluaciones previas a la modificación.

Si el resultado de cargar el archivo es correcto, AutoLISP devuelve el valor de la última evaluación del programa Este valor suele ser el nombre de la última función definida en el archivo.

Por ejemplo, supongamos que el contenido del archivo "dblin.lsp" es el siguiente:

(DEFUN diblinea () "nada")

Este programa define una función llamada "diblinea" [el comando (DEFUN) se explica en el siguiente apartado], que de momento no tiene argumentos y su definición es simplemente el texto “nada “.

Al cargar el programa con LOAD se producen tres efectos:

- Crear una función llamada "diblinea", cuya definición se almacena en memoria y permanece así hasta que se modifique su definición, se libere el espacio nodal reservado para ella (procedimiento que se estudiará más adelante) o se salga del Editor de Dibujo terminando la sesión de trabajo.
- Definir esa función con el valor textual "nada".
- Devolver el valor de la evaluación de (DEFUN), que es el nombre de la función creada; en este caso “diblinea”

Así la secuencia completa desde el Editor de Dibujo seria:

Command: ***(LOAD "dblin")***
DIBLINEA
Command: **(diblinea)**
"nada"
Command:

Lo que se ha hecho tras cargar el programa es evaluar la función “diblinea”, recien definida que devuelve el valor “nada”.

Si la operación de cargar el archivo fracasa, se produce generalmente un error de AutoLISP salvo si se ha especificado un argumento <reserror> (resultado si error). En este caso, AutoLISP devuelve el valor de ese argumento en vez de producir un error.

Por ejemplo, se intenta cargar un archivo "dbcir.lsp" que no existe o no se localiza:

Command: ***(LOAD "dbcir" "No encuentro archivo")*** "No encuentro archivo"
Command:

En vez de producir un error, (LOAD) devuelve el mensaje textual "No encuentro archivo".

Se puede crear un archivo llamado ACAD.LISP (o modificar cuando este existe) con una serie de programas incluidos en él. El contenido de este archivo se carga automáticamente en la memoria cada vez que se entra en el Editor de Dibujo. Para todos los programas contenidos en ACAD.LSP no es necesario utilizar el comando LOAD. Este archivo es muy útil cuando interesa disponer de determinadas rutinas cada vez que se entra en un dibujo.

2.3 2.4. (DEFUN <simb> <lista argum> <expr>...) Definir una función

En AutoLISP una función es directamente un programa, pues su evaluación ofrece un resultado una vez cargado el archivo que contiene su definición. Así, un archivo .LSP puede contener muchos programa, según el número de funciones definidas en él. Para evaluarlas no es necesario volver al archivo que las contiene; basta con cargarlo una sola vez.

(DEFUN) se utiliza precisamente para definir funciones o programas de AutoLISP.

<simb> que es el símbolo o nombre de la función a definir, conviene que los nombres de símbolos contengan como máximo seis caracteres, por razones de espacio ocupado en la memoria.

lista de argumentos (al ser una lista debe ir entre paréntesis), que puede estar vacía o contener varios argumentos y, opcionalmente, una barra inclinada y los nombres de uno o mas símbolos locales de la función. Los argumentos o símbolos globales son variables que se almacenan en la memoria y permanecen en ella lo mismo que los nombres de funciones definidas, de forma que pueden ser utilizados por otros programas o funciones de AutoLISP.

Para extraer el valor actual de esas variables, basta con introducir desde la **Línea de Comando** un signo "**!**" seguido del símbolo o nombre de la variable.

Los símbolos locales son opcionales; La barra inclinada debe estar separada del primer símbolo local y, si lo hubiera, del último global por al menos un espacio en blanco. Los símbolos locales se almacenan en la memoria sólo temporalmente, mientras la función definida está en curso. En cuanto termina desaparecen de la memoria, por lo que no pueden ser utilizados por otros programas o funciones. Se emplean cuando se necesitan únicamente dentro de la función definida, evitando que ocupen memoria inútilmente.

Si el símbolo local se encontrará ya creado antes de ser utilizado en la función

definida, recupera el valor que tenía al principio una vez terminada la función. Si no se especifican como locales al definir la función, todos los símbolos creados con **SETQ** dentro de ella se consideran como globales.

El último elemento de la función definidora DEFUN es la expresión en AutoLISP que va a servir de definición de <simb>. Esta expresión puede ser tan compleja como se quiera, ejecutando otras funciones definidas, cargando archivos .LSP, etc.

Los tres elementos deben existir obligatoriamente. Si la función definida no tiene argumentos ni símbolos locales, es preciso indicarlo mediante una lista vacía "()".

Por ejemplo, se define una función llamada "seno", cuya utilidad es calcular el seno de un ángulo.

Su definición es la siguiente:

> Command: ***(DEFUN seno (x) (SETQ xr (* PI (/ x 18Ø.Ø))) (SETQ s (SIN xr)))***
> SENO

Quizá parezca un poco largo para teclearlo directamente desde la línea de órdenes. Lo lógico es incluir los programas en un archivo de texto, con el Block de notas ,como se ha explicado, Pero en estos ejemplos resulta más sencillo y didáctico introducir las expresiones en AutoLISP desde la línea de órdenes, pues no requiere cargar el archivo con LOAD y se evalúan directamente.

En el ejemplo se utilizan comandos ó subrutinas de AutoLISP, como **SETQ** y las operaciones ***** y **/**, que se estudiaran en seguida en estos primeros capítulos. De momento nos sirven para entender el proceso de creación de funciones de usuario con **DEFUN**.

La función definida "**seno**" utiliza una variable global que es "**x**". Lo que hace esa función es crear una variable llamada "**xr**", que transforma el valor del ángulo "**x**" dado en grados sexagesimales a radianes. Para ello divide "**x**" entre 18Ø.Ø y multiplica ese resultado por **PI**. El valor obtenido en radianes se almacena en "**xr**".

A continuación se crea una nueva variable llamada "**s**". **SIN** es el comando de AutoLISP que calcula el seno de un ángulo expresado en radianes (en este caso el valor contenido en "**xr**"), y el valor de ese seno, que es el resultado final que se persigue, se almacena en la variable "**s**".

Puesto que la utilidad de la función "**seno**" es calcular el seno de un ángulo, lógicamente es una función dormida para una variable (en este caso se la ha llamado "**x**"). Esa variable, más que global, que quizá es un término que induce a confusión, se la puede considerar asociada a la función "**seno**". Cada vez que se llame a esa función habrá que indicar para qué valor de la variable asociada se quiere calcular.

Por ejemplo:

Command: ***(seno 3Ø)*** Ø.5

En este caso se ha llamado a la función "**seno**" para un valor de la variable asociada (que aquí se llama "**x**") igual a 3Ø. La expresión en AutoLISP que define la función "seno" es:

(SETQ xr (* PI (/ 18Ø.Ø x))) (SETQ s (SIN xr))

En esa expresión aparece la variable asociada a la función "**x**" en el cociente (/ 18Ø.Ø x). Lo que hace el programa en AutoLISP es, en la expresión que define la función "seno", buscar dónde aparece la variable asociada "**x**" y sustituirla por el valor de esa variable con que se ha llamado a la función. En el ejemplo anterior ese valor era 3Ø. Por tanto:

(seno 3Ø)

equivale a

(SETQ xr (*PI (/ 18Ø.Ø 3Ø))) (SETQ s (SIN xr))

La variable "**xr**" almacena el valor de 3Ø en radianes, que es Ø.523599. La variable "**s**" almacena el valor del resultado de (SIN Ø.523599), que es Ø.5. La llamada a la función (seno 3Ø) devuelve el resultado de la última expresión evaluada, que es (SETQ s (SIN xr)), y por eso el valor devuelto es Ø.5 que es el que interesa. Ese valor queda almacenado en "**s**" por si interesa utilizarlo posteriormente en otro programa en AutoLISP.

Si se extraen los contenidos de los diferentes símbolos utilizados, los resultados son los siguientes:

!seno ((X) (SETQ XR (* (/ 18Ø.Ø x))) (SETQ s (SIN xr)))
!x nil
!xr Ø.523599
!s Ø.5
!setq <Subr: # 48ØØ>
!sin <Subr: # 3b8Ø>

La función de usuario "seno" contiene una lista con las variables (en este caso sólo la "x") y después la expresión que la define. La variable "x" no almacena ningún valor, puesto que es simplemente una variable asociada a la función "seno". Las variables "xr" y "s" son globales y mantienen los valores almacenados. Los nombres de comandos de AutoLISP, como SETQ y SIN, devuelven su definición de subrutinas.

La función "seno" se ha definido con una variable asociada. Con el comando DEFUN se pueden definir funciones de usuario con tantas variables como se desee. Por ejemplo, una función llamada "suma 3" que tiene como finalidad

realizar la suma de tres números:

Command: ***(DEFUN suma 3 (x y z) (SETQ sum (+ x y z)))***
SUMA 3
Command: ***(suma 3 12 25 3Ø)***
67
Command: ***!sum***
67

Para llamar a la función "suma3" hay que especificar los valores de las tres variables asociadas para los cuales se quiere calcular la suma.

Hasta ahora no se han especificado argumentos locales. La función de usuario "seno" tenía únicamente la variable asociada "x". Si se indica "xr" como argumento local:

Command: ***(DEFUN seno (x / xr) (SETQ xr (* PI (/ x 18Ø.Ø)))***
(SETQ s (SIN xr)))
Command: ***(seno 3Ø)***
Ø.5
Command: ***!xr***
nil

La función "seno" se ejecuta de la misma manera que antes. Pero ahora la variable "xr" ha sido especificada como local y pierde su valor al terminar de ejecutarse "seno". Almacena temporalmente el valor de 3Ø grados en radianes (Ø.523599), pero después lo pierde y se queda en nil, puesto que estaba definida como variable antes de llamar a "seno".

Si se define a "xr" como variable global antes de llamar a "seno":

Command: ***(SETQ xr 55)***
55
Command: ***(seno 3Ø)***
Ø.5
Command: ***!xr***
55

La variable xr ha sido definida como global con el valor almacenado 55. Durante laejecución de “seno” funciona temporalmente como argumento local (en el ejemplo con elvalor de Ø.523599), pero después recupera su valor de 55.

La variable “s” funciona en todo momento como global y conserva el valor almacenado al terminar de ejecutarse “seno”.

Las funciones definidas por el usuario mediante (DEFUN) pueden integrarse en AutoCAD como una orden más, Para ello deben tener un nombre precedido por C: y comprender una lista de variables globales vacía, aunque pueden tener variables locales. Lógicamente, el nombre de la función definida no debe

coincidir con el de una orden existente de AutoCAD ni con el de un comando de AutoLISP
Por ejemplo, si en vez de tener que evaluar la función "diblinea" cada vez que se quiere ejecutar interesa implementarla en AutoCAD como una orden nueva, se definiría de la siguiente manera:

Command: ***(DEFUN c:diblinea () "nada")***
C: DIBLINEA
"nada"

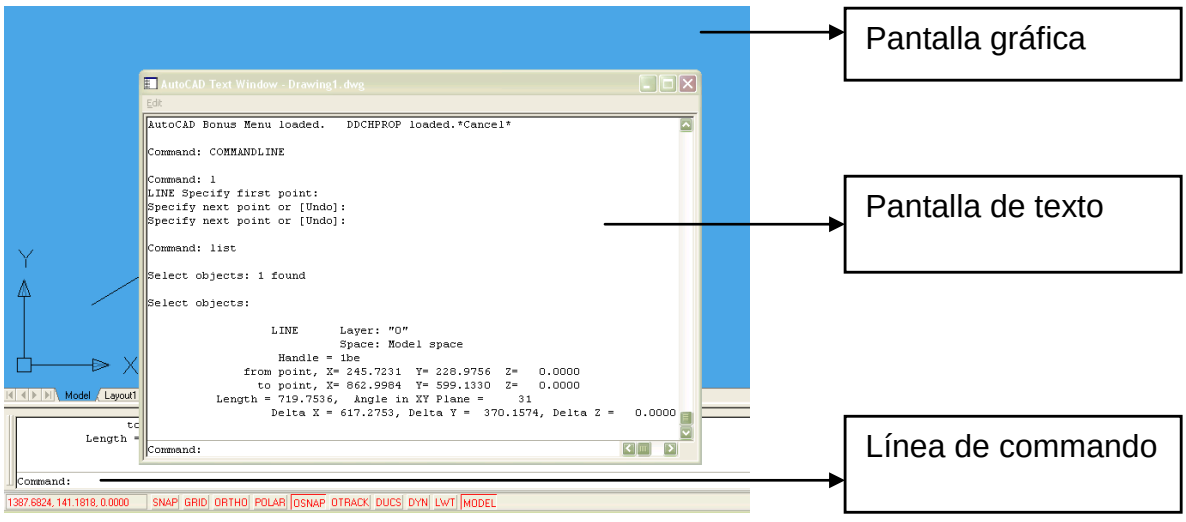
Para ejecutar diblinea ya no es necesario evaluarla entre paréntesis. Ahora funciona como una orden más de AutoCAD, hasta que se libere su espacio en la memoria o se salga de la actual edición de dibujo, La función de usuario "seno" no podría definirse de otra forma:

(DEFUN c:seno (x / xr)……..

puesto que ya se ha dicho que las nuevas órdenes a integrar en AutoCAD no pueden tener argumentos globales o variables asociadas.

Todas las definiciones de funciones incluidas en el archivo ACAD.LSP se cargan automáticamente en la memoria al entrar en el Editor de Dibujo, como ya se ha dicho en el apartado anterior.

2.4 (GRAPHSCR) Pantalla gráfica



Este comando de AutoLISP conmuta de pantalla de texto. Su efecto es el mismo que pulsar la tecla de función "**F1**" desde AutoCAD. Su evaluación devuelve siempre "nil".

Se utiliza para asegurar que el Editor de Dibujo se encuentra en pantalla gráfica, cuando es preciso procesar puntos, entidades y demás elementos gráficos.

2.5 (TEXTSCR) Pantalla de texto

Este comando conmuta a pantalla de texto. Es el complementario del comando anterior. Su evaluación devuelve siempre "nil". Ambos son equivalentes a pulsar sucesivamente la tecla de función "**F1**" desde AutoCAD.

Sirve para asegurar que el Editor de Dibujo se encuentra en la pantalla de texto, cuando se van a mostrar datos no gráficos: por ejemplo, listados de capas, entidades, etc.

En configuraciones con dos pantallas (una de texto y otra gráfica) estos dos comandos no tienen efecto.

2.6 (PROMPT <”mensaje”>) Escribir mensaje

Este comando visualiza la cadena de texto <”mensaje”> en la línea de órdenes de AutoCAD y devuelve "nil". En configuraciones de dos pantallas visualiza el mensaje en ambas. Por esta razón es preferible a otras funciones de escritura como (PRINC), que se estudiarán más adelante.

```
Command: ( PROMPT "Bienvenido a AutoLISP" )
Bienvenido a AutoLISPnil
```

Se observa que el mensaje se visualiza sin comillas. La evaluación de (PROMPT) devuelve "nil" a continuación del mensaje. Para evitar esto se utiliza el comando (TERPRI) explicado en el siguiente apartado.

2.6.1 Ejercicios

Utilizando la función PROMPT

1 Escribir un mensaje diferente

2 Que ocurre con mensajes con los siguientes siguientes caracteres de control

\n
\t
\r
\”
\\

2.7 (TERPRI) Terminar escritura

Este comando mueve el cursor al comienzo de una nueva línea. Se utiliza para saltar de línea cada vez que se escriben mensajes en el área de órdenes de la pantalla gráfica. Existen otros procedimientos como el carácter de control “**\n**” incluido dentro del texto del mensaje, pero a menudo interesa (TERPRI) por razones de claridad.

Esta función no se utiliza para entradas/salidas de archivos. Para ello se

emplea PRINT o PRINC.

```
Command: ( PROMPT "Bienvenido a AutoLISP") (TERPRI)
Bienvenido a AutoLISP
nil
Command:
```

El comando TERPRI salta de línea una vez visualizado el mensaje. Si se hubiera hecho: (PROMPT "\nBienvenido a AutoLISP") se habría producido el salto de línea antes del mensaje, visualizándose éste, pero quedándose al final en la misma línea del mensaje.

2.8 (SETQ <simb1><expr1>[<simb2> <expr2>...]) Atribuir valores

Este comando atribuye el valor de la expresión <exprl> a la variable cuyo nombre es<simb1>, el de <expr2> a <simb2>, y así sucesivamente. Si las variables no estánpreviamente definidas, las crea. La función devuelve e! valor atribuido a la última variable.Los nombres de símbolos no pueden ser un literal (no valdría, por ejemplo, "QUOTE x").

SETQ es la función básica de atribución de valores en AutoLISP.

Por ejemplo:

```
Command: ( SETQ x 5 y "Hola" z ' (a b) w 25.3 )
25.3
```

```
Command:   !x      5
           !y      “Hola”
           !z      ( a b)
           !w      25 . 3
```

Ahora tenemos un ejemplo con una cadena:

```
Command: (SETQ ms1 “hola como estas hoy”)
Command: (PROMPT msg1)
```

Importante: Hay que tener la precaución de no utilizar con SETQ símbolos de variables que coinciden con nombres de comandos de AutoLISP, pues esto descartaría la definición de esos comandos. Por la misma razón, tampoco se deben utilizar nombres de órdenes de AutoCAD.

2.9 Coordenadas X Y Z

En la interfase de AutoCAD, se trabaja básicamente por coordenadas, por lo

que en la pantall gráfica, se representa imaginariamente por un eje X Y, (Z para trabajar en 3 dimensiones), por lo que la desiganación se da por X Y y Z. En consecuencia un punto en su desiganación consta siempre de estos tres valores asi sea 0 el valor para Z.

2.10 (GETPOINT [<pto>] [<"mensaje">]) Introducir un punto

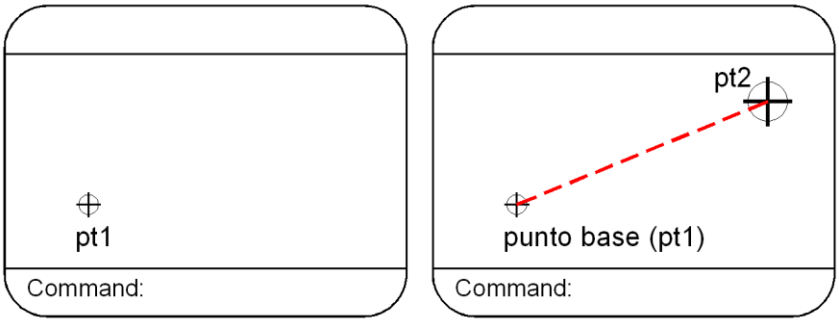
Este comando aguarda a que el usuario introduzca un punto, bien directamente por teclado o señalando en pantalla, y devuelve las coordenadas de ese punto en el SCP actual. Estas coordenadas se devuelven como una lista con los tres valores reales de X Y Z.

Se puede indicar opcionalmente un mensaje para que se visualice en la línea de órdenes, y también un **punto de base <pto>**. En este caso, AutoCAD visualizará una línea "**elástica**" entre el punto de base y la posición del cursor de forma dinámica, NO dibuja la línea como tal, es solo una visualización para referenciar al punto base escojido anteriormente.

Como el punto de base debe ser una lista de dos o tres números reales (sus coordenadas), hay que indicar esa lista sin evaluar, precedida por tanto por QUOTE.

Por ejemplo, si se introduce:

Command: (GETPOINT)



Comando GETPOINT con señalamiento directo y con opción Punto de Base.

aparentemente no sucede nada. Pero la función está esperando a que el usuario designe un punto En ese momento devolverá sus coordenadas. Por ejemplo:

(15Ø.Ø 85.Ø Ø.Ø)

Si no se almacena en una variable, el valor del punto designado no permanece, sino que simplemente se visualiza. Por eso, para almacenar en una variable "**pt1**" ese punto se utilizaría SETQ, como se ha visto en el apartado anterior.

Command: ***(SETQ pt1 (GETPOINT " Introducir un punto: "))***
Introducir un punto:
(se señala un punto en pantalla o se digita las coordenadas)
(15Ø.Ø 85.Ø Ø.Ø)
Command: ***!pt1*** (15Ø.Ø 85.Ø Ø.Ø)

Ese punto queda ya almacenado en una variable global para su posterior utilización,

Command: ***(SETQ pt2 (GETPOINT ' (3Ø 5Ø) "Segundo punto: "))***
Segundo punto: (se señala otro punto en pantalla pt2)
(125.Ø 7Ø.Ø Ø.Ø)

En este ejemplo se ha especificado un punto de base de coordenadas (3Ø 5Ø Ø); la coordenada Z = Ø se toma por defecto. Aparece una línea "**elástica**" entre ese punto de base y la posición del cursor hasta que se "pincha" el segundo punto pt2 (la línea como tal no se dibuja, la línea elastica sirve de guia en base al punto base pt1, hasta señalar o ingresar el segundo punto).

2.11 (COMMAND <argumentos>...) Acceso a COMANDOS de AutoCAD

Este comando sirve para llamar a las órdenes de AutoCAD desde AutoLISP y siempre devuelve "nil", Sus argumentos son nombres de órdenes y opciones de AutoCAD.

Cada argumento es evaluado y enviado a AutoCAD como respuesta a sus mensajes o preguntas.

Los nombres de órdenes y de opciones se envían como cadenas de texto y, por tanto, deben ir entre comillas. Los puntos 2D o 3D se envían como listas de dos o tres números reales, por lo que si se especifican hay que hacerlo precedidos de QUOTE. Cada vez que se necesite un RETURN hay que introducir la cadena vacía " ".

La llamada de COMMAND sin ningún argumento equivale a teclear "CTRL-C", lo que anula la mayoría de órdenes de AutoCAD.

Command: ***(COMMAND "line" '(5Ø 5Ø) pt2 "")***

En el ejemplo se dibujaría una línea entre el punto 5Ø,5Ø y el almacenado en la variable pt2 (que deberá contener, por tanto, una lista de dos o tres números reales). La cadena vacía final introduce un RETURN y termina en AutoCAD la orden LINEA.

Los comandos del tipo "GET...", como GETPOINT del apartado anterior, no se pueden utilizar directamente como argumentos de COMMAND. Por ejemplo, si se pretende dibujar una línea desde el punto 5Ø,5Ø hasta un punto señalado por el usuario:

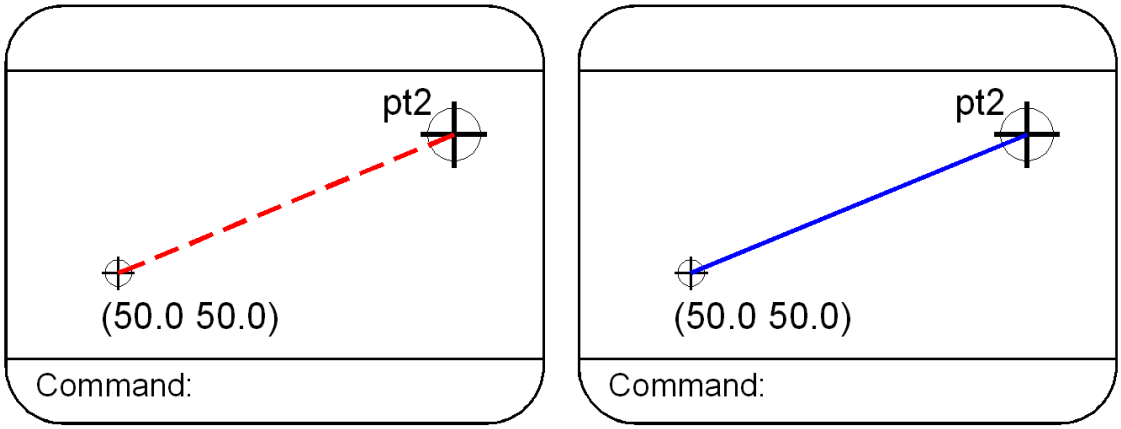
Command: ***(COMMAND "line" '(5Ø 5Ø) (GETPOINT ' (5Ø 5Ø) " Segundo punto; ") "")***

LINE Specify first point: Segundo punto: error: AutoCAD rejected function

Se produce un error y el abandono de la función en curso. Para especificar un punto introducido mediante: GETPOINT (o más datos aceptados por funciones "GET,...") hay que ejecutar previamente esa función y almacenar el valor en una variable.

Command: ***(SETQ pt2 (GETPOINT '(5Ø 5Ø) "Segundo punto: "))***
Segundo punto: (se señala) (12Ø.Ø 7Ø.Ø Ø.Ø)
Command: ***(COMMAND "line" ' (5Ø 5Ø) pt2 " ")***

Ahora la función se ejecuta correctamente y la línea queda dibujada entre el punto 5Ø,5Ø y el punto almacenado en "pt2"



Las órdenes de AutoCAD que leen directamente información del teclado, existen comandos que DTEXT y SKETCH, no se pueden utilizar con la función COMMAND. También conviene evitar la llamada a órdenes que abandonan el control del Editor de Dibujo, como PLOT, PRPLOT o SCRIPT.

Si se llama a una orden de AutoCAD y se especifica corno argumento de COMMAND el símbolo predefinido **pause**, la orden se interrumpe hasta que el usuario introduzca los datos que en ese momento se requieran, pudiendo hacerlo de forma dinámica mediante "arrastre". El mecanismo es similar al uso de las contrabarras en las opciones de menú. En el momento en que se introducen esos datos (puede ser un valor, una opción, un punto, etc.) continúa la ejecución de COMMAND desde AutoLISP.

Si el usuario introduce una orden transparente durante la interrupción de "**pause**" (como 'ZOOM o 'PAN), la función COMMAND permanecerá temporalmente suspendida, Esto permite al usuario efectuar todos los zoom o pan que desee antes de reanudar la función COMMAND introduciendo los datos que sean precisos.

Command: ***(COMMAND "circle" "50,50" "10" "line" "50,50" pause "")***

Este ejemplo dibuja un círculo con centro en 5Ø, 5Ø y radio 1Ø. Después comienza una línea en 5Ø, 5Ø y se interrumpe hasta que el usuario especifica el punto final de la línea, pudiendo hacer todos los zoom y encuadre que desee. En el momento en que se teclea o señala ese punto, la función COMMAND introduce un RETURN y la orden LINE termina.

Obsérvese que los puntos se pueden también especificar como una cadena de texto tal como se introducirían desde el teclado, con las coordenadas separadas por comas (en el ejemplo: **"5Ø, 5Ø"**).

2.11.1 Ejercicios

1 Dibuje una linea, dando valores fijos de los dos puntos
2 Dibuje una linea, pidiendo los puntos mediante GETPOINT
3 Dibuje una linea, espernado por valores que ingrese el usuario
4 Dibuje un traingulo cualquiera
5 Dibuje un cuadrilátero cualquiera

2.12 (QUOTE <expr>) Literal de expresión

Ya se ha dicho que este comando, que se puede abreviar con un apóstrofo, evita que se evalúen los símbolos. Se puede emplear con cualquier expresión en AutoLISP y siempre devuelve el literal de esa expresión, es decir, sin evaluar.

Command: ***(QUOTE (SETQ x 25))***
(SETQ X 25) Command: ***(QUOTE (DEFUN diblin ( ) "nada"))***
(DEFUN DIBLIN nil "nada")

En ambos casos, QUOTE devuelve el literal de la expresión sin evaluar.

AutoLISP siempre convierte todos los nombres de símbolos, funciones de usuario o variables en mayúsculas. Por eso la variable “x” o la función "diblin" se devuelven en mayúsculas. El "nil" que aparece después de la función "diblin" significa que no tiene argumentos.

2.13 2.13. (EVAL <expr» Evaluar expresión

Este comando evalúa la expresión indicada y devuelve el resultado de esa evaluación.

Command: ***(EVAL (SETQ x 15))***
15

Es equivalente a hacer directamente (SETQ x 15). Como se ha dicho, AutoLISP evalúa directamente los paréntesis en cuanto se encuentran cerrados. Sin embargo, EVAL tiene aplicaciones específicas, como el ejemplo que se estudiará al hablar del comando READ en el Capítulo 8.

2.14 Valores predefinidos de AutoLISP

Como ya se ha dicho, existen unas subrutinas de AutoLISP que no son propiamente comandos, sino valores predefinidos. Estas son las siguientes:

PI: Es el valor del número real "pi" 3.1415926...
T: Es el símbolo de True, Cierto (1 lógico),

Por último, el valor de Nada, Falso (Ø lógico), se representa en AutoLISP por **nil**. Este valor aparece siempre en minúsculas y no se puede acceder a él. Por ejemplo,

Command: ***(SETQ nil 2Ø)***
error: bad argument type

no es un símbolo propiamente, sino un valor que representa Vacio o Nada.

2.15 (SET <simbXexpr>) Atribuir valor a un símbolo no evaluado.

Este comando atribuye el valor de la expresión indicada <**expr**> al símbolo <**simb**>. Este símbolo se considera sin evaluar. La diferencia con SETQ es que aquí se atribuye o asocia el valor de la expresión indicada, al literal del símbolo, haciendo ambos equivalentes. Con SETQ se almacenaban valores en símbolos de variables no literales.

Command: ***(SET ' a 25)***
25

AutoLISP rechaza si se hace: (SET a 25). El símbolo debe ser literal.

Únicamente lo admite si previamente se atribuye a ese símbolo un literal de otro.

Command: ***(SET ' x 'a)***
A
Command: ***(SET x 25)***
25
Orden: ***! x***
A
Orden: ***!a***
25

En este caso se ha atribuido al símbolo x el valor literal del símbolo a. Si

extraemos el valor asociado a x, vemos que es A. Al hacer (SET x 25) se atribuye indirectamente el valor 25 al literal asociado a x, que es A. Por eso el valor asociado al símbolo A es ahora 25.

2.15.1 Ejemplo 1; Programa para dibujar una línea simple.

Por ser el primer ejemplo, se trata de un programa muy simple que, manejando los comandos hasta ahora vistos, dibuja una línea entre los dos puntos designados por el usuario.

Para confeccionar el archivo en AutoLISP que contiene este programa, accedemos al Block de Notas. El archivo se llamará DBLIN.LSP y su contenido es el siguiente:

```
(DEFUN diblinea ( / ptl pt2 )
        (GRAPHSCR)(PROMPT "Este sencillo programa dibuja una
      línea" )
        (TERPRI )
        (SETQ pt1 ( GETPOINT "Primer punto: ") )
        (TERPRI )
        (SETQ pt2 ( GETPOINT ptl "Segundo punto: " ) )
        ( TERPRI )
        (PROMPT "MUCHAS GRACIAS" )
        ( TERPRI )
        (COMMAND "línea" pt1 pt2 "" )
)
```

Ya está hecho el archivo. Para cargarlo se utiliza la función "LOAD".

```
Command: (LOAD "dblin")
DIBLINEA
```

Para ejecutar el programa basta llamar a la función *'diblinea" recién creada.

```
Command: (diblinea)
Este sencillo programa dibuja una línea
Primer punto:Segundo punto:
MUCHAS GRACIAS
line of point:
to point:
to point:
Command: nil
```

El segundo punto visualiza una línea "elástica" unida dinámicamente al primero, al especificar este último en el comando GETPOINT.

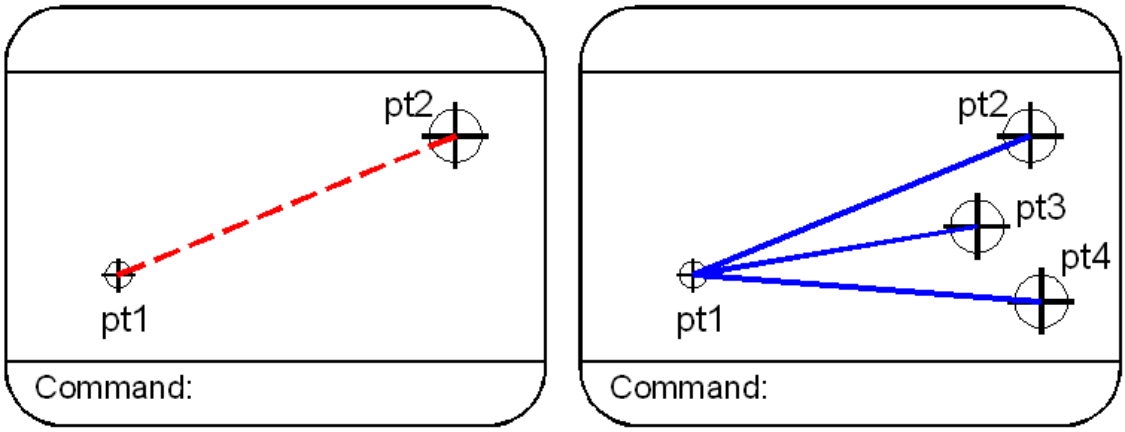
Las variables "pt1" "pt2" son locales y, por tanto, su valor después de ejecutar el programa será “nil”.

2.15.2 Ejemplo 2: Programa para dibujar tres líneas concluyentes

Vamos a completar el ejemplo anterior de forma que, partiendo desde el primer punto "pt1", el nuevo programa dibuje ahora tres líneas hasta los puntos "pt2" "pt3" y "pt4".

```
(DEFUN diblinea3 ( / ptl pt2 pt3 pt4)
        (GRAPHSCR)
        (PROMPT “Este sencillo programa dibuja 3 líneas")
        (TERPRI)
        (SETQ pt1 ( GETPOINT "Primer punto: " ) )
        (TERPRI)
        (SETQ pt2 ( GETPOINT pt1 "Segundo punto: ) )
        (TERPRI)
        (COMMAND "línea" pt1 pt2 " " )
        (SETQ pt3 ( GETPOINT pt1 "Nuevo segundo punto:  “) )
        (TERPRI)
        (COMMAND "línea" pt1 pt3 " " )
        (SETQ pt4 ( GETPOINT pt1 "Nuevo segundo punto; " ) )
        (TERPRI)
        (COMMAND "linea" pt1 pt4 " " )
)
```

La nueva función definida se llama "diblinea3". El programa dibuja en primer lugar una línea desde pt1 hasta pt2, lo mismo que el programa anterior. Pero ahora pide un nuevo segundo punto, visualizando una línea clástica desde pt1. Cuando se introduce, dibuja una segunda línea. Después pide otro nuevo segundo punto, para terminar en cuanto se dibuja la tercera línea



Resultado gráfico del programa para dibujar tres líneas concluyentes.

2.15.3 Ejercicios

1 Dibujar un poligono de tres lados
2 Dibujar un poligono de cuatro lados

3 Capitulo 3: Utilización de listas

Este capitulo comenta los procedimientos básicos para la utilización de listas, cómo crearlas y cómo acceder a todo o parte de su contenido. Ya se ha visto un caso muy importante y habitual de lista: los puntos en dos o tres dimensiones, que son listas de dos o tres elementos reales. Los comandos de AutoLISP que se explican a continuación permiten acceder a los componentes de esas listas, es decir, extraer los componentes X, Y o Z de los puntos según interese.

3.1 (CAR <lista>) Primer elemento de lista

Este comando devuelve el primer elemento de la lista especificada. Si <lista> es vacia "()", devuelve "nil".

Una de sus aplicaciones más importantes es extraer el primer valor de una lista que representa un punto, es decir, la coordenada X.

Por ejemplo, si un punto en 3D "pt1" es "(1Ø 25 5Ø)":

```
Command: ( SETQ x1 (CAR pt1) )
1Ø.Ø
```

El comando SETQ almacena en la variable "x1" el valor de la coordenada X de "pt1".

Si se emplea CAR directamente con listas, hay que tener en cuenta que deben estar sin evaluar (precedidas por QUOTE).

```
Command: (CAR ( 5 15 65 ) )
5
Command: (CAR ' ( ( c x ) 1 2 ) )
( c x)
Command: (CAR ' ( ) )
nil
```

Otra importante función de este comando es el acceso a la Base de Datos de AutoCAD, donde todas las entidades del dibujo se encuentran almacenadas como sucesión de listas. Este aspecto se estudiará con detalle en el Capítulo 12.

3.2 (CDR <lista>) Segundo elemento y resto de lista

Este comando devuelve una lista con los elementos de la lista que se indique, desde el segundo hasta el final; es decir, lodos los elementos menos el primero. Si <lista> es vacía, devuelve "nil".
Si <lista> es un tipo especial de lista llamado "par punteado" con sólo dos elementos (se estudiara más adelante), CDR devuelve el segundo elemento sin incluirlo en una lista. Este tipo de listas son fundamentales en la Base de

Datos de AutoCAD, como se verá en su momento, y de ahí la utilidad de los comandos CAR, CDR y sus combinaciones (véase el siguiente apartado) para acceder a las entidades del dibujo y modificarlas.

Si <lista> es un punto en dos dimensiones, CDR devuelve el segundo elemento, pero, ¡ojo!, incluido en una lista. Para obtener el valor numérico de la coordenada Y es preciso aplicar CAR a esa lista de un elemento.

Command: ***(SETQ pt1 ' (6 9))***
(6 9)
Command: ***(SETQ ly1 (CDR pt1))***
(9)
Command: ***(SETQ y1 (CAR ly1))***
9

La variable "pt1" almacena en forma de lista las coordenadas X Y del punto. En "ly1" sé almacena el resultado del comando CDR, que es una lista con el único elemento "9". Para obtener el valor numérico de la coordenada Y, ha habido que aplicar CAR a "ly1" y almacenar el resultado en "y1".

Estos dos pasos sucesivos se pueden juntar en uno solo, como se verá a continuación.

3.3 (CADR <lista>) Segundo elemento de lista

Este comando equivale a (CAR (CDR <lista>)), por lo que es muy útil para obtener directamente la coordenada Y de una lista que represente un punto.

Command: ***(SETQ pt1 ' (25 36 40))***
(25.Ø 36.Ø 4Ø.Ø)
Command: ***(SETQ y1 (CADR pt1))***
36.Ø

Los comandos CAR y CDR se pueden combinar de manera similar a lo visto, hasta el cuarto nivel de anidación. Por ejemplo, si se tiene

Command: ***(SETQ lis ' ((a b) (x y) 5)) f))***

las siguientes combinaciones equivaldrían a:

(CAAR lis)	a	(CAR (CAR lis))	devuelve a
(CDAR lis)	a	(CDR (CAR lis))	devuelve (b)
(CADDR lis)	a	(CAR (CDR (CDR lis)))	devuelve 5
(CADAR lis)	a	(CAR (CDR (CAR lis)))	devuelve b
(CADDAR lis)	a	(CAR (CDR (CDR (CAR lis))))	devuelve nil

Y así todas las combinaciones posibles. Según lo visto, el comando para obtener la coordenada Z de un punto en 3d (tercer elemento de la lista) es CADDR.

Command: ***(SETQ pt1 ' (3Ø 6Ø 9Ø))***
(3Ø 6Ø 9Ø)
Command: **(CADDR pt1)**
9Ø.Ø

Si el elemento buscado de la lista no existe, la función devuelve nil. Por ejemplo, con el valor de "pt1" del ejemplo anterior:

Command: ***(CDDDR pt1)***
nil

Todas estas combinaciones son extremadamente útiles, tanto para manejar listas en general como para gestionar directamente la Base de Dalos del Dibujo de AutoCAD.

3.3.1 Ejercicios

De la lista (5 4 3 (3 6 7 (23 1 50) 34 66) 8), extraer

1 el elemento 5
2 el elemento 4
3 el elemento 3
4 la lista (23 1 50)
5 el elemento 50

3.4 (LIST <expr>) Creación de lista

Este comando reúne todas las expresiones indicadas en <expr> y forma con ellas una lista, que devuelve como resultado de la evaluación.

Se utiliza sobre todo para definir variables de un punto, una vez obtenidas por separado las coordenadas. Si, por ejemplo, como resultado de un programa se han almacenado en “x1" "y1" y "z1" las coordenadas de un punto y se quiere definir una variable "pt1" que contenga ese punto:

Command: ***(SETQ pt1 (x1 y1 z1)***
Error: bad function
Command: ***(SETQ pt1 ' (x’ y’ z1))***
(x1 y1 z1)

en el primer caso el paréntesis que contiene las tres coordenadas es evaluado por AutoLISPy, como la función "x1" no esta definida, produce un error. En el segundo caso la variable“ptl" almacena el literal de la lista, pero no con los valores de las coordenadas, que son losque interesan. Para conseguirlo hay que utilizar el comando LIST.

Command: ***(LIST x1 y1 z1)***

(3Ø.Ø 6Ø.Ø 9Ø.Ø)
Command: ***(SETQ pt1 (LIST x1 y1 z1))***
(3Ø.Ø 6Ø.Ø 9Ø.Ø)

Ahora sí, el valor del punto está almacenado en "pt1". Obsérvese la diferencia delresultado con valores de variable y con literales:

(LIST x1 y1 z1) devuelve (3Ø.Ø 6Ø.Ø 9Ø.Ø)
(LIST ' x1 ' y1 ' z1) devuelve (x1 y1 z1)

3.5 (GETCORNER <pto> [<"mensaje">]) Introducir otra esquina

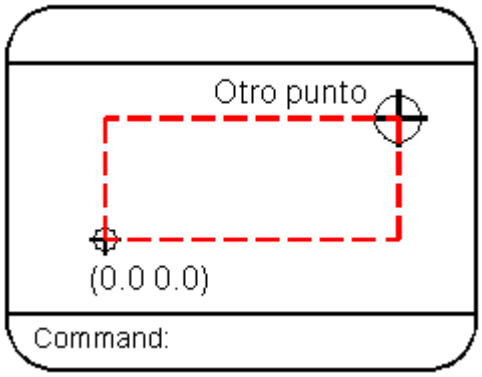
Este comando toma como base el punto especificado <pto> y visualiza de forma dinámica un rectángulo entre este punto de base y la posición del cursor, devolviendo el punto señalado por el usuario.

El resultado es el mismo de GETPOINT (aceptar un punto); lo único que cambia es la forma de visualizar dinámicamente el arrastre: en vez de una línea, aquí es un rectángulo elástico.

El punto de base se expresa respecto al SCT actual. Si se indica un punto 3D, no se tiene en cuenta la coordenada Z: siempre se toma como tal el valor actual de la elevación,

Command: ***(GETCORNER ' (Ø Ø) "Otro punto: ")***
Otro punto: (35.Ø 74.Ø 8Ø.Ø)

En el ejemplo se visualiza el mensaje "Otro punto:" y aparece un rectángulo elástico entre "Ø, Ø" y el cursor, hasta que se "pincha" y la función GETCORNER devuelve ese punto.



Visualización en pantalla de GETCORNER.

3.6 Ejemplo 1: Dibujar un rectángulo

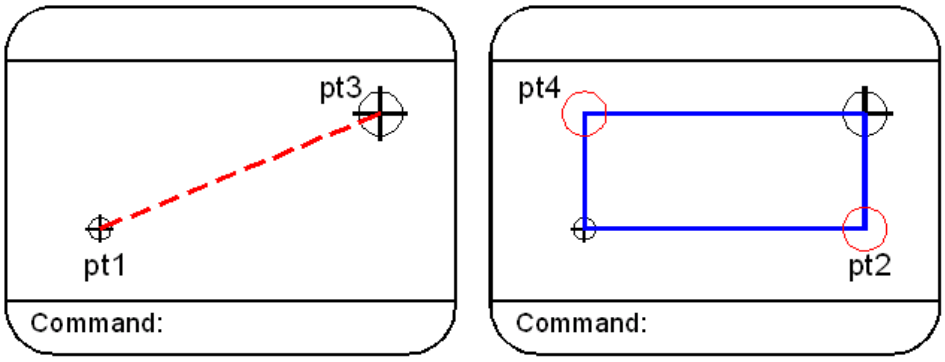
Una aplicación muy sencilla de la utilización de puntos como listas es un programa para dibujar un rectángulo, El usuario introduce las dos esquinas de la diagonal y el programa se encarga de calcular las otras dos para dibujar las cuatro líneas del rectángulo.

En el texto correspondiente a este programa se observa la práctica de ir sangrando los paréntesis, de forma que los que se van abriendo tengan el de cierre en la misma columna en vertical.

El contenido del programa que define una función llamada "sangrado" con las cuatro esquinas como variables locales es el siguiente:

```
(DEFUN rectang ( / pt1 pt2 pt3 pt4)
        (GRAPHSCR)(SETQ ptl ( GETPOINT "Primera esquina: ") )
        (TERPRI)
        (SETQ pt3 (GETPOINT ptl "Segunda esquina: ") )
        (TERPRI)
        (SETQ pt2 (LIST (CAR pt3) (CADR pt1 ) ) )
        (SETQ pt4 (LIST (CAR ptl) (CADR pt3 ) ) )
        (COMMAND "linea" ptl pt2 pt3 pt4 "c" ) )
)
```

El programa asigna a "pt1" y "pt3" las dos esquinas introducidas por el usuario. Para calcular “pt2”' hay que tomar la coordenada X de "pt3" con (CAR pt3) y la coordenada Y de "pt1" con (CADR pt1), reuniendo ambas en una lista con LIST. De manera análoga se almacena en "pt4" el punto obtenido con la coordenada X de "pt1" y la coordenada Y de "pt3". Por último se dibuja la línea entre los cuatro puntos cerrándola al final.

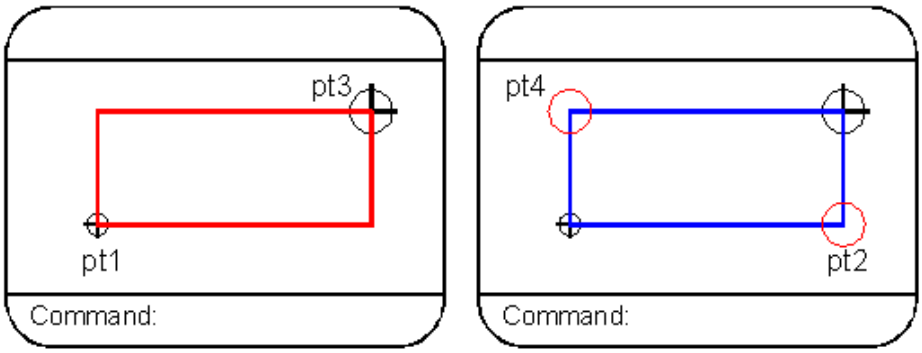


Visualización en pantalla del trazado del rectángulo con GETPOINT.

Para visualizar el rectángulo de forma dinámica se puede mejorar el programa haciendo uso del comando GETCORNER para introducir la segunda esquina

```
(DEFUN rectang ( / pt1 pt2 pt3 pt4 ) (GRAPHSCR)
        (SETQ       pt1     (GETPOINT "Primera esquina: " ) ) (TERPRI)
        (SETQ       pt3     (GETCORNER pt1 "Segunda esquina: " ) ) (TERPRI)
        (SETQ       pt2     (LIST (CAR pt3 ) (CADR pt1 ) ) )
        (SETQ       pt4     (LIST (CAR pt1) (CADR pt3 ) ) )
        (COMMAND "line" pt1 pt2 pt3 pt4 "c" )
```

)



Trazado del rectángulo con GETCORNER

3.7 Ejemplo 2: Dibujar un rectángulo con arcos de empalme

Basado en el ejemplo anterior, este programa dibuja el rectángulo como una polilínea e individuales y después solicita un radio de empalme para aplicarlo a las cuatro esquinas del rectángulo.

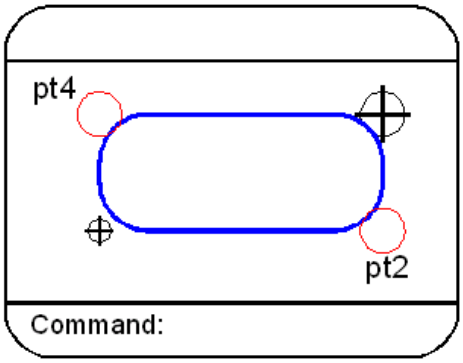
```
(DEFUN rectemp (/ pt1 pt2 pt3 pt4 remp) (GRAPHSCR)
        (SETQ pt1 (GETPOINT "Primera esquina: ") ) (TERPRI)
        (SETQ pt3 (GETCORNER pt1 "Segunda esquina: ") ) (TERPRI)
        (SETQ pt2 (LIST (CAR pt3) (CADR pt1 ) ) )
        (SETQ pt4 (LIST (CAR pt1) (CADR pt3 ) ) )
        (COMMAND "pline" pt1 pt2 pt3 pt4 "c" )
        (SETQ remp (GETREAL "Radio de empalme: ") ) (TERPRI)
        (COMMAND "fillet" " r " remp "")
        ( COMMAND "fillet" "p" "last" )
)
```

La función definida se lama ahora "**rectemp**" por distinguirla de la anterior. Su contenido para dibujar el rectángulo es el mismo, salvo que la orden llamada por COMMAND es "**fillet**".

Después el programa solicita un valor de radio de empalme y lo almacena en la variable "**remp**" definida como una nueva variable local. Para ello utiliza el comando GETREAL explicado en el siguiente capitulo y cuya función no es otra que aceptar el valor introducido como un número real.

A continuación se utiliza la orden "**fillet**" y con su opción "**r**" (Radio) se especifica el valor almacenado en "**remp**". Hay un aspecto a destacar en este punto. La orden "**fillet**" hay que volver a repetirla, una vez señalado el radio, para designar los objetos a empalmar. Desde AutoCAD esto se hace habitualmente con RETURN, que repite la última orden introducida. Pero al ejecutar el programa en AuloLISP, la última orden reconocida por AutoCAD es la que se utilizó antes de cargarlo y no las que han sido llamadas dentro del programa. Por eso es necesario volver a utilizar la función COMMAND para

llamar de nuevo a la orden "**fillet**".
Ahora se escoge la opción "**p**" (Polyline y ya sólo resta designar el rectángulo para que se generen los empalmes con "last" (Ultimo).



Trazado del rectángulo con arcos de empalme.

4 Capitulo 4: Estructura de un programa

Este capítulo comienza a examinar cómo se obtienen con AutoLISP las estructuras básicas de programación: coincidenciales o alternativas y repetitivas. Las primeras cuando la secuencia de instrucciones del programa depende de que se cumplan o no una o varias condiciones establecidas. Las repetitivas cuando es necesario repetir una secuencia un número determinado de veces o hasta que se cumpla una condición, que sirve de control del ciclo o "bucle".

Previamente se explican otros comandos para introducción de datos (hasta ahora se han visto GETPOINT y GETCORNER) y también comandos de comparación y operadores lógicos, necesarios para establecer condiciones en las estructuras de los programas.

4.1 (GETINT [<"mensaje">]) Introducir un número entero

Este comando o función inherente de AutoLISP acepta un número entero introducido por el usuario y devuelve ese número. El valor debe encontrarse entre -32768 y +32767. Si el usuario introduce un número no entero o un dato no numérico, se produce un mensaje de error y AutoLISP solicita que pruebe otra vez.

Command: ***(SETQ nent (GETINT "Introducir numero: "))***
Introducir número: ***25*** (Return)
25
Command: ***! nent***
25

En el ejemplo se introduce el número entero 25, que queda almacenado en la variable global "**nent**".

4.2 (GETREAL [<"mensaje">]) Introducir un número real

Totalmente similar al anterior, salvo que acepta un número real. Este número puede tener todos los decimales que el usuario desee introducir (separados por el "punto" decimal). Si se introduce un número entero, se considera como real. Si no se introduce un número, se produce un mensaje de error y AutoLISP solicita al usuario que pruebe otra vez.

Command: ***(SETQ nreal (GETREAL "Numero real: "))***
Numero real: ***25*** (Return)
25.Ø
Command: ***! nreal***
25.0

El valor introducido debe ser numérico. Si se trata de un ángulo, por ejemplo, no seria aceptado en la notación de grados-minutos-segundos. AutoLISP rechazaría 32d 15' 3Ø".

4.3 (GETDIST [<pt1>] [<"mensaje">]) Introducir una distancia

Este comando acepta el valor de una distancia introducida por el usuario. Este valor se puede introducir directamente desde el teclado o mediante dos puntos del dibujo.

Si se introduce directamente, el Formato debe ser el actual establecido por la orden "Unidades". Pero, independientemente de este formato, GETDIST devuelve siempre un número real.

Si se especifica un punto de base <ptl>, AutoLISP visualiza una línea elástica entre ese punto y la posición del cursor, aguardando a que el usuario introduzca un segundo punto. Esto se puede hacer con cualquiera de los métodos de AutoCAD: coordenadas absolutas, relativas rectangulares, relativas polares, señalando directamente, etc.

Command: ***(SETQ dis (GETDIST ' (1Ø 1Ø) "Segundo punto: "))***
Segundo punto: 1ØØ, 1ØØ
Command: ***!dis***
127.279

Para evaluar una distancia en general, dejando al usuario que señale los dos puntos, sería necesario almacenar el valor del primer punto en una variable.

Command: ***(SETQ ptl (GETPOINT "Primer punto:"))***
(SETQ dis (GETDIST pt1 " Segundo punto: "))
Primer punto: ***5Ø, 5Ø*** Segundo punto: ***1ØØ, 1ØØ***
7Ø. 71Ø7

4.4 (GETSTR1NG [<cr>] l<"mensaje">]) Introducior una cadena de texto

Este comando acepta una cadena de caracteres introducida por el usuario desde el teclado y devuelve esa cadena. Si la cadena contiene más de 132 caracteres, sólo devuelve los 132 primeros caracteres.

Si se especifica <cr>, se toma como señal o condición de que el texto puede contener espacios en blanco, siempre que <cr> no sea "nil".

Si <cr> fuera "nil" o no existiera, al introducir la cadena de texto desde AutoCAD el primer espacio en blanco que se pretenda incluir es tomado como "Return" y termina el texto. Si <cr> es distinto de "nil", la cadena de texto puede contener espacios en blanco y entonces debe terminarse forzosamente con "Return".

Command: ***(SETQ cad (GETSTRING "\nIntroducir texto: "))***
Introducir texto: Textosinespacios (Espacio o Return)
Textosinespacios
Command: ***! cad***
Textosinespacios

Command: ***(SETQ cad (GETSTRING T "\nIntroducir texto: "))***
Introducir texto: Texto con espacios en blanco (Relurn)
Texto con espacios en blanco
Command: ***! cad***
Texto con espacios en blanco

Se observa que en el segundo ejemplo se ha utilizado el valor "**T**" (cierto) como especificación <cr> para asegurar que la señal o "bandera" no es "nil" y admita los espacios en blanco.

Si se introduce solamente "Return", el comando GETSTRING acepta la cadena vacía representada por dos comillas sin nada en medio "".

Command: ***(SETQ cad (GETSTRING "\nIntroducir texto: "))***
Introducir texto: (Return)""
Orden: ***! cad***
""

4.5 (=) (/=) (<) (>) (<=) (>=) Operadores de comparción

Esta serie de comandos de AutoLISP efectúan operaciones de comparación de valores, ya sean numéricos o textuales. Los valores se pueden indicar expresamente o por las variables que los contienen.

- **Igual: (= <átomo> <átomo> ...)** Compara los valores de todos los <átomo> especificados. Si son todos iguales, el resultado de la evaluación es "T'. En caso contrario devuelve "nil".

Command: ***(SETQ x1 5) (SETQ y1 x1) (= x1 y1)***

Los valores de ambas variables son 5 y por tanto la función devuelve "T". Hay que tener muy en cuenta que este comando sólo compara valores y no listas o expresiones. Si, por ejemplo, se tienen dos variables, "ptl" y "pt2", con dos puntos en 2D que son listas de dos elementos, el resultado de (= ptl pt2) es "nil" porque esas variables no contienen valores numéricos, sino listas.

Para comparar la igualdad de ambos puntos habría que recurrir a un comando de operaciones lógicas como **EQUAL** (estudiado en el siguiente apartado).

- **No Igual: (/= <alomo> <átomo>…)** Compara los dos valores y devuelve "T" si electivamente son distintos. La función sólo está definida para dos argumentos.

Command: ***(SETQ a 7 b "Hola") (/ = a b)***
T

- **Menor: (< <átomo> <átomo> ..)** Devuelve "T" si el valor del primer <átomo> es menor que el segundo. La comparación se efectúa según el valor de los códigos ASCII. Por tanto, para valores textuales seria en orden alfabético, siendo el espacio en blanco el carácter de menor valor y las mayúsculas de menos valor que las minúsculas.

Si se indican mas de dos <átomo>, el resultado será “T” si cada uno de ellos es menor que el siguiente; es decir, se encuentran ordenados por valores crecientes de código ASCII.

Command: ***(SETQ v1 1Ø v2 3Ø v3 “A” v4 “A” v5 “f”)***

Dadas esas variables con esos valores asignados, los resultados de emplear el comando "Menor" con varias combinaciones serían los siguientes:

(< v1 v2 v3 v4 v5)	nil
(< v1 v2 v3 v5)	T
(< v3 v5)	T
(< v4 v5 v1)	nil

- **Menor o Igual: (<= <átomo> <átomo> ...)** Similar al comando anterior, funciona de la misma manera pero comparando la condición menor o igual.

Command: ***(SETQ v1 3Ø v2 1Ø v3 3Ø v4 "A" v5 "a" v6 "A")***

Algunas de las combinaciones posibles darían los siguientes resultados:

(<= v1 v3 v4 v6)	T
(<= v2 v1 v4 v5)	T
(<= v1 v3)	T
(<= v5 v4 v6)	nil

- **Mayor: (> <átomo> <átomo> ...)** Compara todos los <átomo> especificados y devuelve "T" sólo si cada valor es mayor que el siguíente; es decir, se encuentran ordenados de mayor a menor:

Command: ***(SETQ v1 2Ø v2 2Ø v3 "C" v4 "f" v5 "C")***

Estos serian algunos de los resultados de la evaluación del comando con esas variables:

(> v1 v2 v3 v5 v4)	nil
(> v1 v3 v4)	nil
(> v4 v5 v3 v2 v1)	nil
(> v4 v3 v2)	T

- **Mayor o Igual : (> = <átomo> <átomo> …)** Similar al anterior, establece la comparación "Mayor o Igual". Devuelve "T" sólo si cada <átomo> es mayor o igual que el siguiente.

Command: ***(SETQ v1 2Ø v2 2Ø v3 "C" v4 "f" v5 "C")***

Estas variables ofrecerían los siguientes resultados:

(>= v5 v4 v3 v1)	nil
(>= v4 v5 v3 v2 v1)	T
(>= v4 v5 v2 v3 v1)	nil
(>= v1 v2)	T

4.6 Operadores lógicos

Existen fundamentalmente cuatro comandos que actúan como operadores lógicos: **AND**, **OR**, **EQUAL** y **NOT**. Estos comandos devuelven únicamente los resultados "T" (cierto) o "nil" (Falso).

- **(AND <expr> ...)** : "Y" lógico Este comando realiza el "y" lógico de una serie de expresiones que son las indicadas en <expr). Evalúa todas las expresiones y devuelve "T" si ninguna de ellas es "nil". En el momento en que encuentra alguna expresión con resultado "nil", abandona la evaluación de las demás y el comando devuelve “nil”.

Su función es detectar si exite algún "nil" en las expresiones indicadas.

Command: ***(SETQ v1 1Ø v2 1Ø) (AND (<= v1 v2) (>= v1 v2))*** devuelve T
(AND (= v1 v2) (> v1 v2)) devuelve nil

Se observa que con el comando AND se puede evaluar cualquier tipo de expresión, por compleja que sea, y cuantas expresiones haga falla.

- **(OR <expr> ...)** : "O" lógico Realiza el “o" lógico de la serie de expresiones indicadas. Evalúa todas las expresiones y devuelve “nil” si todas ellas son “nil”. En el momento en que encuentra un resultado diferente de “nil”, abandona la evaluación y devuelve "T".
-

Se utiliza para detectar si existe alguna expresión diferente de "nil" en las indicadas.

Command: ***(SETQ v1 2Ø v2 2Ø)(OR (< vi v2) (> v1 v2))*** devuelve nil
(OR (<= v1v2) (/= v1 v2)) devuelve T

- **(EQUAL <expr1> <expr2> | <margen>|)** : Identidad lógica Comprueba si las dos expresiones indicadas son idénticas, es decir, si el resultado de ambas evaluaciones es el mismo. En ese caso devuelve "T", o "nil" si no es el mismo.

A diferencia de "**=**", se puede utilizar para comparar cualquier expresión. Una aplicación muy sencilla es comparar si dos puntos son iguales:

Command: ***(SETQ pt1 ' (5Ø 25 Ø) pt2 ' (5Ø 25 Ø) pt3 ' (3Ø 2Ø 1Ø))***
(EQUAL pt1 pt2) devuelve T
(EQUAL pt1 pt3) devuelve nil

El argumento <margen> optativo se utiliza cuando se comparan expresiones cuyos resultados son números reales y puede haber una pequeña diferencia de un decimal. Se suministra un valor del margen respecto al cual se consideran iguales los resultados.

Command: ***(SETQ n1 2 . 5346 n2 2.5347)***
(EQUAL n1 n2) devuelve nil
(EQUAL n1 n2 Ø.ØØØ1) devuelve T

- **(NOT <expr>)** : "NO" lógico Admite una única <expr> como argumento. Devuelve el "no" lógico de esa expresión. Es decir, si el resultado de esa explosión es diferente de "nil", devuelve "nil", y si el resultado es "nil", devuelve "T".

Command: ***(SETQ v1 2Ø v2 3Ø)(NOT (> v1 v2))*** devuelve T
(NOT (< v1 v2)) devuelve nil

Estos operadores pueden utilizarse combinados entre si. Por ejemplo:

Command: ***(AND (OR (NOT (< v1 v2)) (= v2 v3)) (> v1 v3) c)***

En este ejemplo podría sustituirse el operador NOT haciendo uso de su equivalencia:

(NOT (< v1 v2)) equivale a ***(>= v1 v2)***

4.7 (EQ <expr1> <expr2>) Identidad de expresiones

Este comando compara las dos expresiones indicadas y devuelve T (cierto) si ambas son idénticas, o nil (falso) en caso contrario. Se utiliza sobre todo para comparar listas y ver si hay igualdad estructural.

(SETQ list1 ' (x yz))
(SETQ list2 ' (x y z))
(SETQ list3 list2)
(EQ list1 list2) devuelve T
(EQ listi list3) devuelve nil

Se observa que list1 y list2 son exactamente la misma lista por definición. Pero list3 es por definición igual a list2, pero no a list1. Por eso EQ devuelve nil.

La diferencia con **EQUAL** es que este compara los resultados de evaluar las expresiones mientras que EQ compara la identidad estructural de las

expresiones sin evaluar.

4.8 (IF <cond><acción-cierto> [<acción-falso>]) Condicional

Este comando es una de las estructuras básicas de programación: la secuencia alternativa. La función 1 evalúa la expresión indicada corno condición en <cond>. Si el resultado no es "nil", entonces pasa a evaluar la expresión contenida en <acción-cierto>. En este caso devuelve el resultado de esa expresión.

Si <cond> es "nil", entonces pasa a evaluar la expresión contenida en <acción-falso> en el caso de que se haya indicado y devuelve su resultado. Si no se ha indicado <acciónfalso>, devuelve "nil".

Command:

```
( SETQ pt1 (GETPOINT "Primer punto: " ) ) (TERPRI)
( SETQ pt2 (GETPOINT "Segundo punto: " ) ) (TERPRI)
( IF (EQUAL pt1 pt2 )
        ( PROMPT "Son iguales" )
        ( PROMPT "No son iguales" )
) (TERPRI)
```

Primer punto: (se indica un punto)
Segundo punto: (se indica el mismo punto)
"Son iguales"
command:

El ejemplo anterior acepta dos puntos introducidos por el usuario y después compara si las dos listas que representan esos puntos son iguales, con el comando de operación lógica EQUAL. Si son iguales, resultado de <cond> es "T" (cierto) y entonces IF devuelve PROMPT "Son iguales". Si no son iguales el resultado de <cond> es "nil", con lo que IF devuelve PROMPT "No son iguales".

En el ejemplo se han "sangrado" los paréntesis para apreciar mejor la estructura del programa. Desde la línea de órdenes de AutoCAD habría que introducirlo todo seguido. Es práctica habitual, a la he de escribir un archivo con programas en AutoLISP, ir ".sangrando" o desplazando los paréntesis hacia la derecha conforme se van abriendo nuevos paréntesis incluidos en otros. Después los paréntesis de cierre se sitúan al mismo nivel (en la misma columna) que los correspondientes de apertura.

El comando IF requiere indicar como mínimo dos listas: la primera lista como condición y la segunda como acción a efectuar si se cumple ésta (sería la rama del "si" de la estructura alternativa). Una tercera lista optativa especifica la acción a efectuar si no se cumple la condición (la rama del "no" de la alternativa).

4.9 (PROGN <expr>...) Secuencia consecutiva

Este comando admite como argumentos todas las expresiones <expr> que se indiquen y las evalúa secuencialmente Devuelve el valor de la última expresión evaluada.

Es habitual utilizar PROGN en comandos cuyo formato sólo permite especificar una expresión ejemplo muy claro es el condicional IF visto en el apartado anterior. Este sólo permite indicar una acción si la condición se cumple. Con PROGN se pueden ejecutar vanas expresiones.

Command:

```
( IF ( = n 1 )
        ( PROGN
                ( SETQ a ( + 1 n ) )
                (PROMPT "Correcto")
                (TERPRI)
        )
        (PROMPT "Incorrecto")
 )
```

En el ejemplo, si se cumple la condición "n = 1" se ejecuta el comando PROGN, que evalúa secuencialmente los comandos SETQ, PROMPT y TERPRI Si no se cumple la condición, se ejecuta PROMPT

En una estructura alternativa, por tanto, el comando PROGN se puede utilizar para agrupar todas las instrucciones de cada una de las ramas (la del "si" y la del "no").

4.10 (REPEAT <num> <expresión> ...) Repetir N veces

Este comando es la estructura básica de la secuencia repetitiva, cuando se requiere repetir una expresión un determinado número de veces. El número de veces se especifica en <num> y debe ser un entero positivo, El comando REPEAT evalúa la expresión indicada (que puede ser tan compleja como haga falta) <num> veces y devuelve el resultado de la última evaluación.

Command:

```
( DEFUN pg15 ( / pt1 pt2)
        ( SETQ pt1 ( GETPOINT "Introducir punto: " ) )
        ( REPEAT 4
                ( SETQ pt2 ( GETPOINT pt1 "Nuevo punto" ) )
                ( COMMAND "line" pt1 pt2 " " )
                ( SETQ pt1 pt2 )
        )
)
```

El programa define una función "pg15" que dibuja una poligonal con cinco

vértices (es decir, cuatro entidades de línea). Primero se almacena en "pt1" el primer punto señalado por el usuario. A continuación se establece un ciclo repetitivo para dibujar las cuatro líneas. En cada paso de este ciclo, el usuario introduce un nuevo punto y se dibuja una línea entre "pt1" y "pt2".

Con (SETQ pt1 pt2) se consigue tomar como primer punto de la nueva línea el segundo punto de la anterior, con lo que se puede repetir el ciclo. Este proceso de "actualizar variables" es muy habitual en los ciclos repetitivos. La variable "ptl" tiene al principio del ciclo un valor, ese valor se cambia al final del mismo y así entra en la repetición del ciclo con el nuevo valor.

4.11 (WHILE <cond> <acción> ,..) Repetir según condición

Es otro comando para establecer secuencias repetitivas en el programa. En este caso el ciclo no se repite un determinado número de veces, "N", sino mientras se cumpla la condición especificada <cond>. Esta condición sirve, pues, como elemento de control de la repetitiva.

Mientras el resultado de evaluar <cond> sea diferente de "nil", el comando WHILE evalúa la expresión indicada a continuación <acción>, que puede ser tan compleja como se desee. Después vuelve a evaluar <cond>. Esto se repite hasta que al evaluar <cond> resulte "nil". Entonces deja de repetirse el ciclo y WHILE devuelve el resultado de la última expresión <acción> realizada.

Es muy habitual emplear este comando en combinación con IF para establecer una variable de control de la repetitiva y chequear en cada ciclo el valor de esa variable.

Command:

```
( DEFUN pg15 ( / n ctrl pt1 pt2 )
        ( SETQ pt1 ( GETPOINT "Introducir punto: " ) )
        ( SETQ n ( GETINT "Numero de líneas: " ) )
        ( SETQ ctrl T)
        ( WHILE ctrl
                ( SETQ pt2 ( GETPOINT pt1 "Nuevo punto: " ) )
                ( COMMAND "lne" pt1 pt2 " " )
                ( SETQ pt1 pt2 )
                ( SETQ n ( - n 1 ) )
                ( IF ( = n Ø ) ( SETQ ctrl nil ) )
        )
)
```

Se ha utilizado el mismo programa del ejemplo anterior, pero estableciendo un control para la secuencia repetitiva mediante la variable "ctrl".

Se define una variable "n" que almacena el número de líneas de la poligonal, valor introducido por casuario. Cada vez que se ejecuta un ciclo, al valor de "n" se le resta 1, con lo que se lleva la cuenta del número de ciclos que faltan por ejecutar.

La variable "n" se inicializa fuera del ciclo antes de entrar en él, aceptando el valor introducido mediante GETINT. La variable de control "ctrl" se inicializa también fuera del ciclo con el valor "T". Este procedimiento de inicializar determinadas variables antes de entrar por primera vez en el ciclo es muy habitual en las estructuras repetitivas.

El comando IF chequea el valor de "n" y cuando es Ø significa que se acaba de realizar el último ciclo. En ese momento (= n Ø) devuelve "T" (cierto), con lo que se asigna el valor nulo "nil" a la variable de control "ctrl". Esto provoca el final de la repetitiva establecida por WHILE.

Obsérvese que la condición <cond> se ha establecido con el nombre de la variable "ctrl". El ciclo se repetirá hasta que su valor sea "nil".

4.12 Ejemplo 1: Dibujar un haz de líneas

Como primera aplicación sencilla de las estructuras de programación, se va a definir una nueva orden de AutoCAD llamada "haz" que permitirá dibujar, a partir de un punto, un haz de líneas conforme el usuario vaya introduciendo nuevos puntos. El punto de base del haz también lo especifica el usuario.

Puesto que se trata de un archivo de AutoLISP en el que ya empieza a haber estructuras de programa, se van a poner de manifiesto mediante unas llaves que las destaquen.

4.12.1 Listado del programa

```
( DEFUM mlin ( / pt1 pt2 )
        ( SETQ pt1 ( GETPOINT "Punto de base: " ) ) (TERPRI)
        ( SETQ n T)
        ( WHILE n
                ( SETQ pt2 ( GETPOINT pt1 "Punto final: " ) ) (TERPRI)
                ( COMMAND "lnea" pt1 pt2 "" )
        )
)

( DEFUN c :haz ( )
        ( SETVAR "blipmode" Ø ) ( SETVAR "cmdecho" Ø )
        ( mlin )
        ( SETVAR "blipmode" 1 ) ( SETVAR "cmdecho" 1 )
)
```

4.12.2 Observaciones

Se empiezan a observar las prácticas habituales a la hora de escribir programas en AutoLISP. Al cargar el archivo que contiene los programas, el comando LOAD devuelve el nombre de la ultima función definida Como en este

caso se va a definir una nueva orden “haz", interesa que el usuario que cargue el programa sepa cual es el nombre de la nueva orden que tiene que emplear a continuación para ejecutar el programa por eso la definición de la nueva orden se encontrará siempre al final del archivo de texto. En el ejemplo es (DEFUN c:haz ( )....). Al cargar el archivo con LOAD devolverá el nombre C:HAZ.

Por otro lado, en vez de incluir todas las expresiones en AutoLISP del programa dentro de la definición de “c:haz”, se define una función “mlin” (línea múltiple), que es la que contiene las expresiones para dibujar el haz de líneas. Al definir “c:haz” bastará llamar a esa función “mlin”.

Otra práctica habitual en programación de AutoLISP es desactivar las marcas auxiliares y el eco de menú a la línea de órdenes. Esto se consigue accediendo con el comando SETVAR (que se estudiara en detalle en el capítulo correspondiente) a las variables "blipmode" y “cmdecho" y cambiando su valor a Ø. Una vez ejecutado el programa, se vuelven a activar ambas variables cambiando su valor a 1.

La definición de la nueva orden (en el ejemplo, c:haz) contiene, pues habitualmente sólo la desactivación de marcas auxiliares y eco de menú y la llamada a diferentes funciones de usuario previamente definidas en el archivo de texto (en este caso existe solamente una, que es "mlin").

También se observa claramente en el listado del archivo de texto el procedimiento de "sangrar" los paréntesis, de forma que se van desplazando hacia la derecha, y el hecho de situar los paréntesis de cierre en la misma columna (al mismo nivel) que los correspondientes de apertura.

4.12.3 Explicación del programa

Se definen dos funciones de usuario:

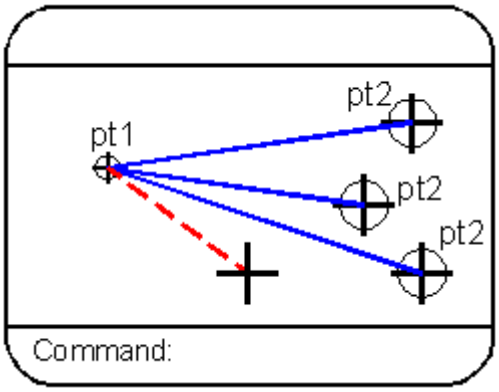
- Función **c:haz**

Desactiva las marcas auxiliares y el eco de órdenes. Llama a la función "mlin". Vuelve a activar las marcas auxiliares y el eco de órdenes.

- Función **mlin**

Solicita del usuario el punto de base del haz y lo almacena en la variable "pt1". Establece un ciclo repetitivo para dibujar cada una de las líneas del haz. Como variable de control de ese ciclo se toma "n", que se inicializa a T (cierto) antes de entrar. El ciclo se repetirá mientras "n" sea cierto, es decir, indefinidamente, de forma que el usuario pueda hacer tantas líneas como desee. Para salir del ciclo habrá que interrumpir el programa con CTRL-C.

Dentro del ciclo se solicita del usuario el punto final de cada línea del haz, el cual se almacena en "pt2". A continuación se dibuja la línea entre el punto de base "pt1" y el punto final indicado "pt2".



Trazado en pantalla del haz de líneas.

4.13 Ejemplo 2: Dibujar una estrella

Este programa dibuja una estrella regular de N puntas. El usuario especifica el número de puntas deseado, el centro de la estrella y los radios de las circunferencias exterior e interior que van a definir su tamaño.

4.13.1 Listado del programa

```
(DEFUN inic ( )
  ( SETQ cen ( GETPOINT "Centro de la estrella: " ) ) (TERPRI)
  ( SETQ rmay ( GETDIST cen "Radio mayor: " ) ) (TERPRI)
  ( SETQ rmen ( GETDIST cen "Radio menor: " ) ) (TERPRI)
  ( SETQ np ( GETINT "Número de puntas: " ) ) (TERPRI)
  ( WHILE ( <= np 1)
    ( PROMPT "Valor debe ser positivo mayor que 1" ) (TERPRI)
    ( SETQ np (GETINT "Numero de puntas: " ) ) (TERPRI)
  )
  ( SETQ dang ( / ( * 2 PI ) np ) )
)

( DEFUN dib ( )
  ( SETQ ang 0)
  ( SETQ ptl (POLAR cen 0 rmay ) )
  ( REPEAT (+ np 1)
    ( SETQ pt2 ( POLAR cen ( + ang ( / dang 2 ) ) rmen ) )
    ( SETQ pt3 ( POLAR cen ( + ang dang ) rmay ) )
    ( COMMAND "line" pt1 pt2 "" )
    ( COMMAND "line" pt2 pt3 "" )
    ( SETQ pt1 pt3)
    ( SETQ ang ( + ang dang ) )
  )
)

( DEFUN c:star ( / ang cen rmay rmen dang pt1 pt2 pt3)
  ( SETVAR "blipmode" 0) (SETVAR "cmdecho" 0)
  ( inic )
  ( dib )
```

```
    (SETVAR "cmdecho" 1)
)
```

4.13.2 Observaciones

En este programa se puede observar otra práctica de programación que es muy importante: prever todos los posibles fallos para tratar de evitarlos. Estos fallos se pueden producir fundamentalmente cada vez que el programa solicita datos del usuario. Es preciso tener en cuenta todos los posibles errores que puede cometer el usuario y añadir al programa las instrucciones necesarias para evitarlos.

Así, en la función "inic" que va solicitando del usuario los datos necesarios para definir la estrella, al introducir el número de puntas el comando GETINT sólo acepta números enteros. Pero sí el usuario indica un número entero negativo o Ø, GETINT lo acepta, con lo que el ciclo repetitivo de la función "dib", que es (REPEAT np ….), produciría un error y el programa quedaría abortado.

Por eso, tras solicitar un valor para el número de puntas, la función "inic" establece un ciclo repetitivo que chequea si el valor de "np" es menor o igual que 1. Si es así, se visualiza el mensaje "Valor debe ser positivo y mayor que 1" (pues se considera que una estrella de una punta no tiene sentido) y se solicita de nuevo el número de puntas. El programa sólo saldrá de este ciclo cuando el valor de "np" sea entero y mayor que 1, con lo que el correcto funcionamiento de la posterior llamada a la función "dib" queda garantizado.

En lo que respecta a los valores de los dos radios, se solicitan mediante GETDIST. Este comando admite valores negativos para los vectores de distancia. Una estrella normal tendría dos radios positivos; Pero el programa funciona también para uno o ambos radios negativos y se obtienen figuras atractivas de estrellas con puntas entrelazadas. Por eso no interesa poner ningún control adicional para esos valores.

4.13.3 Explicación del programa

Define tres funciones de usuario, una de ellas como nueva orden de AutoCAD.

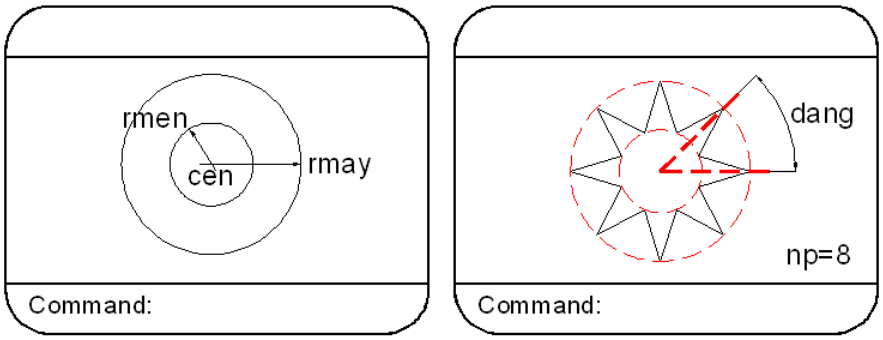
- Función **c:star**

Simplemente desactiva marcas auxiliares y eco de órdenes y llama a las dos funciones de usuario "inic"

- Función **inic**

Solicita del usuario los datos necesarios para dibujar la estrella. En primer lugar, el centro con el comando GETPOINT. Después, los radios mayor y menor, utilizando GETDIST con el centro como punto de base. De esta manera se pueden especificar ambos radios gráficamente. En cualquier caso, el usuario puede introducir directamente por teclado estos valores. Si uno de los

radios es negativo, se obtienen estrellas con las puntas entrelazadas.



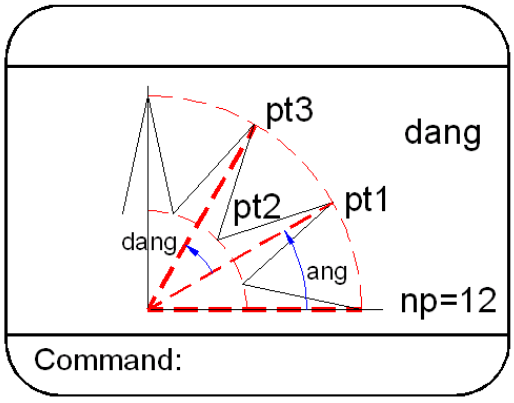
Introducción de datos para dibujar !a estrella

A continuación se pide el número de puntas con GETINT, puesto que debe ser un número entero. Pero se impone la condición de que debe ser positivo y mayor que 1. Por eso se establece un control con WHILE para chequear si el valor introducido es menor o igual que 1. Si esto ocurre, se visualiza un PROMPT de advertencia y se vuelve a solicitar un valor para el número de puntas. Solamente cuando el usuario introduzca un valor correcto, se romperá el ciclo de control.

Por último, la función calcula el ángulo abarcado por cada punta de la estrella, teniendo en cuenta que debe ir en radianes. Ese valor se almacena en la variable de incremento de ángulo "dang" (delta-ángulo).

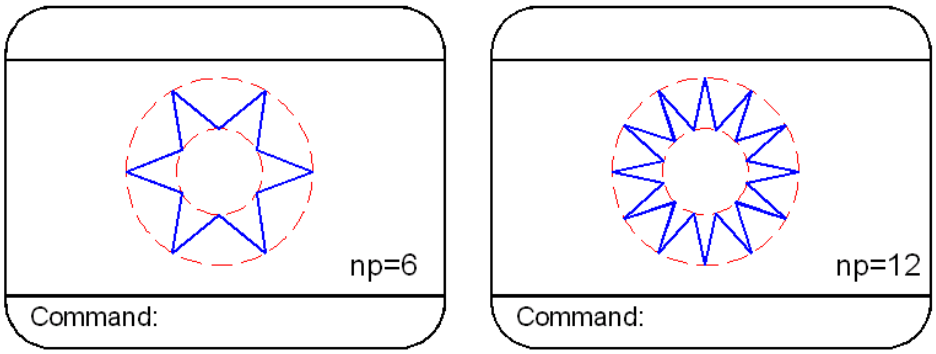
- Función **dib**

Consiste en un ciclo repetitivo que dibuja las dos líneas intermedias entre cada par de puntas. Previamente se inicializa a Ø el valor del ángulo de referencia "ang" para empezar a dibujar cada punta. El punto inicial "pt1" se obtiene la primera vez, fuera de! ciclo. Corresponde al primer cuadrante de la estrella.
El ciclo calcula mediante el comando POLAR, que se emplea aquí por conveniencia y se estudiara en el Capitulo 6, y teniendo en cuenta los valores de los radios mayor y menor, los puntos "pt2" y "pt3" que sirven para dibujar las dos líneas intermedias entre cada par de puntas (Figura 4.3). Al final del ciclo se actualiza la variable “pt1" para que el nuevo ciclo empiece con el valor del último punto del anterior. También se actualiza el ángulo de referencia "ang", sumándole cada vez el incremento de ángulo “dang".



Trazado de cada una de los puntos de la estrella.ç

Este ciclo se repite un número de veces igual al número de puntas de la estrella.



Las dos figuras superiores han sidoobtenidas con los dos radios positivos

4.14 Ejemplo 3: Líneas de texto con cualquier espaciado.

Cuando desde AutoCAD se quieren incorporar una serie de líneas de texto al dibujo, una vez especificado el estilo de texto, su altura, punto de inserción y ángulo de rotación, el espaciado entre líneas queda automáticamente determinado. Si el usuario desea otra medida del espaciado entre líneas, debe calcular cada vez el punto de inserción de cada línea o moverlas hasta situarlas con el espaciado correcto.

En cualquier caso, pierde la posibilidad de empezar la nueva línea simplemente introduciendo "Return" una vez terminada la anterior.

Este programa en AutoLISP define una nueva orden llamada "textos" que, con el estilo actual establecido, permite al usuario escoger el espaciado que desea para las sucesivas líneas de texto.

4.14.1 Listado del programa

```
( DEFUN int ( )
  ( GRAPHSCR )
  ( SETQ pb ( GETPOINT "Comienzo primera linea: " ) )
  ( SETQ alt (GETDIST pb "Altura del texto: " ) )
  ( WHILE ( <= alt 0 )
    ( PROMPT "Altura debe ser positiva: " )
    ( SETQ alt (GETDIST pb "Altura del texto: " ) )
  )
  ( SETQ esp ( GETDIST pb "Espacio entre lineas: " ) )
  ( SETQ nl ( GETINT "Número de lineas: " ) )
```

```
  ( WHILE ( <= nl 0 )
    ( PROMPT "Numero de lineas debe ser positivo: " )
    ( SETQ nl ( GETINT "Numero de lineas: " ) )
  )
)

( DEFUN dibtx ( )
  ( SETQ n 1 )
  ( SETQ pins pb )
  ( WHILE ( <= n nl )
    ( PROMPT "Introducir texto de linea: " ) (PRIN1 n )
    ( SETQ tx ( GETSTRING T ) )
    ( COMMAND "text" pins alt "0" tx )
    ( SETQ n ( + 1 n ) )
    ( SETQ pins ( LIST ( CAR pb ) ( - ( CADR pins ) esp ) ) )
  )
)

( DEFUN c:textos ( / pins pb alt ni n esp tx)
  ( SETVAR "blipmode" 0)
  ( SETVAR "cmdecho" 0)
  ( int )
  ( dibtx)
  ( SETVAR "cmdecho" 1 )
)
```

4.14.2 Observaciones

Tal como se observaba en el programa anterior, los argumentos locales se especifican en la definición de la última función "c:textos". ^slo es así porque las funciones intermedias "int" y "dibtx" necesitan que los valores de esas variables no se pierdan. Por otro lado, conviene establecer todas las variables utilizadas en el programa como argumentos locales, con el fin de que no se agote rápidamente la memoria disponible.

Sin embargo, es conveniente a la hora, de confeccionar un programa no establecer al principio ningún argumento local. Si el programa no funciona bien al primer intento, se puede de esta manera ir explorando los valores almacenados en las variables para ver la marcha del mismo. Sólo al final, cuando el programa esté depurado, se incluirán las variables como argumentos locales.

Por otro lado, cuando el programa empieza a ser complejo y contiene llamadas a varias funciones, resulta aconsejable definir en principio cada función por separado y probarlas hasta comprobar que van bien. Es más fácil detectar los errores en una función aislada que en el programa completo con todas las funciones incluidas.

4.14.3 Explicación del programa.

Se define una nueva orden "textos" y dos funciones de usuario intermedias:

- Función **c:textos**

Define una nueva orden de AutoCAD “textos", que tras desactiva las marcas auxiliares y el eco a menú llama simplemente a las funciones "int" y "dibtx".

- Función **int**

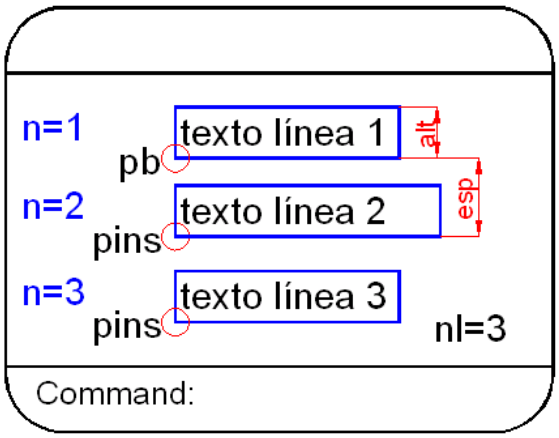
Como siempre, se define para solicitar del usuario todos los datos necesarios para dibujar las líneas de texto. En primer lugar, el punto inicial para la primera línea. Después, la altura del texto y el espaciado entre líneas. A continuación, el número de líneas a dibujar.

Se establece un ciclo de control con WHILE para asegurar que la altura introducida es positiva.

Se establece otro ciclo de control para asegurar que el usuario no introduce un número Ø o negativo de líneas,

Ya se vera al hablar de comandos, como INITGET y GETKWORD, cómo exigen otros procedimientos mas sencillos para establecer los valores permitidos con los comandos del tipo GET... Pero de momento el empleo de ciclos de control con WHILE va a servir para detectar de una manera más clara dónde hay que prever fallos en el programa.

Para el espaciado entre líneas, el comando GETDIST acepta siempre un valor positivo si el usuario "pincha" un punto en pantalla. No obstante, es posible introducir un número negativo por teclado. El programa lo permite, pues no tiene establecido ningún ciclo de control para la variable "esp". En ese caso las líneas se generarían hacia arriba (eje Y positivo).



- Función **dibtx**

Consiste fundamentalmente en un ciclo repetitivo que va dibujando cada una de las líneas de texto. La variable "n" se utiliza como contador del ciclo, para saber que número de línea se está generando en cada momento. Se inicializa antes de entrar por primera vez en el ciclo, con un valor 1. La variable "pins" va a contener el punto de inserción del texto en cada línea. Se inicializa igualándola con "pb", pues para la primera línea el texto comienza en "pb".

A continuación se establece el ciclo con la condición de que el número de línea "n" sea menor o igual que el número total de líneas a dibujar "nl". Se solicita del usuario el texto a incluir en cada línea y se almacena en la variable "tx". Para solicitar ese texto y visualizar al mismo tiempo el número de línea se hace de la siguiente manera:

(PROMPT "Introducir texto de línea ") (PRIN1 n)

El comando PRIN1, que se verá en el Capítulo 9, escribe en la pantalla el valor de la variable "n" que será en cada momento el número de línea correspondiente al actual recorrido del ciclo. Así, la tercera vez que se recorra el ciclo, se solicita del usuario el texto para la línea número tres y el mensaje que se visualiza en la pantalla es:

Introducir texto de línea 3

El texto introducido por el usuario se acepta mediante un comando GETSTRING con el parámetro T para que admita espacios en blanco.

Se conocen ya todas las características del texto y se procede a dibujarlo con (COMMAND "text"….).

La parte final del ciclo, como siempre, actualiza los valores de las variables para entrar en el siguiente recorrido del ciclo. El contador "n" se incrementa en una unidad. Se calcula el nuevo valor del punto de inserción:

(SETQ pins (LIST (CAR pb) (- (CADR pins) esp)))

Se toma la coordenada X del punto de base "pb" (es siempre la misma, los puntos iniciales de todas las líneas tienen la misma coordenada X), y la coordenada Y se calcula restando a la Y del punto de inserción de la anterior línea el valor del espaciado (- (CADR pins) esp). Se reúnen ambas coordenadas en una lista y ese es el nuevo punto de inserción buscado.

5 Capitulo 5: Operaciones numéricas

En este capítulo se presentan los comandos de AutoLISP que permiten realizar operaciones numéricas. Por razones de claridad se han dividido en dos apartados, uno para las operaciones aritméticas básicas y otro para el resto de cálculos numéricos.

Al final del capitulo se explica otro comando de estructura de programa, complementario de los estudiados en el capitulo precedente.

5.1 (+) (-) (*) (/) Operaciones aritméticas.

Estos cuatro comandos de AutoLISP efectúan las cuatro operaciones aritméticas básicas (Sumar, Restar, Multiplicar y Dividir) con los datos numéricos especificados a continuación.

- **Sumar**: **(+ <nurn> <num> …)** Devuelve la suma de todos los <num> especificados a continuación del comando. Si todos los <num> son enteros, el resultado es entero. Si uno de ellos es real, el resultado es real.

Command: ***(SETQ nl 5 n2 7) (SETQ sum (+ 3 nl n2))***
15
Command: ***! sum***
15

Los <num> pueden ser directamente números o bien variables, pero en ese caso su contenido debe ser numérico.

- **Restar**: **(- <num> <num> ...)** Devuelve el resultado de restar al primer <num> todos los demás. Es decir, se resta al primero la suma de todos los demás.

Command: ***(SETQ nl 12 n2 5) (SETQ res (- nl n2 3))***
4
Command: ***! res***
4

El resultado seria equivalente a hacer:

Command: ***(SETQ res (- nl (+ n2 3)))***

Se observa que los valores numéricos de cualquiera de estos cuatro comandos pueden ser resultados de la evaluación de otras funciones. En el ejemplo, el resultado de una evaluación de (+ n2 3) se utiliza como dato numérico pura la evaluación de (- nl <num>).

- **Multiplicar**: **(* <num> <num> ...)** Evalúa el producto de todos los <num> indicados. Como siempre, si uno de los<num> es real el

resultado es real.

Command: ***(SETQ nl 5 n2 6) (SETQ prod (* nl n2 3))***
6Ø
Command: ***! prod***
6Ø

En el ejemplo, todos los números suministrados a la función eran enteros y por eso el resultado es entero.

- **Dividir**: **(/ <num> <num> …)** Divide el primer número por todos los demás. Es decir, divide el primer número porel producto de lodos los demás.

Command: ***(SETQ nl 8Ø n2 5) (SETQ div (/ nl n2 3))***
5
Command: ***(SETQ div (/ ni n2 3.Ø))***
5.3333

Se observa que en el primer ejemplo, al ser los tres números enteros, el resultado de la división es también entero y se truncan los decimales. En el segundo ejemplo se ha indicado el tercer número como real al ponerlo de la forma "3,Ø" y entonces el resultado es un número real. El resultado de dividir más de dos números es equivalente a hacer:

Command: ***(SETQ div (/ nl (* n2 3)))***

Con estas cuatro operaciones básicas y todas sus combinaciones (más los comandos para evaluar funciones trigonométricas, logaritmos, etc., que se estudiarán en el siguiente apartado) es posible introducir en AutoLISP cualquier ecuación.

Por ejemplo, se pretende realizar un programa para el que es preciso calcular los puntos (X, Y) de una curva dada por la fórmula:

$$Y = (4X^3 + X^2 - 5) / (X^2 + 3)$$

El punto (X1, Y1) de la curva se obtendría:

```
(SETQ y1 ( / ( + ( * 4 x1 x1 x1 )
                 ( - ( * x1 x1 ) 5 )
           )
           ( + ( * x1 x1 ) 3 )
           )
)
(SETQ p1 ( LIST x1 y1 ) )
```

Los exponenciales se han obtenido como producto múltiple. Existe un comando EXPT para obtenerlos directamente, que se explicará a continuación.

5.2 Otras operaciones con datos numéricos.

- **(1 + <num>)**: Incrementa una unidad al valor indicado en <num>. Equivale a (+ 1 <num>), pero de una forma más cómoda.

Command: ***(SETQ n (1+ n))***

- **(1 - <num>)**: Resta una unidad al valor indicado en <num>. Equivale a (- <num> 1).

  Command: ***(SETQ n (1 – n))***

- **(ABS <num>)**: Devuelve el valor absoluto del número indicado en <num>.

  Command: ***(ABS - 25.78)*** devuelve 25.78

- **(FIX <num>)**: Convierte el número <num> en un número entero. Es decir, si es un número real con decimales, devuelve sólo la parte entera.

  Command: ***(FIX 32.78)*** devuelve 32

- **(REM <num1> <num2>)**: Divide <num1> por <num2> y devuelve el resto de la división.

  Command: ***(REM 2Ø 7)*** devuelve 6
  Command: ***(REM 25 5)*** devuelve Ø

- **(COS <ang>)**: Devuelve el coseno del ángulo <ang> expresado en radianes.

  Command: ***(COS Ø. Ø)*** devuelve 1.Ø
  (COS PI) devuelve -1.Ø
  (COS (/ PI 2)) devuelve Ø.Ø

- **(SIN <ang>)**: Devuelve el seno del ángulo <ang> indicado en radianes.

  Command: ***(SIN (/ PI 2))*** devuelve 1.Ø
  (SIM 1.Ø) devuelve Ø.841471

- **(ATAN** <**num1**> |<**num2**>|**)**: Devuelve el arco tangente de <num1> en radianes.

  Command: ***(ATAN 1.5)*** devuelve Ø.98

Si se indica un segundo valor <iiuin2>, ATAN devuelve el arco tangente de <num1> dividido por <num2>. Esto permite indicar directamente la tangente como cociente del seno entre el coseno.

- **(SQRT <num>)**: Devuelve la raíz cuadrada del número <num>.

- **(EXP <num>)**: Devuelve "e" elevado a la potencia indicada en <num>.

  Command: ***(SQRT 4)*** devuelve 2.ØØ
  (SQRT 2) devuelve 1.4142
  Command: ***(EXP 1)*** devuelve 2.71828

- **(EXPT <base> <pot>)**: Devuelve el número <base> elevado a la potencia <pot>. Si ambos números son enteros, el resultado es entero; en caso contrario, es real.

  Command: ***(EXPT 2 4)*** devuelve 16
  (EXPT 5.Ø 3) devuelve 125.ØØØØ

- **(LOG <nun>)**: Devuelve el logaritmo neperiano del número indicado <num>.

  Command: **(LOG 4.5)** devuelve 1.5Ø4Ø8

- **(FLOAT <num1>)**: Conviene el número indicado <num> en un número real.

  Command: ***(FLOAT 5)*** devuelve 5.Ø
  (FLOAT 5.25) devuelve 5.25

- **(GCD < num1> < num2>)**: Devuelve el máximo común denominador de los dos números indicados.

  Command: ***(GCD 45 8Ø)*** devuelve 5
  Command: ***(GCD 78 93)*** devuelve 3

- **(MAX <num> <num>…)**: Devuelve el mayor de todos los números indicados. Estos pueden ser enteros o reales.

  Command: ***(MAX 6.52 3 -7.5)*** devuelve 6.52
  Command: ***(MAX -5 -7 -9)*** devuelve -5

- **(MIN <num> <num>...)**: Devuelve el menor de todos los números indicados. Estos pueden ser enteros o reales.

  Command: ***(MIN 23.5 Ø 5.72)*** devuelve Ø.Ø
  Command: ***(MIN Ø - 23 – 89)*** devuelve - 89

5.3 (COND (<cond1 > <res1 >) (<cond2> <res2>) ...) Condicional

Este comando permite establecer una serie de condiciones especificando diferentes actuaciones según cual sea la primera condición en cumplirse.

Los argumentos del comando COND son un número cualquiera de listas, tantas como condiciones se pretendan establecer. Cada lista contiene dos especificaciones: la condición <cond> y el resultado a ejecutar <res> en el caso de que esa condición se cumpla (devuelva un valor diferente de "nil"). La condición <cond> normalmente es una lista con un comando de comparación o un operador lógico (para chequear si la condición se cumple).

La segunda especificación de cada lista es el resultado a ejecutar. Este no tiene por que ser una única lista: pueden ser varias, que se irán ejecutando todas en caso de que la condición se cumpla.

Command:
(SETQ c 1)
(COND ((= c 1) (SETQ n T) (PROMPT "Correcto") (TERPRI)
(T (PROMPT "Incorrecto") (TERPRI))
)

En el ejemplo, el comando COND tiene dos listas como argumento, es decir, establece dos condiciones: en la primera, si el valor de "c" es igual a 1, la lista (= c l) devuelve "T" y entonces la condición se cumple y se ejecutan los comandos siguientes, que son SETQ, PROMPT y TERPRI.

La segunda condición no es una lista, sino "T" (cierto); es decir, se obliga a que sea siempre cierta. El comando COND no ejecuta todas las condiciones establecidas, sino que va chequeando1as hasta encontrar una que devuelva un valor diferente de "nil" y entonces ejecuta las expresiones establecidas como resultado <res>.

Por tanto, COND evalúa las expresiones de la primera condición que encuentre que se cumpla (evaluación diferente de "nil"). Ya no sigue chequeando las siguientes condiciones, aunque haya más que también se cumplan.

Si se cumple una condición y no existe el resultado correspondiente <res> (no está especificado en la lista) entonces el comando COND devuelve el valor de esa condición. Si se especifican las expresiones a ejecutar como <res>, el comando CON D devuelve la evaluación de la última expresión.

Suele ser práctica habitual establecer la última condición como siempre cierta "T". De esta forma, si ninguna de las anteriores se cumple (todas son "nil"), se tiene la seguridad de que el programa llevará a cabo la ejecución correspondiente al resultado <res> de la última lista.

Una aplicación muy corriente de este comando es verificar la respuesta del usuario cuando ésta es del tipo “S” o “N”. Por ejemplo:

Command:
(COND((= ch"S") 1)
((= ch "S") 1)
((= ch "N") Ø)
((= ch "n") Ø)

```
        ( T ( PROMPT "Incorrecto" ) )
)
```

La variable "ch" contiene el carácter introducido por el usuario desde el teclado. Si ese carácter es "S" o "s", el comando COND devuelve l. Si ese carácter es "N" o "n", devuelve Ø. Si es cualquier otro, se visualizara el mensaje "Incorrecto". De esta forma se puede chequear que tipo de carácter ha introducido el usuario.

Las acciones a ejecutar <res> para cada condición se pueden especificar a continuación de cada condición. Se observa que el comando COND es equivalente a especificar varios comandos IF.

Command:

```
( IF ( OR ( = ch "S" ) ( = ch "s" ) ) 1 )
( IF ( OR ( = ch "N") ( = ch "n" ) ) Ø )
( IF (AND( / = ch "S" ) ( / = ch "s" ) ( / = ch "N" ) ( / = ch "n" ) )
(PROMPT "Incorrecto" ) )
```

Cuando existen varias condiciones a analizar, es mas sencillo el comando COND. Además, en el caso de especificar después de cada condición las expresiones a ejecutar, con IF habría que utilizar un PROGN, mientras que con COND se especifican sin más las expresiones.

5.4 Ejemplo 1: Dibujar curvas senoidales

Como aplicación de los comandos que realizan operaciones aritméticas, se presenta aquí un ejemplo para dibujar la curva resultante de la suma de tres senoidales. El programa solicita los coeficientes para cada onda y el resultado visualiza la interferencia resultante de la superposición de las mismas.

El programa completo emplea a presentar cierta complejidad en cuanto al número de funciones y de variables definidas. Por eso se incluye una lista de las variables que intervienen en cada función, con su significado.

5.4.1 Listado del programa

```
( DEFUN intr ( )
  ( SETQ cf1 ( GETREAL "Coeficiente primera onda: " ) ) ( TERPRI )
  ( SETQ cf2 ( GETREAL "Coeficiente segunda onda: " ) ) ( TERPRI)
  ( SETQ cf3 ( GETREAL "Coeficiente tercera onda: " ) ) ( TERPRI )
  ( SETQ prx ( GETREAL "Precisión en X: " ) ) ( TERPRI )

  ( WHILE ( <= prx 0 )
    ( PROMPT "Debe ser positivo mayor que 0" ) ( TERPRI )
    ( SETQ prx ( GETREAL "Precisión en X: " ) ) ( TERPRI )
  )
  ( SETQ xi ( GETREAL "Inicio curva en X: " ) ) ( TERPRI )
  ( SETQ xf ( GETREAL "Final curva en X: " ) ) ( TERPRI )
```

```
  ( WHILE ( <= xf xi)
    ( PROMPT "Debe ser un valor mayor que inicio" ) ( TERPRI )
    ( SETQ xf ( GETREAL "Final curva en X: " ) ) ( TERPRI )
  )
  ( SETQ n ( FIX ( / ( - xf xi) prx ) ) )
)

( DEFUN func (x)
  ( + ( SIN ( * cf1 x ) ) ( SIN ( * cf2 x ) ) ( SIN ( * cf3 ) ) )
)

( DEFUN inic ( / y1 )
  ( SETQ x1 xi )
  ( SETQ y1 ( func x1 ) )
  ( SETQ p1 ( LIST x1 y1 ) )
)

( DEFUN dib ( / x2 y2 p2 )
  ( REPEAT n
    ( SETQ x2 ( + x1 prx ) )
    ( SETQ y2 ( func x2 ) )
    ( SETQ p2 ( LIST x2 y2 ) )
    ( COMMAND "line" p1 p2 "" )
    ( SETQ p1 p2 x1 x2 )
  )
)

( DEFUN ult ( / p2 yf )
  ( SETQ yf ( func xf ) )
  ( SETQ p2 ( LIST xf yf ) )
  ( COMMAND "line" p1 p2 "" )
)

( DEFUN c:ondas ( / cf1 cf2 cf3 prx xi xf n p1 x1 )
  ;( SETVAR "blipmode" 0 )
  ;( SETVAR "cmdecho" 0)
  ( intr )
  ( inic )
  ( dib )
  ( ult )
  ;( SETVAR "blipmode" 1 )
  ;( SETVAR "cmdecho" 1 )
)
```

5.4.2 Observaciones

En lo que respecta a la definición de variables como argumentos locales, cuando el número de funciones es elevado puede resultar conveniente indicarlas para cada función, en vez de hacerlo con todas juntas en c:onda.

Sin embargo, conviene tener muy en cuenta qué variables son efectivamente locales y cuáles deben conservar su valor al salir de la función. Por otro lado, el programa completo define una nueva orden de AutoCAD que va llamando al resto de funciones de usuario definidas. Pero cada una de esas funciones es un programa que podría utilizarse independientemente, por ejemplo, para que el usuario vaya depurando el programa completo y vaya detectando los posibles fallos función por función.

Por eso puede ser una buena práctica de programación establecer una lista de variables en cada función definida y dividirlas en tres grupos:

Variables de entrada: aquellas que deben encontrarse ya definidas antes de llamar a la función y cuyos valores va a utilizar ésta.

Variables o argumentos locales: aquellas que sólo utiliza internamente la función y cuyo valor no interesa una vez que termina de ejecutarse.

Variables de salida: aquellas cuyo valor debe conservarse al terminar la función, pues van a ser utilizadas por otra función posterior. Parte de ellas coincidirán con algunas o todas las variables de entrada.

En cualquier caso, mientras se prueba el programa no se debe especificar ningún argumento local Así se pueden siempre explorar los valores de las variables, si fuera necesario, para ver donde falla el programa. Sólo al final, cuando todo funcione correctamente, se indicarán los argumentos locales con el fin de que no ocupen memoria sin necesidad.

En lo que respecta a la detección de posibles errores en la entrada de datos por parte del usuario (función "mir"), los coeficientes de las tres ondas pueden ser cualquier número real, incluidos negativos y Ø. Sin embargo, la precisión en X debe ser positiva y diferente de Ø, y de ahí que se establezca un control con WHILE.

Además, para prevenir errores en el trazado de la curva resultante, una vez introducido el valor de su inicio en X (variable "xi") se obliga a que el valor de su final (variable "xf') sea mayor para que la curva vaya de izquierda a derecha. Esa es la razón del segundo control con WHILE.

5.4.3 3. Explicación del programa

Se define una nueva orden de AutoCAD y cinco funciones más de usuario.

• Función **c:ondas**

Define la nueva orden de AutoCAD llamada "ondas". Establece como argumentos locales todas las variables restantes que no han sido establecidas como locales en las funciones intermedias.

Llama sucesivamente a las funciones "intr", "imc", "dib" y "ult".

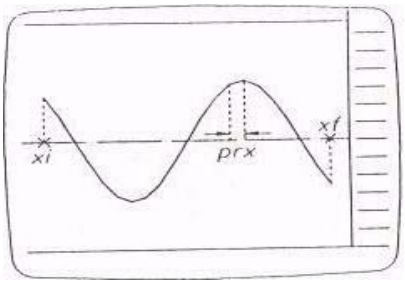
• Función **intr**

Solicita del usuario los datos requeridos. Se trata de dibujar la curva resultante de la suma de tres ondas (funciones senoidales) con diferentes coeficientes para la variable X. La función tiene la siguiente forma:

$$Y = SEN (cf1\ X) + SEN (Cf2\ X) + SEM (cf3\ X)$$

siendo cf1, cf2 y cf3 los tres coeficientes.

La curva se va a dibujar a base de tramos de líneas rectas, de acuerdo con la precisión en X almacenada en "prx". El trozo de curva a dibujar será el comprendido entre los valores ‘‘xi" y "xf”.



Variables para el trazado de la curva de función.

El número de tramos de línea a dibujar será, pues, el resultado de dividir el intervalo entre la precisión en X. Ese valor se almacena en la variable n.

$$(xf - xi)n = prx$$

Con el comando FIX se toma el valor entero de ese cociente y entonces "n" almacena el número de tramos de línea completos a dibujar. Pura completar el intervalo entre "xi" y "xf” con el último "pico' o "resto" que queda, se utilizará la función "ult".

Variables de entrada: ninguna.
Variables locales: ninguna.

Variables de salida:

cf1 Coeficiente de la primera onda.
cf2 Coeficiente de la segunda onda.
cf3 Coeficiente de la tercera onda.
prx Intervalo de precisión en X.
xi Coordenada X de inicio de la curva.
xf Coordenada X de final de la curva.
n Número de tramos de línea completos.

• Función **fun**.

Es una función auxiliar que sirve para calcular todos los puntos de la curva. Simplemente contiene en formato de AutoLISP la definición de la función Y = F(X) con la suma de las tres ondas- Tiene una variable asociada "x" y, cada vez que es llamada para un valor de "x", devuelve un valor que es la coordenada' correspondiente a ese punto de la curva.

• Variables de entrada:

cf1 Coeficiente de la primera onda.
cf2 Coeficiente de la segunda onda.
cf3 Coeficiente de la tercera onda.
x Coordenada X del punto de la curva cuya coordenada Y se quiere obtener.

Variables locales; ninguna.
Variables de salida:

No existe ninguna, pero la evaluación de la propia "func (x)" devuelve un valor que es la coordenada Y buscada.

• Función **inic**

Calcula el primer punto para empezar a dibujar la curva. Para ello toma como coordenada X la correspondiente al inicio de la curva y calcula la coordenada Y de ese punto llamando a la función "func". Reúne "x1 e "y1" en una lista y lo almacena en “p1", que contiene de esta manera el primer punto de la curva.

Como la variable "y1" se ha utilizado simplemente para obtener después "pt", puede ser especificada como local. La variable "x1" no puede ser local, porque va a ser utilizada después por la función “dib”.

Variables de entrada:
xi Coordenada X del inicio de la curva.
Variables locales:
Y1 Coordenada Y del punto de inicio.
Variables de salida:
X1 Coordenada X del punto de inicio.
P1 Punto de inicio de la curva.

• Función **dib**

Dibuja todos los tramos completos de línea con los que se va a aproximar el trazado de la curva resultante. Para ello establece un ciclo repetitivo que dibuja cada tramo. El ciclo se repetirá "n" veces, que es el número de tramos completos.

Al entrar en el ciclo por primera vez existe ya un valor de punto inicial del tramo "p1", que viene de la función "inic". Dentro del ciclo se calcula el punto (mal de cada tramo. Su coordenada X resulta de sumar la precisión "prx" al valor X del

punto inicial "x1". La coordenada Y se obtiene llamando a la función "func" para ese valor de "x2". Se reúnen ambos en una lista y se obtiene el punto "p2".

Se dibuja el tramo de línea entre "pl" y "p2". Por último, se actualiza la variable "p1" de forma que al repetir el ciclo se tome como nuevo punto inicial el final del ciclo anterior. Lo mismo con la variable "x1".

Como variables locales se pueden establecer las correspondientes al punto final del tramo "x2" "y2" "p2", puesto que una vez dibujada la línea de cada tramo se almacena ese punto como nuevo punió inicial "p1", y ese es el valor que interesa conservar.

Al terminar "dib", el último valor de "pl" será utilizado por la función "ult".

Variables de entrada:

n Número de tramos completos de línea.
X1 Coordenada X del punto inicial de curva.
prx Intervalo de precisión en X.
p1 Primer punto del primer tramo.

Variables locales:

x2 Coordenada X del punto final del tramo.
y2 Coordenada Y.
p2 Punto final del tramo.

Variables de salida:

p1 Punto final del último tramo, que será el inicial para dibujar el "resto".

• Función **ult**

Dibuja el último "resto" que queda cuando no hay un número de intervalos completos que lleguen hasta el final de curva "xf". Para ello calcula la coordenada "yf" del final de la curva llamando a func y obtiene el punto final “p2”.

Como se encuentra almacenado en “pl” el punto final del último tramo del cielo repetitivo, pasa a dibujar la línea entre "p1" y "p2".

Variables de entrada:

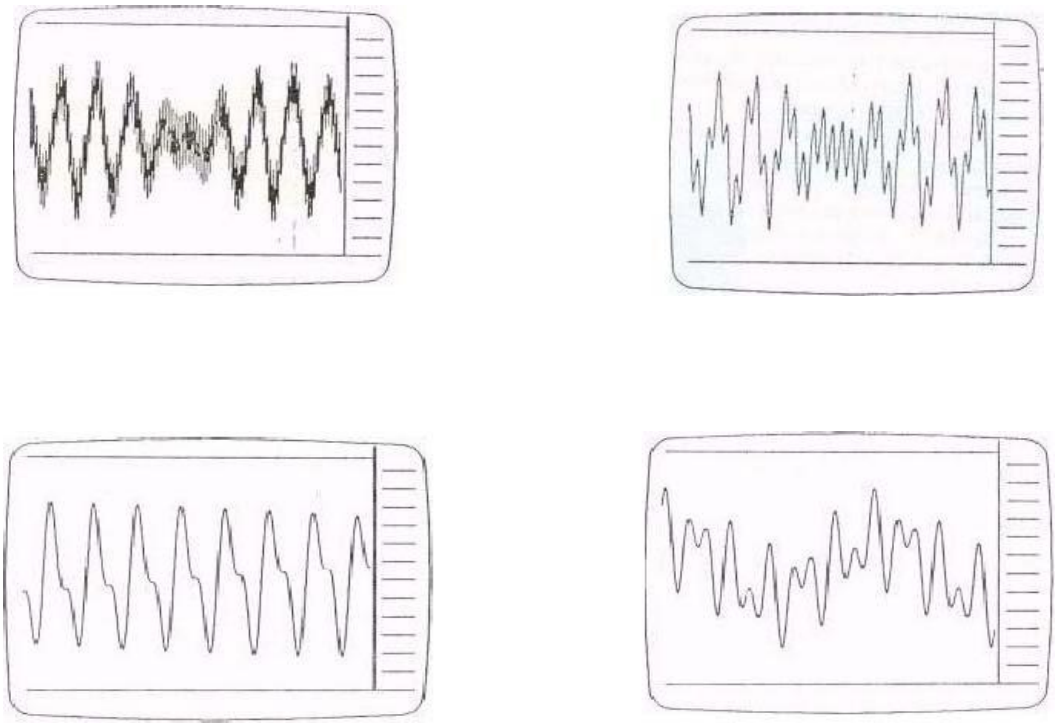
xf Coordenado X del punto final de la curva.pl Punto inicial del "resto".

Variables locales:

yf Coordenada Y del punto final de curva.p2 Punto final de curva.

Variables de salida: ninguna.

En la Figura se muestran algunos trazados posibles de curvas.



Resultados gráficos del trazado de diferentes curvas, con diferentes valores de los tres coeficientes.

6 Capitulo 6: Ángulos y distancias

En este capítulo se revisa un nuevo tipo de datos que pueden ser requeridos por un programa en AutoLISP, como son ángulos y distancias. Es preciso tener en cuenta siempre que los ángulos se miden en radianes dentro de AutoLISP, Como por comodidad se van a escribir casi siempre en grados sexagesimales, interesará tener un pequeño programa de conversión de grados a radianes. Este programa se puede llamar "dir" (grados a radianes en ingles) y su contenido será el siguiente:

```
( DEFUN dtr (g)
        ( * PI ( / g 180.0 ) )
)
```

Un programa similar para realizar la operación inversa, convertir radianes en grados, se podría llamar "cid" y su contenido seria:

```
( DEFUN rtd (r)
        ( / ( * r 180.0) PI )
)
```

Para que estas dos definiciones de funciones se carguen automaticamente nada más entrar en un dibujo, habría que incluirlas en el archivo ACAD.LSP.

6.1 (ANGLE <pt1> <pt2) Ángulo definido por dos puntos.

Este comando devuelve el ángulo determinado por la linea entre los dos puntos <pt1> y <pt2> especificados. El valor del ángulo se mide respecto al eje X actual en el dibujo. El ángulo se mide en radianes y su sentido positivo es el antihorario. Si los puntos indicados son en 3D. se proyectan al plano X-Y actual.

Por ejemplo, para almacenar el ángulo dormido por dos puntos introducidos por el usuario:

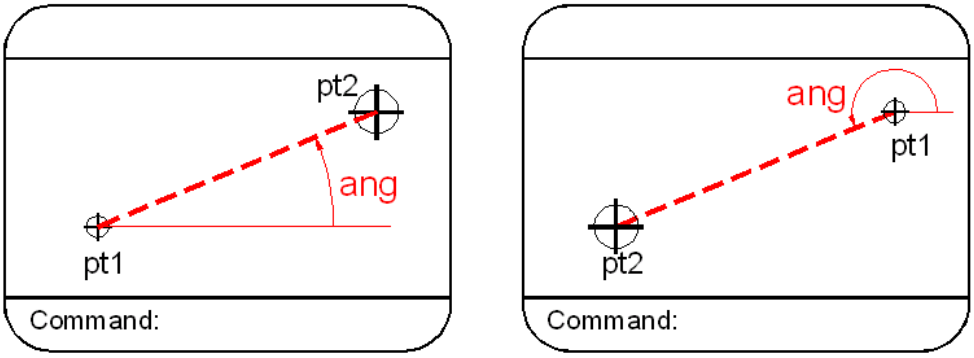
Command:

```
( SETQ pt1 ( GETPOINT "Primer punto: " ) )
( SETQ pt2 ( GETPOINT “pt2 "Segundo punto: " ) )
( SETQ ang ( ANGLE pt1 pt2 ) )
( SETQ ang ( rtd ang ) )
```

La última operación es para convertir el valor devuelto por ANGLE en grados sexagesimales, suponiendo que hayamos cargado en ACAD.LSP la función “rtd" antes definida.

Es importante el orden de introducción de los puntos, pues el ángulo sería diferente para cada caso. No es lo mismo (ANGLE pt1 pt2) que (ANGLE pt2

pt1).



Diferentes ángulos devueltos por ANGLE según el orden de introducción de los puntos.

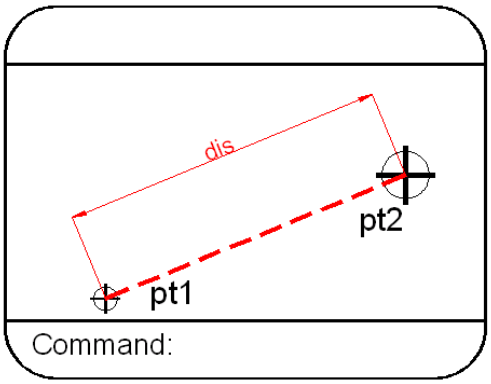
6.2 (DISTANCE <pt1> <pt2>) Distancia entre dos puntos

Este comando devuelve la distancia entre los dos puntos especificados. En este caso, lógicamente, es indiferente el orden de introducción de los puntos.

Command:

(SETQ ptl (GETPOIMT "Primer punto: "))
(SETQ pt2 (GETPOINT pt1 "Segundo punto: "))
(SETQ dis (DISTANCE pt1 pt2))

El valor almacenado en "dis" en el ejemplo es un numero real, que mide la distancia entre ambos puntos de acuerdo con sus coordenadas en el SCP actual. Las unidades son siempre decimales, independientemente del tipo de unidades actuales en el dibujo (Pies y pulgadas, Fraccionario, etc.) (Figura 6.2).



Funcionamiento del comando DISTANCE.

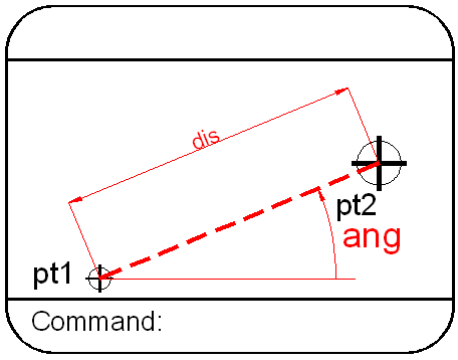
6.3 (POLAR <pt> <ang> <dist>) Punto en coordenadas Polares

Este comando devuelve un punto obtenido en coordenadas relativas polares a partir del especificado <pt>, es decir, desde el punto <pt> se lleva una distancia <dist> en un ángulo <ang> y se obtiene el nuevo punto. Como siempre, el ángulo introducido en <ang> se considera en radianes y positivo en sentido

antihorario. Aunque <pt> puede ser en 3D, el valor de <ang> se toma siempre respecto al plano X-Y actual.

Command:

(SETQ pt1 (GETPOINT "Punto de base: "))
(SETQ dis (GETREAL "Distancia: "))
(SETQ pt2 (POLAR ptl (/ PI 2) dis))



Visualización en pantalla de POLAR.

En el ejemplo se introduce un punto de base "pt1", una distancia "dis" y la variable "pt2" almacena el punto situado a esa distancia en vertical, es decir, a un ángulo de 9Ø grados, que son PI/2 en radianes. Disponiendo de la función "dtr" para convertir grados en radianes, se podría haber puesto:

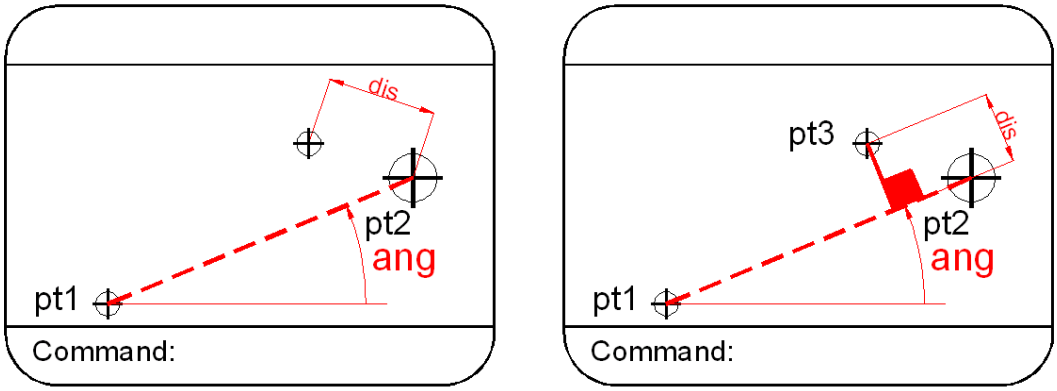
(SETQ pt2 (POLAR pt1 (dtr 90) dis))

El siguiente seria otro ejemplo un poco más completo para obtener un punto situado a la distancia especificada en perpendicular a la línea dada por dos puntos pt1 y pt2:

Command:

(SETQ pt1 (GETPOINT "Primer punto: "))
(SETQ pt2 (GETFOINT pt1 "Segundo punto: "))
(SETQ dis (GETDIST pt2 "Distancia! "))
(SETQ ang (ANGLE ptl pt2))
(SETQ pt3 (POLAR pt2 (+ ang (dtr 90)) dis))

La variable "ang" almacena el valor del ángulo de la línea entre pt1 y pt2. El punto pt3 se obtiene tomando una distancia "dis" a partir de pt2 y con un ángulo igual a "ang" + 9Ø grados, es decir, (+ ang (dtr 9Ø)) .



Ejemplo de utilización de POLAR

El procedimiento explicado en este ejemplo se va a utilizar con el programa propuesto al final del capitulo, para dibujar un muro con espesor a partir de su línea central.

6.4 (GETANGLE [<pt>] [<"mensaje">]) Introducir ángulo.

Este comando espera a que el usuario introduzca un ángulo y devuelve el valor de ese ángulo. El ángulo se puede introducir tecleando directamente en el formato actual de unidades (podrían ser, por ejemplo, grados/minutos/segundos), pero el valor devuelto por GETANGLE será siempre un número real en radianes.

La ventaja de este comando respecto a ANGLE es que se puede especificar el punto de base en <pt> y señalar el segundo punto para obtener directamente el ángulo. En este caso se visualiza una línea "elástica" entre el punto de base y la posición del cursor.

El punto de base <pt>, así como el mensaje a visualizar, son optativos. Si se indica un punto de base en 3D, el ángulo se mide únicamente sobre el plano X-Y actual.

Command:

(SETQ pt1 (GETPOINT "Punto de base: "))
(SETQ ang (GETANGLE pt1 "Segundo punto: "))

En el ejemplo se ha obtenido el ángulo "ang" sin necesidad de almacenar previamente el segundo punto en "pt2" y ejecutar el comando ANGLE.

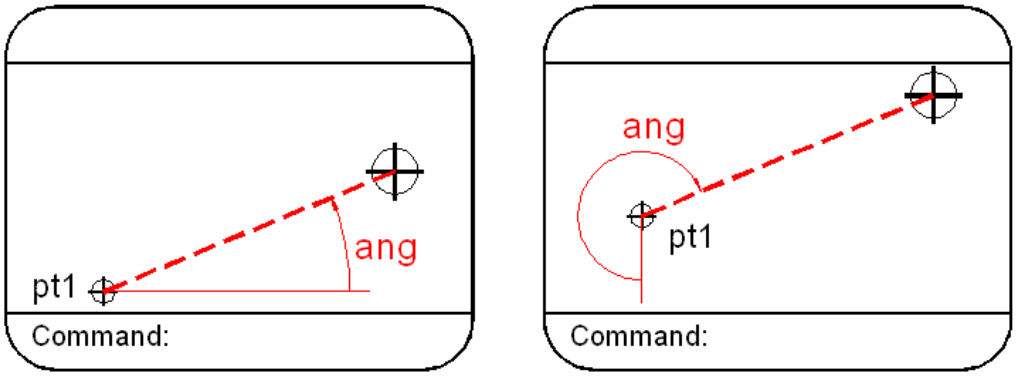
6.5 (GETORIENT [<pto>] [<"mensaje">]) Ángulos según origen y sentido.

A diferencia de GETANGLE, que mide siempre los ángulos desde la posición habitual de Ø grados (la posición de las 3 del reloj) y positivos en sentido antihorario, el comando GETORIENT considera los ángulos con el origen Ø grados y el sentido actuales definidos por la orden "Unidades".

El valor devuelto es siempre en radianes, independientemente del tipo de unidades definido para los ángulos por la orden "Unidades". Si el punto de base es en 3D, el ángulo se mide sólo sobre el plano X-Y actual.

Por ejemplo, si la actual definición de la orden "Unidades" contiene un origen de ángulos en el eje Y negativo (las 6 del reloj) y el sentido positivo como horario (Figura 6.5):

Command:
(SETQ pt1 (GETPOINT "Punto de base: "))
(SETQ ang (GETORIENT pt1 "Segundo punto: "))



Medición de ángulos GETORIENT según origen y sentido de ángulos.

6.6 (ANGTOS <ang>l<modo> [<prec>]]) Conversión de ángulos.

Este comando toma el ángulo especificado en <ang>, que debe ser un número real en radianes, y lo devuelve editado como cada texto según el formato especificado por <modo> y precisión <prec>. Estos valores corresponden a las Variables de Sistema AUNITS y AUPREC, respectivamente. Si no se suministran los argumentos <modo> y <prec>, se

toman los valores actuales de esas dos Variables de Sistema.El argumento <modo> es un número entero entre Ø y 4, cuyos formatos son los siguientes:

Modo Ø	Grados
Modo 1	Grados/minutos/segundos.
Modo 2	Grados centesimales g.
Modo 3	Radianes.
Modo 4	Geodesia.

El argumento <prec> especifica la precisión en decimales con que se va a obtener la cadena de texto.

Por ejemplo, suponiendo que la variable “ang” contiene un valor actual de PI radianes (es decir 18Ø grados), la ejecución de ANGTOS daría los siguientes resultados:

(ANGTOS ang Ø 2) devuelve "18Ø.ØØ"
(ANGTOS ang 1 4) devuelve "18ØdØ'Ø""
(ANGTOS ang 3 4) devuelve "3.1416r"
(ANGTOS, ang 4 3) devuelve "Ø"

El ángulo <ang> puede ser negativo, pero el valor es siempre convenido a positivo entre Ø y 2PI.

6.7 (RTOS <num> [<modo> [<prec>]]) Conversión de números

Este comando toma el número real especificado en <num> y devuelve una cadena según el formato especificado por el modo y la precisión <prec>. Estos argumentos modo y precisión corresponden a las Variables de Sistema LUNITS y LUPREC, respectivamente. Si no se especifican los argumentos, se utilizan los actuales valores de esas dos variables.

El argumento <modo> es un número entero entre 1 y 5, cuyos formatos son los siguientes:

Modo 1	Científico.
Modo 2	Decimal.
Modo 3	Pies-y-pulgadas I (fracción decimal).
Modo 4	Pies-y-pulgadas II (fracción propia).
Modo 5	Fracciones de cualquier unidad.

El argumento <prec> indica la precisión en decimales para la cadena de texto a obtener. Por ejemplo, suponiendo que la variable ‘‘num" contiene un valor actual de 24.75, la ejecución de RTOS podría ofrecer los siguientes resultados:

(RTOS num 1 3) devuelve "2.475E+Ø1"
(RTOS num 2 1) devuelve "24.7"
(RTOS num 3 2) devuelve "1' -5.5Ø""
(RTOS num 4 2) devuelve "1'- 5 1/2""
(RTOS num 5 2) devuelve “24 3/4"

6.8 (INTERS <pt1><pt2><pt3><pt4>[<cs>]) Intersección de lineas.

Este comando toma <ptl> y <pt2> como extremos de una línea, <pt3> y <pt4> como extremos de otra línea, y calcula la intersección de ambas devolviendo ese punto de intersección.

El parámetro <cs> es optativo. Si su valor es "nil", el comando considera las líneas como infinitas y devuelve el punto de intersección aunque se encuentre

en la prolongación de las líneas ("fuera" de ellas). En este caso todas las líneas en 2D tendrían intersección, salvo las paralelas. En 3D las líneas se podrían cruzar y entonces no tendrían intersección.

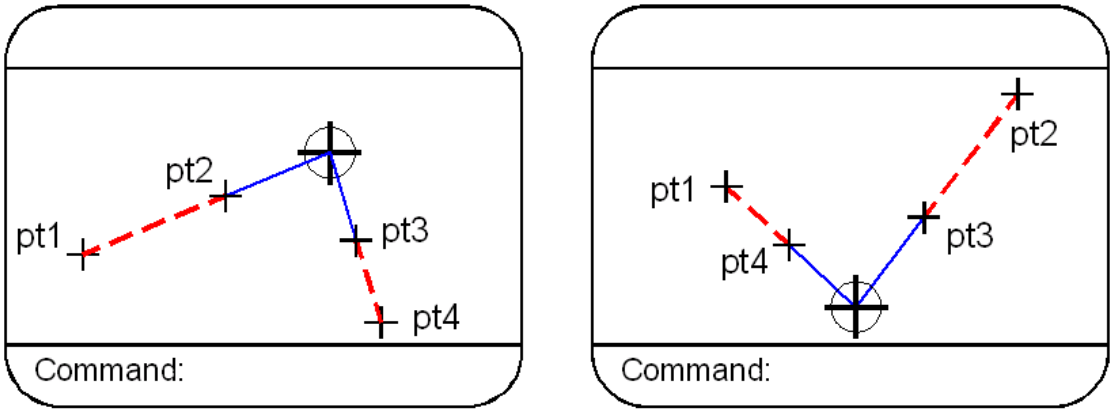
Si el parámetro <cs> no se indica o su valor es diferente de "mi", entonces el punto de intersección sólo se devuelve si se encuentra entre los extremos de ambas líneas ("dentro" de ellas). En caso contrario devolvería "nil".

Command:
(SETQ p1 ' (9Ø 6Ø) p2 ' (16Ø 14Ø) p3 ' (11Ø 5Ø) p4 ' (24Ø 11Ø))
(INTERS p1 p2 p3 p4) devuelve nil
(INTERS p1 p2 p3 p4 T) devuelve nil
(INTERS p1 p2 p3 p4 nil) devuelve (61.77 27.74 Ø.Ø)

En el ejemplo, las dos líneas no tenían una intersección real, y solamente especificando un parámetro <cs> igual a "nil" se devuelve la intersección en la prolongación de ambas lineas.

Hay que tener precaución en indicar los cuatro puntos en el orden correcto, pues en caso contrario las lineas cuya intersección calcula INTERS son diferentes.

(INTERS p1 p3 p2 p4 nil) devuelve (-76Ø.Ø 485.Ø Ø.Ø)



Intersección devuelta por INTERS, según el orden de especificación de los puntos.

6.9 Ejemplo 1: Dibujar un muro con espesor

Este programa dibuja un muro con el espesor indicado. El usuario introduce los puntos inicial y final medidos sobre la línea media del muro, y el programa que lleva la mitad del espesor a cada lado y dibuja las dos líneas paralelas del muro.

6.9.1 Listado del programa

(DEFUN inic ()

```
  (SETQ pin (GETPOINT "Primer punto: "))
  (TERPRI)
  (SETQ e (GETDIST pin "Espesor: "))
  (TERPRI)
  (SETQ pf (GETPOINT pin "Segundo punto: "))
  (TERPRI)
  (SETQ ang (ANGLE pin pf))

)

(DEFUN dib
  (/ p1 p2 p3 p4)
  (SETQ p1 (POLAR pin (+ ang (/ PI 2)) (/ e 2)))
  (SETQ p3 (POLAR p1 (- ang (/ PI 2)) e))
  (SETQ p2 (POLAR pf (+ ang (/ PI 2)) (/ e 2)))
  (SETQ p4 (POLAR p2 (- ang (/ PI 2)) e))
  (COMMAND "line" p1 p2 "")
  (COMMAND "line" p3 p4 "")
  (COMMAND "line" p1 p3 "")
  (COMMAND "line" p2 p4 "")
)

(DEFUN c:pared
  (/ pin e pf ang)
  ;(SETVAR "blipmode" 0)
  ;(SETVAR "cmdecho" 0)
  (inic)
  (dib)
  ;(SETVAR "blipmode" 1)
  ;(SETVAR "cmdecho" 1)
)
```

6.9.2 Observaciones.

En su versión más sencilla este programa dibuja un muro con espesor, definido a partir de su línea media.

Hay que insistir en una precaución importante: no utilizar como nombre de variables los nombres de comandos de AutoLISP. Se puede tener la costumbre de abreviar el nombre de un punto inicial como “pi”. Pero si se utiliza ese símbolo como nombre de variable, esto inhabilita el nombre de comando de AutoLISP “PI” que contiene el valor “3.1416”.

Por este motivo, como nombre de variable de punto inicial, se utiliza en el programa “pin”. Lo mismo cabe advertir respecto al símbolo “T” que representa “Trae, cierto” para AutoLISP. Nunca se deberá usar “t” como nombre de variable.

6.9.3 Explicación del programa Define una nueva orden

Define una nueva orden de AutoCAD y dos funciones de usuario.

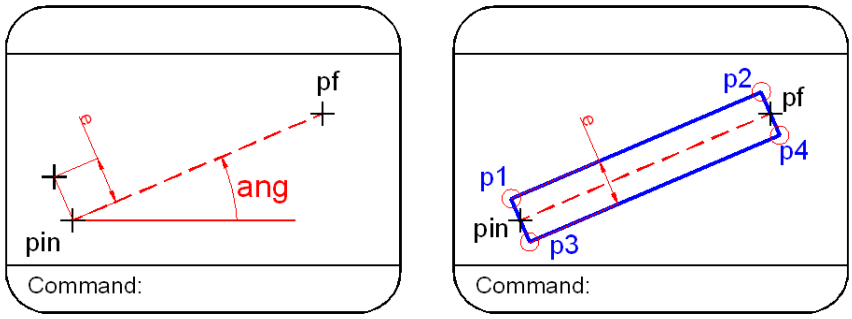
• Función **c:pared**

Añade una nueva orden a AutoCAD llamada "pared". Como siempre, desactiva marcas auxiliares y eco de menú y después llama a las dos funciones de usuario "inic" y "dib".

• Función **inic**

Solicita las variables iniciales necesarias para el programa y calcula el ángulo en el que va a estar orientado el muro.
Solicita el primer punto y lo almacena en "pin". Acepta como dato el espesor del muro y lo almacena en la variable "e". Se utiliza GETDIST para que se pueda indicar el espesor mediante dos puntos si se prefiere (cosa que no podría hacerse con GETREAL).

Se almacena el segundo punto en "pr y se calcula el ángulo mediante ANGLE, almacenándolo en la variable "ang".



Variables para el trazado del muro y puntos a calcular por el programa.

Variables de entrada: ninguna.
Variables locales: ninguna.
Variables de salida:

pin Punto inicial de línea media del muro.
pf Punto final de línea media del muro.
e Espesor del muro.
ang Ángulo de la línea media del muro.

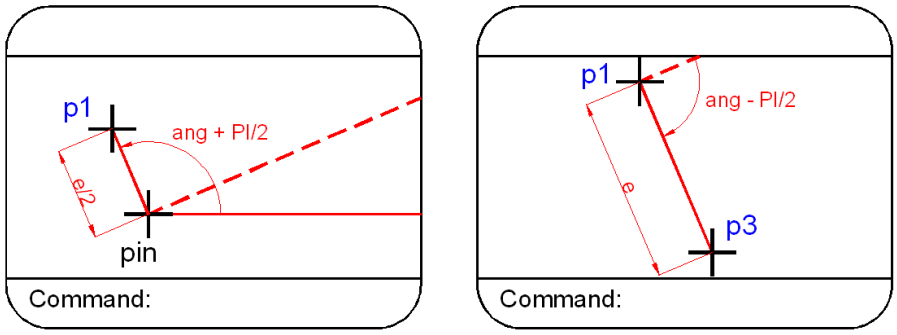
• Función **dib**

Calcula los cuatro puntos de los vértices del muro o pared y lo dibuja mediante cuatro líneas.

El primer punto "p1" se calcula a partir de "pin", llevando una distancia igual a la mitad del espesor a un ángulo de 9Ø grados (P l/2) sumado a "ang". Para ello se emplea el comando POLAR.

El segundo punto "p2" se obtiene de la misma manera, también con el empleo del comando POLAR, pero partiendo en este caso del punto final "pf". Se lleva también una distancia igual a la mitad del espesor a un ángulo igual a ang + P

1/2.



Obtención de los vértices del muro por el programa.

El tercer punto "P3" a partir de "p1", llevando una distancia igual al espesor a un ángulo de 9Ø grados restados a "ang".

Este tercer punto se podía haber obtenido a partir de "pin", llevando una distancia igual a la mitad del espesor a un ángulo "ang – P I/2".

El cuarto punto "p4" se obtiene de la misma manera, bien llevando la mitad del espesor desde "pf", o bien el espesor completo desde "p2". El ángulo es el mismo de antes, ang – P 1/2.

• Variables de entrada:
  pin Punto inicial de línea media del muro.
  pf Punto final de línea media del muro.
  e Espesor del muro.
  ang Ángulo de la línea media del muro.

• Variables locales:
  p1 Primer vértice del muro.
  p2 Segundo vértice.
  P3 Tercer vértice.
  p4 Cuarto vórtice.

• Variables de salida: ninguna.

6.10 Ejemplo 2: Dibujar varios muros con espesor

Como ampliación del ejemplo anterior, este programa permite dibujar una serie de muros encadenados con espesor calculando las esquinas para que queden de forma adecuada.

De momento, el programa no permite ninguna opción como unir la última pared con la primera (opción "Cierra") o la posibilidad de ir eliminando las paredes hacia “atrás” si el usuario se ha equivocado en su introducción (opción "Revoca").

6.10.1 Listado del programa

```
(DEFUN inic
  (/ pf ang)
  (GRAPHSCR)
  (SETQ pin (GETPOINT "Primer punto: "))
  (TERPRI)
  (SETQ e (GETDIST pin "Espesor: "))
  (TERPRI)
  (SETQ pf (GETPOINT pin "Segundo punto: "))
  (TERPRI)
  (SETQ ang (ANGLE pin pf))
  (SETQ p1 (POLAR pin (+ ang (/ PI 2)) (/ e 2)))
  (SETQ p3 (POLAR p1 (- ang (/ PI 2)) e))
  (SETQ p2 (POLAR pf (+ ang (/ PI 2)) (/ e 2)))
  (SETQ p4 (POLAR p2 (- ang (/ PI 2)) e))
  (SETQ pin pf)
  (SETQ n 1)
)

(DEFUN int
  (/ ang)
  (SETQ ang (ANGLE pin pf))
  (SETQ p5 (POLAR pin (+ ang (/ PI 2)) (/ e 2)))
  (SETQ p7 (POLAR p5 (- ang (/ PI 2)) e))
  (SETQ p6 (POLAR pf (+ ang (/ PI 2)) (/ e 2)))
  (SETQ p8 (POLAR p6 (- ang (/ PI 2)) e))
  (SETQ i1 (INTERS p1 p2 p5 p6 nil))
  (SETQ i2 (INTERS p3 p4 p7 p8 nil))
)

(DEFUN act ()
  (SETQ       p1  p5
        p2  p6
        p3  p7
        p4  p8
        pin pf
        ib1 i1
        ib2 i2
  )
  (SETQ n (+ 1 n))
)

(DEFUN ciclo
  (/ pf p5 p6 p7 p8 i1 i2 ib1 ib2)
  (WHILE n
    (IF  (= n 1)
      (PROGN
        (COMMAND "line" p1 p3 "")
        (SETQ pf (GETPOINT pin "\nNuevo punto: "))
```

```
            (int)
            (COMMAND "line" p1 i1 "")
            (COMMAND "line" p3 i2 "")
            (act)
          )
          (PROGN
            (IF (SETQ pf (GETPOINT pin "\nNuevo punto: "))
              (PROGN
                (int)
                (COMMAND "line" ib1 i1 "")
                (COMMAND "line" ib2 i2 "")
                (act)
              )
              (PROGN
                (COMMAND "line" ib1 p2 "")
                (COMMAND "line" ib2 p4 "")
                (COMMAND "line" p2 p4 "")
                (SETQ n nil)
              )
            )
          )
        )
      )
    )
  )

  (DEFUN c:paredes
    (/ pin e p1 p2 p3 p4 n)
    ;( SETVAR "bliprnode" 0 )
    ;( SETVAR "cmdecho" 0 )
    (inic)
    (ciclo)
    ;( SETVAR "blipmode" 1 )
    ;( SETVAR "cmdecho" 1 )
  )
```

6.10.2 Observaciones

El programa empieza a manifestar una cierta complejidad de estructura. Las llaves del listado visualizan el nivel de anidamiento de esas estructuras (el nivel hasta el que se van incluyendo unas dentro de otras).
La repetitiva, en este caso es WHILE, conforma una única estructura (una llave).

La alternativa supone siempre la generación de dos “ramas” en la estructura del programa: la rama del “Si”, que es la acción a efectuar si la condición se cumple, y la rama del "No", que es la acción a efectuar si no se cumple. Si una de las ramas está “vacía" (no hay ninguna acción a efectuar), la estructura del programa se simplifica.

En el programa de este ejemplo, la alternativa incluida en el WHILE presenta acciones a efectuar en las dos ramas. Además, una de ellas incluye otra alternativa que a su vez contiene acciones a efectuar en las dos ramas, lo que complica un poco la estructura del programa.

Por otro lado, se ha incluido un comando PROGN para contener esta última alternativa que realmente no es necesario. Sin embargo, puede ayudar para visualizar de forma mas clara la estructura completa.

Un último aspecto a destacar es el empleo del comando GETPOINT, que solicita el nuevo punto de las paredes a dibujar, como condición de las alternativas IF. Si a la solicitud de ese punto el usuario responde con un RETURN, el comando GETPOINT devuelve “nil”, lo que se puede utilizar como condición no cumplida y, por tanto, el programa efectúa la acción incluida en la rama del "No".

6.10.3 Explicación del programa:

Define una nueva orden de AutoCAD y cuatro funciones más de usuario.

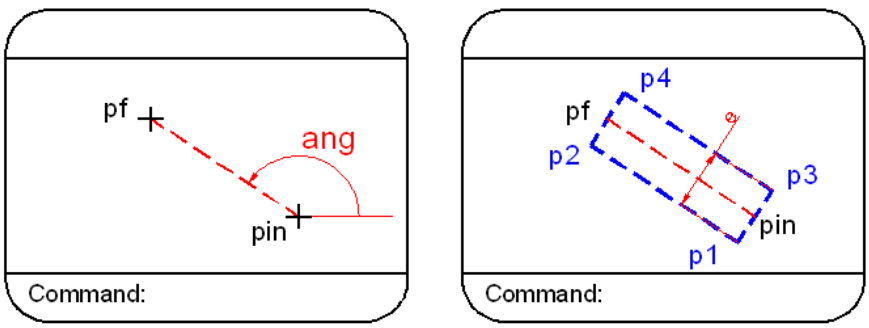
• Función **c:paredes**

Como siempre, se limita a desactivar marcas auxiliares y eco de órdenes y llama a las funciones de usuario que necesita; en este caso, "inic" y "ciclo".

• Función **inic**

Solicita del usuario todos los datos necesarios para obtener la primera pared. El punto inicial se almacena en la variable "pin", el espesor en "e" y el segundo punto en "pf". No es necesario establecer ningún control, puesto que cualquier punto es valido, y un valor negativo o cero del espesor no impide funcionar al programa.

A continuación, tal como se ha explicado en la versión mas sencilla del ejemplo anterior, se calcula el ángulo en el que está orientada la primera pared y los cuatro vértices de esa pared.



Obtención de la primera paied o partir de las variables iniciales.

Al principio se ha añadido el comando GRAPHSCR para que el programa

conmute a pantalla gráfica nada más a empezar a ejecutarse. Al fin se actualiza la variable "pin" para que el programa entre en el ciclo y dibuje la nueva pared tomando como nuevo punto inicial el punto final de la pared anterior.

Por último se inicializa la variable "n" a 1, pues va a ser el contador del número de paredes que se vayan generando. Al mismo tiempo se va a utilizar como variable de control del ciclo.

Las variables "pf" y "ang" son locales y sus valores no se van a utilizar una vez finalizada la función.

Variables de entrada: ninguna

Variables locales:

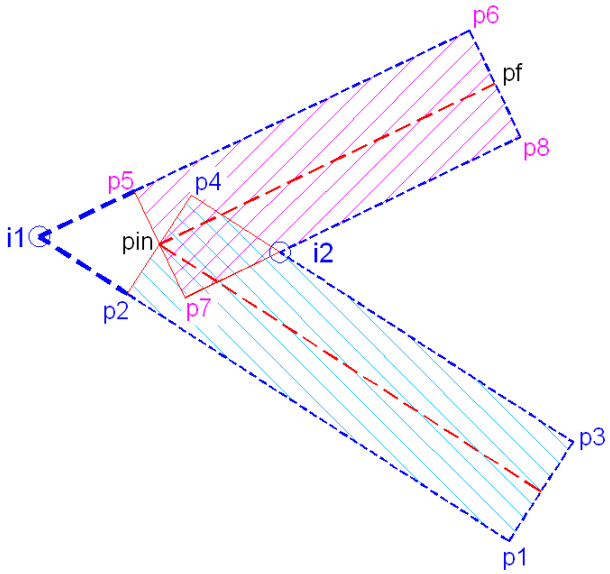
pf Punto final de la primera pared.
ang Ángulo de la primera pared.

Variables de salida:

pin Punto inicial de la nueva pared.
e Espesor homogéneo de todas las paredes.
p1 Primer vértice de la primera pared.
p2 Segundo vértice.
p3 Tercer vértice.
p4 Cuarto vértice.
n Número de pared y control del ciclo.

• Función **int**

Se la llama desde la función "ciclo", una vez, que se ha solicitado el nuevo punto final de la nueva pared. La función "int" (intersecciones) calcula los cuatro vértices de esa nueva pared y los almacena en las variables "p5", "p6", "p7", "p8". A continuación calcula las intersecciones entre la nueva pared y la anterior mediante el empleo del comando INTERS. Los puntos de intersección se almacenan en las variables "i1", "i2". Se emplea INTERS con el parámetro "nil" para que calcule las intersecciones, incluyendo también las prolongaciones de las líneas.



Cálculo de los puntos de intersección entre la primera y la segunda pared.

Variables de entrada:

pin Punto inicial de la nueva pared.
pf Punto final de la nueva pared.
p1 Primer vértice de la pared anterior.
p2 Segundo vértice.
p3 Tercer vértice.
p4 Cuarto vértice.

Variables locales:
ang Ángulo de orientación de la nueva pared.

Variables de salida:

p5 Primer vórtice de la nueva pared.p6 Segundo vértice.
p7 Tercer vértice.
p8 Cuarto vértice.
i1 Primer punto de intersección.
i2 Segundo punto de intersección.

• Función **act**

Actualiza los valores de variables para poder dibujar la nueva pared. Es llamada desde la función “ciclo" cada vez que se han dibujado las líneas de la pared anterior.

Su cometido es almacenar los cuatro vértices de la última pared ("p5" a "p8") en las variables "p1" a "p4" para que sean tomados como vértices de pared anterior al empezar a calcular una nueva.

Al mismo tiempo actualiza la variable "pin" como nuevo punto inicial, necesario para calcular el ángulo de la nueva pared, y guarda los valores de los dos últimos puntos de intersección en "ib1" e "ib2". Estos valores van a ser necesarios para dibujar las líneas de la nueva pared.

Por último, la función incrementa en 1 el valor de "n", que es el contador del número de pared.

Variables de entrada:

pf Punto final de la última pared.
p5 Primer vértice de la última pared.
p6 Segundo vértice.
p7 Tercer vértice.
p8 Cuarto vértice
i1 Primer punto de intersección.
i2 Segundo punto de intersección.
n Número de pared.

Variables locales: ninguna,

Variables de salida:
pin Punto inicial de la nueva pared.
p1 Primer vértice de la pared anterior.
p2 Segundo vértice.
p3 Tercer vértice.
p4 Cuarto vértice.
ib1 Último primer punto de intersección.
ib2 Último segundo punto de intersección.
n Número de la nueva pared.

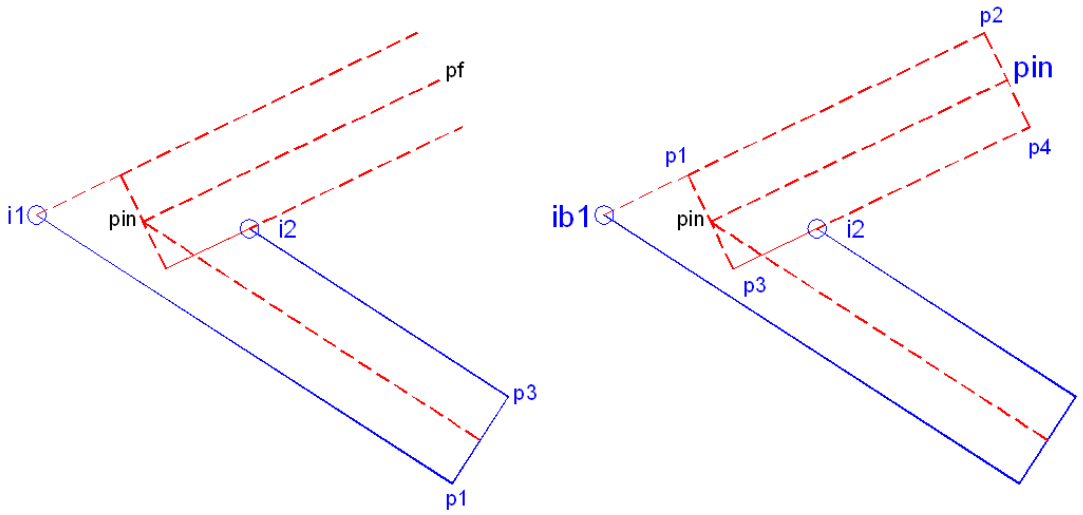
• Función **ciclo**

Es la función que contiene prácticamente la totalidad del programa. Su n estructura básica es un ciclo repetitivo establecido mediante WHILE, con la propia variable “n" como control.

Cada vez que se entra en el ciclo, el programa establece un IF para chequear si el valor de n es I. La primera vez, que se ejecute el ciclo se cumplirá esta condición (puesto que “n” viene con valor 1 desde la función “inic"). En este caso se efectúa la acción incluida en la rama del ‘‘Si”.

• Condición cierta de **IF (= n 1)**

Como es la primera pared, se dibuja la línea que cierra esa pared entre "p1" y “p3”. Se solicita el nuevo punto final para la siguiente pared y se llama a la función "int”, que calculara los cuatro nuevos vértices y las dos intersecciones de esta nueva pared con la primera.



Trazado de la primera pared y obtención de puntos para la segunda.

El mensaje visualizado con GETPOINT para solicitar el punto final de la nueva pared es:

"\nNuevo punto: "

El código de control "\n" origina un RETURN antes de visualizar "Nuevo punto: ".

A continuación, y una vez calculados los dos puntos de intersección, se dibujan las líneas de la primera pared hasta dichos puntos.

Por último se llama a la función "act" para actualizar valores de variables y dejarlos dispuestos para volver a entrar en el ciclo y calcular la siguiente pared. Hay que tener en cuenta que las líneas que delimitan una pared sólo se dibujan cuando el usuario ha indicado el punto final de la siguiente pared y se han calculado los puntos de intersección. Este funcionamiento es similar al de la orden de AutoCAD TRACE.

• Condición no cierta **IF (= n 1)**.

Una vez trazada la primera pared, como la función "act" va a sumar 1 a la variable “n”, el programa va a pasar siempre por la rama del "No".

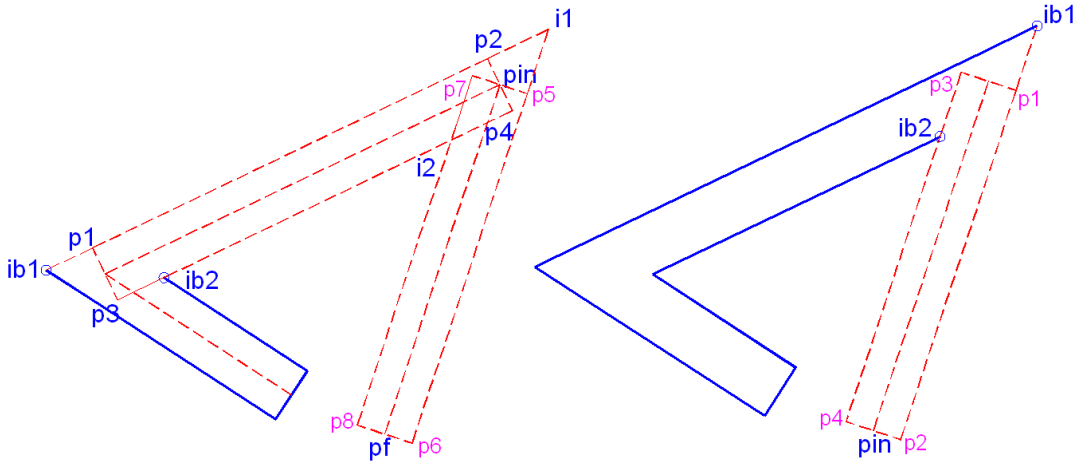
Esta rama es a su vez un condicional, que en el programa está agrupado en un PROGN innecesario, pero incluido por razones de claridad de estructuras. Este condicional utiliza la solicitud de GETPOINT para el nuevo punto final de la nueva pared como condición. Si el usuario selecciona efectivamente un punto, la condición se cumple y el programa efectúa la acción de la rama del "Sí". Si el usuario responde con "Return", el comando GETPOINT devuelve “nil” y entonces la condición no se cumple, con lo que el programa pasa por la rama del "No".

En la rama del "Si", una vez que el usuario ha introducido un nuevo punto para la nueva pared, se llama a la función "int" para calcular los dos nuevos puntos de intersección y se dibujan las dos líneas de la anterior pared, entre "ib1" "i1" y entre "ib2" "i2".

Como se ha dicho, este funcionamiento es similar a la orden TRACE, donde no se dibuja directamente el último tramo cada vez que se pincha, sino el anteúltimo.

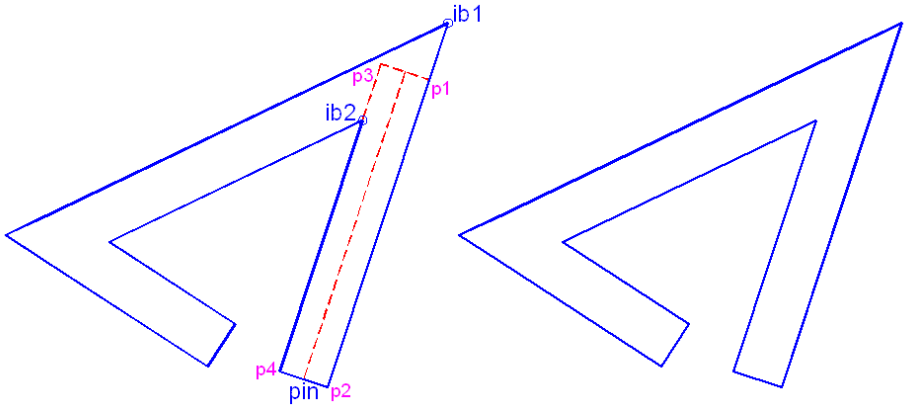
Se llama, por último, a la función "act" para actualizar las variables y quedar en disposición de volver a entrar en el ciclo para seguir calculando mas paredes.

En la rama del "No", una vez que el usuario ha introducido "Return", se da por terminada la generación de paredes. Como se tienen ya almacenados los valores de la última pared, se dibujan directamente las dos líneas correspondientes y la última línea de cierre de esta pared.



Obtención de puntos de la tercera pared y trazado de la segunda.

Se cambia el valor de "n" a nil para provocar la ruptura del ciclo (WHILE n…). Realmente el valor de "n", que es el número de paredes generadas, se va a perder. Hubiera resultado mas lógico conservarlo y utilizar una variable de control diferente "ctr". Pero, puesto que no interesa demasiado conocer el número total de paredes, se ha aprovechado esa variable para emplearla como control.



Trazado de la tercera pared y resultado final.

Una vez roto el control, la función "ciclo" termina y con ella el programe

Variables de entrada:

pin Punto inicial de la nueva pared.
e Espesor homogéneo de todas las paredes.
P1 Primer vértice de la primera pared.
p2 Segundo vértice.
p3 Tercer vértice.
p4 Cuarto vértice.
n Número de pared y control del ciclo.

Variables locales:

pf Punto final de la última pared.
p5 Primer vértice de la última pared.
p6 Segundo vértice.
p7 Tercer vértice.
p8 Cuarto vértice.
i1 Primer punto de intersección.
i2 Segundo punto de intersección.
ib1 Último primer punto de intersección.
ib2 Último segundo punto de intersección.

Variables de salida: ninguna.

7 Capitulo 7: Utilización de funciones del tipo GET

Se ha visto ya el empleo habitual de las funciones del tipo GET…para solicitar del usuario los datos necesarios en los programas de AutoLISP. Es muy frecuente establecer una serie de condiciones a la hora de que el programa acepte esos datos. Hasta ahora se incluían ciclos de control con el comando WH1LE, que chequeaban el valor introducido. Pero es más sencillo establecer las condiciones con comandos específicos como los estudiados en este capítulo.

7.1 (INITGET [<bits>] [<cad>]) Modos de las funciones GET

Este comando especifica el modo en que va a operar el siguiente comando del tipo GET... (excepto GETSTRING y GETVAR). El comando devuelve siempre nil. El valor <bits> es un número entero, cuyos valorcs posibles son los siguientes:

Bits	Modo
1	No admite valores nulos.
2	No admite valor cero.
4	No admite valores negativos.
8	No verifica límites, aunque estén activados.
16	Devuelve puntos 3D en vez de 2D
32	Dibuja la línea o el rectángulo clásticos, con línea de trazos en vez de continua

A la hora de indicar el modo se pueden sumar varios de ellos. Por ejemplo, Si no se desea admitir ni valores nulos ni cero ni negativos y además devolver puntos 3D, el modo resultante sería:

1+2+4+16 = 23

La función de este modo es fijar las condiciones en que va a operar el siguiente comando GET... Si los datos introducidos por el usuario como respuesta a la solicitud del comando GET... no cumplen la condición o condiciones especificadas en el modo, se rechazan esos datos y se visualiza un mensaje que solicita al usuario que pruebe otra vez.

Command:

(INITGET (+1 2 4))
(SETQ coef (GETINT "Intr. coeficiente de ecuación:))

En el ejemplo se pide al usuario mediante un comando GETINT que introduzca el coeficiente de una ecuación. Si el usuario indica un número que no es

entero, el propio comando GETINT lo rechaza pidiendo que pruebe otra vez. Pero si no se especifica un modo con INITGET, el comando GETINT admite un valor Ø o entero negativo o también nulo (si se pulsa Return).

Al establecer con INITGET un modo (+ 1 2 4), el comando GETINT no admite valores Ø ni negativos ni nulos y pide en todos los casos al usuario que pruebe otra vez. Esta es la forma de asegurar que el dato aceptado por GETINT es el correcto.

El modo establecido por INITGET sólo sirve para el siguiente comando GET... Si se vuelve a utilizar un segundo comando GET..., habría que establecer antes el modo con un nuevo comando INITGET.

Los modos establecidos con INITGET sólo se tienen en cuenta para los comandos GET... con los cuales tienen sentido. Así, por ejemplo, no tiene sentido establecer un modo que no permita valores negativos (modo 4) con un comando GETPOINT, puesto que éste devuelve un punto como una lista de dos o tres elementos, y las listas no pueden ser negativas.

El siguiente cuadro muestra los modos que tienen sentido con los diferentes comandos GET...

Comando	Modos que tienen sentido					
	1	2	4	8	16	32
GETINT	SI	SI	SI			
GETREAL	SI	SI	SI			
GETDIST	SI	SI	SI		SI	SI
GETANGLE	SI	SI				SI
GETORIENT	SI	SI				SI
GETPOINT	SI			SI	SI	SI
GETCORNER	SI			SI	SI	SI
GETKWORD	SI					
GETSTRING						
GETVAR						

En la Versión 11 de AutoLISP, los códigos para los modos sufren alguna modificación (véase el apartado 17.3.1 del Capítulo 17).

El argumento opcional <cad> es una cadena de texto que define una serie de respuestas alternativas a los comandos del tipo GET... Si el usuario introduce una respuesta incorrecta a la solicitud del comando GET…, en principio éste daría error y el programa en AutoLISP quedaría abortado. En cambio, si se especifican las respuestas alternativas con INITGET, el comando GET...

verifica si la respuesta del usuario está incluida entre las alternativas, y si es así, devuelve esa cadena en vez de producir error.

La lista de respuestas alternativas es una cadena de texto en que cada respuesta está separada de la siguiente por un espacio en blanco.

Por ejemplo, se dispone de un programa en AutoLISP que dibuja muros con espesor y solicita del usuario mediante GETPOINT que introduzca el nuevo punto final de cada muro. Se desea disponer de una opción "R" para revocar el último tramo de muro. “C” para cerrar el último muro con el primero y "A" para abrir el muro.

```
( INITGET 1 "R C A" )
( SETQ pt ( GETPOINT "intr.. nuevo punto" ) )
```

En este caso el comando GETPOINT no admite un valor nulo (introducir RETURN), pero si admite que el usuario introduzca las respuestas alternativas “R", “C” o “A". Estas pueden ir también en minúsculas (valdrían "r", "c" o "a')
Se pueden aceptar abreviaturas indicándolas en mayúsculas en INITGET y el resto en minúsculas.

```
( INITGET 1 "Revoca" )
```

admitiría "R" o “r”, o la palabra completa "Revoca" como alternativas.

```
( INITGET 1 "REVoca")
```

no admitiría "R" ni "RE", sino solamente "REV" como abreviatura valida (puede estar tanto en mayúsculas como minúsculas).

La abreviatura es, pues, el mínimo número de caracteres en que debe coincidir la respuesta del usuario con la alternativa indicada en INITGET. A partir de ahí se admiten más caracteres por parte del usuario hasta la longitud de la respuesta alternativa indicada en INITGET.

```
( INITGET 1 "Revo" )
```

admitiría "r", "re", "rev" o "revo" (mayúsculas o minúsculas), pero no "revoca", ni "revocar", ni "redo", etcétera.

También es posible indicar la abreviatura junto a la respuesta completa, separada por una coma. (INITGET 1 "REVO,R") equivale a (INITGET 1 "Revo")

7.2 (GETKWORD [<mens>]) Permitir opciones o palabras claves.

Este comando permite solicitar del usuario que introduzca una serie de opciones o palabras clave. La lista de respuestas posibles admitidas se debe especificar previamente con el comando INITGET, visto en el apartado anterior. El usuario debe introducir una de estas respuestas posibles o en caso contrario

GETKWORD solicita que pruebe otra vez.

```
( INITGET 1 "Si No")
( SETQ r ( GETKWORD "Cerrar muro Si/No?: " ) )
```

En este caso. GETKWORD admitiría las respuestas "s", "si", "n" o "no" (mayúsculas o minúsculas), pero no admitiría ninguna otra.

El comando GETKWORD siempre devuelve una cadena de texto, que es una de las respuestas posibles indicadas en INITGET. Si no se ha especificado ninguna opción, devuelve nil.

Hay que tener en cuenta que GETKWORD devuelve la cadena tal como está indicada en INITGET. Así, en el ejemplo, acepta como respuesta "s" y devuelve la cadena correspondiente que es "Si", pero no devuelve "SI" ni "si", porque no están escritas así en INITGET.

7.3 (GETENV <nom-var>» Valor de variable de retorno.

Este comando devuelve una cadena de texto (entre comillas), que es el valor actual atribuido a la variable entorno indicada en <nom-var>. Esta variable habrá que indicarla como un texto, entre comillas.

(GETENV "acadfreeram") podría devolver "24"
(GETENV "lispheap") podría devolver "35ØØØ"

Si la variable no existe o no es encontrada, GETENV devuelve nil.

7.4 Ejemplo 1: Muro con espesor al interior o exterior

Este programa es una variante del Ejemplo 1 del capitulo anterior. En aquel caso, los puntos introducidos por el usuario se median sobre la línea media del muro. Pero, por ejemplo, en las fachadas de los edificios es frecuente que el usuario conozca los puntos del contorno exterior y desee, por tanto, especificar el espesor hacia el interior. En algunos tabiques interiores puede interesar lo contrario.

Para posibilitar esas opciones, en este ejemplo se añade al programa del capitulo anterior la opción de que el usuario decida qué línea va a utilizar para definir el muro: la línea media, la exterior o la interior.

7.4.1 Listado del programa

```
( DEFUN inic ( )
  ( INITGET 6 "Interior Exterior Medio" )
  ( IF (SETQ op (GETKWORD "Interior/Exterior<medio>: " ) )
    ( )
    ( SETQ op "Medio" )
  )
```

```
  (SETQ pin ( GETPOINT "Primer punto: ") )
  ( TERPRI )
  ( SETQ e ( GETDIST pin "Espesor: " ) )
  ( TERPRI )
  ( SETQ pf ( GETPOINT pin "Segundo punto: " ) )
  ( TERPRI )
  ( SETQ ang ( ANGLE pin pf ) )
)

( DEFUN par ( / p1 p2 p3 p4 )
  (IF ( = op "Medio" )
    ( PROGN
      ( SETQ p1 ( POLAR pin ( + ang ( / PI 2 ) ) ( / e 2 ) ) )
      ( SETQ p3 ( POLAR p1 ( - ang ( / PI 2 ) ) e ) )
      ( SETQ p2 ( POLAR pf ( + ang ( / PI 2 ) ) ( / e 2 ) ) )
      ( SETQ p4 ( POLAR p2 ( - ang ( / PI 2 ) ) e ) )
    )
  )
  ( IF ( = op "Exterior" )
    ( PROGN
      ( SETQ p1 pin)
      ( SETQ p3 ( POLAR p1 ( - ang ( / PI 2 ) ) e ) )
      ( SETQ p2 pf)
      ( SETQ p4 ( POLAR p2 ( - ang ( / PI 2 ) ) e ) )
    )
  )
  ( IF ( = op "Interior" )
    (PROGN
      ( SETQ p3 pin )
      ( SETQ p1 ( POLAR p3 ( + ang ( / PI 2 ) ) e ) )
      ( SETQ p4 pf)
      ( SETQ p2 ( POLAR p4 ( + ang ( / PI 2 ) ) e ) )
    )
  )
  ( COMMAND "line" p1 p2 "" )
  ( COMMAND "line" p3 p4 "" )
  ( COMMAND "line" p1 p3 "" )
  ( COMMAND "line" p2 p4 "" )
)

( DEFUN c:pared ( / op pin pf e ang)
  ;( SETVAR "biipmode" 0)
  ;( SETVAR "cmdecho" 0)
  ( inic )
  ( par )
  ;( SETVAR "blipmode" 1 )
  ;( SETVAR "cmdecho" 1)
)
```

7.4.2 Observaciones

En este programa se observa un mecanismo muy habitual a la hora de proponer varias opciones para que el usuario elija una: mostrar una de las opciones entre paréntesis como opción por defecto. De esta manera si el usuario responde con RETURN a la solicitud, el programa toma la opción por defecto. La manera de conseguir esto es muy sencilla, como se mostrará al explicar la función “inic" donde se encuentra incluida.

Una vez seleccionada por el usuario la opción deseada entre las posibles, ésta va a condicionar el funcionamiento del resto del programa. En este caso el ejemplo es muy sencillo. Pero en casos más complejos, si se parte de un programa ya desarrollado y se pretende añadirle la posibilidad de permitir varias opciones, se trata siempre de aprovechar la disposición estructurada del programa y hacer que las modificaciones resulten mínimas.

7.4.3 Explicación del programa

Lo mismo que el programa del capítulo anterior en el que está basado, define una nueva orden de AutoCAl) y dos funciones de usuario.

• Función **c:pared**

Añade una nueva orden "pared" a AutoCAD. Como es habitual, desactiva las marcas auxiliares y el eco de órdenes y llama a las dos funciones "inic" y *'dib". Se definen como variables locales el resto de variables que no se han definido como tales en las funciones de usuario. De esta manera, no queda ninguna variable permanentemente almacenada en la memoria al final del programa.

• Función **inic**

Lo primero que solicita es la opción deseada por el usuario para el trazado de la pared. Las tres opciones propuestas son:

-Exterior.
- Interior.
-Medio.

La opción se va a solicitar mediante el comando de AutoLISP GETKWORD, que visualizará el mensaje:

Interior/Exterior<medio>:

El mensaje se ha realizado de manera similar a los mostrados en las opciones de AutoCAD: la inicial de las opciones en mayúscula y el resto en minúsculas. De esta forma el usuario sabe que "1" y "E" son abreviaturas que sirven para seleccionar la opción deseada. Además se le muestra la opción por defecto <medio>, que será aceptada con un RETURN como respuesta.

Previamente, antes de GETKWORD es preciso especificar una lista de respuestas posibles admitidas, con el comando INITGET.

(INITGET 6 "Interior Exterior Medio")

Hay que incluir como respuesta posible también "Medio", por si el usuario introduce esa opción con sus letras en vez, de RETURN. Las mayúsculas son las abreviaturas de cada una de las opciones. Además de las abreviaturas, el usuario puede introducir más letras de cada opción como respuesta al mensaje de GETWORD.

El comando GETKWORD devuelve una cadena de texto que coincide con la opción correspondiente, tal como aparece en INITGET. Esta cadena de texto es la que se almacena en la variable “op”. Así, por ejemplo, son respuestas permitidas:

"I", "int", "inter", "interior"…….

siendo indiferente introducir las respuestas en mayúsculas o minúsculas. En lodos los casos, GETKWORD devolverá exactamente la cadena de texto "Interior".

De la misma manera, las respuestas posibles;

"e", "ext", "exter", "exterior”…..

devolverán en todos los casos la cadena de texto "Exterior".

Para la opción por defecto "medio" se emplea el mecanismo:

(IF (SETQ op (GETKWORD "Interior/Exterior<medio>: ")) ( )
(SETQ op "Medio"))

La asignación de la respuesta de GETKWORD a la variable "op" se emplea como condición de IF. Si el usuario introduce RETURN como respuesta, GETKWORD devuelve nil y por tanto (SETQ op.…), también devuelve nil. En ese caso la condición no se cumple y se ejecuta la rama del NO de la alternativa que en este caso es (SETQ op "Medio").

Es decir, se hace que hace que la variable "op" almacene la misma cadena de texto que si el usuario hubiese introducido la opción con letras:

"m", "me", "med", "medi" o "medio"

que son las respuestas posibles.

Si el usuario no introduce RETURN, el comando GETKWORD no devuelve nil y entonces la condición se cumple. Se ejecuta, por tanto, la rama del SI de la alternativa que en este caso es una lista vacía.

El resultado de este proceso es que la variable "op" solamente puede almacenar tres cadenas de texto:

"Interior", "Exterior" y "Medio"

Esos serán los tres valores que habrá que chequear en el resto del programa.

Por lo demás, la función "inic" no se ha modificado respecto a la versión anterior del programa y solicita sucesivamente el primer punto, el espesor y el punto final de la pared.

Variables de entrada: ninguna.

Variables de entrada: ninguna.

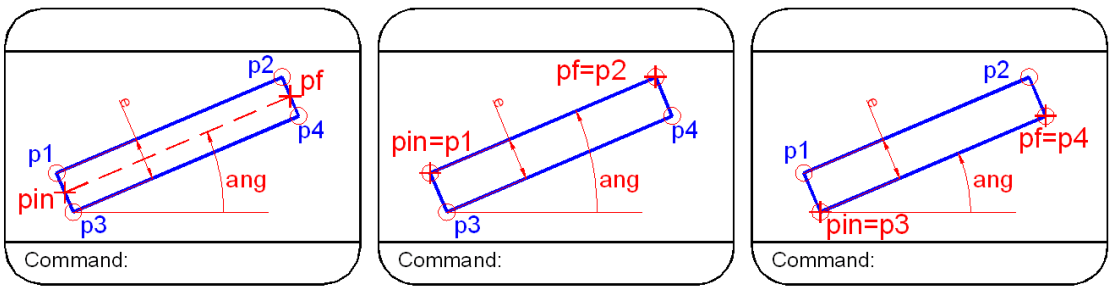
Variables de salida:

op Opción de trazado del muro.
pin Punto inicial de línea media del muro
pf Punto final de línea media del muro.
e Espesor del muro.
ang Ángulo de la línea media del muro.

• Función **par**

Calcula los cuatro vértices del muro o pared y lo dibuja mediante cuatro líneas. El cálculo depende de la opción de trazado del muro seleccionada (Figura 7.1). Si la variable "op" almacena la cadena de texto "Medio", el muro se construye con la mitad del espesor hacia un lado y la mitad hacia el otro. Funciona de manera idéntica a lo explicado en la versión del capítulo anterior. Al cumplirse la condición de IF se ejecuta el PROGN que está en la rama del SI. La rama del NO se encuentra vacía.

Si la variable "op" contiene el texto "Exterior", el muro se construye con el espesor hacia dentro. La línea entre "pin" y “pf” es la línea exterior. Para no alterar demasiado el programa original, se mantienen las cuatro variables de las cuatro esquinas "p1" a "p4" En esta opción "p1' coincide con pin y "p2" con "pf". Las otras dos esquinas, "p3" y "p4", se calculan llevando ángulo de 9Ø grados (Pl/2



Trazado del muro según las tres opciones: Medio, Exterior o Interior. "

radianes) en dirección perpendicular a las anteriores. Todas estas operaciones

se encuentran agrupadas mediante un PROGN en la rama del SI de la alternativa.

Si la variable "op" almacena la cadena de texto "Interior", significa que el usuario ha seleccionado esa opción. El muro se construye entonces hacia afuera y la línea entre "pin" y "pf" es la línea interior. Eso quiere decir que "pin" coincide con "p3" y "pf" con "p4".

Las dos esquinas restantes se obtienen como siempre, llevando simplemente el espesor en la otra dirección. Lo mismo que antes, estas operaciones se agrupan con un PROGN en la rama del SI.

El resultado es obtener las cuatro esquinas, con lo que ya se puede dibujar el muro mediante cuatro COMMAND que llamen a la orden de AutoCAD "línea".

Cuando el número de alternativas es mayor de dos, se puede utilizar el comando COND, estudiado en el Capítulo 5. Sin embargo, la programación estructurada aconseja alternativas de sólo dos ramas, aun cuando sea preciso emplear varios IF para explorar todas las posibilidades. Ello va en favor de la estructura modular del programa.

Variables de entrada:op Opción de trazado del muro.Pin Punto inicial de línea media del muro,pf Punto final de línea media del muro.e Espesor del muro.Ang Ángulo de la línea media del muro.

Variables locales:

p1 Primer vértice del muro.
p2 Segundo vértice.
p3 Tercer vértice.
p4 Cuarto vértice.

Variables de salida: ninguna.

8 Capitulo 8: Manipulación de cadenas textuales

Se explican en este capítulo los comandos de AutoLISP relativos a la manipulación de cadenas de texto. Hay que tener en cuenta que resulta muy habitual visualizar mensajes cada vez que un programa solicita del usuario los datos requeridos. En estos casos se suele componer el mensaje previamente como una cadena de texto, añadiendo si es necesario opciones por defecto.

Además determinadas especificaciones del dibujo se encuentran almacenadas en la Base de Datos como cadenas de texto: nombres de capas, de estilos de texto, tipos de línea, etc. Por otro lado, los textos en AutoCAD suelen requerir una engorrosa manipulación cuando se pretende cambiar su contenido o propiedades. Por estos motivos resulta frecuente utilizar los comandos de AutoLISP que se explican a continuación.

8.1 (STRCASE <cad>[<expr>]) Cambio de mayúsculas o minúsculas.

Este comando toma la cadena de caracteres indicada en <cad> y la convierte en mayúsculas o minúsculas dependiendo del valor de <expr>. Al final devuelve la nueva cadena resultante de esa conversión.

Si la expresión <expr> falta o su valor es "nil", la cadena <cad> se convierte a mayúsculas. Si <expr> tiene cualquier otro valor diferente de "nil", la cadena se convierte en minúsculas.

Este comando es útil para chequear una entrada de texto, por ejemplo del tipo de "Si" "No", en que serian validas todas las posibles combinaciones de mayúsculas y minúsculas: "SI", "si", "NO", "no" o incluso "S", "s", "N", "n".

Command:
(SETQ tx (GETSTRING "Introducir <Si> o <No>: "))
(SETQ cond (STRCASE tx nil))

Ahora en la variable "cond" está almacenado el texto en mayúsculas introducido, por lo que bastará chequear solamente las posibilidades "SI", "S", "NO" o "N".

8.2 (STRCAT <cad1> <cad2>...) Concatenar cadenas de texto

Este comando devuelve una cadena que es la concatenación de todas las cadenas <cad1>, <cad2>, etc., indicadas. Los argumentos tienen que ser todos cadenas, pues en caso contrario se produce un error.

Command:

(SETQ cad1 "Ejemplo de ")
(SETQ cad2 "concatenación")
(SETQ cad3 " de cadenas.")
(SETQ cadt (STRCAT cad1 cad2 cad3))

"Ejemplo de concatenación de cadenas."

Una utilidad muy interesante es cuando el mensaje a visualizar depende del contenido de determinadas variables de texto. Por ejemplo, se pretende insertar un bloque cuyo nombre se pide previamente al usuario, y después se solicita el punto de inserción visualizando en el mensaje el nombre del bloque:

Command:
(SETQ nbl (GETSTRING "Nombre del bloque: "))
(SETQ mens (STRCAT "Punto de inserción del bloque " nbl " : "))
(SETQ pins (GETPOINT mens))

La variable “nbl” almacena el valor textual (cadena) del nombre del bloque. Después en la variar “mens” se concatena un mensaje en que se incluye el nombre del bloque. Este mensaje se visualiza al solicitar un punto de inserción para almacenarlo en "pins".

Si, por ejemplo, el nombre del bloque contenido en "nbl" es "resist", el mensaje visualizado sería:

Punto de inserción del bloque resist:

En el caso de que la variable a concatenar fuera numérica, seria preciso convertirla previamente mediante el comando ITOA, que se estudiará en este mismo capitulo.

8.3 (ATOI <cad>) Conversión de cadena en entero

Este comando convierte la cadena especificada en <cad> en un número entero. Si <cad> tiene decimales los trunca.

Command:
(SETQ cad "38.7")
(SETQ num (ATOI cad)
38

La variable "num" almacenaría en el ejemplo el valor numérico entero de 38.

8.4 8.4. (ATOF <cad>) Conversión de cadena en número real

Convierte la cadena indicada <cad> en un número real Admite el guión inicial de la cadena de texto convirtiéndolo en signo "menos" del número real obtenido. Si la cadena de texto no contiene caracteres numéricos, ATOF devuelve el valor Ø.Ø. Si la cadena contiene al principio caracteres numéricos

y después no, ATOE solamente convierte los numéricos, desechando el resto.

Todas estas posibilidades se contemplan en los siguientes ejemplos:

(ATOF "35.76")	devuelve	35.76
(ATOF "-56")	devuelve	-56.Ø
(ATOF "35,26")	devuelve	35.Ø
(ATOF "24 metros")	devuelve	24.Ø
(ATOF "hola")	devuelve	Ø.Ø

8.5 (ITOA <entero>) Conversión de entero a cadena.

Convierte el valor entero indicado en una cadena de texto. Reconoce el signo "menos", si existe, y lo convierte en guión al principio de la cadena. Si el <entero> especificado tiene decimales o no es numérico, el comando produce un error.

(ITOA 27)	devuelve "27"
(ITOA -35)	devuelve "-35"
(XTOA 45,7)	produce un error
(ITOA 35m)	produce un error

Es útil este comando cuando el mensaje a visualizar depende del contenido de determinadas variables numéricas. Por ejemplo, en el programa del Capitulo 4, para generar líneas de texto con cualquier valor de espaciado (Ejemplo 3), el mensaje para solicitar los textos de cada línea debía visualizar el número de línea correspondiente en cada caso.

Si la variable "n" contiene el valor del número de línea y se pretende almacenar en una variable “mens” el mensaje a visualizar, se produce un error:

Command:
(SETQ mens (STRCAT "Introducir texto de línea" n))
error: bad argument type

El valor de la variable "n" es numérico y no un texto. Para poder concatenarlo con el resto del mensaje, hay que convertirlo previamente en cadena. Si el valor de "n" es un número entero (por ejemplo, 3), se usaría el comando ITOA.

Command:
(SETQ n 3)
(SETQ mens (STRCAT "Coeficiente de grado " (ITOA n)))
"Coeficiente de grado 3"

A continuación bastaría visualizar el mensaje almacenado en la variable "mens" al solicitar el texto de la línea de que se trate con GETSTRING:

(SETQ tx (GETSTRING T mens))

El comando RTOS para convertir un número real en cadena, según un formato especificado, se ha explicado en el Capitulo 6.

8.6 (SUBSTR <cad> <prin> [<long>] Subcadena de una cadena.

Este comando toma la cadena indicada en <cad> y devuelve una subcadena a partir del número de carácter indicado en <prin> (principio de la subcadena), y con una longitud de caracteres <long>. Si no se indica <long>, la subcadena se toma desde <prin> hasta el final.

Tanto el principio de la subcadena <prin> como la longitud de la misma <long> deben ser números enteros positivos.

Command:
(SETQ cad "Buenos dias")
(SUBSTR cad 2 3) devuleve “uen”
(SUBSTR cad 1 1) devuelve "B"
(SUBSTR cad 7) devuelve “ dias”
(SUBSTR cad 7 1) devuelve “ “

8.7 (STRLEN <cad) Longitud de cadena.

Este comando devuelve la longitud de la cadena <cad> indicada. B valor de la longitud es un número entero que expresa el número de caracteres totales de la cadena.

Command:
(SETQ cad "Buenos Días")
(STRLEN cad)
11

En el ejemplo, la longitud de la cadena es de 11 caracteres. Una cadena vacía devolvería una longitud de Ø caracteres:

(STRLEN " ") devuelve Ø

8.8 (ASCII <cad>) Código del primer carácter de cadena.

Este comando devuelve un número entero que es el código ASCII del primer carácter de la cadena indicada <cad>.

(ASCII "6") devuelve 54
(ASCII "A") devuelve 65
(ASCII "hola") devuelve 1Ø4
(SETQ cad "Buenos días")
(ASCII cad) devuelve 66
(SETQ cad (GETSTRING))"RETURN" devuelve “ “
(ASCII cad) devuelve 0

En el último ejemplo, la variable "cad" almacena la cadena introducida mediante GETSTRING se introduce RETURN, lo que devuelve GETSTRING es la cadena vacía " ". El comando ASCII de la cadena vacía (ASCII " ") devuelve Ø.

8.9 (CHR <num>) Caracter correspondiente a código ASCII.

Este comando devuelve el carácter cuyo código ASCII coincide con el número entero <num> especificaddo. Representa la operación inversa del comando anterior.

(CHR 54) devuleve "6"
(CHR 104) devuelve "h"
(CHR 0) devuelve " "
(CHR 13) devuelve " \r "

Se observa en el último ejemplo que el carácter correspondiente al código 13 (que es el RETURN) es “\r".

El comando CHR siempre devuelve un carácter de texto, por tanto entre comillas.

8.10 (READ <cad>) Primera expresión de una cadena.

Este comando devuelve la primera expresión de la cadena indicada <cad>. Si la cadena no contiene paréntesis, es un texto con espacios en blanco, y por tanto READ devuelve el trozo de texto hasta el primer espacio en blanco (en general seria la primera palabra del texto).

Si la cadena contiene paréntesis, éstos son expresiones en AutoLISP, y por tanto READ devuelve la primera expresión. Se recuerda que los caracteres especiales que separan expresiones de AutoLISP son:

( ) ‘ “ ; (espacio en blanco)

A continuación se ofrecen unos ejemplos:

(READ "Buenos días") devuelve BUENOS
(READ "Hola; buenas") devuelve HOLA
(READ "Estoy (mas o menos)bien" devuelve ESTOY

Hay un aspecto muy importante a destacar: READ examina la cadena de texto, pero analiza su contenido como si fueran expresiones en AutoLISP y devuelve no una cadena de texto, sino una expresión de AutoLISP Por eso, en los ejemplos, el resultado está siempre en mayúsculas.

Entonces la utilidad real del comando READ no es analizar contenidos textuales, sino expresiones de AutoLISP almacenadas en cadenas de texto.

Por ejemplo:

(READ " (setq x 5) ")	devuelve (SETQ X 5)
(READ " (setq y 3) (setq z 7) ")	devuelve (SETQ Y 3)
(READ " (setq y (* 5 3)) (setqz 2) ")	devuelve (SETQ Y (* 5 3))

El comando devuelve siempre la primera expresión de AutoLISP contenida en la cadena de texto. Esa expresión se puede evaluar como cualquier otra de AutoLISP, con el comando EVAL:

(EVAL (READ " (setqx 5) ")) devuelve 5

Es equivalente a hacer:

(EVAL (SETQ x 5))

Y esto sería lo misino que haber tecleado directamente desde la línea de órdenes:

Command: ***(SETQ x 5)***

El resultado es en todos los casos crear una variable "x", si es que no existía, y almacenar en ella el valor 5.

La combinación de los comandos READ y EVAL ofrece una importante posibilidad: cargar en la memoria expresiones de AutoLISP sin tener que pasar por la etapa de escribirlas en un archivo de texto y cargarlo después con LOAD.

Por ejemplo, se pretende desarrollar un programa que dibuje la curva de una función y = F(x). La expresión de esta función la introduce el usuario con comandos de AutoUSP y de acuerdo con un formato directamente legible por AutoLISP. Si la función es

$$Y = (\text{sen}^2 (X) - \cos^2 (X)) / (X^2 - 1)$$

el usuario introduciría:

(/ (- (EXPT (SIN X) 2) (EXPT (COS x) 2)) (- (EXPT X 2) 1))

Se almacena esa expresión como una cadena de texto en una variable, utilizando GETSTRING:

(SETQ cad (GETSTRING T))

Ahora se quiere definir una función de usuario llamada, por ejemplo, "fun", que sirve para obtener las coordenadas Y para cada valor de ia variable X, según la función cuya expresión esta almacenada en "cad". La expresión de AutoLISP para crear esa función de usuario seria:

(DEFUN fun (x).....la expresión contenida en "cad" ...)

Entonces a la expresión almacenada en "cad" hay que añadir "(DEFUN fun (x)" al principio y")" al final. Esto se hace con STRCAT:

(DEFUN cad2 (STRCAT " (DEFUN fun (x)" cad ") "))

Ahora "cad2" almacena la expresión completa en AutoLISP para definir la función de usuario “fun”. Para cargarla en la memoria bastaría hacer:

(EVAL (READ cad2)) devuelve FUN

Devuelve el nombre de la función que se acaba de cargar en la memoria.

8.11 Ejemplo 1: Dibujar el trazado de cualquier función.

Una aplicación muy sencilla del empleo de los comandos de AutoLISP, READ y EVAL es el trazado de la curva de una función cualquiera Y = F(x)

La expresión de la curva la introduce el propio usuario con el formato de una expresión de AutoLISP y el programa evalúa dicha expresión y dibuja la curva mediante tramos de línea recta de acuerdo con la precisión indicada.

Su funcionamiento está basado en el programa para dibujar curvas senoidales del Capitulo 5 (Ejemplo 1). Allí la expresión de la función la daba el propio programa como una suma de tres senos. En el presente ejemplo se generaliza para cualquier tipo de función, con las lógicas condiciones de que sea continua y finita en el intervalo que se especifica para su trazado.

8.11.1 Listado del programa

```
( DEFUN intr (/ mens fun fundef xi y1)
  (IF (= funant nil)
    (SETQ funant "")
  )
  (SETQ mens (STRCAT "Expresión de la función en X : <" funant ">
: "))
  (IF (= " " (SETQ fun (GETSTRING T mens)))
    (SETQ fun funant)
  )
  (TERPRI)
  (SETQ fundef (STRCAT " ( defun curvaf ( x ) " fun " ) "))
  (EVAL (READ fundef))
  (SETQ xi (GETREAL "Inicio curva en X: "))
  (TERPRI)
  (SETQ x1 xi)
  (SETQ y1 (curvaf x1))
  (SETQ p1 (LIST x1 y1))
  (SETQ funant fun)
  (SETQ xf (GETREAL "Final curva en X: "))
```

```
  (TERPRI)
  (WHILE (<= xf xi)
    (PROMPT "Debe ser un valor mayor que inicio")
    (TERPRI)
    (SETQ xf (GETREAL "Final curva en X: "))
    (TERPRI)
  )
  (INITGET 7)
  (SETQ prx (GETREAL "Precisión en X: "))
  (TERPRI)
  (SETQ n (FIX (/ (- xf xi) prx)))
)


( DEFUN dib ( / x2 y2 p2 )
  ( REPEAT n
    ( SETQ x2 ( + x1 prx ) )
    ( SETQ y2 ( curvaf x2 ) )
    ( SETQ p2 ( LIST x2 y2 ) )
    ( COMMAND "line" p1 p2 "" )
    ( SETQ x1 x2 p1 p2 )
  )
)

( DEFUN ult ( / p2 yf )
  ( SETQ yf ( curvaf xf ) )
  ( SETQ p2 ( LIST xf yf ) )
  (COMMAND "line" p1 p2 "")
)

( DEFUN c:curva ( / xi xf prx p1 n x1 )
;;;  ( SETVAR "blipmode" 0 )
;;;  ( SETVAR "cmdecho" 0 )
  ( intr )
  ( dib )
  ( ult )
;;;  ( SETVAR "blipmode" 1 )
;;;  ( SETVAR "cmdecho" 1 )
)
```

Puede ingresar algo como esto:

```
(+ (* 2 x x)(- (* 12 x)) 3)
```

Y obtener un reultado como este:

8.11.2 Observaciones

Las mismas consideraciones expuestas para el programa de trazado de curvas senoidales (Ejemplo 1 del Capitulo 5) sirven aquí. Como se ha dicho, la diferencia radica en que el usuario introduce la expresión de la función.

Este es precisamente el punto más delicado, pues la expresión debe ser indicada en el formato de AutoLISP. Si el usuario comete algún error, el programa quedara abortado. Es preciso controlar, pues, este aspecto. El tratamiento de errores en AutoLISP se estudiará más adelante. Pero conviene detectar si la expresión es correcta antes de que el programa solicite el resto de valores. Esto se hace provocando la evaluación de esa expresión, tal como se explica más adelante al hablar de la función "intr".

8.11.3 Explicación del programa

Se define una nueva orden de AutoCAD llamada "curva" y tres funciones más de usuario.

• Función **c:curva**

Desactiva marcas auxiliares y eco de órdenes y llama sucesivamente a las restantes funciones de usuario.

• Función **intr**

Como siempre, su cometido es solicitar los datos requeridos para la ejecución del programa.

En este caso e! primero y fundamental dato es la expresión de la función cuya curva se pretende dibujar, El programa la solicita mediante un comando GETSTRING con el parámetro T para aceptar también espacios en blanco. La expresión queda almacenada en la variable "fun" como una cadena de texto.

NOTA: Se insiste una vez mas en que el usuario debe introducir la expresión en el formato de AutoLiSP. Así, por ejemplo, si se pretende trazar la curva del polinomio.

$$Y = 5X^{2} - 7X + 3$$

el usuario deberá introducir la expresión

(+ (* 5 x x) (-(* 7 x)) 3

u otra equivalente.

Lo mismo con cualquier otro tipo de función, que podrá contener exponenciales, funciones trigonométricas, logarítmicas, etc. Los comandos de AutoLISP correspondientes a esas operaciones se han explicado en el Capítulo 5.

Es muy normal que el usuario pruebe con una misma función para trazar diferentes intervalos o precisiones diversas. Si la función es compleja, resultaría muy engorroso tener que introducirla completa cada vez que se modifica el intervalo o la precisión.

Por eso el programa guarda la última expresión introducida en una variable “funant”. Esa expresión se concatena mediante STRCAT con el mensaje de solicitud en la variable "mens". Se ofrece como opción por defecto para que el usuario no tenga que repetirla.

Si el usuario introduce RETURN como aceptación de esa opción por defecto, el comando GETSTRING devuelve la cadena vacía " ". Luego (SETQ fun (GETSTRING...)) también devuelve nil. Se establece una alternativa para asignar el valor de “funant” con la función por defecto a la variable "fun”, en el caso de que se cumpla esa condición (rama del SI). La rama del NO de la alternativa es una cadena vacía.

Hay que considerar un hecho: la primera vez que se utiliza el programa en el Editor de Dibujo, la variable "funant" se encuentra a nil y esto puede provocar un error. Por eso se establece una alternativa previa que detecta esta posibilidad y asigna a "funant" la cadena vacía " ".

Una vez almacenada la expresión de la función en "fun", el programa debe definir una función a ella mediante DEFUN, con objeto de que pueda ser llamada para calcular los puntos de la curva.

Para ello se forma una cadena de texto mediante STRCAT, añadiendo por delante la cadena "(DEFUN, curvaf (x)" y por detrás el carácter de cierre de paréntesis ")". El resultado de esa concatenación de cadenas se guarda en la variable "fundef".

Por ejemplo, si la variable “fun" de la expresión introducida por el usuario almacena la cadena de texto

" (+ (* 5 x x) (- (* 7 x)) 3) "

la variable "fundef" almacenará la cadena

```
" ( DEFUN curvaf(x)  ( + ( * 5 xx )  ( - ( * 7 x ) ) 3 ) ) "
```

que es precisamente la definición de una función llamada "curvaf" que contiene la expresión que se pretende dibujar.

De momento sólo es una cadena de texto. Pero a continuación el programa hace:

```
( EVAL ( READ fundef ) )
```

El comando READ lee la cadena de texto almacenada en "fundef" y devuelve como resultado expresión en AutoLISP, la cual es evaluada mediante EVAL.

El resultado es que se ejecuta el comando DEFUN y la función "curvaf" definida queda cargada en la memoria para su posterior utilización en el resto del programa.

Si la definición de la función es incorrecta, el programa lo detectará sólo cuando se llame a dicha función. Por eso se cambia un poco el orden del programa respecto al mencionado de las curvas senoidales. En este caso se solicita el valor X inicial de la curva (variable "xi"), se asigna ese valor a la coordenada X del primer punto de la primera línea (variable "xl") y se calcula en seguida el punto "yl" correspondiente, para lo cual se llama a la función "curvaf" recién definida.

```
( SETQ y1 ( curvaf x1 ) )
```

Con esos valores se forma el punto "pl", que será el inicial del trazado de la curva. Aunque lo más lógico hubiera sido introducir primero todos los datos y calcular después el punto, se hace de esta manera para comprobar si la expresión se encuentra bien escrita. En caso de que no sea así, el programa provoca un error y queda abortado en su comienzo. De esta forma no hay que esperar a introducir los valores de precisión, inicio en X, etc., para saber si hay error.

Si todo es correcto, el programa almacena la expresión de la función en "funant" para que se visualice como opción por defecto la próxima vez y continúa solicitando el valor X del final del intervalo a trazar. Se mantiene el control con WHILE para que su valor sea mayor que el inicio.

A continuación viene la precisión en X. En el ejemplo de las curvas senoidales se establecía un control con WHILE para asegurar que ese valor es positivo. En este caso se utiliza el comando INITGET visto en el capítulo anterior.

```
( INITGET 7 )
```

El valor de 7 - 4 + 2 + 1 establece no permitir valores nulos ni Ø ni negativos.

Por último, la función calcula el número de tramos de línea a dibujar,

almacenándolo en la variable “n”. El comando FIX toma el valor entero del cociente:

$$\frac{(xf - xi)}{prx}$$

Es el número de tramos de línea completos a dibujar. Para dibujar el último "pico" o "resto" se utilizará la función "ult".

Variables de entrada: ninguna.

Variables locales:

mens Mensaje para la solicitud de función.
fun Cadena con expresión de función.
fundef Cadena con expresión para def. función.
xi Coordenada X de inicio de la curva.
Y1 Coordenada Y del primer punto de curva.

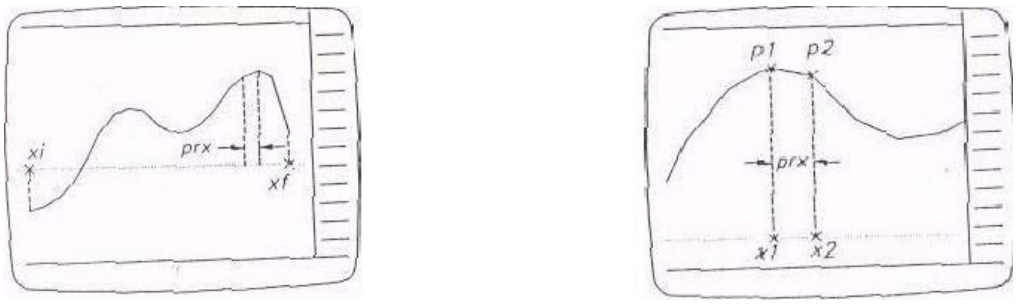
Variables de salida:

funant Expresión de la última función.
prx Intervalo de precisión en X.
x1 Coordenada X del primer punto de tramo.
p1 Primer punto del primer tramo.
xf Coordenada X de final de la curva.
n Número de tramos de línea completos.

• Función **dib**

Es la misma función explicada en el programa del trazado de curvas senoidales. Dibuja los tramos completos de línea que van a conformar el trazado de la curva deseado. Establece un ciclo repetitivo de "n" veces.

La primera vez se entra con un valor de punto inicial en "pl". Dentro del ciclo se calcula el punto final del tramo de línea, de acuerdo con el incremento marcado por la precisión "prx". Se dibuja el tramo de línea entre "pl" y "p2" y se actualiza la variable "p1" para que el punto final del presente tramo sea el inicial del nuevo tramo.



Variables y cálculo de un segmento para el trazado de una curva.

Se establecen las variables locales cuyos valores no van a ser utilizados

posteriormente. El último valor de "p1" con que se sale de la repetitiva va a ser utilizado por la función “ult”.

Para calcular la coordenada Y de cada punto de la curva existe dentro del ciclo una llamada a la función “curvaf”.

```
( SETQ y2 ( curvaf x2 ) )
```

En este punto el programa puede, quedar abortado si, como consecuencia de las características de la función especificada, tiene alguna discontinuidad en el intervalo señalado o tiende a infinito. En cualquier caso, observando el trozo de curva dibujado antes de generarse ese error, es posible observar gráficamente la tendencia de la función.

Variables de entrada:

n Número de tramos completos de línea.
x1 Coordenada X del punto inicial de curva.
prx Intervalo de precisión en X.
p1 Primer punto del primer tramo.

Variables locales:

x2 Coordenada X del punto final del tramo.
y2 Coordenada Y de ese mismo punto.
p2 Punto Final del tramo.

Variables de salida:

p1 Punto Final del último tramo, que será el inicial para dibujar el "resto".

• Función **ult**

Dibuja el último "resto" de la curva. Calcula la coordenada Y correspondiente a "xf” y forma el punto final "p2", dibujando la última línea.

Variables de entrada:

xf Coordenada X del punto final de curva.
p1 Punto inicial del "resto".

Variables locales:

yf Coordenada Y del punto final de la curva.
p2 Punto final de curva.

Variables de salida: ninguna.

• Función **curvaf**

Es en realidad una función más de usuario aunque el programador no la haya escrito en el programa, sino que se ha formado a partir de la expresión introducida por el usuario de dicho programa.

Su contenido es calcular las coordenadas Y correspondientes a cada valor de

X.

Variables de entrada:

x Coordenada X del punto de la curva cuya coordenada Y se quiere obtener.

Variables locales: ninguna.

Variables de salida: No existe ninguna, pero la evaluación de la propia función "curvaf(x)" devuelve un valor la coordenada Y buscada.

8.12 Ejemplo 2: Dibujar hélices con 3DPOL.

Otra aplicación sencilla del empleo de READ y EVAL es el trazado de hélices en 3D con la orden de AutoCAD "3dpol". Este tipo de entidad no permite la opción "Juntar" cuando se edita con "Editpol". Eso quiere decir que los tramos de la hélice no pueden generarse como líneas en una estructura repetitiva.

Si se pretende, por ejemplo, dibujar una hélice con tres "vueltas" y una precisión de 12 tramos rectos en cada vuelta, se necesitan un total de 36 tramos. Una vez que se llama a la orden "3dpol" con COMMAND, ésta no se puede interrumpir. Todos los 37 vértices deben estar definidos antes de llamar a la orden.

Pero la precisión y el número de "vueltas" lo fija el usuario. Por tanto, el programa debe ser capaz de generar variables para esos puntos y de calcular sus valores de acuerdo con los requerimientos del usuario.

Esto se consigue con READ y EVAL, como se verá a continuación.

8.12.1 Listado del programa

```
( DEFUN intr ( / paso)
  ( SETQ cen ( GETPOINT "centro de hélice: " ) )
  ( TERPRI )
  ( SETQ radin ( GETDIST cen "radio inicial! " ) )
  ( TERPRI )
  ( IF ( SETQ radfin ( GETDIST cen ( STRCAT "radio final <" ( RTOS
radin 2 2 )">: " ) ) )
    ( )
    ( SETQ radfin radin)
  )
  ( TERPRI )
  ( INITGET 7 )
  ( SETQ pv ( GETINT "precisión en cada vuelta: ") )
  ( TERPRI )
  ( INITGET 7 )
  ( SETQ nv ( GETINT "número de vueltas: ") )
  ( TERPRI )
  ( INITGET 1 )
```

```
  ( SETQ paso ( GETDIST cen "paso de hélice: ") )
  ( TERPRI )
  ( SETQ ang ( / ( * 2 PI) pv ) )
  ( SETQ dz ( / paso pv ) )
  ( SETQ drad ( / ( - radfin radin ) ( * pv nv ) ) )
)

( DEFUN ptos ( / expr n z0 z xy )
  ( SETQ p0 ( POLAR cen 0 radin) )
  ( SETQ z0 ( CADDR p0 ) )
  ( SETQ n 1 )
  ( REPEAT ( * pv nv )
    (SETQ z ( + z0 ( * dz n ) ) )
    ( SETQ xy (POLAR cen ( * ang n ) ( + radin ( * drad n ) ) ) )
    ( SETQ expr ( STRCAT "( setq p" ( ITOA n ) " ( list (car xy) ( cadr
xy ) z ) )" ) )
    ( EVAL ( READ expr ) )
    ( SETQ n ( + n 1 ) )
  )
)

( DEFUN dib ( / n expr )
  ( SETQ expr "p0  " )
  ( SETQ n 1 )
  ( REPEAT ( * pv nv )
    ( SETQ expr (STRCAT expr "p" ( ITOA n ) " " ) )
    ( SETQ n (+ n 1 ) )
  )
  (SETQ expr (STRCAT " (command " (CHR 34 ) "3dpoly" ( CHR 34 )"
"expr ( CHR 34) ( CHR 34)" )") )
  ( EVAL ( READ expr ) )
)

( DEFUN nul ( / n borexpr )
  ( SETQ n 1 )
  ( SETQ borexpr " (setq p0 nil " )
  ( REPEAT ( * pv nv )
    ( SETQ borexpr ( STRCAT borexpr "p" ( ITOA n) " nil " ) )
    ( SETQ n ( + n 1 ) )
  )
  ( SETQ borexpr ( STRCAT borexpr " )" ) )
  ( EVAL ( READ borexpr ) )
)

( DEFUN c:helice ( / cen radin radfin nv pv dz drad )
;;;  ( SETVAR "blipmode" 0 )
;;;  ( SETVAR "cmdecho" 0 )
  ( intr )
  ( ptos )
  ( dib )
```

```
  ( nul )
;;;  ( SETVAR "blipmode" 1 )
;;;  ( SETVAR "cmdecho" 1 )
)
```

Intente con estos valores sin importar los radios:
precisión en cada vuelta: 10
número de vueltas: 10
paso de hélice: 5

8.12.2 Observaciones

Si en el programa anterior del trazado de una función cualquiera el mecanismo (EVAL (READ …)) se utilizaba para crear una función a partir de la expresión introducida por el usuario, en este caso hay que crear todas las variables de punto necesarias para el trazado de la 3DPOLY.

Además hay que realizar una serie de operaciones matemáticas con tas coordenadas X e Y de cada punto y debe ser el propio programa el que, de acuerdo con el número total de puntos necesario, genere las instrucciones en AutoLISP para realizar esas operaciones.

En cierto sentido, el propio programa va a ser capaz de generar un programa por su cuenta. La hélice se generará en el plano XY del Sistema de Coordenadas (SCP) actual y alineada con el eje Z. Se trata, por tanto, de una hélice "recta", desarrollada perpendicularmente al plano XY actual

8.12.3 Explicación del programa

Se define una nueva orden de AutoCAD llamada "hélice" y cuatro funciones de usuario.

• Función **c:helice**

Define la nueva orden de AutoCAD, establece como argumentos locales todas las variables que quedan de las funciones intermedias y desactiva marcas auxiliares y eco de órdenes.

Llama sucesivamente al resto de funciones de usuario.

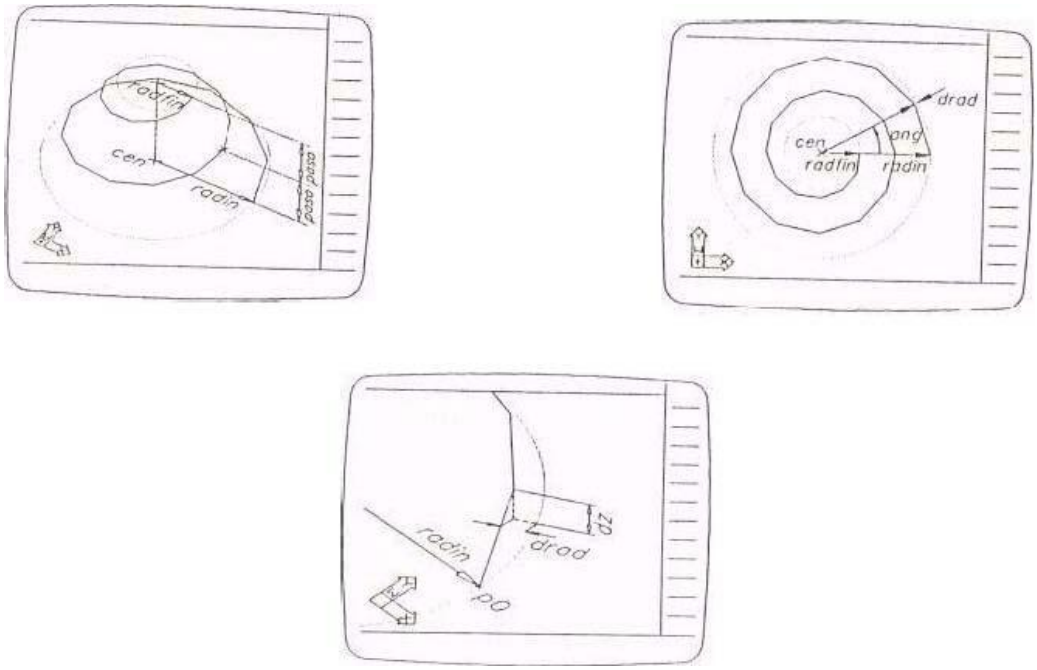
• Función **intr**

Como es habitual, solicita todos los datos que el usuario debe introducir para definir en este caso la hélice. En primer lugar el punto que va a ser el centro de la base de la hélice. A continuación el radio, que no tiene por qué ser constante. Por eso solicita en primer lugar el radio inicial (variable "radin") y a continuación el radio final (variable "radfin").

Se recurre a un procedimiento muy habitual que es visualizar como opción por defecto el valor recién introducido. Para ello se transforma ese valor numérico

en una cadena de texto mediante RTOS.

(RTOS radin 2 2)



Datos iniciales para el trazado de la hélice.

El modo 2 y precisión 2 indica cinc el valor se va a visualizar en formato decimal y con un precisión de dos decimales. La cadena resultante se concatena mediante STRCAT con el texto de solicitud del radio final.

Así, si el usuario introduce un valor de, por ejemplo 35.5 para el radio inicial, el programa visualizara el siguiente mensaje para solicitar el radio final:

Radio final < 35.50 >:

Según el mecanismo ya explicado de utilizar la asignación del valor del radio a la variable "radfin" como condición de un IF, se admite un "Retum" para seleccionar la opción por defecto.

(IF (SETQ radfin (GETDIST ……)) …....)

Si el usuario introduce "Return", GETDIST devuelve nil y la condición no se cumple. Se ejecuta la rama del NO de la alternativa y "radfin" toma el mismo valor que "radin", Si se introduce cualquier otro valor, éste queda almacenado en "radfin" y la rama del SI de la alternativa es una lista vacía.

El siguiente parámetro es la precisión en cada vuelta, que indica el número de segmentos de línea en que se va a aproximar cada vuelta de la hélice, dado que la orden de AutoCAD 3DPOL sólo admite líneas. Ese valor se almacena en "pv". Se establece previamente INITGET 7 para no admitir valores nulos ni Ø ni

negativos.

El número total de vueltas se almacena en "nv". Lo mismo que antes, se establece INITGET 7. El último valor necesario es el paso de hélice, cuyo valor si puede ser Ø o negativo, aunque no nulo (INITGET 1). Se almacena en "paso".

Con esto terminan todas las especificaciones que debe introducir el usuario. El programa calcula ahora los valores que necesitan para el trazado de la 3DPOL.

El ángulo en radianes abarcado por cada segmento de línea se calcula de la siguiente forma:

$$ang = \frac{2\ PI}{pv}$$

La fracción de paso que la hélice va a "crecer" en e1 eje z (positivo o negativo) para cada segmento de línea es:

$$dz = \frac{paso}{pv}$$

La variación del radio para cada segmento de línea es la variación total, dividida por el número total de segmentos de línea, que será el producto de número de segmentos en cada vuelta por el número de vueltas.

radfin − radindrad = pv * nv

Variables de entrada: ninguna.
Variables locales:
paso Valor del "paso" de la hélice.
Variables de salida:
ccn Centro de la base de la hélice.
radin Valor del radio inicial.
radfin Valor del radio final.
pv Número de segmentos en cada vuelta.
nv Número de vueltas.
ang Fracción de ángulo para cada segmento.
dz Fracción de "paso" para cada segmento.
drad Incremento de radio para cada segmento.

• Función **ptos**

Es la función que calcula todos los puntos necesarios para definir la 3DPOLY. El número total de puntos será el producto del número de segmentos en cada vuelta por el número total de vueltas. Cada punto se calcula de la siguiente forma :

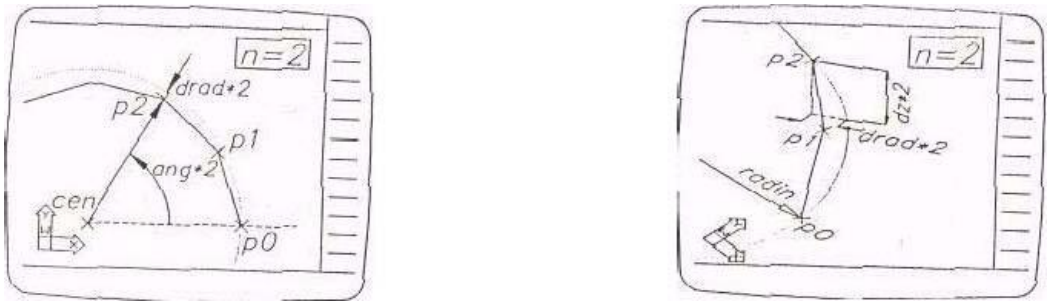
Coordenadas XY:
con POLAR desde el centro llevando un ángulo y una distancia.
Coordenada Z:
calculando la fracción de "paso" que corresponda.

El punto se obtendrá construyendo una lista con las tres coordenadas.Las variables "pØ", "p1", "p2", "p3", etc., contienen cada uno de los puntos.Así, por ejemplo, el punto "p3", que es el cuarto punto de la 3DPOL, se obtendría de la siguiente manera:

```
( SETQ z ( + zØ ( * dz n ) ) )
```

El valor de "n" es en este caso 3, pues es el tercer punto que se calcula a partir del inicial "pØ". E1 producto de "dz" por "n" es la fracción de "paso" que corresponde a ese punto. El valor de “z” es la coordenada Z del punto "p3 deseado.

```
( SETQ xy ( POLAR cen ( * ang n ) ( + radin ( * drad n ) ) ) )
```



Forma de obtener el tercer punto de la polilínea (Punto "p2").

El producto (* ang n) es la fracción de ángulo correspondiente al punto "p3”. El producto (* drad n) es el incremento de radio correspondiente a ese mismo punto. Sumando ese incremento al radio inicial se obtendrá el valor del radio para el punto actual.

Llevando, por tanto, desde el centro "cen" ese ángulo y esa distancia mediante el comando POLAR se obtiene un punto sobre el plano XY (coordenada Z la misma de "cen", por tanto).

Interesan únicamente las coordenadas XY de ese punto, las cuales se obtienen con los comandos CAR y CADR.

(CAR xy) es la coordenada X
(CADR xy) es la coordenada Y

El comando LIST construye una lista con los valores de ambas coordenadas, más la coordenada Z calculada anteriormente. Con los tres valores se forma el punto "p3".

```
( SETQ p3 ( LIST ( CAR xy ) ( CADR xy ) z ) )
```

Por tanto, la secuencia completa sería:

```
( SETQ z ( + zØ ( * dz n ) ) )
( SETQ xy ( POLAR cen ( * ang n ) ( + radin ( * drad n ) ) ) )
( SETQ p3 ( LIST ( CAR xy ) ( CADR xy ) z ) )
```

Para cualquier otro punto, la secuencia sería la misma con sólo cambiar en la última línea "p3" por el punto que corresponda. Se observa además que el número de la variable de punto coincide con el valor de la variable "n" de la repetitiva.

Para el ciclo número 5, el valor de "n" será 5 y el punto calculado será el correspondiente a "p5".

Por tanto, la última expresión hay que formarla en cada ciclo de acuerdo con el valor de "n". Esto se consigue construyendo una cadena de texto:

```
( STRCAT "( setq p" ( itoa n ) " ( list ( car xy ) ( cadr xy ) z) ) " )
```

El valor de "n" convertido en texto mediante ITOA es el que se añade a "p" para formar la variable de punto. El resto de la expresión se concatena y el total se almacena en "expr".

Ya sólo falta convertir esa cadena de texto en una expresión AutoLISP y evaluarla:

```
( EVAL ( READ exp. ) )
```

Al final del ciclo se suma uno a la variable de control "n". El número de veces que se repite el ciclo es el número total de segmentos a generar (* pv nv).

Antes del REPEAT se ha calculado el primer punto pØ y su coordenada zØ, que se va a utilizar como referencia para calcular el resto de coordenadas Z. Se inicializa el valor de "n" a 1.

El resultado final es la creación de todas las variables de punto de la polilínea con sus valores.

Variables de entrada:

cen Centro de la base de la hélice.
radin Valor del radio inicial.
radfin Valor del radio final.
pv Número de segmentos en cada vuelta.
nv Número de vueltas.
ang Fracción de ángulo para cada segmento.
dz Fracción de "paso" para cada segmento.
drad Incremento de radio para cada segmento.

Variables locales:

zØ Coordenada Z punto inicial de hélice.
n Contador del ciclo repetitivo.
xy Coordenadas XY de cada punto de la hélice.
z Coordenada Z de cada, punto de la hélice.
expr Expresión para generar cada punto.

Variables de salida:
pØ, p1 ... Todos los vértices de la hélice.

• Función **dib**

Una vez creadas todas las variables de punto con sus valores, hay que dibujar la 3DPOL.

Para ello se llama con COMMAND a la orden de AutoCAD y se suministran todos los puntos. Como este número es variable, se establece un ciclo repetitivo para formar la expresión de AutoLISP que contenga todos esos puntos.

Se inicializa esa expresión con la variable "pØ" y después se van concatenando el resto de puntos hasta completar el total, que es el producto de "pv" por "nv".

Por ejemplo, si el usuario ha especificado n = 5, el contenido de la variable "expr" al final del ciclo seria:

"pØ p1 p2 p3 p4 p5"

Para dibujar una 3dpol con esos puntos se necesita formar la expresión:

(COMMAND "3dpol" pØ p1 p2 p3 p4 p5 " ")

Hay un problema para concatenarla con STRCAT, y es que la expresión contiene en su interior comillas. Si éstas se indican sin más, STRCAT las toma como final de cadena y no como un carácter mas. Para indicarlo hay que emplear el código ASCII de las comillas, que es 34, con CHR.

Una vez formada la expresión sólo falla evaluarla y la 3dpol queda dibujada.

Variables de entrada:pØ, pl ..... Todos los vértices de la hélice.pv Número de segmentos en cada vuelta.nv Número de vueltas.

Variables locales:n Contador del ciclo repetitivo.expr Expresión de la llamada a 3DPOLY.

Variables de salida: pØ, p1 ..... Todos los vértices de la hélice.

• Función **nul**

La polilínea ya ha sido dibujada. Sin embargo, todas las variables creadas para

cada vértice permanecen cargadas en la memoria. Si, como es lo más habitual, el número de segmentos resulta elevado, no se pueden tener tantas variables ocupando inútilmente memoria.

Esta última función de usuario tiene, pues, por objeto liberar todas esas variables de punto poniendo su valor en nil.

La expresión a formar para conseguir eso sería:

```
( SETQ pØ nil p1 nil p2 nil p3 nil .....)
```

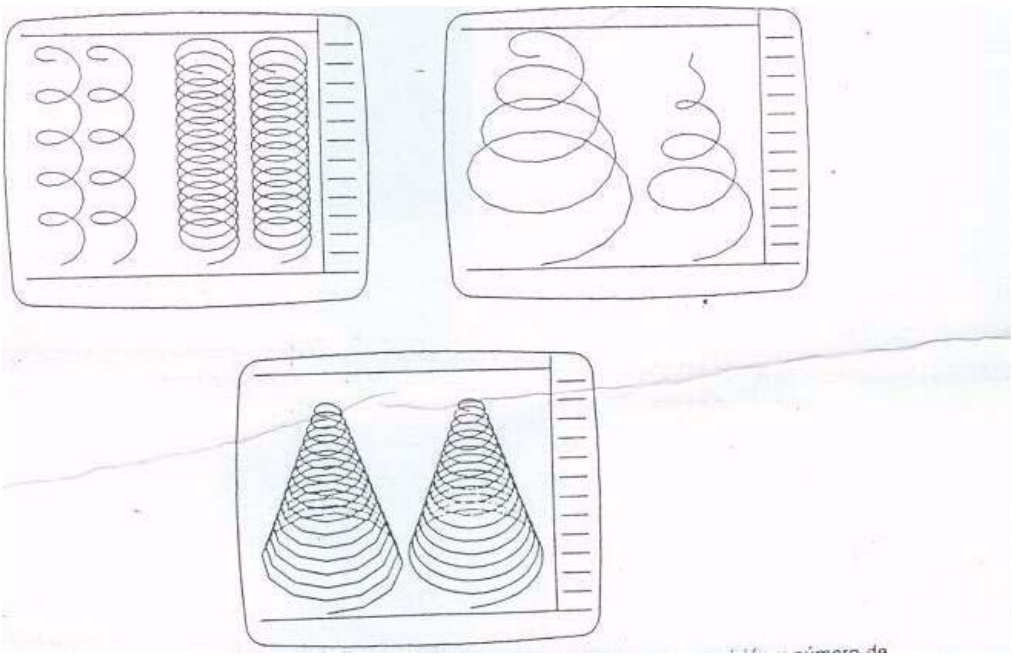
Por tanto, se inicializa esa expresión en una variable "borexpr" y en un ciclo se le van añadiendo todos los puntos con su correspondiente nil.

Al final del ciclo se evalúa esa expresión y las variables quedan anuladas.

Variables de entrada:
pØ, p1 … Todos los vértices de la hélice.
pv Número de segmentos en cada vuelta.
nv Número de vueltas

Variables locales:
borexpr Expresión para cancelar variables.
n Contador del ciclo repetitivo.

Variables de salida: ninguna.



Resultados gráficos de hélices para diferentes valores de radios, paso,precisión y número de vueltas. En el primero y tercer gráfico se ofrecen las polilíneasoriginales y adaptada curva.

9 Capitulo 9: Operaciones con archivos (ficheros)

En este capitulo se estudia una importante aplicación de ios programas en AutoLISP: la posibilidad de gestionar directamente archivos de texto. Se puede abrir cualquier archivo de texto escrito en caracteres ASCII y después acceder a su contenido, bien para leerlo desde el programa o incluso para crearlo, escribiendo líneas generadas por el propio programa.

AutoCAD utiliza en muchas aplicaciones los archivos de texto: tipos de línea, patrones de sombreado, menús, archivos de guión, órdenes externas, etc. Desde un programa en AutoLISP se puede modificar cualquiera de esos archivos generando, por ejemplo, patrones de sombreado para añadir en el ACAD.PAT, o archivos de guión .SCR específicos.

Por ultimo, puesto que los propios programas de AutoLISP se encuentran contenidos en archivos de texto, es posible crear un programa o modificarlo desde otro programa en AutoLISP.

9.1 (OPEN <nombre de archivo> <modo>) Abrir archivo (fichero)

Este comando abre un archivo para permitir el acceso a las funciones de entrada/salida de AutoLISP. El archivo se indica con el nombre completo, con extensión y entre comillas. OPEN devuelve un valor de descriptor de archivo que habrá que indicar para todas las operaciones a realizar con ese archivo. Por eso se debe almacenar ese valor en una variable.

```
( SETQ fich ( OPEN "ejemplo.txt" "r" ) )
```

El valor de <modo> especificado indica si el archivo se abre en modo lectura o escritura y es un carácter que puede ser solamente "r", "w" o "a". La significación de cada uno de estos tres caracteres es la siguiente:

Modo	Descripción
“r”	Abre en lectura. Sólo se pueden extraer datos del archivo. Si el nombre indicado no existe, devuelve nil
“w”	Abre en escritura. Si el archivo no existe, OPEN lo crea. Si ya existe, OPEN reescribirá los nuevos datos en él, destruyendo los existentes.
“a”	Abre en modo aditivo. Si el archivo no existe, OPEN lo crea. Si existe, OPEN lo abre y se sitúa en su final para ir añadiendo los nuevos datos a continuación de los ya existentes.

Por ejemplo, OPEN podría devolver los siguientes valores:

(SETQ fich (OPEN "nuevo.txt" "w")) devuelve <File: #5236>
(SETQ fich (OPEN "noexiste. doc" "r")) devuelve nil

(SETQ fich (OPEN "ejemplo.ser" "a")) devuelve <File: #f3652>

Si el archivo no se encuentra en el directorio actual hay que indicar e1 camino para localizarlo. La contrabarra se obtiene con el carácter de control "\ \".

(SETQ fich (OPEN "a:\ \guiones\ \ejemplo.scr" "a"))

9.2 (CLOSE <descr. archivo>) Cerrar archivo (fichero)

Este comando cierra el archivo cuyo descriptor se indica. Este descriptor ha tenido que obtenerse, lógicamente, con el comando OPEN. Una vez cerrado el archivo se devuelve nil y ya no puede hacerse ninguna operación con él.

(CLOSE fich) devuelve nil

9.3 (FINDFILE <nombre-archivo>) Explorar camino acceso.

Este comando explora el camino de acceso a bibliotecas de AutoCAD para encontrar el archivo indicado en <nombre-archivo>. El nombre debe darse completo, con extensión y entre comillas, puesto que es una cadena de texto.
Los caminos de acceso a bibliotecas de AutoCAD son los siguientes:

El directorio actual del Sistema Operativo desde el que se ha llamado a AutoCAD,

El directorio que contiene el dibujo actual que se está editando.

El directorio especificado en la variable de entorno ACAD, que indica dónde buscar los archivos de soporte (dibujos prototipo, fuentes de texto, menús, programas AutoLISP y dibujos a insertar) que no se encuentran en el directorio actual.

El directorio que contiene los archivos de programa de AutoCAD.

Por tanto, si no se indica nada más que el nombre del archivo con su extensión, FINDFILE busca en los caminos de acceso a bibliotecas de AutoCAD. Si se indica el nombre de archivo precedido de unidad de disco y camino completo, FINDF1LE busca sólo en ese directorio, sin tener en cuenta los caminos de acceso a bibliotecas.

Si se encuentra el archivo, FINDFILE devuelve el nombre completo con el camino de acceso incluido. Este nombre completo es totalmente compatible con el comando OPEN. Por tanto, se puede utilizar FINDFILE para encontrar un archivo antes de abrirlo con OPEN.

Por ejemplo, las siguientes expresiones podrían devolver:

(FINDFILE "ejemplo.lsp") "devuelve "C:\ \SOPORTES\ \EJEMPLO.LSP"
(FINDFILE "chalet, dwg") devuelve "C:\ \CAD1Ø\ \CHALET.DWG"
(FINDFILE "a:\ \bloques\ \mesa.dwg") devuelve "A:\ \BLOQUES\ \MESA.DWG"
(FINDFILE "demo.scr") devuelve nil

El archivo "ejemplo.lsp" se encuentra en el directorio de archivos de soporte "c:\soporles". El dibujo "chalet" se encuentra en el directorio de programas de AutoCAD "c\cad1Ø". El dibujo "mesa" se busca únicamente en el directorio indicado "a:\bloques" y al encontrarse devuelve el nombre con el camino completo. El archivo "demo.ser" no está en ninguno de los caminos de acceso a bibliotecas de AutoCAD; por eso FINDFILE devuelve níl.

9.4 (PRIN1 <expr> [<descr-arch>]) Escribir expresión.

Este comando escribe la expresión indicada <expr> y devuelve la propia expresión. Si nno se indica un descriptor de archivo, la expresión se escribe en pantalla (se visualiza en la línea de órdenes). Si se indica un descriptor valido de un archivo previamente abierto a la escritura, se escribe en ese artículo y devuelve la expresión.

La expresión no tiene por qué ser una cadena de texto; puede ser cualquiera. Se escribe sin añadirse ningún espacio en blanco o interlínea.

(SETQ x 25.Ø a "Buenos dias")

(PRIN1 x) visualiza 25.Ø devuelve 25.Ø
(PRIN1 a) visualiza "Buenos días" devuelve "Buenos días"
(PRIN1 'a) visualiza A devuelve A

Si se indica una expresión, PRIN1 escribe el resultado, de esa expresión, a no ser que esté precedida por un literal:

(PRIN1 (SETQ x 5)) visualiza 5 devuelve 5
(PRIN1 ' (SETQ x 5)) visualiza (SETQX 5) devuelve (SETQ X 5)

Si la expresión es una cadena con caracteres de control, PRIN1 escribe esos caracteres con contrabarra y el número de su código octal. Si la cadena incluye caracteres cuyo código ASCII es superior al 126, que es el máximo reconocido por AutoCAD V. 1Ø, también escribe o visualiza esos caracteres con contrabarra y un número octal. Por ejemplo:

(PRIN1 "mañana") visualiza y devuelve "ma\244ana"
(PRIN1 (CHR 2)) visualiza y devuelve "\002"
(PRIN1 (CHR 13)) visualiza y devuelve "\r"

Indicando un descriptor de archivo funcionaría de manera totalmente similar a los ejemplos descritos, escribiendo en el archivo en vez de visualizar en pantalla.

(SETQ fich (OPEN "demo.ser"))

(PRIN1 "delay 1000" fich)

Se puede utilizar PRIN1 sin indicar ninguna expresión, y entonces escribe el símbolo de cadena nula. Al incluir PRIN1 como última expresión de un programa, visualizará en pantalla una línea en blanco y terminará el programa de una manera limpia.

9.5 (PRINT <expr> [<descr-arch>]) Escribir con interlinea.

Totalmente idéntica a PRIN1, salvo que salta a nueva línea antes de visualizar o escribir la expresión y añade un espacio en blanco después.

Command: ***(PRIN1 (CHR 65))***
"A" "A"
Command: ***(PRINT (CHR 65))***
"A" "A"

9.6 (PRINC <expr> [<descr-arch>]) Escribir expresión ASCII.

Este comando funciona de manera análoga al de PRIN1, escribiendo la expresión en pantalla o en el archivo indicado por su descriptor y devolviendo la expresión. Pero la diferencia es que escribe todo el juego de caracteres ASCII, incluidos los de número superior a 126, aunque devuelva el código de control octal igual que PRIN1.

Así, por ejemplo:

(PRINC "mañana") escribe "mañana" devuelve "ma\244ana"

La utilidad de PRIN1 es que escribe expresiones compatibles, por ejemplo, con LOAD y COMMAND, que llama a órdenes de AutoCAD cuyos mensajes no admiten, por tanto, todos los códigos ASCII. Mientras que PRINC escribe cualquier carácter admitido en un archivo de texto y las expresiones pueden ser leídas directamente con comandos como READ-LINE, que se estudiará más adelante en este capitulo.

9.7 (READ-CHAR [<descr-arch>]) Leer caracter.

Este comando lee un carácter de la memoria temporal de entrada del teclado ("buffer") y devuelve su código ASCII. Si la memoria temporal contiene varios caracteres en el momento en que se ejecuta READ-CHAR, devuelve el primer carácter. Sucesivas llamadas a READ-CHAR irán devolviendo los demás caracteres hasta llegar al último de la memoria temporal (que será el último tecleado).

Si en el momento de ejecutar READ-CHAR la memoria temporal del teclado se encuentra vacía, espera a que el usuario introduzca una serie de caracteres y

cuando pulsa RETURN devuelve el primer carácter introducido. A partir de ese momento, sucesivas llamadas a READ-CHAR devuelven los demás.

Si se especifica un descriptor de archivo valido, el comando lee el primer carácter de ese archivo y devuelve su código ASCI. Después, sucesivas llamadas a READ-CHAR van devolviendo los demás caracteres del archivo. El cambio de línea del archivo de texto "Nueva línea/Retorno de carro" es el código ASCII 1Ø.

(READ-CHAR) se introduce Hola y devuelve 72
(READ-CHAR) devuelve 111
(READ-CHAR) devuelve 108
(READ-CHAR) devuelve 97
(READ-CHAR) devuelve 10

En el ejemplo, la primera vez que se utiliza READ-CHAR la memoria temporal se encuentra vacía y el usuario introduce la cadena "Hola*'. El comando devuelve el código ASCII del primer carácter "H", que es 72. Las demás llamadas van devolviendo los sucesivos códigos ASCII hasta llegar al RETURN final, que es el 1Ø.

Por compatibilidad con el Sistema Operativo UNIX, en que el final de línea es siempre el código ASCII 1Ø, el comando devuelve ASCII 1Ø cuando se utiliza en MS-DOS y se detecta el carácter "Retorno de carro o Return", a pesar de que en realidad el código ASCII de este carácter es 13.

9.8 (READ-LINE [<descr-arch>]) Leer línea.

Este comando lee una cadena completa introducida desde el teclado o una línea de un archivo de texto si se indica un descriptor de archivo válido. Devuelve el valor de la cadena leída (por tanto, entre comillas).

Al llegar al final del archivo de texto devuelve nil

Por ejemplo, si el contenido de un archivo llamado "dtr.lsp" es el siguiente:

```
( DEFUN dtr ( g )
        ( * PI ( / g 180.0 ) )
)
```

se abre ese archivo a la lectura:

(SETQ arch (OPEN "dtr.lsp", "r"))

Las sucesivas llamadas de READ-LINE devolverían:

(READ-LINE fich) " (DEFUN dtr (g)"
(READ-LINE fich) " (* PI (/ 9 18Ø.Ø))"
(READ-LINE fich) ") "
(READ-LINE fich) nil

9.9 (WRITE-CHAR <num> t<descr-arch>J) Escribir carácter ASCII.

Este comando escribe el carácter cuyo número de código ASCII es el indicado en <num>, bien en pantalla o bien en el archivo de texto cuyo descriptor se indica. Este archivo de texto debe lógicamente encontrarse abierto a la escritura. El comando devuelve el valor ASCII del carácter escrito.

(WRITE-CHAR 72) escribe carácter "H" y devuelve 72
(WRITE-CHAR 32) escribe espacio en blanco y devuelve 32

Por las mismas razones de compatibilidad con UNIX que en el caso de READ-CHAR, en el Sistema Operativo MS-DOS tanto el código ASCII 1Ø como el 13 generan la misma secuencia "Retomo de carro/Fin de línea".

9.10 (WRITE-LINE <cad> [<descr-arch>]) Escribir línea.

Este comando escribe la cadena indicada <cad> en la pantalla o en el archivo cuyo descriptor se especifica. El comando devuelve la cadena entre comillas, pero omite las comillas al escribir en el archivo.

```
( SETQ fich ( OPEN "ej.scr" "a" ) )
( WRITE-LINE "línea 5Ø,5Ø 1ØØ,1ØØ " fich )
( WRITE-LINE "cambia u  p col rojo " fich )
( CLOSE fich )
```

En el ejemplo se abre un archivo en modo aditivo, que va a ser un archivo de guión de AutoCAD. Los dos comandos WRITE-LINE escriben dos líneas en ese archivo.

9.11 Ejemplo 1: Menú en pantalla con bloques definidos.

Es habitual en un dibujo con unos cuantos bloques definidos (Construcción, Circuitos, Topografía, etc.) que el usuario no se acuerde del nombre de un determinado bloque cuando pretende insertarlo. Esto obliga a listar los existentes con la opción "?" de la orden BLOQUE. Además, el usuario debe introducir después el nombre de ese bloque por teclado.

Resultaría mucho más cómodo si el nombre de cada bloque fuera incorporándose como opción a un menú de pantalla, conforme van siendo definidos en el dibujo actual. El usuario tendría en todo momento a la vista los bloques disponibles, y para insertar cualquiera de ellos le bastaría con seleccionar en la pantalla la opción correspondiente.

Esto es posible hacerlo creando un sencillo menú de escritura de bloques en la pantalla, el cual puede ser llamado cada vez que interese.

El programa de este ejemplo va actualizando dicho menú cada vez que el

usuario define un nuevo bloque en su dibujo. Para ello es necesario redefinir la orden BLOQUE, tal como se explica a continuación.

9.11.1 Listado del programa

Nota: Hay que tener cuidado al correr este programa, ya que reemplazara el menú que tiene la interfase.

```
(DEFUN inimnu (/ fich)
  (SETQ fich (OPEN "c:\\bloques.mnu" "w"))
; siempre y cuando se haya direcionado al directorio en busqueda
;por omision
  (WRITE-LINE "***SCREEN" fich)
  (WRITE-LINE " [AutoCAD ] menu acad" fich)
  (WRITE-LINE " " fich)
  (WRITE-LINE "[---------] " fich)
  (WRITE-LINE "[Menú de  ] " fich)
  (WRITE-LINE "[BLOQUES  ] " fich)
  (WRITE-LINE "[---------] " fich)
  (WRITE-LINE " " fich)
  (CLOSE fich)
)

(inimnu)

(SETVAR "cmdecho" 0)
;(COMMAND "anuladef" "bloque")
(COMMAND "menu" "bloques")
(PROMPT "Creado nuevo menú llamado BLOQUES.")
(PROMPT "La opción AutoCAD vuelve al menú ACAD")

(DEFUN bloc (/ pins ent)
  (SETQ nbl (GETSTRING "Nombre del bloque: "))
  (TERPRI)
  (WHILE (= nbl "?")
    (COMMAND ".bloque" "?" " ")
    (SETQ nbl (GETSTRING "Nombre del bloque: "))
    (TERPRI)
  )
  (SETQ pins (GETPOINT "Punto de base o inserción: \n"))
  (SETQ       ent (SSGET)
  (COMMAND ".bloque" nbl pins ent " ")
)

(DEFUN actmnu    (/ fich)
    (SETQ fich (OPEN ".bloques.mnu" "a"))
    (WRITE-LINE (STRCAT "[" nbl ": )" "Insert " nbl) fich)
    (CLOSE fich)
    (IF  (= (GETVAR "menuname") "bloques")
      (COMMAND "menú" "bloques")
```

```
    )
)

(DEFUN c:bloque (/ nbl)
;;;   (SETVAR "blipmode" 0)
;;;   (SETVAR "cmdecho" 0)
   (bloc)
   (actmnu)
;;;   (SETVAR "blipmode" 1)
;;;   (SETVAR "cmdecho" 1)
)

  (PRINT "Orden BLOQUE redefinida")
```

9.11.2 Observaciones

Este programa utiliza el mecanismo de la redefinición de la orden de AutoCAD "Bloque" para crear una orden con el mismo nombre, pero que realiza unas operaciones diferentes de la orden original.

Para ello hay que anular previamente la definición de esa orden con la orden ANULADEF, pues en caso contrario la nueva definición hecha desde el programa no tendría efecto. Para volver a la definición original, bastará emplear la orden REDEFINE.

El programa tiene un aspecto interesante a resaltar: no se limita a crear funciones de usuario que sólo se ejecutarán cuando sean llamadas, sino que él mismo llama a una determinada función y a órdenes de AutoCAD. De esta forma se ejecutarán al cargar el archivo en AutoLISP con LOAD.

9.11.3 Explicación del programa

Redefine la orden de AutoCAD BLOQUE (e INSERT) y define tres funciones de usuario, además de ejecutar la llamada a una de esas funciones y varios comandos más de AutoLISP.

• Función **c:bloque**

Al dar como nombre el de una orden ya existente, ésta queda redefinida. Para que la redefinición tenga efecto hay que anular previamente la orden original con ANULADEF. La llamada a esta orden de AutoCAD se ejecuta en el programa como se explica más adelante.

La orden redefinida llama a las funciones de usuario "bloc" y "actmnu”.

• Función **inimnu**

Inicializa un nuevo archivo de menú llamado "bloques.mnu". Esto se hace con el comando

(OPEN "bloques.mnu" "w")

El parámetro "w" hace que se abra el archivo en modo escritura. Si ese archivo no existe OPEN lo crea Si existe, borra su contenido y lo inicializa de nuevo. El descriptor de ese archivo se almacena en la variable “fich”.

Se pretende que ese archivo de menú contenga un menú de pantalla cuyo contenido es el siguiente:

```
***SCREEN
[AutoCAD] menu acad
[---------------]
[Menú de ]
[BLOQUES ]
[---------------]
```

El menú comienza con el identificador de sección (en este caso SCREEN, que es pantalla). Después contiene la" opción AutoCAD para llamar al menú "acad". Luego visualiza una serie de líneas que indican el nombre del incio y que no llaman a ninguna opción.

A este contenido se le irán añadiendo posteriormente los nombres de los bloques que vaya definiendo el usuario, con las correspondientes llamadas a su inserción.

El contenido inicial del archivo de menú se va escribiendo con sucesivos comandos WRITE-LINE. Al final se cierra el archivo con CLOSE. Con esto termina la definición de la función de usuario "inimnu".

Pero se hace necesario inicializar ese archivo de menú nada mas cargar con LOAD el archivo de AutoLISP, Así, cada vez, que se utilice la orden Bloque redefinida, se podrá añadir el nuevo bloque al menú.

Por eso se incluye en el archivo de AutoLISP la llamada a la función. Cuando se carga el archivo con LOAD, la primera función "inimnu" se carga en la memoria en primer lugar. A continuación viene la llamada

```
( inimnu )
```

Esto hace que se evalúe dicha función, cuyo resultado es inicializar un archivo de menú "bloques.mnu",

A continuación se anula la definición de la orden "bloque" con la orden de AutoCAD "anuladef”. Se carga el menú recién creado, con la orden de AutoCAD "menú", y se visualiza un mensaje de advertencia:

"Creado nuevo menú llamado BLOQUES. La opción AutoCAD vuelve al menú ACAD"

Así el operario que ha cargado el archivo con LOAD sabe lo que ha ocurrido. En pantalla se visualiza el menú "bloques" con el aspecto inicial ya indicado

anteriormente.

Variables de entrada: ninguna.

Variables locales:
fich Descriptor del archivo de menú.

Variables de salida: ninguna.

• Función **bloc**

Contiene la nueva definición de la orden “bloque". En primer lugar solicita el nombre del bloque a crear (variable "nbl"). A continuación establece un control para detectar si el usuario ha respondido con “?”.

En la orden original de AutoCAD, esta opción provoca un listado con todos los bloques actualmente definidos en el dibujo. Al haberla redefinido no se dispone de esta opción, por lo que hay que incorporarla mediante un control. Si el usuario responde con "?", se llama a la orden original de AutoCAD que se consigue poniendo un punto por delante (Orden ".bloque"), y a su opción "?" preguntando de nuevo por el nombre del bloque a crear.

Si el usuario responde con cualquier otro nombre de bloque, se pasa a solicitar el punto de intersección (variable pins"). Obsérvese el empleo de "\n" en vez de TERPRI para provocar un salto de línea.

Lo único que resta es designar los objetos que forman el bloque, lo cual se hace con el comando SSGET, que se estudiará mas adelante en el Capitulo 13 y cuya función es almacenar un conjunto designado (variable “ent" en este caso).

Se llama a la orden original de AutoCAD con ".bloque", se suministran todos los parámetros y el bloque queda definido.

Variables de entrada: ninguna.

Variables locales:
pins Punto de inserción del bloque.
ent Entidades que forman el bloque.

Variables de salida:
nbl Nombre de bloque a crear.

• Función **actmnu**

Su cometido es actualizar el contenido del archivo de menú "bloques.mnu", añadiendo los nuevos bloques cada vez que el usuario utilice la orden redefinida.

En primer lugar abre dicho archivo de menú con el parámetro "a", es decir, en

modo aditivo, quiere decir que cada línea se añadirá al final del archivo a las ya existentes.

Si, por ejemplo, el usuario crea un nuevo bloque llamado "transis", la línea a añadir al archivo deberá ser la siguiente:

[transis:] insert transis

De esta forma, al seleccionar esa opción del menú en pantalla, se insertará el bloque. Para escribir esa línea se concatena el texto de la siguiente forma:

(STRCAT "[" nbl ":] insert " nb1)

Por último, la función explora cuál es el menú actual que tiene cargado el usuario. Si éste utiliza la orden "bloque" redefinida desde el menú de AutoCAD, el archivo de menú "bloques.mnu" se va actualizando, pero el menú cargado no cambia cada vez, lo cual podría ser molesto.

En cambio, si el usuario ha cargado el menú "bloques" porque le interesa trabajar con él, lo más interactivo es que cada vez que defina un nuevo bloque y éste se añada al menú "bloques.mnu" se cargue el menú actualizado para que el nuevo bloque se refleje automáticamente en pantalla.

Por eso la función examina el nombre del menú actual cargado, en la Variable de Sistema "menuname", y si se trata de "bloques" So llama con (COMMAND "menú" ..).

Variables de entrada:

nbl Nombre del bloque recién creado.

Variables locales:

fich Descriptor del archivo de menú.

Variables de salida: ninguna.

Habitualmente la definición de una nueva orden se coloca en el final del archivo de AutoLISP para que al cargar devuelva el nombre de la nueva orden. En este caso, como se trata de redefinir una orden ya existente, interesa mas visualizar un mensaje de advertencia.

Por eso al final del archivo de AutoLISP se incluye un PROMPT con el mensaje. El último PRIN1 es para terminar LOAD sin que salga el nil que devuelve el comando PROMPT.

Así se consigue que al cargar el archivo en AutoLISP con LOAD el operario de AutoCAD vea en pantalla los siguientes mensajes:

Creado nuevo menú llamado BLOQUES.La opción AutoCAD vuelve al menú ACAD.Orden BLOQUE redefinida

10 Capitulo 10: Acceso a variables de AutoCAD

Todas las Variables de Sistema de AutoCAD pueden ser gestionadas desde los programas de AutoLISP bien para extraer los valorcs actuales contenidos en ellas, bien para introducir nuevos valores (salvo las que sean de sólo lectura).

Esto permite controlar desde el programa prácticamente todos los aspectos de cualquier dibujo, puesto que todos se encuentran definidos por una o más variables.

Se ha hablado ya, por ejemplo, de la variable CMDECHO, cuyo valor es Ø ó 1, y que controla el eco de las instrucciones de AutoLISP a la línea de órdenes. También de BLIPMODE, que controla la visualización de marcas auxiliares.

10.1 (SETVAR <var> <valor) Introducir valor en variable.

Este comando asigna el <valor> especificado a la variable indicada en <var>. El nombre de la variable <var> debe ir entre comillas. Si la variable es de sólo lectura, no se puede introducir ningún valor y se produce un error.

Command: ***(SETVAR "filletrad" 2)***

En el ejemplo, el comando SETVAR asigna a la variable "filletrad", que controla el valor del radio de empalme, un radio igual a 2.

Si se ejecuta el comando SETVAR cuando una orden se encuentra en curso, sería equivalente a emplear la orden de AutoCAD "Modivar" de forma transparente (con el apóstrofo delante). En ese caso puede ocurrir que la modificación introducida en la variable sólo surta efecto en la siguiente orden o regeneración.

Command: ***(COMMAND "erase") (SETVAR "pickbox" 2)***

COMMAND llama a la orden de AutoCAD "erasea", la cual se queda esperando en "Select objets: ". Después SETVAR cambia el valor de la mirilla de designación contenido en la variable "pickbox" a 2. Este cambio se efectúa de manera transparente y la orden “erase" sigue pidiendo designar objetos, pero ahora visualiza la mirilla con el nuevo valor de "pickbox". Si el programa en AutoLISP introduce a continuación cualquier conjunto designado, éste será aceptado como respuesta a la orden "erse".

10.2 (GETVAR <var>) Extraer valor de una variable.

Este comando extrae el valor actual de la variable indicada en <var> y devuelve ese valor. Lógicamente, también se puede acceder a las variables de sólo lectura. Si la variable indicada no existe, devuelve nil . El nombre de la

variable debe ir entre comillas.

Command: ***(SETQ v (GETVAR "aperture"))***

En el ejemplo el valor de "aperture" que es la mirilla del retículo para modos de referencia a entidades, queda almacenado en la variable "v".

10.3 Valores posibles de OSMODE (MODOs de designación).

El significado y funnción de cada una de las Variables de Sistema viene determinado desde el propio AutoCAD. No obstante, hay una variable que conviene reseñar en este capitulo por su utilidad para los programas en AutoLISP que requieren designar puntos pertenecientes a entidades del dibujo.

Esta Variable de Sistema es OSMODE y su contenido es el del modo o modos de referencia de entidades, vigente actualmente en el dibujo. Es decir, los modos almacenados con la orden de Auto "Refent"

Cada vez que haya que introducir un punto en un programa de AutoLISP designado en pantalla conviene tener muy en cuenta cual es el valor actual de OSMODE y cambiarlo si fuera preciso. Los valores posibles de OSMODE y sus significados son los siguientes:

Valores	**Significados**
Ø	NIN (Ninguno)
1	END (Punto final
2	MID (Punto medio)
4	CEN (Centro)
8	NODE (Punto)
16	QUA (Cuadrante)
32	INT (Intersección)
64	INS (Punto de inserción)
128	PER (Perpendicular)
256	TAN (Tangente)
512	NEA (Cerca)
1Ø24	QUICK, (Rápido)

Hay que tener en cuenta que el modo "Rápido" no puede establecerse aisladamente, sino en combinación con algún otro modo de referencia; de ahí la coma "," de que va seguido. Se pueden combinar varios modos de referencia

separados por comas. En este caso el valor de "Osmode" será la suma valores de cada modo.

Por ejemplo:

Command: ***(COMMAND "osmode" "quick,end,int")***
(GETVAR '"osmode")
1057

Al almacenar con la orden "Refent" los tres modos de referencia, el valor de "Osmode" resulta la suma de los correspondientes valores de "quick", "end" e "int": 1Ø24,+1 + 32 = 1057.

El resultado hubiera sido e1 mismo almacenando directamente ese valor en "Osmode" con el comando SETVAR.

Command: ***(SETVAR "osmode" 1057)***
1Ø57

10.4 (OSNAP <pt> <modos>) Aplicar MODO de referencia.

Este comando aplica el modo o modos de referencia indicados <modos> al punto <pt> y devuelve un punto como resultado de esa aplicación. El modo o modos debe ser una cadena de texto y por tanto debe ir entre comillas. Si los modos indicados son varios, irán separados por comas.

Orden: ***(SETQ pmed (OSNAP ptl "mid"))***

En el ejemplo seria equivalente a haber seleccionado un modo de referencia "punto medio" y haber pinchado en el punto ptl. Dependiendo del valor de "apertura", si se encuentra una entidad dentro de la mirilla, su punto medio quedaría almacenado en la variable "pmed". En caso de no encontrar ninguna entidad o no existir un punto medio, devuelve "nil".

Se observa que el hecho de que se encuentre o no el punto buscado depende en gran medida del valor actual de "apertura". Un valor demasiado pequeño dificulta la operación. Un valor demasiado grande puede atrapar otro punto próximo al que se pretendía.

Para indicar el modo de referencia valdría "med" o "medio". Si se especifican varios:

Command: ***(SETQ pin (OSNAP ptl "quick,mid,int"))***

10.5 (MENUCMD <cad>) Llamada a Menus de AutoCAD.

Este comando llama a un submenú del menú actual cargado por AutoCAD. De este modo se puede ejecutar un programa en AutoLISP asociándole un menú

(ya sea de pantalla, tableta, despiegable, etc.) que podría contener, por ejemplo, opciones a seleccionar por el usuario desde el propio programa en AutoLISP.

El comando MENUCMD devuelve siempre nil. La cadena indicada <cad> es una cadena de texto (entre comillas) con la forma:

sección = nombre de submenu

La sección se refiere a la inicial del idenlificador de sección de menú. Estas iniciales son las siguientes:

Iniciales	**Significado**
B	Buttons: Menú de pulsadores.
S	Screen: Menú de pantalla.
1	Icon: Menú de símbolos.
P1-P1Ø	Popl-1Ø: Menús desplegables 1-1Ø.
T1-T4	Tabletl-4: Menús de tablero 1-4.
A1	Aux1: Menú teclado auxiliar.

El nombre de submenú al que se llama tiene que existir en la sección de menú correspondiente (tiene que haber un identifícador de submenu con "***" y el nombre en cuestión).

Por ejemplo:

(MENUCMD "S-REFENT")

haría aparecer en el área de menú de pantalla (sección S) el submenú REEENT".

(MENUCMD "P2 = MALLAS")

En este caso se llama a un submenú "mallas" en la sección de menú doblegable 2 (POP2) Este submenú debe existir en esa sección con su correspondiente identificador "***MALLAS".

10.6 (TRANS <pto> <desde> <hasta> [<desplz>]) Convertir de un SCP a otro.

Este comando convierte un punto o un vector de desplazamiento desde un Sistema de Coordenadas hasta otro. El valor del punto o del desplazamiento se indica en <pto> como una lista de tres números reales. Si se indica el argumento optativo <desplz> y su valor es diferente de NIL, entonces la lista de tres números reales se considera un vector de desplazamiento.

Los argumentos <desde> y <hasta> son los que indican en que Sistema de Coordenadas se encuentra actualmente el punto o desplazamiento y en qué nuevo Sistema de Coordenadas debe expresarse el punto o desplazamiento devuelto.

Estos dos argumentos para expresar Sistemas de Coordenadas pueden ser los siguientes:

Un código que indique el Sistema de Coordenadas. Los tres códigos admitidos son:

Un nombre de entidad que indica el Sistema de Coordenadas relativo a esa Entidad (SCE). Esto es equivalente a la opción Entidad de la orden SCP de AutoCAD.

Un vector de altura de objeto en 3D, indicado como lista de tres números reales. Este vector expresa la orientación de la altura de objeto en el nuevo SCP con respecto al Sistema Universal SCU. No sirve este procedimiento cuando la entidad ha sido dibujada en el Sistema Universal (su SCE coincide con el SCU).

Ø Sistema de Coordenadas Universal (SCU).

1 Sistema de Coordenadas Personal (SCP) actual.

2 Sistema de Coordenadas Vista actual (SCV).

El comando TRANS devuelve el punto o vector de desplazamiento como una lista de tres elementos expresada en el nuevo Sistema de Coordenadas indicado en <hasta>.

Por ejemplo, si el SCP actual se ha obtenido girando desde el SCU 9Ø grados sobre el eje Y:

(TRANS ' (1 2 3) 0 1) devuelve (-3.Ø 2.Ø 1.Ø)
(TRANS ' (-3 2 1) 1 0) devuelve (1.Ø 2.Ø 3.Ø)

En el primer caso, el punto (1 2 3) en el SCU (código Ø) se expresa en el SCP actual (código 1) como (-3 2 1). En el segundo caso se hace la operación inversa, desde el SCP actual (código 1) hasta el SCU (código Ø),

A la hora de introducir coordenadas o desplazamientos para utilizar órdenes de AutoCAD, hay que tener muy presente que siempre se consideran respecto al SCP actual (salvo que vayan precedidas de asterisco). Por eso, si se dispone de unas coordenadas calculadas en otro sistema, hay que pasarlas siempre al SCP actual mediante el comando TRANS.

En la Base de Datos de AutoCAD los puntos característicos de cada entidad se encuentran expresados en el Sistema de Coordenadas de la entidad SCE. Es necesario siempre tener en cuenta cual es el SCP actual y utilizar el comando TRANS para convertir esos puntos.

El Sistema de Coordenadas Visualización (SCV) es el sistema hacia el cual se convierten las imágenes antes de ser visualizadas en pantalla. Su origen es el centro de 1a pantalla y el eje Z la línea de visión (perpendicular hacia la pantalla). Es importante cuando se pretende controlar como van a visualizarse las entidades.

Si el punto o vector desplazamiento indicado en TRANS en 2D, el propio comando lo convierte en 3D suministrando la coordenada Z que falta. Esta coordenada Z dependerá de cual es el Sistema de Coordenadas desde el cual se considera el punto:

Si es el Universal (SCU), la coordenada Z es Ø.Ø.
Si es el Personal actual (SCP), la coordenada Z es el valor de la elevación actual.
Si es el Sistema de la Entidad (SCE), Z es Ø.Ø.
Si es el Sistema de Visualización (SCV), el valor de Z es la proyección del punto en el plano XY actual de acuerdo con la elevación actual.

Este comando presenta alguna modificación en AutoLISP 11 (véase el apartado 1732 del Capitulo 17).

10.7 (VPORTS) Configuración de Ventanas actual.

Este comando devuelve una lista con la configuración de ventanas actual. La lista contiene en forma de sublistas los descriptores de todas las ventanas de la configuración actual. Cada descriptor es, a su vez, una lista con tres elementos: número de identificación de ventana (correspondiente a la Variable de Sistema de AutoCAD CVPORT), esquina inferior izquierda y esquina superior derecha de cada ventana.

Las dos esquinas aparecen en fracciones de anchura y altura de pantalla, igual que en el listado de la orden VENTANAS opción "?". Así, (Ø Ø) corresponde al vértice inferior izquierdo de la pantalla (1 1) al superior derecho y (Ø.5 Ø.5) al centro de la pantalla.

Por ejemplo, si la configuración actual en pantalla es de cuatro ventanas iguales de tamaño, el comando VPORTS podría devolver:

```
(
   ( 3 ( Ø.5 Ø.5 ) ( 1.Ø 1.Ø ) )
   ( 1 ( Ø.5 Ø.Ø ) ( 1.Ø Ø.5 ) )
   ( 6 ( Ø.Ø Ø.5 ) ( Ø.5 1.Ø ) )
   ( 9 (Ø.Ø Ø.Ø ) ( Ø.5 Ø.5 ) )
)
```

El primer número de identificación que aparece (en el ejemplo, el 3) es el de la ventana activa actual.

10.8 (REDRAW [<nom-ent> [<mod>]]) Redibujar.

Este comando efectúa un redibujado. Si se suministra sin ningún argumento, redibuja toda la ventana gráfica actual. Equivale en este caso a la orden REDIBUJA de AuToCAD:

(REDRAW)

Si se indica un nombre de entidad como argumento en <nom-ent>, solamente esa entidad será redibujada Esto es interesante cuando se ha despejado la ventana gráfica con el comando (GRCLEAR), que se estudiará más adelante. Con REDRAW se pueden entonces redibujar las entidades que se deseen y sólo éstas se harán visibles en el área granea.

(REDRAW <nom-ent>)

Una aplicación práctica puede ser cuando se seleccionan entidades con utilización de filtros (comando SSGET, que se estudiará en el capitulo siguiente). Por ejemplo, en un dibujo de circuito electrónico se desea averiguar cuántos bloques de resistencia se encuentran insertados. Con el comando SSGET se extraen de la Base de Datos todas las inserciones de ese bloque. Con GRCLEAR se despeja la ventana gráfica actual y con REDRAW proporcionando el nombre de entidad de todas las inserciones de bloque extraídas se irían redibujando en la pantalla.

Modo	**Efecto**
1	Redibujar la entidad en pantalla
2	Despejar la entidad en pantalla
3	Visualizar la entidad en relieve (vídeo inverso o doble intensidad)
4	Suprimir la visualización en relieve

Si el nombre de entidad indicado es una entidad compuesta (bloque con atributos o polilínea), el efecto de REDRAW afecta a la entidad principal con todos sus componentes simples. Si el código de modo se indica con signo menos (-1, -2, -3 o -4), el efecto de REDRAW sólo afectará a la entidad principal (el "encabezamiento” en la Base de Datos).

10.9 Ejemplo 1: Archivos de foto de múltiples ventanas.

La Versión 1Ø de AutoCAD bajo el entorno de Sistema Operativo MS-DOS permite hasta un máximo de cuatro ventanas en la pantalla. De ellas sólo una se encuentra establecida como ventana activa actual. La utilidad de las ventanas múltiples es que permiten visualizar al mismo tiempo en la pantalla diferentes aspectos o puntos de vista del dibujo (normalmente en 3D).

Si el usuario ha establecido una configuración de ventanas con esos diferentes puntos de vista, incluyendo por ejemplo supresión de líneas no visibles con la orden HIDE, y pretende obtener un archivo de "foto" (extensión .SLD), no es posible hacerlo con todas las ventanas a la vez. Seria preciso ir cambiándose a cada una de las ventanas y utilizar la orden MSLIDE cada vez. Para proyectar esas fotos habría que realizar la misma operación con VSLIDE.

El programa de este ejemplo realiza esa operación de una sola vez, y de forma más cómoda para el usuario. En la versión más, sencilla, esta previsto para un máximo de cuatro ventanas. En otros Sistemas Operativos, como, por ejemplo, UNIX, AutoCAD V. 1Ø permite hasta dieciséis ventanas. También existe una versión de AutoCAD 386 que permite dieciséis ventanas bajo MS-DOS.

El programa revisado para funcionar con cualquier número de ventanas se encuentra en el capítulo siguiente, como ejemplo de aplicación de los comandos para manejar listas.

Por último, la más reciente versión AutoCAD V. 11 incorpora la posibilidad de definir un "Espacio de papel" (orden PSPACE) donde se pueden proyectar diferentes vistas del dibujo e incluso modelizaciones ("Shades"). Con la orden MSLIDE se obtiene un único archivo .SLD con el conjunto de todas las vistas y modelos.

El programa aquí explicado tiene, por tanto, su mayor utilidad para quien no disponga de la última versión, pero en todo caso sirve para ilustrar el mecanismo de acceso a Variables de AutoCAD.

10.9.1 Listado del programa

```
(DEFUN intr ()
  (INITGET 6)
  (IF (SETQ nv (GETINT "Número de ventanas configuración actual
<1>: "))
    ()
    (SETQ nv 1)
  )
  (TERPRI)
  (WHILE (> nv 4)
    (PROMPT "Número máximo es 4")
    (TERPRI)
    (INITGET 6)
    (IF (SETQ nv (GETINT "Número de ventanas configuración actual
<1>:  "))
      ()
      (SETQ nv 1)
    )
    (TERPRI)
  )
```

```
  (IF (SETQ cam (GETSTRING  "Unidad y directorio para almacenar
fotos <C:> "))
    ()
    (SETQ cam "C:\\")
  )
  (SETQ pref (GETSTRING "Prefijo para nombre de fotos (hasta 6
caracteres ) : "))
  (TERPRI)
  (SETQ pref (SUBSTR pref 1 6))
)

(DEFUN sacaf (/ n vact)
  (SETQ vact (GETVAR "cvport"))
  (SETVAR "cvport" 1)
  (COMMAND "mslide" (STRCAT cam pref "$ 1"))
  (SETQ n 1)
  (REPEAT (- nv 1)
    (SETVAR "cvport" (* n 3))
    (COMMAND "mslide" (STRCAT cam pref "$" (ITOA (+ n 1))))
    (SETQ n (+ n 1))
  )
  (SETVAR "cvport" vact)
)

(DEFUN c:sacatodo (/ cam pref nv)
;;;  (SETVAR "blipmode"  0)
;;;  (SETVAR "cmdecho" 0)
  (intr)
  (sacaf)
;;;  (SETVAR "blipmode" 1)
;;;  (SETVAR "cmdecho" 1)
  (PRIN1)
)

(DEFUN busca ()
  (WHILE (NOT (FINDFILE (STRCAT cam pref "$1.sld")))
    (PROMPT "Archivo de foto no encontrado: ")
    (TERPRI)
    (IF (SETQ cam (GETSTRING "Unidad y directorio para almacenar
fotos <C:> "))
      ()
      (SETQ cam "C:\\")
    )
    (SETQ pref (GETSTRING "Prefijo para nombre de fotos (hasta 6
caracteres) : " ))
    (TERPRI)
    (SETQ pref (SUBSTR pref 1 6))
  )
)
```

```
(DEFUN miraf (/ n vact)
  (SETQ n 1)
  (SETQ vact (GETVAR "cvport"))
  (SETVAR "cvport" 1)
  (COMMAND "vslide" (STRCAT cam pref "$1"))
  (REPEAT (- nv 1)
    (SETVAR "cvport" (* n 3))
    (COMMAND "vslide" (STRCAT cam pref "$" (ITOA (+ n 1))))
    (SETQ n (+ n 1))
  )
  (SETVAR "cvport" vact)
)

(DEFUN c:miratodo (/ cam pref nv)
;;;  (SETVAR "biipmode" 0)
;;;  (SETVAR "cmdecho" 0)
  (intr)
  (busca)
  (miraf)
;;;  (SETVAR "blipmode" 1)
;;;  (SETVAR "cmdecho" 1)
  (PRIN1)
)

(PROMPT "Ordenes nuevas SACATODO y MIRATODO definidas")
(PRIN1)
```

10.9.2 Observaciones

Este programa accede a la Variable de Sistema de AutoCAD llamada "cvport", que almacena un número con la ventana activa actual.

La mayor efectividad del programa se obtendría accediendo a la lista con la configuración de ventanas actual mediante el comando (VPORTS). La forma de realizar esto se explica en el capítulo siguiente.

De momento es el propio usuario el que tiene que indicar el número de ventanas de la configuración actual cuando el programa se lo pida.

El programa define dos nuevas órdenes de AutoCAD. Al cargarlo con LOAD sólo devolvería el nombre de la última función definida. Para que el usuario sepa que existen dos nuevas órdenes, el programa visualiza un mensaje con PROMPT.

10.9.3 Explicación del programa

Define dos nuevas órdenes de AutoCAD, las cuales llaman a cuatro funciones de usuario. Además, visualiza un mensaje al final cuando se carga con LOAD.

• Función **c:sacatodo**

Define una nueva orden. Desactiva marcas auxiliares y eco de menú y llama a las funciones "intr" y "sacaf". Establece como variables locales todas las que quedan pendientes de las funciones intermedias

• Función **intr**

En primer lugar solicita el número de ventanas de la configuración actual mediante GETINT. Como es valor no puede ser Ø ni negativo, establece ambas condiciones con INITGET 6 (suma de 4 + 2). Permite el valor nulo para que el usuario pueda acoplar la opción por defecto <1> pulsando "Return". Se utiliza el mecanismo ya explicado en otros programas de emplear el SETQ como condición de una alternativa IF.

El número máximo de ventanas previsto en el programa es de 4. Por eso se establece un control mediante WHILE para visualizar un mensaje de advertencia y volver a pedir el dato si el usuario indica un valor mayor de 4.

Al filial de este proceso, la variable "nv" almacena el numero de ventanas. Sólo podrá contener los valores 1, 2, 3 ó 4.

Se podría establecer un control más sencillo; por ejemplo, de la siguiente forma:

```
( SETQ nv Ø )
( WHILE ( OR ( < nv 1 ) ( > nv 4 ) )
-
-
-
)
```

El primer valor es para obligar al programa a entrar en el cielo. Después ya no sale hasta que el valor solicitado pura nv" esté entre 1 y 4.

A continuación el programa necesita los datos para crear los archivos de "foto", uno para cada ventana. La orden SACAFOTO solicita el nombre de archivo. Hay que indicarlo precedido del camino si se pretende grabar en otra unidad o en otro directorio diferente del actual.

Así, por ejemplo, si se quiere grabar un archivo de foto con el nombre "detalle" en un directorio "fotos" de la unidad A:, habría que hacer:

Command: ***SACAFOTO***
Nombre de fichero de foto: ***a: \fotos \detalle***

El programa solicita en primer lugar la unidad y directorio para almacenarlo en la variable "cam". Como opción por defecto ofrece C: para que los archivos de foto queden en el directorio actual de la unidad de disco. Si el usuario tiene instalado AutoCAD o esta trabajando en otra unidad (por ejemplo, D:), debe tener la precaución de indicarlo. Para el programador basta con cambiarlo en el

programa si le conviene.

Hay que indicar la última contrabarra (o /, que también es admitida por AutoCAD). El programa, para diferenciar sus archivos de foto de otros que puedan existir, va a incluir el carácter $ y un número del 1 al 4. Por eso solicita del usuario un prefijo de hasta seis caracteres, dado que el número máximo de un archivo es de 8 en MS-DOS.

Así, por ejemplo, si el usuario va a grabar las fotos en su disquete, en el directorio "fotos" y co" el nombre "chalet", la secuencia sería:

Unidad y directorio para almacenar fotos <C:> ***a: \fotos***
Prefijo para nombre de fotos (hasta 6 caracteres) : ***chalet***

Si la configuración actual es de cuatro ventanas, el programa creará cuatro archivos de foto llamados:

chalet$1, chalet$2, chalet$3, chalet$4

Si el usuario introduce mas de seis caracteres, el programa los trunca seleccionando la subcadena con SUBSTR de los seis primeros caracteres. Se almacenan en la variable "cam".

Lógicamente, si el usuario indica una unidad de disco no existente o un directorio equivocado, el programa producirá un error al intentar grabar la primera foto y quedara abortado.

Esta misma función va a servir para visualizar las fotos al ser llamada por la nueva orden "c:miratodo". También en este caso se necesita especificar el número de ventanas y los datos de unidad, directorio y prefijo de seis caracteres para buscar las fotos. Por eso no es necesario volver a escribirlo.

Variables de entrada: ninguna.

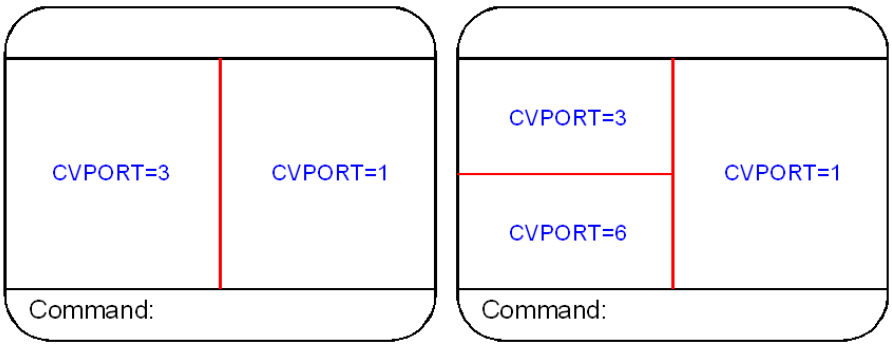
Variables locales: ninguna.

Variables de salida:

nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.

• Función **sacaf**

La Variable de Sistema de AutoCAO “cvport” almacena el número de la ventana activa actual. Su valor es siempre 1, 3, 6 ó 9, según el número de ventanas (Figuras 10.1 y 10.2).



Ejemplos de configuración de dos y tres ventanas.

La función "sacaf" almacena en primer lugar el valor de la ventana actual en "vact", para restituirla al final. Así el usuario permanece en la ventana en que estaba cuando llama al programa.

Después activa la primera ventana introduciendo con GETVAR el valor 1 en "cvport". Ya se puede sacar la primera foto. El nombre del archivo se obtiene concatenando con STRCAT las variables "cam" y "pref.", como se ha explicado en la función anterior, y "$l". Si el usuario ha escogido la opción defecto C: y, por ejemplo, un prefijo "chalet", el comando STRCAT devolverá "c:chalet$l" con lo esa foto se grabará en el directorio actual del disco duro.

Para obtener las fotos de las demás ventanas se utiliza una variable de control "n", que se inicializa a un valor de 1. Se establece un ciclo repetitivo con "nv – 1". Si el número de ventanas es, por ejemplo 4, el ciclo se repetirá tres veces, pues la primera foto ya esta obtenida fuera del ciclo.

Dentro del ciclo, la segunda ventana tiene en "cvport" un valor de 3, la tercera de 6 y la cuarta de 9. Por tanto, basta multiplicar el valor de "n" por 3. Para el nombre del archivo de foto, éstos serán:

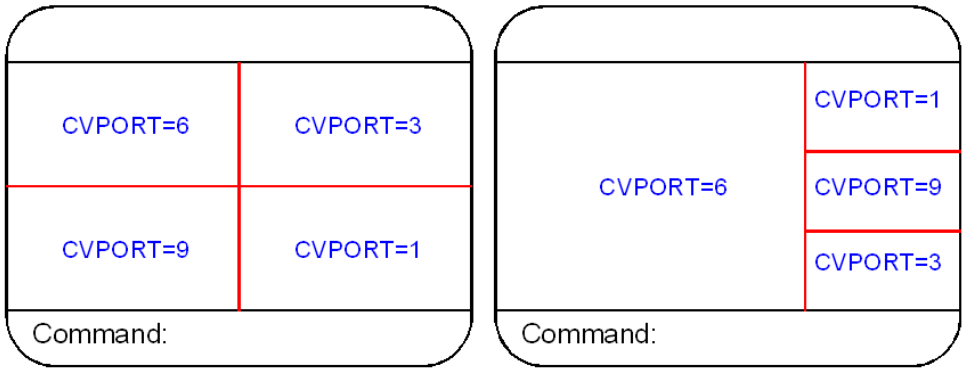
chalet$2, chalet$3 y chalet$4

Por tanto, hay que concatenar una cadena con "cam". "pref." y el carácter dólar más un número que en n + 1 convertido en cadena con ITOA.

A continuación se saca la foto con SACAFOTO y se actualiza la variable "n" sumando 1.

Lo último que queda es dejar activa la ventana que estaba, restituyendo a "cvport" su valor original.

Si el usuario indica un número de ventanas equivocado (por ejemplo 4, cuando en realidad son sólo 2), el programa no produce error. Simplemente los dos últimos valores de "cvport" no producen efecto y la foto de la última ventana se repetiría dos veces mas. Habría cuatro archivos de foto pero los tres últimos contendrían la misma foto. Este inconveniente queda subsanado en la versión ya mencionada, que se encuentra en el capítulo siguiente:



Ejemplos de dos configuraciones con cuatro ventanas.

Variables de entrada:

nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.

Variables locales:

n Contador y control del ciclo.
vact Valor actual de variable “cvport".

Variables de salida: ninguna.

• Función **c:miratodo**

Desactiva marcas auxiliares y eco de órdenes. Llama a la función "intr" ya explicada y después a las funciones "busca” y "miraf”.

• Función **busca**

Tras la llamada a la función "intr", la orden "miratodo" necesita localizar los archivos de foto para proyectarlos en las diferentes ventanas. Con el fin de detectar el posible error de que no se encuentren (con lo que el programa quedaría abortado), se llama a esta función.

Consiste en un ciclo establecido con WHILE, que impone como condición que se pueda localizar el primer archivo de foto "........$.SLD" especificado. Para ello se utiliza el comando FINDDILE. Una vez que el usuario ha indicado la unidad y directorio y el prefijo de hasta seis caracteres, se concatena con STRCAT el nombre completo del primer archivo de foto. Por ejemplo, si "cam" contiene "C:" y la variable "pref" contiene "chalet", el nombre del archivo seria:

c:chalet$1.sld

El comando FINDFILE busca ese archivo y si no lo encuentra devuelve nil. En este caso el comando NOT (negación lógica) devuelve "T” (cierto) y la condición se cumple, con lo que el ciclo visualiza un mensaje de advertencia y vuelve a solicitar los datos.

Se puede producir un error si, por ejemplo, sólo hay dos archivos de foto y la

configuración actual es de tres ventanas, Al intentar localizar la tercera foto el programa quedaría abortado. Esto se puede solucionar haciendo que FINDFILE localice todos los archivos de acuerdo con el valor de "nv". Pero no merece la pena complicar demasiado el programa con estos aspectos.

Variables de entrada:nv Número de ventanas configuración actual.cam Unidad y directorio de almacenamientopref. Prefijo para nombre de fotos.

Variables locales: ninguna.

Variables de salida

nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.

• Función **miraf**

Su estructura y funcionamiento es totalmente similar a la función "sacaf" ya explicadasin más que cambiar la llamada a la orden de AutoCAD SACAFOTO por una llamada a laorden MIRAFOTO.

Es posible, para no repetir otra vez todas las expresiones definir una función, llamadapor ejemplo “fotos”, que dependa de una variable de texto “txt”. La llamada a las órdenes deAutoCAD se hace depender entonces del valor de esa variable.

```
( DEFUN fotos ( txt / n vact )
        ( SETQ vact .....)
        ……..
        ……..
        ( COMMAND txt ( STRCAT cam pref "$1" ) )
        ……..
        ( REPEAT ....
                ……
                ( COMMAND txt ( STRCAT cam pref  "$" ( ITOA ( + n 1 )
                ) ) )
                …….
        )
)
```

Para utilizar esta función en la nueva orden "Sacatodo", bastará con especificar la llamada:

```
( fotos "sacafoto" )
```

En la nueva orden "Miratodo" la llamada seria:

```
( fotos "mirafoto" )
```

Así, en cada caso, la variable "txt" en la función "fotos" se sustituye por

"Sacafoto" o "Mirafoto" según convenga.

En el ejemplo se ha preferido dejar ambas funciones separadas, por razones didácticas.

Variables de entrada:

nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.

Variables locales:

n Contador y control del ciclo.
vact Valor actual de variable "cvport".

Variables de salida: ninguna.

10.10 Ejemplo 2: Inserción de bloques unitarios partiendo líneas.

El programa incluido en este ejemplo permite la inserción de bloques en medio de líneas ya dibujadas, partiendo automáticamente el tramo de línea ocupado por el bloque. Puede servir, por tanto, para insertar bloques en cualquier tipo de circuitos: resistencias, diodos, transistores, etc., en círculos electrónicos; válvulas en circuitos de tuberías... También serviría por ejemplo, para insertar puertas y ventanas en la línea de un tabique en un dibujo de construcción, etc.

La condición impuesta es que el bloque sea unitario en la dirección de la línea del circuito a partir. Es decir, que en su definición mida una unidad. Esto es necesario para que el programa pueda calcular los dos puntos entre los cuales debe abrir la línea.

Con el fin de facilitar el proceso, el programa hace uso de la Variable de Sistema "osnap" ya explicada para establecer en cada caso los modos de referencia más adecuados.

10.10.1 Listado del programa

```
(DEFUN intr (/ ap)
  (GRAPHSCR)
  (SETQ nbl (GETSTRING "Nombre de bloque unitario: "))
  (TERPRI)
  (SETQ os (GETVAR "osmode"))
  (SETQ ap (GETVAR "aperture"))
  (SETVAR "osmode" 512)
  (SETVAR "aperture" 5)
  (INITGET 1)
  (SETQ pti (GETPOINT "Punto de inserción CERca a: "))
  (TERPRI)
```

```
  (INITGET 2)
  (IF (SETQ escx (GETREAL "Factor de escala X <1>: "))
    ()
    (SETQ escx 1)
  )
  (TERPRI)
  (INITGET 2)
  (IF (SETQ   escy (GETREAL (STRCAT "Factor de escala Y <" (RTOS
escx 2 2) ">: " ) ) )
    ()
    (SETQ escy escx)
  )
  (TERPRI)
  (IF (SETQ ang (GETANGLE pti "Ángulo de rotación < 0 >: "))
    ()
    (SETQ ang 0)
  )
  (TERPRI)
  (SETVAR "aperture" ap)
)

(DEFUN insbl (/ pf dis pck pent angins)
  (SETQ angins (/ (* ang 180) PI))
  (COMMAND "insert" nbl pti escx escy angins)
  (SETQ pf (POLAR pti ang escx))
  (SETQ      dis      (*  2  (/  (GETVAR  "viewsize")  (CAR  (GETVAR
"screensize")))))
  (SETQ pck (GETVAR "pickbox"))
  (SETVAR "pickbox" 1)
  (SETQ pent (POLAR pti (+ ang PI) dis))
  (SETVAR "osmode" 0)
  (COMMAND "break" pent "f" pti pf)
  (REDRAW)
  (SETVAR "pickbox" pck)
  (SETVAR "osmode" os)
)

(DEFUN c:intercala (/ os nbl pti escx escy ang)
;;;  (SETVAR "blipmode" 0)
;;;  (SETVAR "cmdecho" 0)
  (intr)
  (insbl)
;;;  (SETVAR "blipmode" 1)
;;;  (SETVAR "cmdecho" 1)
  (PRIN1)
)
```

10.10.2 Observaciones

El programa accede a determinadas Variables de Sistema de AutoCAD y

modifica sus valores de acuerdo con las necesidades en cada momento. Esto supone que los valores actuales de esas variables se pierden.

En concreto se modifican los valores de las variables "osmode". "pickbox y apertura". Con el fin de que el usuario no vea alterados los valores que el utiliza, una vez que ha llamado al programa se recurre a una practica habitual que es almacenar esos valores para después volver a restituirlos.

En teoría debería hacerse lo misino con las variables "blipmode" y "cmdecho" antes de desactivarlas. Pero en todos los ejemplos se ha supuesto, por comodidad, que interesa mantenerlas activadas. Por ese motivo todos los programas finalizan poniéndolas a 1.

También se accede en e! ejemplo a dos variables interesantes: "viewsize" y "screensize" Son de sólo lectura, no se pueden alterar. Pero su valor es útil para saber en qué escala de visualización se encuentra el dibujo en cada momento.

10.10.3 Explicación del programa

Se define una nueva orden de AutoCAD y dos funciones de usuario.

• Función **c:intercala**

Desactiva marcas auxiliares y eco de órdenes y llama a la dos funciones de usuario "intr" e "insbl". Establece como variables locales las que quedaban pendientes de las funciones intermedias.

• Función **intr**

Solicita, como siempre, los datos necesarios para el programa. En primer lugar conmuta a pantalla gráfica por si el usuario se encontraba en pantalla de texto. Después requiere el nombre del bloque unitario que se pretende insertar. Lógicamente, si este bloque no existe, el programa dará error y quedará abortado.

Si el bloque es un dibujo externo, se podría detectar su existencia con el comando FINDFILE, como se ha hecho en el ejemplo anterior. Pero si es un bloque definido en el dibujo, habría que explorar la Base de Datos para detectarlo. De momento el programa se deja así.

A continuación el programa va a modificar el valor de las Variables de Sistema "aperture" y "osnap". Por eso previamente almacena su contenido actual en "ap" y "os".

Con el fin de que el usuario disponga del modo de referencia CERca para indicar el punto de inserción del bloque, establece un valor de 512 para "osnap". Al mismo tiempo establece un valor 5 para "aperture", considerando que es un tamaño cómodo. La dimensión de esta apertura en pantalla dependerá de la resolución de la tarjeta gráfica y el monitor que se estén

empleando.

Se solicita ese punto de inserción del bloque con un comando GETPOINT. Previamente, con INITGET 1 se impiden valores nulos para que el usuario no pueda introducir, por ejemplo, "Return". Se supone que el punto de inserción forma parte de alguna línea previamente dibujada. Si el usuario señala un punto cualquiera que no pertenece a una línea, el programa lo acepta también, pero puede fallar a la hora de intentar partir una entidad no existente o no prevista.

Por supuesto, si el usuario quiere indicar otro modo de referencia diferente de CERca, puede hacerlo sin problemas.

El siguiente valor es la escala de inserción del bloque. Se iniciativa con INITGET 2 para no permitir el valor Ø. Con esta salvedad, el factor de escala puede ser cualquier número real positivo o negativo. Se ofrece la opción por defecto 1 para que el usuario la acepte con sólo pulsar "Return".

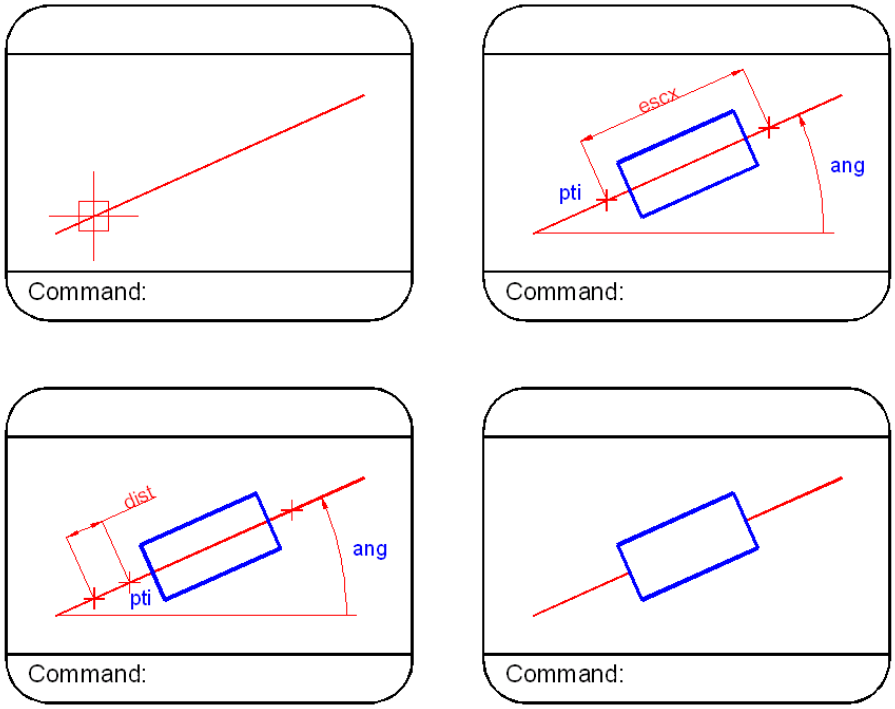


Figura 10.3. Datos iniciales para la inserción del bloque unitario.

Se solicita un factor de escala en Y por si el usuario quiere indicarlo. La opción por defecto es un valor igual al de escala X. Para visualizar este valor en la escala de texto se concatena el valor de “escx” recién introducido con (RTOS escx 2 2), es decir, en modo decimal y con dos decimales de precisión.

Por último se solicita el ángulo de inserción proponiendo por defecto el valor Ø. El usuario puede introducir ese valor en las unidades actuales (por ejemplo, grados sexagesimales), pero el comando GETANGLE devuelve el valor en radianes. Se admite cualquier valor positivo, negativo o nulo (sería el “Return” para la opción por defecto).

El último paso es restablecer el valor primitivo de “aperture”, puesto que ya no se va a necesitar. No se altera de momento “osmode”, pues se va a utilizar más adelante.

Variables de entrada; ninguna.

Variables locales:
ap Valor original de "aperture".

Variables de salida:
nbl Nombre del bloque unitario.
pti Punto de inserción del bloque.
escx Factor de escala X de inserción.
escy Factor de escala Y de inserción.
ang Ángulo de inserción en radianes.
os Valor original de "osmode".

• Función **insb**

La orden de AutoCAD "INSERT" requiere el valor del ángulo en grados (se supone aquí que el usuario no tiene establecidas otro tipo de unidades). Como la variable "ang" contiene ese ángulo en radianes, se crea una variable "angins" con el valor en grados.

A continuación se procede a insertar el bloque con los valores indicados.

Se calculan ahora los dos puntos por los que hay que partir la línea en la cual se ha insertado el bloque. El primer punto será el de inserción "pti". El segundo, "pf", se obtiene llevando una distancia igual a la escala en X (por eso el bloque tiene que ser unitario en su definición) un ángulo en radianes "ang".

Para partir la línea, la orden "parte" requiere en primer lugar designar la entidad a partir y la designación debe ser por un punto. El programa calcula entonces un punto adecuado a una distancia suficiente para no designar el bloque recién insertado, pero al mismo tiempo lo bastante cerca para no pillar otra entidad no deseada. Como esa distancia va a depender de la escala de visualización actual, el programa calcula la dimensión en unidades de dibujo de cada punto de luz o "pixel" en la pantalla.

La Variable de Sistema "viewsize" de sólo lectura contiene la altura actual de visualización en unidades de dibujo. La variable "screensize" contiene esa misma altura pero en "pixels", de acuerdo con la precisión de la tarjeta gráfica instalada. Su cociente proporciona la relación deseada. Se multiplica por dos para asegurar que el punto de designación no va a pillar el bloque.

Hay que tener en cuenta que "screensize" contiene una lista con los valores de altura y anchura. Por eso hay que usar CAR para extraer el primer elemento de la lista.

La distancia "dis", por tanto, contiene el valor en unidades de dibujo, que corresponde a dos "pixels" en la pantalla. El programa lleva esa distancia desde "pti" hasta un ángulo de "ang" + Pl/2. El punto resultante "pent" es el de designación de la entidad.

Antes de llamar a la orden “PARTE", se establece un modo de referencia nulo para que no afecte a la designación de la entidad a partir. La orden parte requiere en primer lugar la designación, que será "pent". Después se especifica la opción "p" para que pida el primer punto, que es "pti . El segundo punto de la partición es "pf”.

Se redibuja para apreciar bien en la pantalla la operación y se restituyen las variables pickbox y "osmode" a sus valores originales.

Variables de entrada:

nbl Nombre del bloque unitario.
pti Punto de inserción del bloque.
escx Factor de escala X de inserción.
escy Factor de escala Y de inserción.
ang Ángulo de inserción en radianes.
os Valor original de "osmode".

Variables locales

pf Punto final para partir línea.
dis Distancia desde inserción, para designar línea a partir.
pent Punto para designar línea a partir.
pck Valor original de "pickbox".
angins Ángulo de inserción en grados.

Variables de salida: ninguna.

11 Capitulo 11: Otros comandos de operaciones con listas.

Se detallan en este capitulo una serie de comandos de manipulación de listas que complementan a los ya vistos en el Capítulo 3. Puesto que, como ya se ha repetido, la Base de Datos del Dibujo en AutoCAD se encuentra organizada a base de listas, todos ellos tienen su utilidad.

Sin embargo, los más habituales en la gestión de la Base de Datos de AutoCAD son los comandos ASSOC, CONS y SUBST, que se explican en primer lugar.

11.1 (ASSOC <elem> <list>) Lista asociada.

Este comando busca el elemento especificado <elem> en la lista <list> y devuelve la lista asociada cuyo primer elemento es <elem>. Por eso la lista <list> debe ser una lista de asociaciones, que contenga sublistas con elementos asociados. Este tipo de listas son las que contiene la Base de Datos del dibujo, como se explica en el capítulo siguiente.

Por ejemplo, una lista que contenga cuatro listas de asociaciones podría ser:

((cuerpo poliedro) (largo 25) (alto 12) (ancho 6))

Si la lista se llama "II", el comando ASSOC devolvería:

(ASSOC ' cuerpo 11) devuelve (CUERPO POLIEDRO)
(ASSOC ' alto 11) devuelve (ALTO 12)
(ASSOC ' volumen 11) devuelve nil

11.2 (CONS <pr-elem> <list>) Añadir primer elelemento a la lista.

Este comando toma la lisia indicada en <list> y le añade un nuevo primer elemento <prelem>, devolviendo la nueva lista resultante.

Command:
(SETQ p1 ' (1Ø 35 Ø))
(SETQ p2 ' (2Ø 35 Ø))
(SETQ p3 (CONS (CAR p1) (CDR p2)))

En el ejemplo, p3 se construye tomando la lista (CDR p2) que son las coordenadas Y Z del punto “p2” y añadiendo por delante como nuevo primer elemento (CAR p1) que es la coordenada X del punto “p1”

Si como segundo elemento <list> se especifica no una lista sino un valor concreto o de una variable, entonces CONS construye un tipo especial de lista

de dos elementos llamada par punteado, puesto que tiene un punto como separación entre ambos elementos. Los pares punteados ocupan menos memoria que las listas normales y son muy utilizados en la Base de Datos del dibujo en AutoCAD.

Con los comandos CAR y CDR se pueden extraer directamente el primer y segundo elemento de los pares punteados.

(CONS ' tipo 32) devuelve (TIPO . 32)

11.3 (SUBST <nue-elem> <ant-elem> <lis>) Sustituir elemento.

Este comando busca en la lista indicada <lis> el elemento actual <ant-elem> y lo sustituye por el nuevo elemento <nue-elem>, devolviendo la nueva lista resultante. Si el elemento actual no se encuentra, devuelve la lista <lis> tal como esta, sin cambios.

Si, por ejemplo, "lin" contiene la lista de una línea que se encuentra en la capa "Ø"\ la lista asociada correspondiente es (8 . “Ø”). Si se quiere cambiar la entidad a una nueva capa sería:

(SUBST ' (8 . "Pieza") ' (8 . "Ø") lin)

El nuevo par asociado en "lin" es (8 . "Pieza") que corresponde a una capa "Pieza". Para actualizar en la Base de Datos habría que emplear ENTMOD como se verá en el Capítulo 13.

11.4 (APPEND <list1 > <list2>,..) Juntar listas.

Este comando reúne todos los elementos de las listas especificadas <list1>, <tist2>, etc., en una lista que los engloba.

Command : ***(SETQ 1t (APPEND ' (vi v2) ' (v3 v4)))***

La variable "lt" almacena la lista final.

! lt devuelve (vi v2 v3 v4)

Se observa que APPEND no devuelve ((vi v2) (v3 v4)). Es decir, lo que reúne son los elementos de las listas indicadas y no las listas mismas, Otro ejemplo:

(APPEND ' (v1 (v2 v3)) '(v4 (v5))) devuelve (v1 (v2 v3) v4 (v5))

Los argumentos de APPEND tienen que ser siempre listas.

Los siguientes ejemplos serian incorrectos:

(APPEND (v1 v2) (v3 v4)) listas deben ser literales
(APPEND ' (vi v2) v3) v3 no es lista sino elemento

En esto se diferencia del comando LIST estudiado en el Capitulo 3, que reúne elementos y forma una lista con ellos.

11.5 (LENGTH <list>) Longitud de lista

Este comando devuelve un número entero, que es la longitud de la lista indicada <list>. Es decir, devuelve el número de elementos contenidos en la lista.

(LENGTH ' (n 1 p1 p2 p3)) devuelve 5
(LENGTH ' (n 1 (p1 p2 p3))) devuelve 3
(LENGTH ' ( )) devuelve Ø

11.6 (LAST <list>) Ultimo elemento de lista.

Devuelve el último elemento contenido en la lista especificada.

(LAST ' (n 1 v1 v2)) devuelve v2
(LAST ' (n 1 (v1 v2))) devuelve (vi v2)
(LAST ' (n)) devuelve n
(LAST ' ( )) devuelve nil

Se observa que LAST puede devolver un átomo o una lista, según sea el último elemento de <list>. Según lo explicado, el comando LAST extrae directamente la coordenada Z de una lista que represente un punto. Pero no es aconsejable emplearlo para este cometido, porque si el punto se ha indicado en 2D con sólo dos coordenadas, LAST devolvería la Y. En cambio, con CADDR se tendría la seguridad de obtener siempre la Z, pues devolvería "nil" en caso de no existir.

Por estas razones y por su utilidad para gestionar la Base de Datos donde se encuentran almacenados los puntos del dibujo, es preferible emplear CAR, CDR y sus combinaciones como se ha explicado en el Capítulo 3.

11.7 (MEMBER <elem> <list>) Lista a partir de elemento.

Este comando busca el elemento indicado <elem> en la lista <list> y devuelve el resto de la lista a partir e elemento, incluido el mismo. Si el elemento se repite en la lista, MEMBER busca la primera aparición y devuelve el resto de la lista. Si el elemento indicado no está incluido en la lista, MEMBER devuelve "nil".

(MEMBER ' x ' (n 1 x s t1 t2)) devuelve (x s t1 t2)
(MEMBER ' s' (n 1 s t1 s t2) devuelve (s t1 s t2)
(MEMBER ' 1 ' (n 1 (t1 t2) 1)) devuelve (1 (t1 t2) 1)
(MEMBER ' t1 ' (n 1 (t1t2) s)) devuelve nil.

En el último caso, "t1" no existe como elemento en la lista indicada. En cambio, si se hace:

(MEMBER ' (t1 t2) ' (n 1 (t1 t2) s)) devuelve ((t1t2) s)

11.8 (NTH <num> <list>) Elemento enésimo de lista.

Este comando devuelve el elemento "enésimo" de la lista indicada <list>. Es decir, examina la lista y devuelve el elemento situado en la posición indicada en <num>. Hay que tener cuidado, pues el primer ciclo de la lista es el número Ø. Así el último elemento de una lista con cuatro elementos ocuparía la posición 3.

Si la posición especificada <num> es mayor que la posición del último elemento de la lista, NTH devuelve “nil”.

(NTH 2 ' (v1 n x p1 p2 p3)) devuelve x
(NTH 0 ' (v1 n x p1 p2 p3)) devuelve vi
(NTH 1 ' (n (p1 p2 p3) 1 s)) devuelve (p1p2 p3)
(NTH 5 ' (n x s 1 v1)) devuelve nil

Se observa que NTH puede devolver un átomo o una lista. Si, por ejemplo, la variable "p1" contiene una lista de punto:

(NTH 0 p1) devuelve coordenada X
(NTH 1 p1) devuelve coordenada Y
(NTH 2 p1) devuelve coordenada Z

11.9 (REVERSE <list>) Lista invertida.

Este comando devuelve la lista indicada <list> con todos sus elementos invertidos de orden.

(REVERSE ' (v1 v2 v3 v4)) devuelve (v4 v3 v2 v1)
(REVERSE ' (x y (p1 p2 p3) z)) devuelve (z (p1 p2 p3) y x)
(REVERSE ' (x)) devuelve (x)
(REVERSE ' ( )) devuelve nil

11.10 (FOREACH <nom> <list> <expr>) Aplicar expresióna lista.

Este comando procesa cada elemento de la lista <list> y les aplica la expresión <expr>. Para ello utiliza como símbolo el indicado en <nom>, que debe aparecer dentro de la expresión <expr>. Entonces el modo de funcionamiento es el siguiente; toma cada elemento de la lista y lo hace intervenir en la expresión en los lugares en que aparece el símbolo <nom>. Después evalúa cada una de las expresiones resultantes para cada elemento de la lista.

(FOREACH p ' (p1p2 p3) (PRINTp))

equivale a:

(PRINT p1)
(PRINT p2)
(PRINT p3)

El comando FOREACH devuelve el resultado de la última expresión evaluada (el comando PRINT se ha estudiado ya en el Capítulo 9).

Hay que tener precaución con la manera en que se utiliza este comando. Por ejemplo:

Command: ***(SETQ p1 ' (20 20 0) p2 ' (50 50 0) p3 ' (100 100 0))***
Command: ***(FOREACH p ' (p1 p2 p3) (PRINT p))***
p1
p2
p3

Con SETQ se han almacenado en las variables "p1", "p2" y "p3" tres puntos en tres dimensiones (listas de tres elementos). Al aplicar FOREACH al literal de la lista formada por los tres símbolos, éstos no participan con el valor de punto que contienen en ese momento. Entonces el resultado de la expresión (PRINT p) aplicada al literal de la lista es imprimir en la línea de órdenes los literales de los símbolos, no sus valores.

La lista a la que interesa aplicar (PRINT p) es la que contiene los valores actuales de punto. Y para formar esa lista es preciso emplear el comando LIST ya conocido. Si se hace:

(LIST p1 p2 p3) devuelve ((2Ø 2Ø Ø) (5Ø 5Ø Ø) (1ØØ 1ØØ Ø))

Esta es precisamente la lista que nos interesa. Entonces la manera correcta de emplear FOREACH en este caso seria:

Command: ***(FOREACH p (LIST p1 p2 p3) (PRINT p))***
(2Ø.Ø 2Ø.Ø Ø.Ø)
(5Ø.Ø 5Ø.Ø Ø.Ø)
(1ØØ.Ø 1ØØ.Ø Ø.Ø)

Ahora si se imprimen en la línea de órdenes los valores de punto contenidos en las variables. Es equivalente a haber hecho:

(FOREACH p ' ((20 20 0) (50 50 0) (100 100 0)) (PRINTp))

Otro ejemplo de aplicación de este comando para dibujar líneas desde el punto "Ø,Ø" hasta cada uno de los tres puntos anteriores sería:

(FOREACH p (LIST p1 p2 p3) (COMMAND "línea" "0,0" p " "))

11.11 (APPLY <func> <list>) Aplicar función a la

lista.

Este comando aplica la función dada <fune> a todos los elementos de la lista indicada <list>. La función <func> puede ser inherente (comando de AutoLISP) o bien una función de usuario. Esta función toma como argumentos todos los elementos de la lista <list>.

(APPLY ' * ' (3 2 4)) devuelve 24
(APPLY ' strcat ' ("Buenos" " " "días")) devuelve "Buenos días"

Se observa que tanto el nombre del comando o función como la lista son literales (se les aplica el mando QUOTE). Lo mismo que en el apartado anterior, si se pretende aplicar una función a los valores actuales de variables, hay que utilizar el comando LIST.

(SETQ x 5 y 6 2 3)
(APPLY ' * (LIST x y z)) devuelve 9Ø
(SETQ a "Un " b "ejemplo" c " sencillo.")
(APPLY ' strcat (LIST a b c)) devuelve "Un ejemplo sencillo."

11.12 (MAPCAR <func> <list1>...<listn>» Aplicar función a elementos sucesivos de listas.

Este comando aplica la función <func> tomando como argumentos el primer elemento de <list1>, el primer elemento de <list2> y así sucesivamente hasta el primer elemento de <list2>. Por tanto, el número de listas indicadas debe coincidir con el número de argumentos requeridos para la función <func>.

El resultado de aplicar <func> se almacena como primer elemento de la lista de resultados que va a volver MARCAR, A continuación se aplica <func> tomando como argumentos los segundos elementos cada una de las listas. El resultado se almacena como segundo elemento en la lista de resultados. Y así sucesivamente con el tercer elemento, el cuarto, etc., hasta cubrir lodos los elementos contenidos en las listas.

Se observa, por tanto, que las listas <list1> a <listn> indicadas deben tener todas el mismo numero de elementos.

Command: ***(MAPCAR '+ ' (5 4 6) ' (2 3 3) ' (Ø1 3))***
(7 8 12)

El ejemplo es equivalente a hacer:

(+ 5 2 Ø)
(+ 4 3 1)
(+ 6 3 3)

La diferencia es que MAPCAR devuelve los tres resultados agrupados en una lista.

Hay que tener en cuenta la misma consideración del comando anterior respecto a los literales de listas con variables. Si se hace

(SETQ x 5 y 7 z 1Ø)
(MAPCAR '+ '(x y z) ' (2 2 2))

intentaría aplicar "+" a los literales de "x" “y” "z" y daría error. Para formar una lista con los valores actuales de esas tres variables habría que utilizar el comando LIST y entonces se haría:

(MAPCAR '+ (LIST x y z) ' (2 2 2)) devuelve (7 9 12)

La utilidad fundamental de este comando respecto al anterior de APPLY es que permite aplicar la función especificada más de una vez. Con APPLY sólo se podía especificar una lista y la función tomaba todos los elementos de esa lista como argumentos. Con MARCAR se especifican tantas listas como argumentos y el número de elementos de cada lista es el número de veces que se aplica la función.

Volviendo al ejemplo sencillo de las líneas, se puede definir una función que dibuje líneas entre dos puntos, especificando como variables asociadas a la función "pin" y "pf", que serian los puntos extremos de la línea a dibujar.

La definición seria:

(DEFUN dblin (pin pf) (COMMAND "línea" pin pf " "))

Se supone que existen seis variables que almacenan seis valores de punto. Esos valores de punto pueden haber sido introducidos por el usuario a través de comandos GETPOINT, por ejemplo. Las variables se pueden llamar "pin1", "pin2", "pin3", "pf1", "pf2" o "pf3".

Para dibujar directamente las tres líneas entre cada par de puntos, se podría hacer con MARCAR de la siguiente forma:

(KAPCAR ' dblin (LIST pin1 pin2 pin3) (LIST pf1 pf2 pf3))

Sería equivalente a hacer:

(dblin pin1 pf1)
(dblin pin2 pf2)
(dblin pin3 pf3)

En este caso MAPCAR, dibujaría las tres líneas y devolvería una lista (nil nil nil), puesto que la función de usuario "dblin" llama a COMMAND y éste siempre devuelve "nil".

11.13 (LAMBDA <lista-argum> <exp>) Definir función temporal.

Este comando define una función de usuario sin nombre. Su formato y funcionamiento es el mismo que DEFUN pero al no tener nombre, sólo puede utilizarse en el momento de definirla y no puede ser llamada posteriormente. Se utiliza cuando se necesita definir una función sólo momentáneamente y no se desea ocupar espacio en la memoria de manera innecesaria

El comando LAMBDA devuelve el valor de la última expresión evaluada en <expr>, lo mismo que DEFUN. Se usa sobre todo en combinación con los comandos APPLY y MAPCAR, para aplicar una función temporal a los elementos de una o varias listas.

(APPLY ' (LAMBDA (x y z) (/ (- x y) z))' (25 5 2)) devuelve 1Ø

En el ejemplo se define una función temporal con LAMBDA que tiene tres variables. Su cometido es restar X - Y y dividir esa resta entre Z. Se aplica esa función con APPLY a la lista que suministra los tres argumentos requeridos. El resultado será (25 - 5) dividido por 2, es decir, 1Ø.

De manera similar, se utiliza con MARCAR cuando se quiere obtener una lista de resultados. Por ejemplo, en el apartado anterior se ha definido una función llamada "dblin" para dibujar líneas. Si no se desea almacenar en la memoria esa función, se define sin nombre con el comando LAMBDA y se aplica directamente a las listas que contienen los puntos iniciales y finales de las líneas.

(MAPCAR ' (LAMBDA (pin pf) (COMMAND "línea" pin pf " "))(LIST pin1 pin2 piri3)(LIST pf1 pf2 pf3))
devuelve (nil nil nil)

El ejemplo dibujaría las tres líneas y devolvería una lista con los tres resultados de COMMAND, que son siempre nil.

11.14 Ejemplo 1: Archivos de foto de múltiples ventanas para más de cuatro ventanas.

Como se ha dicho ya, este ejemplo propone el mismo programa del capítulo anterior (Ejemplo 1), pero completado con el tratamiento de la lista de configuración de ventanas.

Gracias al examen de esta lista se puede conocer el número de ventanas de la configuración actual sin necesidad de que sea el usuario quien lo indique.

Esto hace que el programa funcione para cualquier número de ventanas (hasta 16 bajo Sistema Operativo UNIX o en versión de AutoCAD para equipos 386).

11.14.1 Listado del programa

```
(DEFUN intr ()
  (SETQ nv (LENGTH (VPORTS)))
  (IF (SETQ cam (GETSTRING "Unidad y directorio para
almacenar fotos <C:> " ))

    (SETQ cam "C:\\")
    ()
  )
  (SETQ pref (GETSTRING "Prefijo para nombre de fotos ( hasta 6
caracteres ):  " ))
  (TERPRI)
  (SETQ pref (SUBSTR pref 1 6))
)

(DEFUN sacaf (/ n vact)
  (SETQ vact (CAAR (VPORTS)))
  (SETQ n 0)
  (REPEAT nv
    (SETVAR "cvport" (CAR (NTH n (VPORTS))))
    (COMMAND "mslide" (STRCAT cam pref "$" (ITOA (+ n 1))))
    (SETQ n (+ n 1))
  )
  (SETVAR "cvport" vact)
)

(DEFUN c:sacatodo ();(/ cam pref nv)
;;;    (SETVAR "blipmode" 0)
;;;    (SETVAR "cmdecho" 0)
    (intr)
    (sacaf)
    (SETVAR "blipmode" 1)
    (SETVAR "cmdecho" 1)
    (PRIN1)
)

(DEFUN busca (/ n)
  (SETQ nf 0)
  (WHILE (= nf 0)
    (intr)
    (SETQ n 1)
    (WHILE (FINDFILE (STRCAT cam pref "$" (ITOA n) ".sld"))
      (SETQ n (+ n 1))
    )
    (SETQ nf (- n 1))
    (PROMPT (STRCAT "Encontrados " (ITOA nf) " archivos de foto "
))
    (TERPRI)
  )
)
```

```
(DEFUN miraf (/ n vact)
  (SETQ vact (CAAR (VPORTS)))
  (SETQ n 0)
  (REPEAT nv
    (SETVAR "cvport" (CAR (NTH n (VPORTS))))
    (COMMAND "vslide" (STRCAT cam pref "$" (ITOA (+ n 1))))
    (COMMAND "delay" 5000)
  )
  (SETQ n (+ n 1))
  (SETVAR "cvport" vact)
)

(DEFUN opcion     ()
  (IF    (= nv nf)
    (miraf)
    (PROGN
      (PROMPT "Número de archivos de foto no coincide con num.
ventanas " )
      (TERPRI)
      (INITGET "Si No")
      (IF (SETQ op (GETKWORD "Continuar? <No>: " ))
        ()
        (SETQ op "No")
      )
      (IF (= op "Si")
          (PROGN
          (miraf)
            (COMMAND "delay" 5000)
          )
      )
    )
  )
)

(DEFUN c:miratodo (/ cam pref nv)
;;;  (SETVAR "blipmode" 0)
;;;  (SETVAR "cmdecho" 0)
  (busca)
  (opcion)
  (SETVAR "blipmode" 1)
  (SETVAR "cmdecho" 1)
  (PRIN1)
  (PROMPT "Órdenes nuevas SACATODO y MIRATODO definidas")
  (PRIN1)
)
```

11.14.2 Observaciones

Aprovechando que ahora es posible conocer desde el programa la

configuración de ventanas, se completa la función de "busca" con el fin de detectar el número de archivos de foto creados con la nueva orden "sacatodo". Al pretender utilizar la nueva orden "miratodo", se detecta si el número de ventanas actual coincide con el número de fotos grabadas.

En caso de que no coincidan, se visualiza un mensaje de advertencia y se da al usuario la posibilidad de continuar o de dar por terminado en ese momento el programa.

Para conservar la estructura modular del programa, utilizan funciones de usuario que llaman a su vez a otras funciones de usuario.

11.14.3 Explicación del programa

Define dos nuevas órdenes de AutoCAD, y cinco funciones más de usuario.

• Función **c:sacatodo**

Permanece inalterada respecto a la versión de este programa ya explicada en el Ejemplo 1 del Capítulo 10. Llama simplemente a las funciones “intr" y "sacaf".

• Función **intr**

Presenta una única modificación respecto a la versión del capítulo anterior. En vez de ser el usuario el que indique el número de ventanas de la configuración actual, éste se obtiene directamente de la forma:

(SETQ nv (LENGTH (VPORTS)))

El comando VPORTS devuelve una lista con todas las ventanas de la configuración actual. El comando LENGTH devuelve el número de elementos de esa lista, que será el número de ventanas. La variable "nv" almacena ese valor.

A partir de ahí la función solicita la unidad y directorio, y el prefijo para los nombres de los archivos de foto.

Variables de entrada: ninguna.

Variables locales: ninguna.

Variables de salida:

nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.

• Función **sacaf**

Para explorar todas las ventanas de la configuración actual se utiliza la lista

devuelta por VPORTS. Cada elemento de esa lista corresponde a una ventana. El comando NTH extrae el elemento enésimo, empezando por Ø. Cada uno de esos elementos es a su vez una lista, cuyo primer elemento es el número de esa ventana. Por tanto, el número que identifica a la ventana enésima será:

(CAR (NTH n (VPORTS))

Basta introducir ese número en la Variable de Sistema de AutoCAD "cvport" para poder obtener archivo de foto de esa ventana con la orden "sacatodo"

Ejemplo de una configuración de ventanas en AutoCAD 386.

Previamente se ha almacenado en “vact” el número de la ventana activa actual, con el fin de restituirlo al final de la función. Al principio del ciclo se inicializa la variable "n" a Ø.

Los nombres de los archivos de foto se forman con el criterio ya explicado en el capítulo anterior.

Variables de entrada:

Variables locales:

nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.
n Contador y control del ciclo.
vact Valor actual de variable "cvport". • Variables de salida: ninguna.

• Función **c:miratodo**

Sufre importantes modificaciones respecto a la versión del Capitulo 10. Llama a dos funciones de usuario. Una de ellas, “busca", agrupa a la función del mismo nombre y a la llamada a "intr" del capítulo anterior. Además llama a una nueva función que es "opción", que engloba la llamada a "miraf”.

• Función **busca**

Sufre modificaciones respecto a la explicada en el Capitulo 10. En primer lugar se crea una variable "nf". Contiene el número de archivos de foto encontrados. Se le asigna un valor Ø para obligar al programa a recorrer la repetitiva la primera vez.

Se establece esa repetitiva con WHILE para ejecutarla mientras se cumpla la condición de que nf = Ø. Dentro del ciclo se llama a la función "intr". Esta determina el número de ventanas de la configuración actual, la unidad y directorio, y el prefijo de los archivos de foto a proyectar en este caso.

Se establece un segundo ciclo incluido dentro del anterior, con la variable de control "n". Este ciclo busca todos los archivos de foto que responden al prefijo especificado en orden correlativo, desde el que termina con $1 en adelante.

Por ejemplo:

chalet$1.sld chalet$2.sld chalet$3.sld …….

En el momento en que no encuentra uno, FINDFILE devuelve nil y termina el ciclo. El valor de n - 1 en ese momento contiene el número de archivos de foto correlativos encontrados. Se almacena en nf. Se visualiza un mensaje que indica el número encontrado.

Si es Ø, se volvería al principio de la primera repetitiva, que volvería a llamar a "intr" para solicitar de nuevo los datos de unidad, directorio y prefijo. Esto se repetirá hasta que se localice por lo menos un archivo de foto.

Variables de entrada: ninguna.

Variables locales:
n Número de archivo de foto o localiza

Variables de salida:
nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.
nf Número de archivos de foto localizados.

• Función **opcion**

Es una función nueva respecto a la versión del capitulo anterior.

Consiste en una alternativa que chequea la condición de que el número de ventanas de la configuración actual contenido en "nv" coincida con el número de archivos de foto localizados "nf". Si coinciden, no hay ningún problema; se llama a la función "mirar" y se proyectan las fotos.

Si no coinciden, se ejecuta la rama del NO de la alternativa, agrupada en un PROGN. Se visualiza n mensaje de advertencia y se solicita del usuario una decisión para continuar o no con la ejecución del programa.

Para ello se emplea GETKWORD con la opción por defecto de "No" continuar. El comando INITGET previo sólo permite las respuestas "S", "Si", "N" o "No", en mayúsculas o minúsculas. Se almacena esa respuesta en "op".

Se establece una nueva alternativa que chequea esa respuesta, Para ello basta examinar las permitidas en INITGET. Si es afirmativa, se llama a la función "mirar" y se proyectan las fotos. Si es negativa, la una del NO de la alternativa se encuentra vacía y el programa termina sin proyectar nada.

Variables de entrada:
nv Número de ventanas configuración actual.
cam Unidad y directorio de almacenamiento.
pref Prefijo para nombre de fotos.

nf Número de archivos de foto localizados.

Variables locales:
op Opción para continuar o no,

Variables de salida: ninguna.

• Función **miraf**

Totalmente similar a "sacar" ya explicada, cambiando la llamada a la orden de AutoCAD "sacafoto" por la orden "mirafoto". La única diferencia es el número de veces de la repetitiva indicado en REPEAT. Es e1 menor de "nv" (número de ventanas) y "nf (número de archivos encontrados). Así no se produce error al intentar proyectar un archivo que no existe.

Variables de entrada: nv Número de ventanas configuración actual. cam Unidad y directorio de almacenamiento. pref. Prefijo para nombre de fotos.

Variables locales: n Contador y control del ciclo. vact Valor actual de variable "cvport".

Variables de salida: ninguna

12 Capitulo 12: Organización de la Base de Datos de AutoCAD.

En este capitulo se pasa revista a un aspecto fundamental a la hora de utilizar los programas en AultLISP: el acceso directo a la Base de Datos de AutoCAD, sobre cuya estructura se proporcionará una introducción para conocer cómo se encuentran organizados los datos de entidades en ella, para en capítulos posteriores, explicar cómo se pueden gestionar desde AutoLISP.

Esta facilidad para acceder y modificar directamente la Base de Datos de un dibujo es lo que proporciona potencia real a AutoLISP.

12.1 Estructura de la Base de Datos para entidades.

Todas las entidades de un dibujo de AutoCAD se encuentran contenidas en la Base de Datos. Esta consiste en un conjunto de listas, una para cada entidad, más las listas correspondientes a tablas de símbolos, cuya función se estudia en el apartado correspondiente.

La lista de cada entidad es, a su vez un conjunto de listas de asociaciones. Cada asociación contiene dos valores; el primero es el código que indica el tipo de dato contenido en el segundo elemento, y éste contiene el dato concreto (coordenadas de punto, ángulo, nombre, etc.).

Según el tipo del dato almacenado, la lista de asociación podría contener más elementos Por ejemplo cuando se trata de un punto, contiene cuatro elementos: el código y las tres coordenadas del punto.

En otros casos contiene el color, el tipo de línea, la capa en que se encuentra la entidad, etc.

Por ejemplo, la lista correspondiente a una línea podría ser la siguiente:

((-1 . <Entity name: 6ØØØØ32C>)(Ø . "LINE")(8 . "PIEZA")(62 . 1)(6 . "TRAZOS")(1Ø Ø.Ø Ø.Ø Ø.Ø)(11 1Ø.Ø 1Ø.Ø Ø.Ø))

Se trata de una lista que contiene siete listas incluidas.

Cada una de ellas representa lo siguiente:

(-1 . <Entity name: 6ØØØØ32C>)

El código -1 es específico para el nombre de la entidad de que se trata. El punto que sepata los dos elementos de la lista indicada que se trata de un par punteado. El segundo elemento de la lista contiene el “nombre” de la entidad, que es en realidad un número de identificación en hexadecimal para distinguir esa entidad del resto.

(0 . "LINE")

El código Ø es específico para el tipo de entidad. El segundo elemento de esta lista, que es un par punteado, indica el tipo de entidad. En este caso se trata de una "línea". Hay que tener en cuenta que el tipo de entidad viene siempre en inglés y en mayúsculas, por lo que hay que indicarlo así en los programas de AutoLISP que acceden a la Base de Datos. Además se trata de una cadena de texto y por eso va entre comillas.

(8 . "PIEZA")

El código 8 es indicativo de la capa en que se encuentra la entidad. En este caso es la capa PIEZA. El segundo elemento de la lista es la cadena de texto con el nombre de la capa siempre en mayúsculas.

(62 . 1)

El código 62 es indicativo del número de color de la entidad. El segundo elemento de la lista (que es un par punteado) especifica ese número de color. En este caso es el 1, que corresponde al rojo.

(6 . "TRAZOS")

El código 6 es indicativo del tipo de línea de la entidad. El segundo elemento de la lista de asociación es el nombre del tipo de línea, entre comillas, por tratarse de un texto, y en mayúsculas.

(1Ø Ø.Ø Ø.Ø Ø.Ø)

En este caso el código 1Ø es indicativo del punto inicial de la línea. Para otro tipo de entidades puede cambiar. La lista ya no es un par punteado, sino que continuar cuatro elementos (los puntos son los decimales de los tres valores numéricos), Los tres elementos que van tras el código son las coordenadas de ese punto inicial, en este caso (X = Ø. Y = Ø, Z = Ø).

(11 1Ø.Ø 1Ø.Ø Ø.Ø)

De manera similar a la lista anterior, el código 11 indica para las entidades que son líneas el punto final. En este caso sus tres coordenadas son (X = 1Ø, Y = 1Ø, Z = Ø).

Con todas estas listas, la entidad línea queda perfectamente definida.

Para las entidades compuestas, del tipo de las polilíneas, la estructura de la Base de Datos cambia un poco. Por ese motivo se estudian en un apartado diferente.

12.1.1 Entidades simples

Las listas de asociaciones contienen todas las propiedades y características

que definen la entidad. Al ser listas, pueden tratarse sin problemas desde un programa en AutoLISP con los comandos relativos al manejo de listas.

Para acceder a las propiedades y características de la entidad es preciso ante todo conocer los códigos normalizados que se encuentran en el primer elemento de cada sublista de asociaciones. Estos códigos pueden ser comunes a todas las entidades o depender del tipo de entidad de que se trate.

Así, por ejemplo, el código 8, que indica la capa en que se encuentra la entidad, es independiente del tipo de entidad.

En cambio el código 1Ø puede indicar el punto inicial para una línea, la inserción en una entidad de punto, el centro para un circulo o arco, el punto de inserción para un texto o bloque, etc.

Hay que tener en cuenta un aspecto muy importante cuando se trabaja en tres dimensiones: las coordenadas de estos puntos característicos se encuentran referidas al Sistema de Coordenadas de la Entidad (SCE). Por tanto, para poder trabajar con ellos en el SCP actual habrá que convertirlos previamente con el comando TRANS.

Los códigos empleados en las sublistas de asociaciones de la Base de Datos se detallan en los cuadros siguientes, divididos en dos grupos; códigos comunes a todas las entidades y códigos que cambian según el tipo de entidad.

Cuadro 12.1.

Códigos comunes	**Tipo de dato**
- 1	Nombre de la entidad.
Ø	Tipo de entidad
6	Nombre de tipo de línea. Si no aparece, se asume que es “porcapa
7	Nombre de estilo de texto.
8	Nombre de capa.
38	Valor de elevación. Si no aparece, se asume que es Ø.
39	Valor de altura de objeto. Si no aparece, se asume que es Ø.
62	Número de color (Ø = porbloque, 256 = porcapa). Si no aparece, es "porcapa".
66	Señal de que el bloque contiene atributos.
210	Vector de orientación de la altura de objeto en 3D.

Cuadro 12.2

Otros Códigos	**Tipo de dato para cada tipo de entidad**
1	TEXTO, ATRIBUTO – Contenido de texto
2	BLOQUE – Nombre ATRIBUTO – Identificador

1Ø	LINEA – Punto inicial CIRCULO, ARCO – Centro TEXTO, ATRIBUTO – Punto inicio izquierda BLOQUE – Punto inserción PUNTO – Coordenadas POLILINEA – Punto inicial SÓLIDO, 3DCARA – Primer vértice TRAZO – Primera esquina punto inicial
11	LINEA – Punto final TEXTO, ATRIBUTO – Punto de justificación SÓLIDO, 3DCARA – Segundo vértice TRAZO – Segunda esquina punto inicial
12	SÓLIDO, 3DCARA – Tercer vértice TRAZO – Primera esquina punto final.
13	SÓLIDO, 3DCARA – Cuarto vértice TRAZO – Segunda esquina punto final.
4Ø	ARCO, CÍRCULO – Radio TEXTO, ATRIBUTO – Altura texto POLILINEA – Grosor inicial
41	TEXTO, ATRIBUTO – Factor proporción texto BLOQUE – Factor escala X. POLILINEA – Grosor final
42	BLOQUE – Factor escala Y. POLILINEA – Factor curvatura poliarco.
43	BLOQUE – Factor escala Z
44	BLOQUE – Distancia entre columnas INSERTM.
45	BLOQUE – Distancia entre filas INSERTM.
5Ø	ARCO – Ángulo inicio TEXTO, ATRIBUTO – Ángulo rotación BLOQUE – Factor escala X. POLILINEA – Dirección tang adaptar curva.

Otros Códigos	**Tipo de dato para cada tipo de entidad**
51	ARCO – Ángulo final. TEXTO, ATRIBUTO – Ángulo inclinación estilo
7Ø	POLILINEA – Señal de abierta o cerrada BLOQUE – Número columnas de INSERTM ATRIBUTO – Señal tipo atributo: Valor Ø – Normal Valor 1 – Invisible Valor 2 – Constante Valor 4 – Verificable
71	BLOQUE – Número de filas de INSERTM TEXTO, ATRIBUTO – Señal generación texto: Valor Ø – Generación normal Valor 2 – Texto reflejado Valor 4 – Texto cabeza abajo
72	TEXTO, ATRIBUTO – Señal tipo justificación: Valor Ø – Izquierda. Valor 1 – Centro. Valor 2 – Derecha. Valor 3 – Situar. Valor 4 – Rodear Valor 5 – Ajustar

12.1.2 Entidades compuestas

Las entidades compuestas que aparecen en la Base de Datos pueden ser de dos tipos:

1. Polilíneas: Tanto polilíneas 2D como 3D y también mallas.
2. Bloques con atributos.

Hay que tener en cuenta que las inserciones de un bloque son referencias de ese bloque y están consideradas como entidades simples. La definición de las entidades que forman ese bloque se encuentra en la Tabla de Símbolos, como se explicará en el apartado siguiente.

Únicamente si el bloque contiene atributos se considera como una entidad compuesta, de forma que la referencia del bloque es la entidad global y cada uno de, los atributos las entidades simples incluidas en él.

• **Polilíneas**

Las entidades compuestas del tipo de las polilíneas aparecen en la Base de Datos de la siguiente manera:

Primero la entidad compuesta, con la lista correspondiente (tipo de entidad "POLYLINE"), conteniendo las propiedades y características globales de la polilinea. Es la llamada "cabecera" de la entidad en la Base de Datos,

A continuación, un conjunto de listas de los vértices de la polilinea, una lista para cada vértice, con el tipo de entidad "VERTEX" (lista de asociación con el código Ø) y conteniendo las propiedades y características individuales de cada elemento de la polilinea.

Por último, una lista de "Fin de secuencia", que es un tipo de entidad especial llamado "SEQEND" (código Ø), que indica el final de las listas de vértices y, por tanto, de la polilinea. Esta entidad contiene un código especial, -2, que indica el nombre de la entidad principal, es decir, de la "cabecera".

Para saber cuántos vértices tiene una polilinea hay que explorar entonces la Base de Datos desde la entidad principal o "cabecera" hasta encontrar un tipo de entidad "SEQEND", que indicará que ahí terminan los vértices de dicha polilinea.

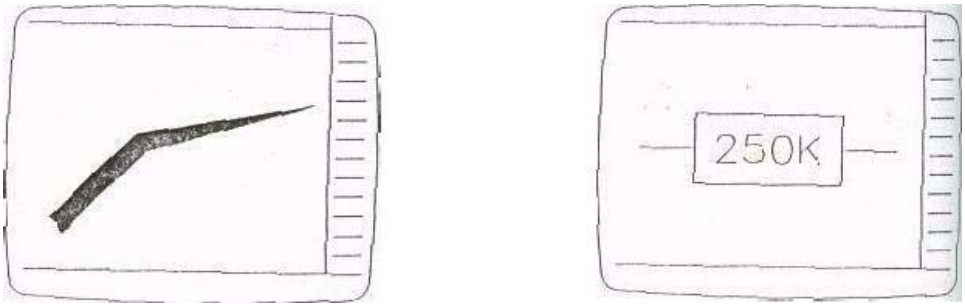
Así, por ejemplo, una polilínea sencilla (Figura I2.l), de sólo dos segmentos rectos con diferentes espesores podría contener en la Base de Datos las siguientes listas:

```
( ( -1 . <Entity ñame: 6ØØØØØØ3Ø> )( Ø . "POLYLINE" )
        ( 8 . "PIEZA" )
        ( 66 . 1 )( 1Ø Ø.Ø  Ø.Ø Ø.Ø )( 7Ø . Ø )( 4Ø . 5.Ø )
        ( 41 . 5.Ø )( 21Ø Ø.Ø Ø.Ø 1.Ø )
        ( 71 . Ø )
```

```
        ( 72 . Ø )
        ( 73 . Ø )
        ( 74 . Ø )
        ( 75 . Ø )
)
```

Esta primera lista de cabecera contiene el tipo de entidad "POLYLINE", que es la entidad principal o compuesta. En este caso se encuentra en la capa “PIEZA" Su color es "porcada”, pues no aparece ninguna 1ista con el código 62. El grosor inicial y el grosor final globales de la polilínea (códigos 4Ø y 41) son ambos de 5.Ø. El resto de sublistas de asociaciones hacen referencia a aspectos como, si la polilínea es abierta o cerrada, si está adaptada la curva, la orientación de su altura de objeto respecto al SCP actual, etcétera.

```
( ( -1 . <Entity name: 6ØØØØØ48> )( Ø . "VERTEX")
        ( 8 . "PIEZA" )( 1Ø 1ØØ.Ø  11Ø.Ø Ø.Ø )
```



Polilínea y bloque con atributo para examinar en la Base de Datos.

```
        ( 7Ø . Ø )
        ( 4Ø . 5.Ø )
        ( 41 . 5. Ø )
        ( 42 . Ø.Ø )
        ( 5Ø . Ø.Ø )
)
```

Esta segunda lista, con el tipo de entidad "VERTEX", condene las sublistas de asociaciones para el primer vértice de la polilínea. Su capa es "PIEZA"; las coordenadas (código 1Ø) son en este caso (X = 1ØØ Y = 11Ø Z= Ø). El grosor inicial y final del segmento que empieza en ese vértice (códigos 4Ø y 41) son ambos 5.Ø. El resto de datos se refieren a si hay curva adaptada, la dirección de la tangente, etc.

```
( ( -1 . <Entity name: 6ØØØØØ6Ø> )( Ø . "VERTEX" )
        ( 8 . "PIEZA" )( 1Ø 12Ø.Ø 13Ø.Ø Ø.Ø )( 7Ø . Ø )( 4Ø . 5.Ø )
        ( 41 . Ø.Ø )
        ( 42 . Ø.Ø )
        ( 5Ø . Ø.Ø )
)
```

Esta tercera lista corresponde al segundo vértice. En este caso sus coordenadas son (X = 12Ø Y = 13Ø Z = Ø). El grosor inicial del segmento que

empieza en ese vértice es 5, pero el grosor final aquí es Ø.

```
( ( -1 . <Entity name: 6ØØØØØ78> )( Ø . "VERTEX" )
        ( 8 . "PIEZA" )( 1Ø 16Ø.Ø 14Ø.Ø  Ø.Ø )( 7Ø . Ø )
        ( 41 . Ø.Ø )
        ( 42 . Ø.Ø )
        ( 5Ø . Ø.Ø )
)
```

Esta cuarta lista contiene los datos del tercer vértice. Sus coordenadas son (X = 160 Y = 14Ø Z = Ø). Al ser una polilínea abierta, el grosor inicial y final coinciden, puesto que no hay ningún segmento que parta de este vértice. El grosor en este caso es Ø.

```
( ( -1 . <Entity name: 6ØØØØØ9Ø> )
        ( 0 . "SEQEND" )
        ( 8 . "PIEZA" )
        ( -2 . <Entity name: 6ØØØØØ3Ø> )
)
```

Esta última lista indica que la polilínea ya no contiene más vértices. El tipo de entidad es “SEQEND”. El código -2 indica el nombre de entidad de cabecera que repite el ya indicado en la primera lista.

• Referencia de bloque

Las referencias de bloque que contienen atributos se organizan de la misma manera; en primer lugar, la lista con la referencia de bloque (tipo de entidad "INSERT"); después, las listas para cada atributo (tipo de entidad "ATTRIB"), y, por último, la lista de fin de secuencia, "SEQEND".

Así, por ejemplo, un sencillo bloque de una resistencia que contiene un atributo con el valor de esa resistencia (Figura: 12.1) tendría en la Base de Datos las siguientes listas de asociaciones.

```
( ( -1 . <Entity name: 6ØØØØ1Ø> )( Ø . "INSERT" )
        ( 8 . "ELEMENTOS" )
        ( 66 . 1 )
        ( 2 . "RES" )( 1Ø 8Ø.Ø 1ØØ.Ø Ø.Ø )
        ( 41 . 1.Ø )
        ( 42 . 1.Ø )( 5Ø . Ø.Ø )
        ( 43 . 1.Ø )( 7Ø . Ø )
        ( 71 . Ø )
        ( 44 . Ø.Ø )
        ( 45 . Ø.Ø )
        ( 21Ø Ø.Ø Ø.Ø  1.Ø )
)
```

El tipo de entidad "INSERT"' indica que se trata de una referencia de bloque. El bloque se llama "RES" (código 2). Se encuentra insertado en la capa

"ELEMENTOS". El código 66 con un valor 1 indica que el bloque contiene atributos. El código 1Ø indica las coordenadas del punto de inserción del bloque. Los códigos 41, 42 y 43 indican las escalas X, Y y Z de inserción del bloque (en este caso, ambas 1). El código 5Ø es el ángulo de rotación de la inserción (aquí, Ø). Los códigos 70, 71, 44 y 45 son para las inserciones múltiples con la orden "INSERTM", y en este ejemplo tienen todos el valor Ø. El último código es para la orientación de la Altura de Objeto.

La siguiente lista que aparece en la Base de Datos es la correspondiente al atributo incluido en el bloque:

```
( ( -1 . <Entity name: 6ØØØØ12Ø> )
        ( 0 . "ATTRIB" )
        ( 8 . "ELEMENTOS" )( 1Ø 12Ø.Ø 8Ø.Ø Ø.Ø )( 4Ø . 16.Ø )
        ( 1 . "25ØK" )
        ( 2 . "RESIS" )( 7Ø . 4 )
        ( 73 . Ø )( 5Ø . Ø.Ø )
        ( 41 . 1.Ø )
        ( 51 . Ø.Ø )
        ( 7 . "TS1" )
        ( 71 . Ø )
        ( 72 . 4( 11 15Ø.Ø 1ØØ.Ø Ø.Ø )( 21Ø Ø.Ø Ø.Ø 1.Ø )
)
```

El tipo de entidad es “ATTRIB" y se encuentra en la misma capa que la inserción del bloque. El punto de inserción del texto del atributo es (X = 12Ø Y = 8Ø Z = Ø). La altura de este texto (código 4Ø) es 16. Los códigos 1 y 2 indican el valor del atribulo (aquí es "25ØK") y el identificador (aquí "RESIS”). El código 7Ø con valor 4 indica que se trata de un Atributo Verificable. El código 5Ø indica ángulo de rotación del texto Ø. Los códigos siguientes hacen referencia al estilo de texto del atribulo. En este caso tiene factor de proporción 1, ángulo de inclinación Ø, estilo "TS1”, generación normal, y es un texto con opción "Rodear" (código 72 con valor 4). El punto indicado para "Rodear" está en el código 11.

La última lista con el tipo de entidad "SEQEND", puesto que ya no hay más atributos, sería la siguiente:

```
( ( -1 . <Entity name:  6ØØØØ138> )( Ø . "SEQEND" )
     ( 8 . "ELEMENTOS" )( -2 . <Entity name: 6ØØØØ1l38> )
```

El último código -2 contiene el nombre de la entidad principal o de "cabecera", que es la referencia del bloque.

Con el comando ENTNEXT (se explicará en el capítulo siguiente) se pueden recorrer tanto 1os vértices de una polilínea como los atributos de un bloque, hasta encontrar el tipo de entidad SEQEND. De esta forma es posible acceder directamente a las subentidades que forman esas entidades compuestas.

En AutoLISP 11, las Referencias Externas de Bloque y los sólidos son tratados

como referencias de bloque. AutoLISP 11 se explica en el Capítulo 17.

12.2 Estructura de la Base de Datos para tablas de símbolos.

Las "Tablas de Símbolos" consisten también en una serie de listas similares a las precedentes, pero no contienen entidades, sino definiciones y características propias de AutoCAD, y con las que se trabaja en el entorno del dibujo.
Estas características son concretamente siete:

- Capas.
- Estilos de texto.
- Tipos de línea.
- Definiciones de bloques.
- Vistas almacenadas.
- SCPs almacenados,
- Configuraciones de ventanas almacenadas.

Su organización es la misma que para entidades, a base de sublistas de asociaciones, con un código como primer elemento, que indica el tipo de dato que viene a continuación. Los códigos se muestran en el Cuadro 12.3.

Cuadro 12.3

Código	**Tipo de dato**
Ø	Tipo de símbolo: “LAYER” – Capa “STYLE” – Estilo de texto “VIEW” – Vista almacenada “LTYPE” – Tipo de línea “BLOCK” – Definición de bloque “UCS” – SCP almacenado “VPORT” – Configuración de ventanas
2	Nombre de capa, estilo, vista, etc.
3	Archivo tipo de letra estilo texto.
5	Nombre de tipo de línea asociado a capa.
1Ø	Coordenadas origen del bloque.
4Ø	Altura estilo texto.
41	Factor de proporción estilo texto.
5Ø	Ángulo inclinación estilo texto.
62	Número de color asociado a capa. Si es negativo, significa que está DESACTIVADO.
-2	Nombre de entidad que forma parte de una definición de bloque.

En el caso de las definiciones de bloque, a continuación de las características globales vienen las listas con todas las entidades que componen el bloque.

En AutoCAD V. 11 se incorporan dos nuevas Tablas de Símbolos: "DIMSTYLE" y "APPID". Sus características se explican en el Capítulo 17.

• Capas

La lista de definición de una capa en la Tabla de Símbolos podría ser la siguiente:

```
( ( Ø . "LAYER" )
        ( 2 . "EJES" )
        ( 7Ø . 64 )
        ( 62 . -7 )
        ( 6 . “TRAZO_Y_PUNTO" ) )
)
```

En este caso el código Ø "LAYER" indica que se trata de una capa. Su nombre es “EJES”. El color asociado es blanco, y como el número de color se encuentra con signo menos (-7), esto quiere decir que se encuentra en este momento desactivada. El tipo de línea asociado a la capa es “TRAZO_Y_PUNTO”.

• Estilos de texto

La lista de definición de un estilo de texto podría ser como la que sigue:

```
( ( Ø . "STYLE" )
        ( 2 . "TS1" )
        ( 7Ø . 64 )
        ( 4Ø . Ø.Ø )
        ( 41 . 1.Ø )
        ( 5Ø . Ø.Ø )
        ( 71 .Ø )
        ( 42 . 3.5 )
        ( 3 . "simplex" )
        ( 4 . " " )
)
```

El nombre del estilo de texto es "TS1". La altura en la definición del estilo (código 4Ø) es Ø. El factor de de proporción (código 41) es 1. El ángulo de inclinación del estilo (código 5Ø) es Ø. La generación estilo es normal (código 71 igual a Ø). La altura actual de los textos por defecto (código 42) es 35 archivo de tipo de letra en que está basado el estilo es "simplex".

• Tipos de línea

La lista correspondiente a la definición de un tipo de línea cargado en el dibujo presenta un aspecto como el siguiente:

```
( ( Ø . "LTYPE" )
        ( 2 . "TRAZOS" )
```

```
        ( 7Ø . 64 )
        ( 3 . " -- -- -- -- -- -- -- -- -- " )
        (72 . 65 )
        ( 73 . 2 )( 4Ø . Ø.75 )
        ( 49 . Ø.5 )
        ( 49 . -Ø.25 )
)
```

El nombre del tipo de línea es "TRAZOS". El aspecto que presenta en el archivo de definición "ACAD.LIN" es el que aparece asociado como una cadena de texto al código 3. El código 72 indica factor de alineamiento A, que, como se sabe, es obligatorio para los tipos de línea de AutoCAD. El código 73 indica el número de parámetros (sucesión de tramos, huecos y puntos) que definen el patrón de la línea. En este caso son 2: un trazo y un hueco. El código 4Ø proporciona la longitud total del patrón código 49 da en varias listas los valores de ese patrón; positivos para trazos, negativos para huecos y Ø para puntos.

• Definiciones de bloques

La lista de "cabecera" de La definición de un bloque tiene una característica especial y es que contiene un código -2 con el nombre de la primera entidad de las que componen ese bloque. Por ejemplo:

```
( (Ø . "BLOCK" )
        ( 2 . "RES" )
        ( 7Ø . 66 )
        ( 1Ø Ø.Ø  Ø.Ø Ø.Ø )
        ( -2 . <Entity name: 4ØØØ12ca> ) )
)
```

El bloque definido se llama "RES", su punto de inserción es el (X = Ø Y = Ø Z = Ø) y el nombre de 1a primera entidad de la que está compuesta el bloque es 4ØØØ12ca. A continuación vendrían todas las líneas con las entidades que componen el bloque exactamente del mismo tipo de las vistas para las entidades sueltas Aquí no se necesita ninguna lista final del tipo “SEQEND", pues al explorar todas las listas de las entidades en orden correlativo no hay ninguna lista que siga a la última entidad.

• Vistas almacenadas

La lista correspondiente a la definición de una vista almacenada podría ser la siguiente:

```
( ( Ø . "VIEW" )
        ( 2 . "DETALLE" )
        ( 7Ø . Ø )
        ( 4Ø . 8Ø.Ø )
        ( 1Ø 1Ø5.Ø 22Ø.Ø )
        ( 41 . 11Ø.Ø )
```

```
        ( 11 Ø.Ø Ø.Ø 1.Ø )
        ( 12 Ø.Ø Ø.Ø Ø.Ø )
        ( 42 . 5Ø.Ø)
        ( 43 . Ø.Ø )
        ( 44 . Ø.Ø )
        ( 5Ø . Ø.Ø )
        ( 71 . Ø)
)
```

La vista se llama "DETALLE". Los códigos 4Ø y 41 indican la altura y anchura de la vista. El código 1Ø proporciona las coordenadas del centro de la vista. Los códigos 11 y 12 dan la orientación del punto de vista en 3D.

• Sistemas de coordenadas almacenados

Una lista correspondiente a un SCP almacenado tiene el siguiente aspecto:

```
( ( Ø . "UCS" )
        ( 2 . "FRONTAL" )
        ( 7Ø . Ø )
        ( 1Ø Ø.Ø  Ø.Ø Ø.Ø )
        ( 11 1.Ø Ø.Ø Ø.Ø )
        ( 12 Ø.Ø -1.837772e-16 1.Ø )
)
```

El nombre del SCP es "FRONTAL". Los códigos 1Ø, 11 y 12 dan la orientación de ese SCP con respecto al Universal. A veces aparecen valores exponenciales muy próximos a Ø, pero que no son exactamente Ø, como en el ejemplo. Esto es para permitir moverse el cursor aunque el punto de vista del plano actual XY sea a "ras de suelo" (símbolo de "lápiz partido" para el SCP).

• Configuraciones de ventanas almacenadas

La lista correspondiente puede presentar un aspecto parecido al que sigue:

```
( ( Ø . "VPORT" )
        ( 2 . "DESPIECE" )
        ( 7Ø . Ø )
        ( 1Ø Ø.5 Ø.Ø )
        ( 11 1.Ø Ø.5 )
        ( 12 1Ø5.53 22Ø.Ø )
        ( 13 Ø.Ø Ø.Ø)
        ( 14 1Ø.Ø 1Ø.Ø )
        ( 15 1Ø.Ø 1Ø.Ø )
        ( 16 Ø.Ø Ø.Ø 1.Ø )
        ( 17 Ø.Ø Ø.Ø  Ø.Ø )
        ( 4Ø . 8Ø.Ø )
        ( 41 . 1.38824 )
        ( 42 . 5Ø.Ø )
        --
```

```
        --
)
```

El nombre de la configuración es "DESPIECE". A continuación viene una serie bastante numerosa de listas de asociaciones que van proporcionando todos los datos de la división de la pantalla en ventanas y del punto de vista correspondiente, a cada una de ellas.

13 CAPITULO 13: Trabajo con entidades de dibujo y acceso a Base de Datos.

En este capitulo se presentan, en primer lugar, los comandos de AutoLISP para trabajar con conjuntos designados de AutoCAD. Son de los mas utilizados, pues cualquier operación de edición que quiera realizarse en un dibujo para modificar entidades requiere, en primer lugar, una designación de esas entidades.

Lo que identifica a una entidad es su nombre hexadecimal, tal como aparece en la lista correspondiente al código -2 en la Base de Datos. Existen una serie de comandos de AutoLISP que gestionan los nombres de entidades.

A continuación se estudian los comandos que permiten, una vez conocido el nombre de una entidad, acceder a la lista completa con los datos de esa entidad, modificarla y actualizarla en la Base de Datos.

Por útiimo, existen comandos específicos pura gestionar los datos de las Tablas de Símbolos, que son los estudiados en el apartado final.

13.1 Comandos para la selección de entidades de dibujo.

Se explican en este apartado los comandos que permiten introducir en el programa en AutoLISP entidades o conjuntos designados de entidades. Hasta ahora, con los comandos del tipo "GET..." se ha visto que es posible introducir puntos, ángulos, distancias, cadenas de texto o valores numéricos. Pero es evidente que la mayoría de órdenes de AutoCAD manejan entidades, y para ello existen estos comandos.

13.1.1 (SSGET [<modo>] [<pt1>] [<pt2>]) Introducir conjunto.

Este comando acepta un conjunto designado de entidades. Las posibilidades de utilización son las siguientes:

• Sin ningún argumento, solicita del usuario una designación completa. Se pueden emplear todos los modos de designación: Ventana, Captura, Previo, Añadir, Quitar, etc., y en el momento en que se pulsa RETURN, el conjunto designado es aceptado por SSGET.

Command: ***(SETQ conj (SSGET))***
Selec objets:
<Selection set: 1>

El comando devuelve una señal de conjunto designado con un número. En el ejemplo indica que se trata del conjunto designado numero 1. El conjunto

quedaría almacenado en la variable "conj", Todos los conjuntos devueltos por SSGET en todas sus opciones son aceptados por AutoCAD en respuesta a la pregunta "Designar objetos:".

Por ejemplo, para desplazar el conjunto "conj" una distancia de 1Ø unidades en el eje X se podría hacer:

Command: ***(COMMAND "move" conj "" "1Ø,Ø" "")***

(SSGET “P”): Designa el último conjunto previamente designado.

(SSGET "U"): Designa la última entidad añadida a la Base de Datos del dibujo de las visibles en pantalla, es decir, la última entidad dibujada y no borrada de las visibles en pantalla.

(SSGET "V" p1p2): Designa el conjunto resultante de la ventana cuyos vértices son los puntos "p1 y "p2". Esos puntos hay que indicarlos. Si se introduce sólo (SSGET "V"), se produce un error.

(SSGET "C" p1p2): Designa el conjunto resultante de la captura cuyos vértices son los puntos "p1 y "p2". Ambos puntos hay que indicarlos.

(SSGET p1: Designa la entidad que pasa por el punto "p1. Es equivalente a señalar ese punto en pantalla. El resultado dependerá del modo de referencia actual, es decir, el valor de la variable "osmode".

(SSGET "X" <lista de fíltros>): El modo "X" es un modo especial que explora toda la Base de Dalos del dibujo y crea un conjunto designado con todas las entidades del dibujo que cumplan las condiciones establecidas por la lista de filtros. Esas condiciones hacen refererencia a las propiedades de la entidad: Tipo de Entidad, Capa, Color, Tipo de Linea, Elevación, etc.

Para construir las listas de asociaciones que se desea imponer como condición en la lista de filtros se utiliza el comando CONS. Por ejemplo, las siguientes asociaciones definirían diferentes tipos de propiedades:

(CONS Ø "circle") Entidades de círculo.
(CONS 8 "cotas") Entidades en la capa "Cotas".
(CONS 7 "trazos") Texts con estilo "Trazos".
(COMS G2 1) Entidades con color 1 (rojo).
(CONS 2 "mesa") Referencias del bloque "mesa”

Una vez construidas las asociaciones desceadas se reúnen en una lista con el comando LIST y esa lista se puede indicar como argumento de (SSGET "X").

El comando SSGET con el parámetro "X" no admite todos los códigos posibles en la Base de Datos. Los que admite son los siguientes (Cuadro 13.1):

Cuadro 13.1. Códigos admitidos paro tn lisia de filtros do SSGET "X".

Código	Significado
Ø	Tipo de entidad. Los nombres están en ingles en la Base de Datos (Line, Circle, Point etc.).
2	Nombre de bloque para referencia de bloque (obtenida con INSERT).
6	Nombre de tipo de linea.
7	Nombre de estilo de texto.
8	Nombre de capa
38	Valor de elevación.
39	Valor de altura de objeto.
62	Número de color (Ø = porbloque, 256 = porcada).
66	Señal de que el bloque contiene atributos.
210	Vector de orientación de la altura de objeto en 3D.

Por ejemplo para designar el conjunto de entidades de círculo situados en la capa "Pieza" y con el color rojo se haría de la siguiente manera:

```
( SETQ conj ( SSGET "X"  ( LIST
        ( CONS  Ø "circle" )
        ( CONS 8 "pieza" )
        ( CONS62 1 ) ) )
)
```

El comando SSGET con el parámetro ‘‘X” explora toda la Base de Datos del dibujo y va almacenando en la variable "conj" todas las entidades que cumplen los tres requisitos especificados.

En AutoLISP V. 11 es posible incluir caracteres "comodin" en las cadenas de texto indicadas en la lista de filtros (véase el apartado 17.3.4 del Capítulo 17).

Los conjuntos designados ocupan archivos temporales de AutoCAD. Por esa razón existe una limitación en cuanto al número máximo de conjuntos almacenados en variables, que es de seis a la vez. Una vez alcanzado ese límite, SSGET rechaza la posibilidad de crear un nuevo conjunto y devuelve "nil". Para acceder a más conjuntos designados es preciso eliminar alguno de los almacenados poniendo la variable a "nil":

Los conjuntos designados se pueden utilizar para procesarlos en las órdenes de AutoCAD sin más que dar el nombre de la variable que los contiene tras la llamada a la orden con COMMAND

13.1.2 (SSLENGTH <conj>) Número de entidades de un conjunto.

Este comando determina el número de entidades contenidas en el conjunto designado <conj>. El número de entidades es siempre un entero positivo salvo si es mayor que 32767, en cuyo caso es un número real.

Command:
(SETQ conj (SSGET "U"))
(SSLEMGTH conj)
1

En el ejemplo, (SSGET "U") almacena el conjunto de la última entidad dibujada visible en pantalla. Por eso SSLENGTH devuelve 1.

13.1.3 (SSNAME <conj> <ind>) Nombre de entidad en conjunto

Las entidades se encuentran referenciadas en la Base de Datos con un "nombre" que es en realidad un número en hexadecimal (por ejemplo, 6ØØØØA3C). Este "nombre" es único para cada entidad, de forma que la identifica inequívocamente y la distingue de todas las demás del dibujo.

El comando SSNAME devuelve el "nombre" de entidad del conjunto designado <conj>, situada en el lugar indicado por el índice <ind>. Las entidades del conjunto designado se empiezan a numerar desde Ø. Así, un conjunto de cuatro entidades estarían numeradas de Ø a 3.

Command:
(SETQ conj (SSGET))
(SSNAME conj 1)
<Entity name: 6ØØØØA3C>

Una vez obtenido el nombre de la entidad, puede ser procesado por los comandos relativos a nombres de entidades, que se explican en el siguiente apartado.

Los nombres son sólo de entidades principales. El comando SSNAME no puede acceder a las que forman parte de una polilínea o de un bloque.

13.1.4 (SSADD [<nom-ent> [<con>]]) Añadir entidad a conjunto

Este comando se utiliza para añadir una entidad a un conjunto ya designado. Si se empile el comando sin ningún argumento, construye un conjunto designado “vacio”, sin elementos:

(SSADD)

Si se indica únicamente un argumento con un nombre de entidad <nom-ent>, construye un conjunto designado que contiene sólo esa entidad:

(SSADD <nom-ent>)

Si se indica un nombre de entidad y un conjunto previamente designado <conj>, el comando SSADD añade la entidad al conjunto designado, con lo que este pasa a tener un elemento más:

(SSADD <nom-ent> <conj>)

El comando SSADD siempre devuelve un valor de conjunto designado. En los dos primeros casos como crea un conjunto nuevo, devuelve su valor. Si se indica un conjunto ya existente en <conj>, el comando devuelve ese mismo valor especificado, puesto que el efecto es añadirle una entidad, pero el identificador del conjunto sigue siendo el mismo.

13.1.5 (SSDEL <nom-ent> <conj>) Eliminar entidad de conjunto.

Este comando elimina la entidad cuyo nombre se indica <nom-ent>, del conjunto designado <conj>. El nombre del conjunto sigue siendo el mismo y por eso SSDEL devuelve ese nombre. Si la entidad no existe en el conjunto, SSDEL devuelve nil.

13.1.6 (SSMEMB <nom-ent> <conj>) Ver si entidad está en conjunto.

Este comando examina el conjunto indicado <conj> para ver si la entidad cuyo nombre se especifica <nom-ent> está en él. Si la encuentra, devuelve el nombre de esa entidad. Si no la encuentra, devuelve nil.

13.2 COMANDOS RELATIVOS A NOMBRES DE ENTIDADES

Para acceder y modificar las caracteristicas de una entidad directamente en la Base de Datos es preciso identificar sin lugar a dudas la entidad o entidades de que se trate. Para ello se utiliza el "nombre" de la entidad, que como se ha visto es único.

Aunque con SSNAME es posible obtener los nombres de entidades en un conjunto designado, los comandos explicados en este apartado manejar, directamente nombres de entidades, sin necesidad de tener que indicar un conjunto designado.

13.2.1 (ENTNEXT [<nom-ent>]) Nombre de entidad siguiente.

Este comando devuelve el nombre de la primera entidad de la Base de Datos que sigue a la entidad cuyo nombre se indica en <nom-ent>. Si no se especifica ningún nombre como argumento, devuelve el de la primera entidad no eliminada de la Base de Datos.

Command:
(SETQ enti (ENTNEXT))(SETQ ent2 (ENTNEXT enti))(SETQ ent3 (EMTNEXT ent2))

En el ejemplo se almacenan en las variables los nombres de las tres primeras entidades de la Base de Datos. Estableciendo una secuencia repetitiva, se podría ir recorriendo toda la Base de Datos obteniendo los nombres de todas las entidades existentes en el dibujo

Este comando ENTNEXT accede no sólo a las entidades principales, sino también a las contenidas en ellas. Si, por ejemplo, "entl" es un nombre de entidad del tipo de polilínea, ENTNEXT ira recorriendo en la Base de Datos todas las entidades simples que componen la polilínea, incluida la entidad de tipo de SEQEND

Así, por ejemplo, dado un vértice de una polilínea (tipo de entidad "VERTEX") si se quiere saber a qué polilínea pertenece, habrá que ir explorando todas las entidades siguientes con ENTNEXT hasta encontrar una que sea del tipo “SEQEND” Se mira entonces el valor asociado al código -2 y ése es el nombre de la entidad principal.

13.2.2 (ENTLAST) Nombre de la última entidad principal.

Este comando devuelve el nombre de la última entidad principal no eliminada de la Base de Datos. Se utiliza fundamentalmente para obtener el nombre de las entidades recién añadidas a la Base de Datos con COMMAND. Presenta la gran ventaja de que obtiene el nombre de la entidad aunque no sea visible en pantalla o se encuentre en una capa INUTilizada.

Command:
(COMMAND "linea" p1 p2 "")(SETQ lin (ENTLAST))

La variable "lin" almacena el nombre de la línea recién dibujada, sin necesidad de tener que designarla. Con ese nombre ya es posible acceder a la Base de Datos y realizar todas las operaciones con la línea en cuestión directamente.

13.2.3 (ENTSEL <mens>) Nombre de entidad con punto de designación.

Existen determinadas órdenes de AutoCAD que procesan entidades teniendo en cuenta el punto de designación de las mismas, como, por ejemplo, la orden "Parte". El comando ENTSEL espera que el usuario designe una única entidad mediante un punto y devuelve una lista cuyo primer elemento es el nombre de la entidad, y su segundo elemento el punto de designación. De esta forma se tienen asociados ambos valores para procesarlos posteriormente.

Esta lista se puede indicar en las llamadas a las órdenes de AutoCAD que

requieren señalar la entidad a procesar por un punto.

Se puede indicar una cadena de texto en <mens>, que será el mensaje visualizado al solicitar designar una entidad.

Command:
(SETQ ent (ENTSEL "Designar una entidad mediante punto: "))
Designar una entidad mediante punto:
(<Entity name: 6ØØØØ32A> (1Ø.Ø 2Ø.Ø Ø.Ø))

En el ejemplo, el nombre de la entidad designada es "6ØØØØ32A" y el punto de designación (1Ø.Ø 2Ø.Ø Ø.Ø).

13.2.4 (HANDENT <rot>) Nombre de entidad asociada a rótulo.

Este comando devuelve el nombre de la entidad asociada al rótulo indicado en <rot>. Hasta ahora se ha visto que los nombres de entidades eran únicos y las identificaban inequivocamente. Pero al terminar la sesión y salir del Editor de Dibujo esos nombres se pierden. En cambio, si en el dibujo de AutoCAD se ha activado la característica de los rótulos con la orden ROTULOS, cada entidad lleva asociado uno. Ese rótulo no cambia en las diferentes entradas y salidas del Editor de Dibujo.

13.3 Comandos relativos a datos de entidades.

Los comandos explicados en este apartado extraen y modifican directamente los datos contenidos en la Base de Datos, es decir, gestionan las listas que definen todas las entidades del dibujo.

13.3.1 (ENTGET <nombre-ent>) Introducir lista de entidad.

Este comando busca en la Base de Datos e1 nombre de entidad indicado y devuelve la lista completa correspondiente a esa entidad. Se observa que el comando requiere indicar el nombre de la entidad y por tanto es preciso obtener previamente este nombre (con SSNAME, ENTNEXT, ENTLAST etc.).

Por ejemplo, para obtener la lista correspondiente a un círculo dibujado desde un programa en AutoLISP se haría de la siguiente manera:

Command:
(COMMAND "circulo" "1Ø,1Ø" “5”)
(SETQ ent (ENTGET (ENTLAST)))

El comando ENTLAST devuelve el nombre de la última entidad dibujada, que es eL círculo, y ENTGET devuelve la lista correspondiente a ese círculo. Si el color y tipo de línea actuales son "Porca-pa" y la capa actual es la "Ø", el contenido de la variable "ent" podría ser:

```
( ( -1 .  <Entity name:  6ØØØØC3A> )
( Ø .  "CIRCLS" )
( 8 . "&" )
( 1Ø  1Ø.ØØØØ  1Ø.ØØØØ  Ø.ØØØØ)
( 4Ø . 5.ØØØØ )
)
```

La forma más sencilla de acceder a los datos es mediante ASSOC, obteniendo la lista de asociaciones deseada y después sus componentes con CAR, CDR, CADR, etc.

Por ejemplo, para obtener la capa en que se encuentra el circulo del ejemplo anterior se haria:

Command: ***(SETQ cap (CDR (ASSOC 8 ent)))***

La evaluación de (ASSOC 8 enQ devuelve la lista asociada al código 8 en la lista de entidad "ent", es decir, (8 . "Ø"). Después CDR devuelve el segundo elemento "Ø", que es el nombre de capa que se pretendía obtener y que queda almacenado en la variable "cap".

Para obtener, por ejemplo, la coordenada Y del centro del circulo sería.

Command: ***(SETQ ceny (CADDR (ASSOC 10 ent)))***

La evaluación de (ASSOC 1Ø ent) devuelve la lista asociada (1Ø 1Ø.ØØØØ 1Ø.ØØØØ Ø.ØØØØ). Después CADDR es equivalente a (CAR (CDR (CDR)}). El primer CDR devuelve la Lista (1Ø,ØØØØ 1Ø.ØØØØ Ø.ØØØØ). Aplicando a esa lista CDR quedaría (1Ø.ØØØØ Ø.ØØØØ). Por ultimo, aplicando CAR queda (1Ø.ØØØØ), que es el valor de la coordenada Y.

En AutoliSP 11 el formato del comando es:

(ENTGET <nombre-ent> [<Ustap>])

El parámetro optativo <listap> (lista de aplicaciones) hace referencia a nombres registrados de aplicaciones ADS. Véase el apartauo 17.6.1 (comando REGAPP) y el apartado 17.1 (para el concepto de ADS) en Capitulo 17.

13.3.2 (ENTDEL <nombre-ent) Borrar o recuperar entidad.

La entidad cuyo nombre se indica es borrada si existe en la Base de Datos del dibujo actual, o recuperada si había sido previamente borrada en la actúal sesión de dibujo. Esto quiere decir que las entidades borradas con ENTDEL pueden ser después recuperadas con el mismo ENTDEL, antes de salir de la actual edición del dibujo.

Las entidades borradas con ENTDEL se eliminan definitivamente al salir del

Editor de Dibujo, por lo que ya no podrían ser ya recuperadas.

Este comando accese solamente a las entidades principales. No es posible borrar vértices de polilíneas o atributos ni entidaes de un bloque

Command: ***(COMMAND "linea" "5,5" "2Ø,2Ø" "")(ENTDEL (ENTLAST))***

En el ejemplo se borra la entidad de línea recién dibujada. Se podría volver a recuperar antes de salir del dibujo.

13.3.3 (ENTMOD <lista-ent>) Actualizar lista de entidad.

Este comando actualiza la lista de una entidad en la Base de Datos. Recibe la lisia de la entidad (obtenida con ENTGET y modificada después) y la inserta en la Base de Datos sustituyendo a la que existia. El mecanismo de modificación de la Dase de Datos es, pues, el siguiente:

- Extracción de la lista actual de la entidad con ENTGET.
- Acceso al dato de la lista que interesa modificar (con ASSOC, CAR. CDR, etc.).
- Modificiición de ese dato y construcción de la nueva lista (con CONS. LIST o. de forma mas cómoda, con SUBST).
- Inserción de la lista modificada en la Base de Datos sustituyendo a la lista original

(con ENTMOD).

El comando ENTMOD tiene algunas restricciones en cuanto al tipo de dato que puede actualizar para una entidad. No puede modificar ni el nombre de entidad ni el tipo. Si se modifica el nombre de estilo de texto, tipo de línea o nombre de bloque, éstos deben estar previamente definidos o cargados en el dibujo. Si se modifica el nombre de capa, en cambio, se crea una nueva capa si no existiera previamente.

Por ejemplo, si se pretende cambiar de capa el circulo del ejemplo anterior, pasándolo de la capa "Ø" a una capa "Pieza", el proceso sería:

Command:

(COMMAND "circulo" "1Ø,1Ø" "5")
(SETQ ent (ENTGET (ENTLAST)))
(SETQ entac (SUBST (CONS 8 "Pieza") (ASSOC 8 ent) ent))
(ENTMOD entac)

La variable "ent" contiene la lista de la actual definición del circulo. La variable "entac" recibe la lista una vez actualizada. Para ello se ha construido una lista de asociación en (CONS 8 "Pieza"), que contiene la definición de la nueva capa y con el comando SUBST se sustituye esa nueva asociación en la antigua (ASSOC 8 ent), dentro de la lista "ent". Para introducir esa nueva lista en la Base de Datos sólo resta hacer (ENTMOD entac).

13.3.4 (ENTUPD <nom-ent>) Regenerar entidad compuesta.

Con el comando ENTMOD visto en el apartado anterior se actualizaba en la Base de Datos una entidad a la que se habían cambiado sus características. Si se trata de una entidad principal, ENTMOD regenera directamente la entidad y ésta se visualiza en pantalla ya actualizada. Pero si se modifica un componente de una entidad compuesta (por ejemplo, un vértice de polilínea), aunque se actualice en la Base de Datos el aspecto de la entidad en pantalla no cambia hasta que se produzca una regeneración de todo el dibujo

Con el comando ENTUPD, indicando simplemente el nombre de la subentidad (por ejemplo, el vértice de polilínea que se ha modificado), busca cuál es la cabecera de esa entidad y la regenera, con lo que se actualiza su aspecto en pantalla.

13.4 Comandos relativos a Tabla de Símbolos.

Existen dos comandos específicos para acceder a las definiciones contenidas en las Tablas de Símbolos Este acceso es de sólo lectura: no se puede modificar el contenido de las tablas.

13.4.1 (TBLNEXT <norn-tabla>) Extraer contenido de tabla.

Este comando devuelve la lista con el contenido de la Tabla de Simbolos cuyo nombre se indique El nombre tiene que ser "LAYER", "LTYPE". "VIEW", "STYLE", "BLOCK", "UCS" o "VPORT", que son los únicos admitidos.

El comando devuelve la primera Tabla de Símbolos existente de ese tipo la primera vez que se utiliza. Después va devolviendo las siguientes conforme se vuelve a utilizar.

Por ejemplo, en un dibujo con tres capas, "Ø", "pieza" y "oculta", el comando TBLNEXT se utilizaría tres veces para obtener las características de las tres capas:

(TBLNEXT "layer")

```
( ( Ø . "LAYER" )
( 2 . "Q" )
( 6 . "CONTINUA" )
( 7Ø . Ø )
( 62 . 7 )
)
```

La capa "Ø" tiene asociados un tipo, de línea "continua" y un color 7 "blanco". Empleando de nuevo TBLNEXT devuelve la siguiente definición de capa:

(TBLNEXT "layer")

```
( ( Ø . "LAYER" )
```

```
( 2 . "PIEZA" )
( 6 . "CONTINUA" )( 7Ø . Ø )
( 62 . 1 )
)
```

La capa "Pieza" tiene asociados un tipo de linea "continua" y un color 1 "rojo".

(TBLNEXT "layer")

```
( ( Ø "LAYER" )
( 2 . "OCULTA" )
( 6 . "TRAZOS" )
( 62. 3 )
)
```

La capa "Oculta" tiene asociados un tipo de linea "Trazos" y un color 3 "verde".

Si se emplea TBLNEXT LAYER por cuarta vez, devolvería nil, puesto que ya no existen más definiciones de capas.

13.4.2 (TBLSEARCH <nom-tabla> <simb>) Buscar tabla con nombre.

Este comando busca en el tipo de tabla que se indique en <nom-tabla> el nombre de símbolo indicado en <simb> y devuelve la lista correspondiente. De esta forma se puede buscar, por ejieplo, directamente la lista correspondiente a la capa llamada "pieza", haciendo:

Command:
(TBLSEARCH "layer" "pieza")

Si se pretendiera acceder a !a definición de un estilo de texto definido en el dibujo y llamado “tsl” bastaría hacer:

(TBLSEARCH "style" "tsl")

```
( ( Ø . "STYLE" )
( 2 . "TS1" )
( 3 . "simpiex" )
( 4 . " " )( 7Ø . Ø )( 4Ø . Ø,Ø )
( 41 . 1.Ø )( 5Ø . Ø.Ø)
( 71 . Ø )
)
```

El contenido de la tabla informa que el estilo "tsl" está basado en el archivo de tipo de letra “simplex", con altura Ø, factor de proporción 1, ángulo de inclinación Ø y generación normal.

13.5 Ejemplo 1: Cambiar capa actual designando entidad de referencia.

Resulta muy frecuente en AutoCAD querer situarse en una capa que contiene determinadas entidades para modificar algo o añadir algo en esa capa. Pero si la estructura de capas del dibujo es un poco compleja (lo que ocurre enseguida en cuanto el dibujo se complica), puede ocurrir que el usuario no recuerde cómo se llama la capa en cuestión y debe acudir a la orden LIST para averiguarlo y después a la orden CAPA DEF pan establecer esa capa como actual.

En pantallas gráficas de alta resolución y dependiendo de la tarjeta gráfica, como la orden LIST provoca un cambio a pantalla de texto, es frecuente que se produzca un redibujado al volver a la pantalla gráfica con la consiguiente espera.

Como aplicación muy elemental de las posibilidades de la gestión de la Base de Datos, se presenta aquí una pequeña utilidad que permite establecer como actual la capa en que se encuentra una entidad, sin otro csfucr/o que señalar esa entidad.

13.5.1 Listado del programa.

```
( DEFUN c:iracapa ( / ent nent cap )
    ( SETVAR "blipmode" Ø )  ( SETVAR  "cmdecho" Ø )
    ( WHILE ( NOT  ( SETQ ent  ( ENTSEL "Designar entidad:  " ) ) ) )
    ( SETQ nent ( CAR enfc ) )
    ( SETQ cap ( CDR  ( ASSOC 8  ( ENTGET nenfc ) ) ) )
    ( COMMAND "capa" "def" cap “ " )
    ( SETVAR "blipmode" 1 ) (SETVAR "cmdecho”  1) ( PRIN1 )
)
```

13.5.2 Observaciones.

La rutina es tan sencilla que se ha definido la orden directamente, sin funciones de usuario intermedias

El mecanismo de utilización de los comandos de acceso a la Base de Datos es siempre muy similar:

En primer lugar, designación de entidades con comandos del tipo SS.... vistos en el apartado 1 de este capítulo.

A continuacón, gestión de los nombres de esas entidades con los comandos del tipo ENT.., vistos en el apartado 2.

Por último, acceso y modificación de la Base de Datos con los otros comandos del tipo ENT vistos en el apartado 3.

A veces es posible saltarse el primer caso y obtener directamente los nombres de entidades con los comandos ENT... del apartado 2, como ocurre en este ejemplo con ENTSEL.

13.5.3 Explicación del programa.

Define directamente una nueva orden de AutoCAD llamada "iracapa".

• Función **c:iracapa**

Se utiliza el comando ENTSEL porque este sólo permite designar una entidad por un punto. De esta forma se obliga al usuario a que designe la entidad cuya capa desea establecer como actual en el dibujo. Si el usuario designara varias, podría ocurrir que estuvieran en diferentes capas y surgiría una ambigüedad.

El control con WHILE hace que la petición para designar una entidad se repita en el caso de que el usuario falle al "apuntar".

El comando ENTSEL devuelve una lista con el nombre de la entidad y las coordenadas del punto de designación. Por eso se utiliza CAR para almacenar el nombre en "nent".

La siguiente expreción extrae la sublista correspondiente al código 8, que contiene el nombre de la capa de caá entidad, el cual se obtiene con CDR y se almacena en "cap".

Sólo resta utilizar la orden "capa", opción "def", para definir esa capa como actual en el dibujo.

13.6 Ejemplo 2: Cambiar un texto a mayúsculas o minúsculas.

Una tarea incómoda con AutoCAD es cambiar el contenido de textos insertados en el dibujo. Aunque la nueva Versión 11 viene preparada con un Letrero de Dialogo que permite editar el texto a cambiar, no es posible convertirlo a mayúsculas o minúsculas todo él a un tiempo.

El programa contenido en este ejemplo permite esa conversión para varios textos a la vez sin más que designarlos.

13.6.1 Listado del programa.

```
( DEFUN inic ( / n nent lent op )
   ( PROMPT "Designar textos a cambiar: " ) ( TERPRI )
   ( WHILE ( NOT conj ) ( SETQ conj  ( SSGET ) ) )
   ( SETQ num  ( SSLENGTH conj ) )
   ( SETQ n Ø )
   ( REPEA.T num
         ( SETQ nent ( SSNAME conj n ) )
         ( SETQ lent ( ENTGET  nent ) )
         ( IF ( / = ( CDR ( ASSOC Ø  Lent ) )  "TEXT" )
```

```
                ( PROGN ( SSDEL nent conj ) ( SETQ n ( - n 1 ) ) )
            )
            ( SETQ n ( + n 1 ) )
        )
        ( SETQ num ( SSLENGTH  conj ) )
        ( IF ( = num Ø ) ( PROMPT "Ningún texto encontrado\n" )
            ( PROGN ( INITGET 6 "MAY MIN" )
                ( IF ( SETQ op ( GETKWORD  "MAY o MIN? <MAY>: ") )  ( )
                    ( SETQ op "MAY" ) )
                ( TERPRI ) ( IF ( = op "MIN" )  (SETQ ex T ) )
            )
        )

    )
    ( DEFUN cambtx ( / n tx txant txn txln )
        ( SETQ n Ø )
        ( REPEAT num
            ( SETQ tx  ( SSNAME conj n ) )
            ( SETQtxant ( CDR ( ASSOC  1 ( ENTGET tx ) ) ) )
            ( SETQ txn  ( STRCASE txant ex ) )
            ( SETQ txln
                ( SUBST ( CONS 1 txn ) ( ASSOC  1 ( ENTGET tx ) )  ( ENTGET tx )
                ) )
            ( ENTMOD txLn )
            ( SETQ n ( + 1 n ) )
        )
    )
    ( DEFUN c:txmay  (  /  num  conj ex )
        ( SETVAR "bLipmode" Ø )  ( SETVAR "cmdecho" Ø )
        ( inic )
        ( cambtx )
        ( SETVAR "blipmode" 1 )  ( SETVAR "cmdecho" 1 ) ( PRIN1 )
    )
```

13.6.2 Observaciones.

El programa utiliza en su primera parte (primera función de usuario) los comandos del tipo SS... para gestionar el conjunto designado por el usuario, accediendo después con ENTGET a la Base de Datos para comprobar si todas las entidades del conjunto son textos, eliminando aquellas que no lo son.

En su segunda parte, y siempre que el número de entidades de texto encontradas no sea nulo, procesa los textos camnbiándolos a mayúsculas o minúsculas.

Para cambiar los textos, el programa utiliza el mecanismo básico de actualización de la Base de Datos.

13.6.3 Explicación del programa.

Define una nueva orden de AutoCAD y dos funciones de usuario.

• Función **c:txmay**

Define la nueva orden de AutoCAD "txmay" que llama a las dos funciones de usuario que se explican a continuación.

• Función **inic**

Tiene como misión pedir al usuario que designe los textos que desea cambiar, y una opción para cambiarlos a mayúscuhis o minúsculas.

Toda la primera parte de la función se emplea en controles para seleccionar únicamente las entidades de texto, por si el usuario ha designado alguna más.

En primer lugar, y tras visualizar un mensaje de solicitud, el comando WHILE utiliza la variable "conj" como condición. Si el usuario falla y no designa ninguna entidad, la variable "conj" queda a nil y el ciclo WHILE vuelve a pedir un conjunto designado. La única manera de salir de ese control es que el usuario designe al menos una entidad.

Con SSLENGTH se almacena en “num” el número de entidades seleccionado. Ahora es preciso examinar ese conjunto y ver cuáles de esas entidades son textos. Para ello se establece una repetitiva, con la variable "n" como control.

Con SSNAME se extrae el nombre de cada una de las entidades del conjunto usando "n" corno índice de posición de cada entidad en el conjunto. ENTGET obtiene la lista completa de esa entidad en la Base de Datos.

La expresión (CDR (ASSOC Ø lent)) obtiene el tipo de entidad. Con IF se examina si no es "TEXT". Si lo es, la rama del NO de la alternativa está vacía. Se incrementa directamente en 1 el vaior de "n" y se vuelve a repetir el ciclo examinando la siguiente entidad del conjunto "conj".

Si se cumple que la entidad no es un texto, se ejecuta la rama del SI de la alternativa, que es un PROGN. Su misión es eliminar esa entidad que no es un texto con SSDEL. Al eliminarla, el conjunto pierde una entidad y la siguiente pasa a ocupar el puesto vacante dejado. Por eso la nueva entidad a examinar ocupa el mismo índice que la recién eliminada. Eso obliga a restar 1 de "n" para que al incrementarse en 1 fuera de la alternativa se quede como estaba, y al volver a recorrer el ciclo desde el principio examine la nueva entidad en esa posición.

Al salir de REPEAT se vuelve a utilizar SSLENGTH, que ahora si almacenará en "num" el número de entidades únicamente de texto seleccionadas. Si no hubiera textos (n = Ø), el programa fallaría. Por eso se utiliza un último control.

La rama del SI de la alternativa en el caso de que no haya textos es visualizar un mensaje que asi lo indica.

La rama del NO, cuando se ha encontrado al menos un texto, es un PROGN que solicita con GETK-WORD al usuario una opción para cambiar a

mayúsculas o minúsculas. La opción por defecto es MAY (mayúsculas). Previamente se inicíaiiza el comando con INITGET para permitir sólo esas dos opciones.

Si la opción indicada es minúsculas, se asigna a una variable "ex" el valor T para utilizarla después en el comando STRCASE de la siguiente función.

Variables de entrada; ninguna.

Variables locales:

n Indice posición de entidad en conjunto.
nent Nombre de entidad en posición "n".
lent Lista de entidad en Base de Datos.
op Opción mayúsculas o minúsculas.

Variables de salida;

conj Conjunto de entidades de texto.
num Número de entidades de texto en conjunto.
ex Variable nil o T para STRCASE.

• Función **cambtx**

Su misión es cambiar los textos designados en "conj" según la opción seleccionada (controlada por "ex").

Se establece un ciclo con REPEAT, que se repite tantas veces como textos haya en “conj”. Si no existen textos, simplemente el ciclo no tiene efecto y el programa termina.

El ciclo examina cada una de las entidades del conjunto en la posición dada por “n”, obtiene la lista en la Base de Datos, y en la sublista de asociaciones correspondiente al código 1 el valor actual del texto. Con STRCASE se cambia según el valor de "ex": si es nil, se cambia a mayúscula; si es T, a minúsculas.

Una vez alamcenado en “txn", se construye una sublista de asociaciones de la forma:

(cons 1 txn)

Resultado gráfico del cambio a mayúsculos de varios textos diferentes.

Mediante el comando SUBST se sustituye la lista actual por esa nueva lista construida. El resultado es "txln". Para incluirla en la Base de Datos se utiliza ENTMOD.

Se repite la misma operación hasta procesar todos los textos designados (Figura 13.1).

Variables de entradaconj Conjunto de entidades de texto.num Número de entidades de texto en conjunto.ex Variable nil o T para STRCASE.
Variables locales:n Indice posición de entidad en conjunto.tx Nombre de

entidad de texto posición n.txant Contenido actual del texto.txn Nuevo contenido del texto.txln Nueva lista con texto modificado.

Variables de salida: ninguna.

13.7 Ejemplo 3: Cambio global de estilo de varios textos.

Otra labor engorrosa que suele surgir cuando se trabaja con textos en AutoCAD es que, una vez insertados en el dibujo, no es posible modificar globalmente las propiedades de varios textos a la vez con la orden CAMBIA.

AutoCAD va pidiendo los cambios de los textos uno a uno, y además para cada texto va solicitando las propiedades a cambiar una a una: punto de inserción, estilo, altura, ángulo de rotación y contenido del texto. La labor de cambiar alguna de estas propiedades se suele convertir en una sucesión de "Return" seguidos, hasta que el usuario llega a la que le interesa.

Lógicamente, no tiene sentido cambiar a un mismo punto de inserción lodos los textos. Tampoco sería muy lógico cambiar el ángulo de rotación manteniendo el mismo punto de inserción de cada linea (en este caso resulta más coherente usar la orden GIRA y rotar todaslas lineas). Únicamente para el estilo y la altura resulta lógico cambiar todos los textos a la vez.

Esto se puede hacer con comodidad accediendo a la Base de Datos y modificando en la lista de cada texto el contenido de la propiedad que se indique, tal como se expone en el programa de este ejemplo.

13.7.1 Listado del programa.

```
( DEFUN UNICA ( / n nent lent conj num adv )
   ( GRAPHSCR )
   ( WHILE ( NOT ( SETQ conj ( SSGET ) ) ) )
   (SETQ num  ( SSLENGTH conj ) )
   ( SETQ n Ø )
   ( REPEAT num
        ( SETQ nent ( SSNAHE conj n ) )
        ( SETQ lent ( ENTGET nenT ) )
        ( IF ( / = ( CDR ( ASSOC Ø ient ) )  "TEXT" )
             ( PROGN ( SSDEL nent conj ) ( SETQ n ( - n 1 ) ) )
        )
        ( SETQ n ( + n 1 ) )
   )
   ( SETQ num  ( SSLENGTH conj ) )
   ( IF ( -num Ø ) ( PROMPT "Ningún texto encontrado\n" )
            ( PROGN
                 ( INITGET 1 "Estilo Altura" )
                 ( SETQ op ( GETKWORD  "Propiedad a cambiar Estilo/Altura: " ) )
                 ( TERPRI )
                 (  IF  (  =  op  "Estilo"  )  (
                 camest ) )
```

```
                ( IF ( = op "Altura" ) ( camalt
                ) )
            )
        )

)
( DEFUM camest ( / n lent estact nest )
    ( SETQ estact ( CDR ( ASSOC 7  ( ENTGET  ( SSNAME conj Ø ) ) ) ) )
    ( IF ( SETQ nesfc ( GETSTRING
                    ( STRCAT "Nuevo estilo <" estact " >: " ) ) )
            ( ) ( SETQ nest estact )
    ) ( TERPRI )
    ( SETQ n Ø )
    ( REPEAT num
            ( SETQ lent ( ENTGET  ( SSNAME conj n ) ) )
            ( SETQ lent ( SUBST  ( COMS 7  nest ) ( ASSOC 7 lent ) lent ) )
            ( ENTMOD lent )
            ( SETQ n ( +  1 n ) )
    )
)
( DEFUN camalt  (  /  altact  nalt n lent )
    ( SETQ altact ( CDR ( ASSOC  4Ø ( ENTGET  ( SSNAME  conj Ø ) ) ) ) )
    ( INITGET 6 )
    ( IF ( SETQ nalt ( GETREAL ( STRCAT
            "Nueva altura <" ( RTOS altact 2 2 )  ">: " ) ) )
            ( ) ( SETQ nalt altact ) ) ( TERPRI )
    ( IF ( > num 1 ) ( chkalt ) )
    ( SETQ n Ø )
    ( IF adv ( PROGN
            ( INITGET "Si No" )
            ( IF ( SETQ op ( GETKWORD "Nueva altura excesiva.
            Los textos pueden superponerse. Continuar? S/N <No>: “ ) )
                ( ) ( SETQ op  "NO" )
            ) )
    )
    ( IF ( OR ( NOT adv )  ( = op "Si" ) )
            ( REPEAT num
                ( SETQ lent ( ENTGET  ( SSNAME conj  n ) ) )
                ( SETQ Ient ( SUBST ( CONS 40 nalt ) ( ASSOC 7 Ient ) )
                ( ENTMOD Ient )
                ( SETQ n  ( + 1 n ) )
            )
    )


)
( DEFUN chkalt  (  /  pinl pin2 ang  dis sep )
    ( SETQ pin1 ( CDR ( ASSOC  1Ø ( ENTGET  ( SSNAME conj  Ø ) ) ) ) )
    ( SETQ pin2 ( CDR ( ASSOC 13  ( ENTGET  ( SSNAME  conj 1 ) ) )
    ) )
    ( SETQ ang ( CDR ( ASSOC 5Ø ( ENTGET  ( SSNAME conj Ø ) ) )
    ) )
    ( SETQ dis ( DISTANCE pinl pin2 ) )
    ( SETQang  ( -( ANGLE pin1 pin2 )  ang ( / PI 2 ) ) )
```

```
    ( SETQ sep ( ABS ( * dis ( COS ang ) ) ) )
    ( IF ( < sep nalt ) ( SETQ adv T ) )
)
( DEFUN c:protext ( )
    ( SETVAR “blipmode" Ø ) ( SETVAR "cmdecho"  Ø )
    ( única )
    ( SETVAR "blipmode" Ø ) ( SETVAR  "cmdecho" Ø ) ( PRIN1 )
)
```

13.7.2 Observaciones.

En vez de ir llamando a todas las funciones de usuario sucesivamenle, el programa tiene la particularidad de que llama sólo a una (se ha definido como "única" para resaltar esta característica). Es esa función la que llamara a otras, dependiendo de ciertas condiciones.

A su vez, una de estas funciones llama a otra, con lo que se produce un "anidamíento", es decir, funciones incluidas unas dentro de otras.

13.7.3 Explicación del programa.

Se define una nueva orden de AutoCAD y cuatro funciones más de usuario.

• Función **c:protext**

Llama únicamente a la función de usuario "única'*. Antes, como siempre, establece como locales las variables que no han podido ser establecidas como tales en las funciones intermedias y desactiva marcas auxiliares y eco de órdenes.

En este caso, como sólo llama a "única", todas las variables pueden ser establecidas como locales ella.

• Función **unica**

Su contenido es muy similar a la función inicial del programa anterior y está en realidad tomado de ella. Solicita del usuario que designe textos, le obliga con un control a designar al menos una entidad, examina esas entidades y elimina las que no son textos. Si al final no ha sido seleccionado ningún texto, la variable “num” almacena un valor Ø. En ese caso se visualiza un mensaje y el resto del programa toma en cuenta ese valor de "num" para no tener efecto.

Si "num" no es Ø (rama del SI de la alternativa), se establece un PROGN. Aquí es cuando la función cambia respecto a la del programa anterior. En este caso se solicita del usuario qué propiedad desea cambiar y se ofrecen con INITGET dos alternativas: Estilo/Altura.

Como no va a existir ninguna opción por defecto (aunque no habría ningún inconveniente en poner una), se impide con INITGET1 que ei usuario dé

respuesta nula pulsando "Return".

Se examina la opción "op"; si es "Estilo", se llama a la función para cambiar el estilo y si es “Altura”, se llama a la función para cambiar la altura.

Variables de entrada: ninguna.

Variables locales:

n Indice posición de entidad en conjunto.
nent Nombre de entidad en posición "n".
lent Lista de entidad en Base de Datos.
num Número de entidades de texto.
conj Conjunto de entidades de texto.
op Opción Estilo o Altura.

Variables de salida: ninguna.

• Función **camest**

Su misión es cambiar el estilo de los textos designados. En primer lugar extrae el estilo de texto actual para la primera entidad del conjunto (índice Ø), examinando la sublista de asociaciones con el código 7. Ese estilo se almacena en "estact".

Después solicita el nuevo estilo, ofreciendo como opción por defecto el actual de esa primera entidad. Si los textos tienen diferentes estilos, el que se ofrece por defecto es el del primero del conjunto. Ese nuevo estilo se almacena en "nest".

Se establece una repetitiva para ir cambiando ese estilo en todos los textos designados. Se hace como siempre, sustituyendo la nueva sublista de código 7 en este caso por la antigua y actualizando la Base de Datos con ENTMOD (Figura 13.2).

Lógicamente, si el nuevo estilo no existe, el programa dará error y quedara abortado. Para averiguar si el estilo existe, habría que examinar la Tabla de Símbolos correspondiente a los estilos de textos. La forma de gestionar las Tablas de Símbolos se expone en los dos ejemplos siguientes. En cualquier caso, es muy fácil añadir un ciclo repetitivo a la función "camest" para explorar si existe el estilo nuevo y visualizar un mensaje de advertencia en el caso de que no exista.

Variables de entrada:
num Numero de entidades de texto.
conj Conjunto de entidades de texto.

Resultado gráfico del cambio de varias líneas de texto a un nuevo estilo

Variables locales:

estact Estilo actual DEl primer texto.
nest Nuevo estilo para todos los textos.
n Índice posición de entidad en conjunto.
nent Nombre de entidad en posición "n".
lent Lista de entidad en Base de Datos.

Variables de salida: ninguna.

• Función **camalt**
Tiene por misión cambiar la altura de todos los textos designados.

En primer lugar almacena en "altact" la altura actual del primer texto del conjunto, extrayéndola de la lista de asociaciones con el código 4Ø. Después solicita la nueva altura, con INITGET 6 para no permitir valor Ø ni negativos, Ofrece la opción por defecto igual a la altura actual de ese primer texto.

Ahora se establece un control. Sí el número de textos designados es mayor de 1, como no va a cambiar el punto de inserción de los mismos, puede ocurrir que la altura sea excesiva y los textos se superpongan. Entonces, para comprobar si esto es así, se llama a una función de chequeo que es "chkalt". Esa función va a crear una variable "adv", con una advertencia en el caso de que la altura sea excesiva.

Por tanto, se establece como condición esa propia variable; si no es nil, significa que existe y entonces se visualiza el mensaje de advertencia y se solicita una opción de continuar Si/No. Se ofrece como seguridad la opción por defecto "No".

Por último, si no existe la advertencia (lo que quiere decir que la altura examinada por "chkalt" no ha resultado excesiva), o bien a pesar de existir el usuario ha respondido que "Si" quiere continuar en la opción "op", entonces se ejecuta un PROGN que modifica en la Base de Datos los textos dándoles la nueva altura (Figura 13.3).

Figura 13.3. Resultado gráfico del cambio de varias lineas de texto a una nueva altura, primero menor que la inicial y después mayor.

Variables de entrada:
num Número de entidades de texto
conj Conjunto de entidades de texto.
adv Advertencia de altura excesiva (entra tras la llamada a "chkalt"),

Variables locales:
altact Altura actual primera entidad de texto.
nalt Nueva altura para todos los textos.
n Índice posición de entidad en conjunto.
lent Lista de entidad en Base de Datos.
op Opciún continuar con altura excesiva.

Variables de salida: ninguna.

• Función **chkalt**

Como ya se ha dicho, se utiliza para examinar si la altura indicada por el usuario resulta excesiva cuando hay más de un texto a cambiar.

Se va a suponer que todos los textos a cambiar se encuentran insertados uno a continuación del otro y, por tanto, con la misma separación. Esto es un caso muy frecuente cuando se pretende cambiar globalmente la altura en una lista de materiales, por ejemplo.

Por tanto, basta examinar los dos primeros textos (índices Ø y 1 con SSNAME) para saber su separación. Se extraen directamente de la Base de Datos los puntos de inicio (código 1Ø y se almacenan en “pin1” y “pin2".

Suponiendo que el ángulo de inserción es el mismo para todas las líneas de texto, se extrae del primero (lista de código 5Ø) y se almacena en "ang". Por si acaso, los puntos de inicio no se encuentran uno debajo de otro (por ejemplo, textos con opción Centrar. Rodear, Derecha, etc.), se calcula la distancia entre los dos puntos de inicio y se proyecta sobre la perpendicular a la linea del primer texto.

Para ello se calcula el ángulo que forma el vector entre "pin1" y "pin2" y se le resta el ángulo de inserción del texto y también Pl/2. El resultado es almacenar en "ang" el ángulo formado por la distancia entre los dos puntos de inicio y la perpendicular a la línea de texto. Se multiplica la distancia por el coseno de ese ángulo y se obtiene la separación entre las dos líneas de textos. Se toma el valor absoluto para que no influya el signo (Figura 13.4).

Sí esa separación es menor que la nueva altura pretendida "nalt", se pone la variable "adv" a T para indicar que la altura resulta excesiva.

Cuando los textos se encuentran justificados por la izquierda (opción por defecto de la orden "Texto" de AutoCAD), el punto inicial de la línea coincide con el punto de inserción. En las opciones Centrar, Rodear y Derecha, el punto de inserción es diferente del punto inicial de la linea. En la Base de Datos la lista asociada al código 1Ø contiene el punto inicial del texto. La lista asociada al código 11 contiene el punto de inserción.

Si en el programa todos los textos se encuentran justificados de la misma manera (todos Centrados, o todos con Rodear, o todos justificados a la Derecha), se podría calcular la separación como la distancia entre esos puntos, sin necesidad de proyectarla. En este caso la función "chkalt" se simplifica.

Sin embargo, en el programa se ha preferido conservar el procedimiento mas largo por si acaso los textos se encuentran con diferentes justificaciones.

13.8 Ejemplo 4: Listado de bloques con número de

inserciones.

Como primer ejemplo de acceso a Tablas de Símbolos, este sencillo programa ofrece un listado de todos los bloques definidos en el dibujo, de forma análoga a la orden de AutoCAD "BLOQUE ?". Pero aquí se añade una cifra que representa el número de inserciones presentes de cada bloque. Esto puede ser útil en dibujos con elementos normalizados para saber cuantas válvulas de tal tipo tiene una instalación, cuantos transitores un circuito, cuantos baños una planta de construcción, etc.

Figura13.4. Variables que intervienen en la función de chequeo y resultado gráfico cuando las líneas de texto se solapan.

13.8.1 Listado del programa

```
( DEFUN  lista  (  /  listbl nom conj num )
   ( IF ( SETQ listbl ( TBLNEXT "block"  T ) )
        ( PROGN
           ( SETQ nom  ( CDR (ASSOC 2 listbl ) ) )
           ( IF ( SETQ conj ( SSGET "X" ( LIST ( CONS Ø "INSERT" )
                                               ( CONS 2 nom ) ) ) )
              ( SETQ num ( SSLENGTH  conj ) )
              ( SETQ num Ø ) )
           ( PROMPT " \n\n" )
           ( PROMPT " LISTADO DE BLOQUES \n" )
           ( PROMPT " Y\n" )
           ( PROMPT "NUMERO DE INSERCIONES\n" )
           ( PROMPT "----------------------------------------\n" ) ( TERPRI )
           ( PROMPT ( STRCAT  nom  "\t" ( ITOA num ) "\n" ) )
        )
        ( PROMPT "Ningún bloque definido en el dibujo\n" )
   )
   ( WHILE ( SETQ  listbl ( TBLNEXT "block" )
   )
        ( SETQ nom ( COR ( ASSOC 2 listbl )
```

```
            ) )
            ( IF ( SETQ conj ( SSGET "X" ( LIST ( CONS Ø "INSERT" )
                                                  ( CONS 2 nom ) ) ) )
                ( SETQ num ( SSLENGTH conj ) )
                ( SETQ num Ø ) )
        ( PROMPT ( STRCAT  nom  "\t" ( ITOA num )  "\n" ) )
        )
        ( TEXTSCR )
)
( DEFUN c:listabl ( )
      ( SETVAR "cindecho" Ø ) ( SETVAR "blipmode" Ø )
      ( lista )
      ( SETVAR "cmdecho"  1 ) ( SETVAR "blipmode" 1 ) (PRIN1)
)
```

13.8.2 Observaciones.

El programa explora la Tabla de Símbolos de tipo “BLOCK" y va extrayendo los nombres de todos los bloques definidos en el dibujo. Para cada uno de ellos determina cuantas inserciones (tipo de entidad “INSERT”) existen mediante SSGET utilizado con filtro "X".

Con el fin de presentar el listado en dos columnas en pantalla, escribe un tabulador con el código “\t”. Estos códigos se encuentran detallados en el Capitulo 1.

13.8.3 Explicación del programa.

Define una nueva orden que llarna a una función de usuario. Atendiendo a la naturaleza del programa resulta más conveniente separar dos funciones de usuario: por un lado, la cabecera fija del listado, y por otro, cada uno de los bloques. Esto se hará en el siguiente ejemplo, En este caso, dada la sencillez del programa, se ha preferido por comodidad dejar una sola función "lista".

• Función **c:listabl**

Define una nueva orden de AutoCAD llamada "listaba”. Llama simplemente a la función de usuario "lista". El último PRIN1 es para que el programa termine sin visualizar la última evaluación de SETVAR.

• Función **lista**

Presenta dos partes diferenciadas. El primer bloque del dibujo se obtiene mediante (TBLNEXT "block" 'I'). La lista correspondiente a la definición de ese bloque se almacena en "listbl”.

Se establece un control con IF para saber si existen bloques definidos en el dibujo. En caso de no existir ninguno, se ejecuta la rama del NO de la alternativa y se visualiza un mensaje de advertencia.

Si existe al menos un bloque, la rama del SI de la alternativa contiene un

PROGN que engloba todas las operaciones a realizar. El nombre del bloque "nom" se obtiene de la lista de asociación con código 2. Para saber cuántas inserciones de ese bloque existen en el dibujo se construye una Hsta de filtros de la forma:

(SETQ conj (SSGET "X" (LIST (CONS Ø "INSERT") (CONS 2 nom))))

La variable "conj" almacena el conjunto de todas las inserciones (tipo de entidad "iNSERT") del bloque "nom".

El número de inserciones es la longitud de ese conjunto. Sin embargo, existe un inconveniente: si el bloque está definido pero no existe ninguna inserción, el conjunto "conj" es vacío pero SSLENGTH no devuelve un número de entidades Ø, sino que devuelve nil. Por eso hay que establecer un control con IF, para que en ese caso la rama del NO asigne a "num" el valor Ø.

Con PROMPT se escriben en la pantalla de texto los RETURN (código "\n") y las cabeceras para presentar el listado. Debajo se escribe ya el nombre del primer bloque y el número de inserciones de ese bloque Se escribe en medio un espacio de tabulador "\t" para separar en dos columnas.

La secunda parte de la función de usuario establece un ciclo con WHlILE para extraer el resto de bloques. E1 comando TBLNEXT se emplea ahora sin el parámetro T para que vaya explorando toda la Tabla de Símbolos del tipo “BLOCK". El resto es similar a lo visto para el primer bloque.

La función termina conmutando a pantalla de texto con TEXTSCR para que se visualice el listado escrito en ella.

Variables de entrada: ninguna.

Variables locales:

listbl Lista con la definición del bloque.
nom Nombre del bloque.
conj Conjunto con todas las inserciones.
num Número de inserciones del bloque.

Variables de salida: ninguna.

13.9 Ejemplo 5: Hacer Bladisco de todos los bloques del dibujo.

A la hora de extraer todos los bloques de un dibujo para formar una biblioteca de bloques suele resultar una operación ciertamente engorrosa tener que emplear la orden BLAD1SCO con cada uno de ellos.

El programa contenido en este ejemplo explora la Tabla de Simbolos de tipo "BLOCK." y efectúa un "Bladisco" con todos ellos, grabando la biblioteca de bloques en la unidad y directorio que el usuario indique.

13.9.1 Listado del programa.

```
( DEFUN  todobl  (  /  dir listbl ulcam ) ( PROMPT "Unidad [ y directorio] para
      almacenar los bloquea  <C:> " ) ( SETQ cam " " ) ( IF ( = "" ( SETQ dir ( READ-
      LINE ) ) )
     ( SETQ cam "c:" ) ( SETQ cam  ( STRCAT cam dir ) ) ) ( SETQ ulcam ( SUBSTR
cam  ( STRLEN cam ) 1 ) ) ( IF ( OR ( = ( CHR 92 )ulcam )  ( = "/" ulcam ) (= " : " ulcam
) )
( )
     ( SETQ cam ( STRCAT  cam  "\ \" ) ) ) ( IF ( SETQ listbl  ( TBLNEXT "block"  T ) )
( PROGN ( SETQ total ( LIST nil ) )
          ( disco )
        )

      ( PROMPT "Ningún bloque definido en el dibujo\n" ))( WHILE ( SETQ  listbl (
     TBLNEXT "block" ) )
        (disco)
     ) ) ( DEFUN disco ( / nom  nfich cafich )
     ( SETQ nom ( CDR ( ASSOC 2 listbl ) ) )( SETQ nfich ( SUBSTR nom  1 8 ) )(
     SETQ cafich ( STRCAT cam nfich ) )( IF ( FINDFILE ( STRCAT  cafich ".dwg" ) )
           ( opción ) ( PROGN ( COMMAND "bladisco"
           cafich nom ) ( SETQ totbl ( CONS nfich total
           ) ) )
     ) ) ( DEFUN opción  (  /  op )
     ( INITGET "Si No " )
     ( IF  ( SETQ op ( GETKWORD  ( STRCAT "Archivo " nfich " ya existe. Desea
          sustituirlo <N>: " ) ) ) ( ) ( SETQ op "No" ) )
           ( IF ( = op "Si" ) ( PROGN ( COMMAND "bladisco"
                cafich "s" nom ) ( SETQ totbl ( CONS nfich
                totbl ) ) ( COMMAND "bladisco" cafich " " ) )
     ) ) ( DEFUN  lista  (  /  n  nbl )
     ( SETQ nbl ( -( LENGTH total )  1 ) )( IF ( >  nbl Ø )
          ( PROGN ( PROMPT "\n \n" ) ( PROMPT " LISTADO DE BLOQUES\n" ) (
              PROMPT " ALMACENADOS\n" ) ( PROMPT ( STRCAT  "EN
              DIRECTORIO " cam "\n" ) ) ( PROMPT "---------------------------------------\n" )
              ( TERPRI ) ( SETQ n ( -nbl 1 ) ) ( REFEAT nbl
                           ( PROMPT ( NTH  n total ) ) ( TERPRI )
                      ( SETQ n ( -n 1 ) ))( TEXTSCR )
          )
     ) ) ( DEFUN  c:tododisco  (  /  totbl cam )
     ( SETVAR "cmdecho"  Ø ) ( SETVAR "biipmode" Ø )( todobl )( lista )( SETVAR
     "cmdecho"  1 ) ( SETVAR "blipmode"  1 ) ( PRIN1 )
```

13.9.2 Observaciones.

El aspecto más llamativo de este programa es la utilización de READ-LINE para leer desde el teclado la cadena de texto que indica la unidad y directorio para el almacenamiento de los archivos. Esto se hace asi porque GETSTRING lee el carácter “\” tal cual es. En cambio, READ-LINE lo leee de la forma “\ \”, que es como lo necesita FINDDF1LE para explorar la existencia de un archivo con el mismo nombre.

13.9.3 Explicación del programa.

Define una nueva orden y cuatro funciones de usuario. Las funciones se encuenttran anidadas, ya que una de ellas, "todobl", llama a otra “disco”, la cual a su vez llama a otra "opción".

• Función **c:tododisco**

Define una nueva orden de AutoCAD con ese nombre y lama sucesivamente a las funciones de usuario “todobl" y "lista".

• Función **todobl**

Su cometido es explorar 1a Tabla de Símbolos con todos los bloques del dibujo y efectuar un "Bladisco" con ellos, controlando si ya existe un archivo con el mismo nombre.

Se visualiza un mensaje para solicitar del usuario que indique la unidad y, optativamente, el directorio para almacenar los archivos. Se lee la respuesta con READLINE para que las contrabarras "\" se lean como "\ \". Para indicar los directorios se pueden emplear tanto "\" como "/". Se inicializa la variable "cam" como una cadena vacía. Se ofrece la opción por defecto "c:".

Se concatena una cadena de texto con "cam" (que puede tener la unidad por defecto "c:” o " ") y "dir", que es el directorio especificado por el usuario.

Así, por ejemplo, si el usuario responde con "Return", la variable "cam" almacenará:

```
c:
```

Si el usuario, por ejemplo, responde con "\bloques", la variable "cam" almacenaría:

```
\\bloques
```

Para que el camino quede en disposición de añadírsele después el nombre del archivo, hay que añadir al final una barra o contrabarra. Pero no siempre interesa hacer esto.

Es posible que el usuario introduzca, por ejemplo. "c:\" para grabar los archivos en el directorio raíz. En ese caso hay que evitar que el programa añada "\\" al final de ese camino.

Lo mismo si el usuario quiere grabar los archivos en el directorio actual de "a:". Por eso se establece un control a continuación. Así se extrae del camino introducido por el usuario el último carácter y se almacena en "ulcam".

Si ese último carácter es "\ \" o "/" o ":”, esto querrá decir que el usuario ha indicado una unidad de disco o se esta refiriendo a algún directorio raíz. Se

establece esa condición con OR teniendo la precaución de indicar "\\" para la contrabarra, porque si no AutoLISP no lo admite como carácter de texto. También se puede indicar (CHR 92), que es e) código ASCII. Si esto es asi la rama del SI se encuentra vacia. Si no es así, la rama del NO añade una ultima contrabarra al camino "cam".

Con (TBLNEXT "block" "1') se extrae de la Base de Datos la lista con la definición del primer bloque. Se establece una alternativa para que en el caso de no existir ningún bloque (rama del NO), se visualice un mensaje de advertencia.

Si ese primer bloque existe, la rama del SI inicializa una lista "totbl" que va a contener todos los bloques de los cuales se extraen archivos con "Bladisco", con el fin de escribir en pantalla un listado al final. Como hay problemas para crear una lista vacia, se crea una lista con un elemento nil, al cual se irán añadiendo posteriormente los bloques extraídos con "Bladisco".

A continuación se llama a la función "disco", cuya misión es obtener el nombre del bloque y aplicar "Bladisco" para generar un archivo.

Una vez realizada esa operación con el primer bloque, se establece un ciclo con WHILE para explorar el resto de bloques, llamando para cada bloque a la función "disco".

Variables de entrada: ninguna

Variables locales:

dir Directorio para almacenar bloques.
ulcam Ultimo carácter del camino para archivo.
lisibt Lista de bloque en Tabla de Símbolos.

Variables de salida:

cam Camino completo para grabar bloques.
totbl Lista con todos los bloques grabados.

• Función **disco**

Tiene como finalidad averiguar el nombre de cada bloque y obtener un archivo con "Bladisco".

El nombre "nom" se obtiene en la sublista de código 2. Por si el nombre del bloque tiene más de ocho caracteres, se obtiene con SUBSTR una subcadena con los ocho primeros caracteres que será el máximo permitido para el nombre del archivo. Este nombre se almacena en "nfich". El camino completo para grabar el archivo se obtiene concatenando "cam" con "nfich".

En los supuestos anteriores, si por ejemplo el nombre del bloque es "ventana", la variable "cafich" podría contener:

c: ventana \\bloques\\ventana

Se concatena "cafich" con la extensión ".dwg'* y se explora con FINDFILE si ya existe un archivo con ese nombre. Si existe la rama del SI llama a la función "opción" para que el usuario decida si desea volver a grabar el archivo. En caso contrario, la rama del NO efectua el "Bladisco" y añade el nombre del archivo obtenido a la lista "totbl".

Variables de entrada:
cam Camino para grabar archivos de bloques.
totbl Lista con archivos grabados.

Variables locales:

Variables de salida:
totbl Lista con archivos grabados actualizada.
nom Nombre de bloque.
nfich Nombre de archivo a grabar.
cafich Camino completo de archivo a grabar.

• Función **opcion**

Advierte al usuario mediante un mensaje que el archivo ya existe y solicita una respuesta. Se admiten con INITGET solamente las respuestas afirmativa o negativa, Por defecto se ofrece la negativa.

Se establece una alternativa. Si la respuesta del usuario almacenada en "op" es negativa, la rama del NO se encuentra vacia y el archivo no se graba. Si la respuesta es afirmativa, se llama a la orden "Bladisco" y se introduce "s" para que el archivo sea sustituido. Se añade el nuevo archivo a la lista "ttibl".

Variables de entrada:
cam Camino para grabar archivo de bloques.
totbl Lista con archivos grabados.

Variables locales:
nom Nombre de bloque.
nfich Nombre de archivo a grabar.
cafich Camino completo de archivo a grabar

Variables de salida:
totbl Lista de bloques grabados actualizada.

• Función **lista**

Tiene como misión visualizar en pantalla un listado con todos los archivos obtenidos con “Bladisco” El número total de archivos grabados se calcula con LENGTH, que proporciona la longitud de la lista “total”. Hay que rest 1 porque esa lista contiene un elemento de más, que es nil, al iniciarla.

A la listra se han ido añadiendo elementos mediante CONS, que los añade al

principio. Eso quiere decir que los nombres de archivos se encuentran al revés de como se lian ido obteniendo Por ejemplo si se han grabado en ese orden los archivo.

- ventana
- armario
- silla
- mesa

la lista "totbl" contendrá:

("mesa" "silla" "armario" "ventana" nil)

Aunque realmente da lo mismo el orden en que se muestre el listado, en el programa se respeta el inicial.

Antes de ofrecer el listado, se establece un control chequeando si "nbl" es mayor que Ø. Así, si no se ha encontrado ningún bloque en el dibujo o no se ha grabado ningún archivo, no se escribe en pantalla la cabecera del listado (cosa que no tendria sentido para una lista vacia).

Si existe algún nombre de archivo grabado, se escriben en pantalla dos retornos de carro y después la cabecera en tres lineas y un subrayado con guiones. Para ello se utiliza PROMPT. El contenido y presentación de esta cabecera queda a gusto del usuario. En la cabecera se visualiza también el directorio donde quedan grabados los archivos (variable "cam"). Es una información mas que se ofrece al usuario.

Mediante un ciclo con REPEAT se van extrayendo de la lista los nombres de archivos. Para la lista del ejemplo anterior, el número de archivos "nbl" es 4. Los elementos de la lista se numeran Ø, 1, 2, 3 y 4. Por tanto, el primer nombre a escribir en pantalla es el que ocupa la posición 3 (en este caso "ventana"), Por eso hay que restar 1 a "nbl" para obtener la posición dei primer nombre del listado. Ese valor se almacena en "n".

El ciclo se repite tantas veces como nombres de archivos hay que listar, restando en cada caso 1 a "n" para obtener la posición del siguiente nombre. El último a listar ocupará la posición Ø en "total".

La última instrucción cambia a pantalla de texto con el fin de visualizar el listado.

Variables de entrada:

totbl Lista final con archivos grabados.
cam Unidad y directorio de archivos grabados.

Variables locales:

nbl Número de archivos grabados.n Posición del nombre de archivo a escribir en pantalla.

Variables de salida: ninguna.

14 Capitulo 14: Acceso a pantalla gráfica y dispositivos de entrada.

En este capítulo se estudian los comandos que proporcionan un acceso directo a la pantalla gráfica de AutoCAD y a los periféricos de entrada de datos. Estos comandos son muy específicos y la mayoría de las aplicaciones en AutoLISP puede hacerse sin utilizarlos.

14.1 (GRCLEAR) Despejar ventana gráfica.

Este comando despoja la ventana gráfica actual. Es como si se hubiera visualizado con MIRAFOTO un archivo de foto de AutoCAD "en blanco", sin ninguna entidad contenida en el. Sólo afecta al área gráfica, no a la linea de estado, lineas de órdenes o área de menús de pantalla.

Se vuelve a la situación anterior con un simple redibujado, comando (REDRAW) de AutoLISP o también orden REDIBUJA de AutoCAD.

14.2 (GRDRAW <de> <a> <col> [<rel>]) Dibujar vector virtual.

Este comando dibuja un vector virtual en el área gráfica de pantalla, entre los dos puntos indicados en <de> y <a>. Los puntos son, como siempre, listas de dos o tres números reales. Las coordenadas se refieren al SCP actual del dibujo.

El vector se visualiza con el número de color indicado en <col>. Si se indica -1 como número de color, el vector se visualiza en "O-exclusivo" (borra al dibujarse por encima de algo, y es borrado si se dibuja algo nuevo por encima de él). Si se especifica un argumento de relieve <rel> y es diferenle de nil, el vector se visualiza con "puesta en relieve" (video inverso, brillo u otro sistema).

Los vectores dibujados con este comando son virtuales, es decir, no forman parte del dibujo y desaparecerán con un redibujado o una regeneración.

Por ejemplo:

```
( GRDRAW ' ( 5Ø 5Ø ) ' ( 2ØØ 2ØØ ) 1 T )
```

dibujará un vector desde el punto “5Ø,5Ø” hasta el punto “2ØØ,2ØØ” en color "1" (rojo) y con puesta en relieve.

14.3 (GRTEXT [<casilla> <texto> [<rel>]]) Escribir en áreas de texto.

Este comando se utiliza para escribir textos virtuales en las áreas de texto de la pantalla gráfica de AutoCAD. Según en cuál de las tres áreas de texto se

quiera escribir, se utilizara el comando de manera diferente.

• Área de menús de pantalla

Se indica el número de casilla del área de menús. Este número debe ser un entero positivo o Ø. Las casillas se numeran de arriba abajo empezando por el Ø, hasta el número máximo de líneas permitidas por la interfaz gráfica. Por ejemplo, una tarjeta gráfica VGA permite hasta 26 lineas en el área de menús; por tanto, las casillas se numeran de Ø a 25.

Una vez indicado el número de casilla, se especifica el texto (entre comillas) que se desea visualizar en esa casilla. El texto se truncará si no cabe entero en la casilla, o se completará con blancos en el caso de que sobren caracteres.

Si se indica el argumento de relieve <rel> y su valor es diferente de Ø, el texto se pondrá de relieve en la casilla correspondiente. Si el valor de <rel> es Ø, el texto vuelve a su visualización normal. Al indicur un argumento de relieve no cambia el texto de la casilla, sino sólo su visualización.

Por ejemplo:

```
( GRTEXT 8 "HOLA" )
( GRTEXT 8 "QUE TAL" 1 )
```

La primera utilización del comando GRTEXT escribe el texto "HOLA" en la casilla 8. En la segunda utilización se produce una puesta de relieve en esa casilla, pero el texto sigue siendo "HOLA", no ha cambiado. Por eso hay que escribir primero el texto que se desee y después ponerlo de relieve:

```
( GRTEXT 8 "HOLA" )
( GRTEXT 9 "HOLA" 1 )
```

En este caso, para poner de relieve el texto de la casilla 8 se ha indicado el mismo texto que ya tiene escrito. Si se suministra un valor de texto diferente, se pueden producir comportamientos anómalos del programa en AutoLISP.

El texto escrito en la casilla indicada es virtual, no modifica la opción de menú contenida debajo. En cuanto se cambie de submenú o se produzca un redibujado del área de menús, desaparecerán los textos virtuales escritos con GRTEXT. Como el menú de SCREEN sumnistrado por AutoCAD utiliza 21 lineas, todo lo que se escriba más abajo con GRTEXT permanecerá normalmente en pantalla.

• Linea de estado (Visualización de Capa y Modos)

Para visualizar un texto en la zona izquierda de la linea de estado (Visualización de Capa y Modos) hay que especificar un número de casilla -1. La longitud máxima del texto depende de la tarjeta gráfica (generalmente se admiten más de 40 caracteres). El argumento de puesta en relieve no tiene efecto.

```
( GRTEXT -1 "DISEÑO ASISTIDO POR COMPUTADORA" )
```

• Línea de estado (Visualización de coordenadas)

Para escribir el texto en la linea de estado en la zona de visualización de coordenadas hay que indicar un número de casilla -2. El argumento de puesta en relieve no tiene efecto.

```
( GRTEXT -2 "EJECUTANDO AutoLISP" )
```

Si las coordenadas se encuentran activadas, en cuanto haya un movimiento del cursor se redibujará esta área para visualizar las coordenadas, con lo que el texto desaparecera.

Por último, si se llama al comando sin argumentos, restablecerá todas las áreas a su estado original, desapareciendo todos los textos virtuales que se hayan escrito.

14.4 (GRREAD [<cont>]) Lectura directa de datos de entrada.

Este comando permite la lectura directa de dispositivos de entrada. Si se suministra el argumento <cont> con un valor diferente de "nil", se activa la lectura continua de dispositivos señaladores en movimiento. En este caso GRREAD acepta el primer valor del dispositivo, sin esperar a que se pulsen botones selectores.

En todos los casos GRREAD devuelve una lista cuyo primer elemento es un código que indica el tipo de dato que viene a continuación. El segundo elemento es el dato concreto de que se trate.

En el Cuadro 14.1 se detallan los códigos para el primer elemento.

Código	**Tipo de dato**	**Segundo elemento**
2	Carácter del teclado.	Código ASCII.
3	Punto designado.	Lista coordenadas.
4	Casilla de menú designada.	Número de casilla.
5	Coordenadas dinámicas (sólo si <cont> no es "nil").	Valor de coordenadas.
6	Opción de menú BUTTONS.	Número de pulsador.
7	Opción de menú TABLET1.	Número de casilla.
8	Opción de menú TABLET2.	Número de casilla.
9	Opción de menú TABLET3.	Número de casilla.

1Ø	Opción de menú TABLET4.	Número de casilla.
11	Opción de menú AUX1.	Número de casilla.
12	Coordenadas de opción 6 (siempre acompañada a la lista de código 6)	Valor de coordenadas en el mometo de pulsar el botón.
13	Opción de pantalla designada mediante teclado.	Número de casilla.

Este comando sólo se empleara en aplicaciones muy especializadas.

15 Capitulo 15: Funciones de chequeo y operaciones binarias.

Se estudian en este capítulo los comandos de AutoLISP que se utilizan para examinar los símbolos, ya sean variables, funciones, valores concretos, etc., y detectar una característica concreta de esos símbolos. Se puede conocer asi si un dato es Ø o no, si una variable existe o es nil, si un valor almacenado es una lista o un átomo, de que tipo es un elemento de una expresión, etc.

15.1 (ATOM <elem>) Detectar átomo.

Este comando examina el elemento indicado <elem> y devuelve T si es un atomo o "nil" si es una lista. Todo lo que no sea una lista se considera un atomo.

(ATOM ' (3 5)) devuelve nil
(ATOM 3) devuelve T
(ATOM c) devuelve T

En el último ejemplo, aunque la variable "c" no esté creada (su contenido es "nil"), como no es una lista ATOM devuelve T.

(ATOM (SETQ x 5)) devuelve T
(ATOM ' (SETQ x 5)) devuelve nil

15.2 (BOUNDP <simb>) Detectar valor asociado a símbolo.

Este comando examina el símbolo indicado <simb> y devuelve T si tiene asociado un valor diferente de nil. El símbolo puede ser una variable, un nombre de función definida por elusuario o incluso un nombre de comando de AutoLISP. Únicamente si el símbolo es nil BOUNDP devuelve nil.

Puesto que el comando BOUNDP examina nombres de símbolos, hay que indicar estos sin evaluar, precedidos por tanto de QUOTE.

(SETQ x 25)
(DEFUNdelta ( ) (SETQ n (+ 1 n)))
(SETQ y x)

(BOUNDP ' x) devuelve T
(BOUNDP ' delta) devuelve T
(BOUNDP ' Y) devuelve T
(BOUNDP ' dblin) devuelve nil
(BOUNDP ' setq) devuelve T

Suponiendo que el símbolo "dblin" no tiene valor asociado (no se ha creado una variable o una función de usuario con ese nombre, o bien se ha eliminado de la memoria igualándola a nil), entonces BOUNDP aplicado a ese símbolo devuelve nil.

15.3 (LISTP <elem>) Detectar lista.

Este comando explora el elemento indicado <elem> y devuelve T en el caso de que se trate de una lista y “nil” si no lo es.

(SETQ 11 ' (1Ø 25 Ø))
(SETQ x 3Ø)

(LISTP 11) devuelve T
(LISTP ' 11) devuelve nil
(LISTP x) devuelve nil
(LISTP (LIST x 25)) devuelve T

15.4 (MINUSP <elem>) Detectar valor numérico negativo.

Este comando devuelve T si el valor del elemento indicado <elem> es un número (entero o real) negativo. En caso contrario devuelve nil.

Es decir, examina el resultado de la evaluación de <elem>. Este resultado tiene que ser numérico. Si no es así, MINUSP produce un error.

(MINUSP -5) devuelve T
(MINUSP -1.234) devuelve T
(MINUSP 28) devuelve nil
(MINUSP (SETQ z -5)) devuelve T

Como MINUSP examina resultados, se puede indicar como elemento una expresión en AutoLISP.

15.5 (NULL <elem>) Detectar valor asociado nulo.

Este comando examina el valor asociado al elemento que se indica <elem>. Devuelve T si ese valor asociado es nil y devuelve nil en caso contrario.

(SETQ x 25)
(DEFUN delta ( ) (SETQ n (+ 1 n)))
(SETQ Y x)

(NULL x) devuelve nil
(NULL delta) devuelve nil
(NULL ' delta) devuelve nil
(NULL dbiin) devuelve T
(NULL w) devuelve T

Existe una diferencia importante con BOUNDP. En este caso se examinan resultados de evaluación de símbolos y no los propios símbolos. Por eso no interesa indicar literales con NULL, puesto que el resultado de la evaluación de un literal de simbolo es el propio nombre de ese símbolo. Como ese valor no es nil, NULL aplicado a literales devolverá siempre nil.

15.6 (NUMBERP <elem>) Detetctar valor numérico.

Este comando examina el valor del elemento <elem> (el resultado de su evaluación) y devuelve T si es un número entero o real. De lo contrario, devuelve nil.

(SETQ x 25)
(SETQ a "Buenos dias")

(NUMBERP x) devuelve T
(NUMBERP a) devuelve nil
(NUMBERP (SETQ y 32)) devuelve T
(NUMBERP ' x) devuelve nil

Puesto que NUMBERP examina resultados, se puede indicar como elemento una expresión en AutoLISP.

15.7 (ZEROP <elem>) Detectar valor numérico igual a cero.

Este comando examina el valor del elemento indicado <elem> (su resultado) y devuelve T si este resultado es Ø, y nil en caso contrario. El resultado tiene que ser numérico, pues si no se produce un error.

(SETQ x 25)
(SETQ y 0)

(ZERO x) devuelve nil
(ZERO y) devuelve T
(ZERO ‘ y) produce un error
(ZERO (- x 25)) devuelve T

Se puede indicar como elemento una expresión en AutoLISP.

15.8 (TYPE <elem>) Tipo de elemento.

Devuelve el tipo de elemento indicado en <elem>. Este puede ser un símbolo, un valor concreto, una expresión en AutoLISP, un nombre de comando, etc.

Los tipos devueltos por TYPE pueden ser los siguientes:

REAL	Valor numérico real.

INT	Valür numérico entero
STR	Valor textual (cadena de caracteres).
FILE	Descriptor de archivo.
PICKSET	Conjunto designado de AutoCAD.
ENAME	Nombre de entidad de AutoCAD.
SYM	Símbolo.
LIST	Lista (y también funciones de usuario).
SUBR	Comando de AutoLISP o subrutina.
PAGETB	Tabla de paginación de funciones.
EXSUBR	Subrutina externa a función ADS: Nuevo tipo incorporado en la Versión 11 (véase Capítulo 17).

Por ejemplo, según el tipo de elemento indicado, el comando TYPE devolvería los siguientes valores:

(SETQ x 53 y 27.5 cad "HOLA" '(a b c))
(SETQ fich (OPEN "ej.lsp" "a"))
(SETQ conj (SSGET) nent (SSNAME conj 0))

(TYPE x) devuelve INT
(TYPE 'x) devuelve SYM
(TYPE y) devuelve REAL
(TYPE fich) devuelve FILE
(TYPE conj) devuelve PICKSET
(TYPE nent) devuelve ENAME
(TYPE cad) devuelve STR
(TYPE 1) devuelve LIST
(TYPE setq) devuelve SUBR
(TYPE (setq z 27)) devuelve INT

En el último ejemplo se ha indicado como argumento de TYPE una expresión. En este caso la evaluación de esa expresión devuelve un número entero (el 27) y TYPE devuelve como tipo de elemento INT.

Las tablas de paginación de funciones (tipo PAGETB) son elementos que se añaden como primer término de las listas que definen funciones de usuario, cuando la paginación virtual de funciones se encuentra activada. De esto se hablara al explicar el comando VMON.

El tipo de demento devuelto por TYPE está siempre en mayúsculas y es un nombre, no una cadena de texto (no va entre comillas).

15.9 (TRACE <fun>) Marcar función con atributo de

rastreo.

Este comando marca la función indicada <func> con un atributo de rastreo y devuelve el nombre de la última función. Se utiliza como ayuda para la puesta a punto del programa en AutoLISP (su "depuración"). Cada vez que la función <func> es evaluada, se visualiza en pantalla la llamada a esa función y el resultado de su evaluación. La llamada aparece con un "sangrado" (espacios en blanco al principio de la línea) proporcional al nivel de anidación de esa función.

El nombre de función tiene que ser el de una función definida de usuario.

Tomando como ejemplo sencillo la poligonal del apartado 4.8 del comando REPEAT, se modifica para que haga un número indeterminado de lineas:

```
( DEFUN dibl ( )
        ( SETQ pt2 ( GETPOINT  ptl "Nuevo punto: " ) ) (TERPRI )
        ( COMMAND "line" pt1 pt2 " " )
        ( SETQ pt1 pt2 )
)

( DEFUN C: pgl ( )
        ( SETQ ptl ( GETPOINT  "Introducir punto: " ) ) ( TERPRI )
        ( REPEAT 4
                (dibl )
        )
)

( TRACE dibl )
```

Se carga el archivo que contiene los dos programas definidos, “dibl” y "c:pgl", y se llama a la nueva orden de AutoCAD “pgl”. Como se ha activado el atributo de rastreo para la función “dibl”, cada vez que sea llamada dentro de la repetitiva se visualizará el resultado de su evaluación. Este resultado será el de su última expresión que os (SETQ ptl pt2), por tanto el valor en ese momento de la variable pt2.

```
Command:pgl
        Entering DIBL:
Command: Result: ( 2ØØ.Ø 5Ø.Ø Ø.Ø )
        Entering DIBL:
Command: Result: ( 2Ø5.Ø 65.Ø  Ø.Ø )
        Entering DIBL:
Command: Result: ( 23Ø.Ø 7Ø.Ø Ø.Ø )
        Entering DIBL:
Command: Result: ( 25Ø.Ø 1ØØ.Ø Ø.Ø )
```

Cada una de las cuatro veces que se recorre la repetitiva, la llamada a la función “dibl" se visualiza con el mensaje "Entering DIBL", que aparece “sangrado" dos espacios en blanco, porque esa llamada está incluida dentro de

(DEFUN c:pgl) y a su vez dentro de (REPEAT 4), por tanto en un segundo nivel de anidación. Una vez evaluada esa función, se visualiza el resultado de la forma "Result: ".

15.10 (UNTRACE <func>) Desactivar atributo de rastreo.

Este comando desactiva el atributo de rastreo de activado por TRACE. Devuelve el nombre de la función <func> indicada.

Command: ***(UNTRACE dibl)***

15.11 *ERROR* <cad> Función de ERROR.

No es un comando, sino una función de manipulación de errores (hay que emplearla con DEFUN). Si se ha especificado esta función, cada vez, que ocurra un error al ejecutar un programa de AutoLISP se ejecutará esa función. El único argumento que admite es una variable en la que AutoLISP va a almacenar la cadena de texto que describe el tipo de error ocurrido. Por ejemplo: "bad argument type".

Command:

(DEFUN *error* (cader)(IF (= cader "bad argument type") .....))

En el ejemplo la variable "cader" almacena la cadena de texto que describe el error. Se utiliza aquí la función *error* para detectar si el error es del tipo "bad argument type". Y mediante un comando IF efectuar una acción en el caso de que esto ocurra.

En todos los programas vistos al final de los diferentes capítulos se ha insistido en procurar tener previstos los posibles errores. Sin embargo, siempre existe un riesgo de que por la indebida utilización del programa o por una acción no prevista por el programador, se produzca un error y éste quede abortado. AutoLISP devuelve entonces un listado de la parte del programa en que se ha producido el error, además de todas las funciones que engloban esa parte (es decir, hasta el paréntesis más amplio, que es normalmente un DEFUN).

Este listado puede resultar molesto para el usuario. Una rutina muy sencilla de tratamiento de errores que evita esto es la siguiente:

```
( DEFUN *error* ( cad )
        ( PRINC "Error:  " ) ( PRINC cad )  ( TERPRI )
        ( PROMPT "*NO vale*" )  ( PRIN1 )
)
```

Al producirse un error de AutoLISP, el texto de ese error (variable asociada “cad”) se escribe con PRINC, y en vez de ofrecer todo el listado del programa se visualiza simplemente un PROMPT "*No vale”.

Esta rutina puede añadirse sin ningún problema a todos los programas explicados.

La versión 11 genera un código cada vez que se produce un error de AutoLISP. Véase el apartado 17.3.3 del Capitulo 17.

15.12 (VER) Versión de AutoLISP.

Este es un comando sin argumentos, simplemente informativo. Devuelve una cadena de texto que contiene el número de versión de AutoLISP cargado.

(VER) podría devolver "AutoLISP Release 1Ø.Ø"

Si está cargado AutoLISP AT, el comando VER también lo indica:
(VER) "AutoLISP AT Release 1Ø.Ø"

15.13 Operaciones a nivel binario.

En este apartado se agrupan una serie de comandos de AutoLISP que realizan operaciones a nivel binario. Esto quiere decir que, aunque admiten que se indiquen los números en decimal o en hexadecimal, los consideran como binarios (conjunto de ceros y unos). Su utilidad, por tanto, es más bien específica para programadores.

15.13.1 (~ <num>) Negación lógica.

Este comando devuelve la negación lógica de una cifra binaria, es decir, el complemento a I. El número indicado tiene que ser entero.

(~ 5) devuelve -6
(~ -6) devuelve 5
(~ Ø) devuelve -1
(- 54) devuelve -55

El carácter de negación ~ corresponde al código ASCII número 126.

15.13.2 (BOOLE <func> <ent1> <ent2>...) Operación Booleana.

Este comando realiza una operación Booleana general a nivel binario. El argumento <func> es un número entero entre Ø y 15 que representa una de las 16 funciones Booleanas posibles. Los valores enteros que después se indiquen, <entl>, <ent2>, etc., se combinarán bit a bit de acuerdo con la función Booleana especificada.

Es decir, el primer bit de <entl> se combina con el primero de <ent2>, y asi sucesivamente. El resultado final será Ø ó 1 según la tabla de la verdad de la función Booleana indicada. Lo mismo con todos los demás bits de los números

enteros especificados. La combinación final de todos los bits resultantes dará el número entero final que devuelve et comando.

Las operaciones Boolenaas con “nombre propio” son AND, OR, XOR y NOT. Los valores de <func> que corresponden a ellas y el tipo de operación son los siguientes:

Función	Operación	Bit resultante es 1 (cierto) si:
1	AND	Todos los bits de entrada son 1.
6	XOR	Sólo uno de los bits es 1.
7	OR	Al menos 1 de los bits es 1.
8	NOT	Todos los bits de entrada son Ø.

Por ejemplo:

(BOOLE 6 8 12 7) equivale a un XOR de:

8 que es el binario 1ØØØ12 que es el binario 11ØØ7 que es el binario 111

Por tanto la operación XOR se hará cuatro veces:

XOR de 1 1 Ø devuelve Ø
XOR de Ø 1 1 devuelve Ø
XOR de Ø Ø 1 devuelve 1
XOR de Ø Ø 1 devuelve 1

El resultado final es el número binario ØØ11, que equivale a 3. Por tanto:

(BOOLE 6 8 12 7) devuelve 3

15.13.3 (LOGAND <ent1> <ent2>...} Operación Y lógico.

Este comando realiza una operación de AND (Y lógico) a nivel binario de todos los números indicados.

Estos tienen que ser enteros y el resultado es también un entero.

Es un caso particular de operación Booleana visto antes. LOGAND equivale a hacer BOOLE 1.

(LOGAND 5 7 12 14) devuelve 4
(BOOLE 1 5 7 12 14) devuelve 4

15.13.4 (LOGIOR <ent1> <ent2>...) Operación O lógico

Devuelve el resultado de un OR (O lógico) a nivel binario de todos los números indicados.

Estos deben ser enteros y el resultado también es entero.

Es un caso de operación Booleana. LOGIOR equivale a BOOLE 7.

(LOGIOR 1 4 9) devuelve 13
(BOOLE 7 1 4 9) devuelve 13

15.13.5 (LSH <ent> <numbits>) Desplazamiento a nivel binario.

Este comando devuelve el desplazamiento a nivel binario de un registro del número entero indicado <ent> en un valor del número de bits especificando <numbits>. Si este es positivo, el entero se desplaza hacia la izquierda. Si el número de bits es negativo, se desplaza hacia la derecha.

Los datos que faltan entran como Ø y se pierden los dígitos salientes. Si entra o sale un dígito I como bit más significativo (será el 16 en MS-DOS o el 32 en estaciones de trabajo) de un entero, cambia su signo.

En el cuadro siguiente se muestran varios ejemplos donde se indica, en primer lugar, el comando en AutoLISP; después, el número entero original indicado en el comando; a continuación, el binario equivalente a ese entero; después, el valor del desplazamiento especificado; a continuación, el binario resultante una vez aplicado ese desplazamiento; por último, el entero equivalente a ese binario, que es el resullado final que devuelve el comando.

Comando AntoLISP	Número entero	Binario equivalen t	Valor de desplazamiento	Binario resultante	Entero equivalemte
(LSH 2 1)	2	1Ø	1	1ØØ	4
(LSH 2-1)	2	1Ø	-1	1	1
(LSH 4Ø 2)	4Ø	1Ø1ØØØ	2	1Ø1ØØØØØ	16Ø
(LSH 11 -1)	11	1Ø11	-1	1Ø1	5

En MS-DOS, si se hace (LSH 16384 I), como e! máximo de dígitos piirn un número entero es de 56 bits, al desplazar 1 hacia la izquierda.el bit número 17 queda con valor 1.Pero ese bÍL en MS-DOS es el indicador, de signo negativo y por eso el comando devolvería -32768.

16 Capitulo 16: Gestión de la memoria

Este capítulo no contiene comandos de AutoLISP que gestionan la memoria disponible para almacenar las variables, definiciones de funciones y valores de los programas cargados. Resultan muy útiles cuando los requerimientos de memoria son grandes o la memoria disponible es pequeña, lo que puede provocar problemas.

Hay que tener siempre presente que cuanto mayor es el número de variables y de funciones definidas, mayores son las necesidades de memoria. Al mismo tiempo, si sus nombres ocupan más de seis caracteres, se consume mas memoria. Todas estas circunstancias pueden forzar al usuario a liberar memoria o a activar la paginación virtual de funciones, concepto que se explica a continuación.

16.1 (VMON) Paginación virtual de funciones.

Este comando activa la paginación virtual de funciones. Cuando se tiene cargado en memoria un programa extenso de AutoLISP o varios programas, puede ser insuficiente el espacio de memoria nodal disponible (controlado por la variable de entorno LISPHEAP). Se recuerda aquí que la memoria nodal almacena todos los símbolos de funciones de usuario y de variables cargados en memoria. Estos símbolos son los también llamados "nodos".

Si esto ocurre, se puede emplear el comando VMON para activar la paginación virtual de funciones. Desde el momento en que se llame al comando, esa paginación virtual afectara únicamente a las funciones de usuario (definidas mediante DEFUN) cargadas a partir de ese momento. Cada vez que se carguen nuevas funciones de usuario y la memoria nodal sea insuficiente. AutoLISP ira evacuando las funciones poco utilizadas a un archivo temporal controlado por la paginación de archivos de AutoCAD. Si esas funciones evacuadas se volvieran a necesitar, AutoLISP volvería a cargarlas en memoria nodal. Estos intercambios son automáticos y transparentes para el usuario.

Si se dispone de suficiente memoria RAM ampliada o paginada, la utilización de esa memoria será lógicamente mucho más rápida que la paginación en disco.

Las funciones de usuario cargadas antes de la llamada a VMON no pueden ser evacuadas de la memoria nodal; por tanto, seguirán ocupando espacio. De la misma manera, todos los símbolos de variables continúan en memoria nodal y no son afectados por VMON. Esto quiere decir que, aunque se hayan ampliado mucho las posibilidades de cargar programas largos, sigue existiendo una limitación de memoria nodal disponible (según el valor de LISPHEAP).

La paginación virtual de funciones no afecta al valor de la memoria montón (variable LISPSTACK). Una vez activada con VMON ya no puede ser desactivada hasta salir de la actual sesión del Editor de Dibujo.

Con VMON activada, todas las nuevas DEFUN colocan un nuevo nodo llamado "tabla de paginación" como primer elemento añadido a la lista de AutoLISP que define la función. Este primer elemento no es visible a la hora de escribir (en pantalla o en un archivo) esa función. Pero se puede obtener con CAR El comando TYPE devuelve el tipo de elemento PAGETAB para esas "tablas de paginación".

16.2 (GC) Recuperación de memoria inutilizada.

Cada vez que se pretende cargar en memoria un nuevo símbolo de función o de variable. AutoLISP busca en la memoria nodal nodos libres donde almacenarlo. La simple operación de crear una variable y atribuirle un valor, por ejemplo:

```
( SETQ var 27.53 )
```

requiere tres nodos: uno para incluir el símbolo de esa variable en la lista de ATOMLIST; el segundo para almacenar en memoria el nombre del símbolo (en este caso, var), y el tercero para almacenar su valor (aquí, 27.53).

Si el nombre del símbolo tiene seis caracteres o menos, se almacena directamente en el nodo. Si tiene más, se toma espacio adicional de la memoria montón. De ahí la importancia de utilizar nombres de símbolos (funciones o variables) con menos de seis caracteres.

Si no existen nodos libres para almacenar ese símbolo, AutoLISP recupera automáticamente el espacio inutilizado (nodos que se han liberado porque ya no tienen ningún símbolo asociado). Si esto resulta insuficiente, solicita memoria adicional del montón (LISPHEAP) para crear nuevos nodos. En el momento en que se agote esta memoria tendrá que recurrir a la paginación virtual de funciones si se encuentra activada (si se ha empleado el comando VMON).

El espacio de memoria tomada del montón (LISPMEAP) para crear nuevos nodos ya no puede ser devuelta al montón hasta salir de AutoCAD. Sin embargo, existe la posibilidad de forzar la recuperación de esa memoria mediante el comando GC:

```
( GC )
```

Esta recuperación de memoria lleva bastante tiempo y su utilidad es limitada. Es preferible dejar que AutoLISP vaya recuperando automáticamente los nodos liberados en memoria sin necesidad de llamar a este comando.

16.3 (ALLOC <num>) Tamaño de segmentos en memoria nodal.

Los nodos en AutoLISP tienen un tamaño de 1Ø octetos para MS-DOS y 12

octetos para estaciones de trabajo. AutoLISP AT usa nodos de 12 octetos. Para evitar un fraccionamiento excesivo de la memoria "heap", los nodos se agrupan en segmentos. Cada segmento tiene un tamaño implícito de 512 nodos, es decir, 521Ø octetos para MS-DOS.

Si se desea especificar un número de nodos diferente a 512 como tamaño del segmento, se puede hacer con el comando ALLOC:

```
( ALLOC 1Ø24 )
```

asignará a cada segmento un tamaño de 1Ø24 nodos, lo que representará 1Ø 24Ø octetos en MS-DOS.

Asignando un tamaño adecuado a los segmentos se pueden reducir las operaciones de recuperación de memoria inutilizada, ganando en velocidad de ejecución del programa. De todas formas, salvo que se tenga una buena experiencia en programación AutoLISP, es preferible dejar la tarea de gestión de memoria al mecanismo automático de atribución de nodos de AutoLISP, explicado en el apartado anterior.

16.4 (EXPAND <num>) Numero de segmentos en memoria nodal.

El espacio de memoria para almacenar valores de cadenas se toma de la misma memoria montón “heap” que los segmentos de nodos. Estos valores de cadenas son de todo tipo: nombres de símbolos de mas de seis caracteres, valores concretos de variables de texto, textos de mensajes, textos de opciones de menú, etc.

Por tanto, en la memoria de montón “heap”, además, de la utilizada para los segmentos de nodos, se requiere una parte para cadenas de texto.

El comando EXPAND reserva explícitamente para nodos un número determinado de segmentos de la memoria montón "heap".

```
( EXPAND 1Ø )
```

reservará 1Ø segmentos que, si son de 512 nodos cada uno, representarán en MS-DOS un espacio de 51 2ØØ octetos, que es más de lo disponible bajo DOS con AutoLISP (como máximo 45 ØØØ octetos para la suma de LISPHEAP y LISPSTACK).

Por eso es muy frecuente que EXPAND no reserve todos los segmentos que se le indiquen, sino aquellos que realmente ha podido obtener de la memoria montón disponible. El comando devuelve el valor del número de segmentos que ha conseguido reservar.

No es posible reservar toda la memoria montón disponible para segmentos de nodos. AutoLISP necesita que exista una parte libre para la memoria de

cadenas.

16.5 (MEM) Estadística de la memoria.

Este comando visualiza el estado actual de la memoria en AutoLISP y devuelve NIL.

Por ejemplo, el comando podría devolver:

```
( MEM )
```

Nodes: 2Ø56Free nodes: 17ØSegments: 4Allocate: 514Collections: 13

Cada término indica lo siguiente:

Nodes: el número total de nodos atribuidos hasta ese momento. Tiene que ser igual al número de segmentos multiplicado por el tamaño en nodos de cada segmento.

Free nodes: es el número de nodos que se encuentran "libres" como consecuencia de la recuperación de memoria no utilizada.

Segments: es el número de segmentos atribuidos.

Allocate: indica el tamaño en nodos del segmento actual.

Collections: es el total de recuperaciones de memoria no utilizada que han sido efecctuadas (ya sea automática o manualmente).

Si se encuentra activada la paginación virtual de funciones (comando VMON), se visualizan dos nuevos campos:

Swap-ins: número de funciones que se han cargado del archivo de paginación creado.

Page file: es el tamaño del archivo de paginación creado para almacenar temporalmente las funciones transferidas de la memoria nodal.

17 Capitulo 17: AutoLISP Versión 11. Nuevos comandos.

La nueva versión de AutoLISP que acompaña a AutoCAD V. 11 incorpora una serie de nuevos comandos que se explican en este capitulo. En primer lugar se tiene en cuenta el nuevo tipo de entidad de la Versión 11, que es la Referencia Externa de Bloque. También la posibilidad de cargar programas desarrollados en Lenguaje C a través del ADS (AutoCAD Development System). Existen mejoras en la especificación de cadenas de texto y en la conversión de valores a diferentes unidades.

Por último se dispone de nuevos comandos para la gestión de la Base de Datos, pudiéndose incluso generar una entidad completamente nueva construyendo sus listas de asociaciones mediante ENTMAKE. Hay que mencionar también los dos nuevos tipos de Tablas de Símbolos: "DIMSTYLE" y "APPID".

17.1 Cconceptos previos.

El llamado ADS (AutoCAD Development System) es un entorno que permite desarrollar programas en Lenguaje C para AutoCAD. Estos programas consisten en un conjunto de funciones externas que pueden ser cargadas o llamadas desde un programa en AutoLISP.

La utilización de Lenguaje C posibilita realizar aplicaciones muy complejas y de una programación especializada, incluyendo especificaciones en el Sistema Operativo y Hardware.

Cuando un conjunto de rutinas de ADS se carga desde un programa en AutoLISP, éstas son incorporadas como un nuevo tipo de objeto en AutoLISP llamado "Subrutinas externas" o "Funciones ADS", que debe distinguirse de las "Funciones inherentes" o Comandos, y de las "Funciones de usuario" enumeradas junto con el resto de objetos AutoLISP en el apartado 1.3 del Capitulo 1.

Este nuevo tipo de objeto se detecta con el comando TYPE, que lo devuelve como EXSUBR. Los tipos completos devueltos por TYPE se detallan en el Capitulo 15, apartado 15.8.

Los comandos que permiten cargar y descargar aplicaciones ADS, XLOAD y XUNLOAD se estudian más adelante en este mismo capitulo.

En cuanto a las convenciones de AutoLISP para interpretar los valores de cadenas de texto, se añade un nuevo carácter de control para incluir las comillas dentro de las cadenas. En la Versión 1Ø habia que indicarlo de la forma (CHR 34), mientras que ahora se puede especificar el carácter de control:

\" que significa el carácter comillas"

Para todos los códigos de control reconocidos, véase el apartado 1.5 del Capitulo 1.

17.2 Nuevos comandos relativos a unidades y cadenas de texto.

Se explica en este apartado los nuevos comandos de la Versión 11, dejando aparte los relativos al ADS y al acceso a la Base de Datos, más específicos.

17.2.1 (CVUNIT <valor> <desde> <a>) Convertir valor a nuevas unidades.

Este comando conviene el valor indicado, desde las unidades especificadas en <desde> hasta las unidades especificadas en <a>. Devuelve el resultado de la conversión de ese valor, Si alguna de las unidades no existe o la conversión resulta incoherente, devuelve nil.

Las unidades se indican con sus nombres como cadenas de texto (por tanto, entre comillas). Esos nombres deben existir en el archivo ACAD.UNT, que es un nuevo archivo de definición de unidades incorporado en AutoCAD V. 11.

Este archivo contiene las maneras más habituales de indicar los nombres de unidades. Así, por ejemplo, los "metros", se pueden indicar:

metro (s) , m

Es decir, son válidas las formas "metro", "metros" y "m".

Para las unidades cuadradas o cúbicas, el exponente se indica de la forma "'2" y "'3". Así los metros cuadrados y cúbicos se indican:

"mˆ2" y "mˆ3"

Se pueden convertir de unas unidades a otras no sólo valores numéricos, sino también valores de punto (listas de dos o tres coordenadas). A continuación se muestran algunos ejemplos:

```
( CVUNIT 18Ø  "grado" "radian" )        devuelve 3.14159
( CVUNIT 1Ø  "cm" "pulgada" )           devuelve 3.937Ø1
( CVUNIT 25 "celsius" "kelvin" )        devuelve 298.1.5
( CVUNIT 1.25 "hora" "segundo" )        devuelve  45ØØ
( CVUNIT 25ØØ "m 2"  "acre" )           devuelve Ø.617763
( CVUNIT 15 "kg" "libra" )              devuelve 33.Ø693
( CVUNIT ' ( 2 5 7 ) "mm" "pulgada" )   devuelve ( Ø.Ø7874Ø2
                                                  Ø.19685
                                                  Ø.275591 )
( CVUNIT 76Ø  "grado" "círculo" )       devuelve 2.11111
```

Los nombres de unidades contenidos en el archivo ACAD.UNT se pueden obtener, por ejemplo, con un “type" o simplemente sacando ese archivo por impresora.

Para la conversión de unidades, el programa en AutoLISP necesita acceder cada vez a ese archivo de texto y leer su contenido. Esto resulta bastante lento. Por eso, si un programa requiere una frecuente conversión de unidades, es preferible calcular el factor con CVUNIT sólo la primera vez y después emplear ese factor directamente sin repetir CVUNIT.

17.2.2 (TEXTPAGE) Conmutar a Pantalla de Texto borrando.

Este comando conmuta a Pantalla de Texto, de la misma forma que TEXTSCR (véase el Capitulo 2, apartado 2.6) La diferencia radica en que TEXTPAGE efectúa además un borrado de pantalla (un “clear screen”). Es decir, las últimas órdenes realizadas desde AutoCAD se borran de la pantalla de texto y la pregunta "Orden;" se visualiza al principio de la pantalla.

Este comando es útil cuando se escriben listada en pantalla desde un programa en AutoLISP, y no se desea el efecto de "persiana" producido cada vez que se escribe un listado y desplaza al anterior hacia “arriba" en la pantalla. El comando TEXTPAGE siempre devuelve nil.

17.2.3 (WCMATCH <cad> <fil>) Comparar cadena de texto con filtro.

Este comando aplica el filtro o patrón indicado en <fil> a la cadena de texto <cad>. Compara la cadena con el filtro para ver si lo cumple: si es así devuelve T, y en caso contrario, nil.

El filtro se forma con un conjunto de caracteres globales o "comodín"; cuya relación y significado se muestran en el Cuadro 17.1. Es posible indicar varios filtros diferentes separados por coma. En este caso WCMATCH devolverá T (cierto) si la cadena cumple con uno cualquiera de los filtros. Los filtros se indican en una cadena de texto (entre comillas).

Cuadro 17.1

Carácter	Significado
# (Número)	Corresponde a un dígito numérico.
@ (Arroba)	Corresponde a un carácter alfabético.

. (Punto)	Corresponde a un caracter no alfanumérico.

* (Asterisco)	Corresponde a cualquier combinación de caracteres a partir de su posición.
? (Interrogacion)	Corresponde a un carácter cualquiera.
~ (Tilde)	Negación, indica la correspondencia contraria al filtro que va despues.
[.. ..]	Corresponde sólo a los caracteres indicados entre corchetes.
[~....]	Corresponde a los carateres no indicados entre corchetes
-(Guión)	Se utiliza para indicar un rango de caracteres entre corchetes
, (Coma)	Separa dos filtros diferentes.
` (Apóstrofo invertido)	Corresponde al literal del carácter indicado a continuación.

Por ejemplo, para detectar sí una cadena empieza con la letra "B" se haría;

(WCMATCH "Bloques" "B*") devuelve T

Para detectar si la cadena contiene cinco caracteres se haría:

(WCMATCH "Bloques"' "?????") devuelve nil

Para detectar si la cadena contiene una letra "q" se haria:

(WCHATCH "Bloques" "*q*") devuelve T

Para defecto si la cadena no contiene ninguna letra “q” se haría:

(WCMATCH "Bloques" "~*q*") devuelve nil

Para detectar si la cadena contiene una coma, hay que indicar el literal de la coma:

(WCMATCH "Bloques,armario" “*',*") devuelve T

Como ya se ha dicho, es posible indicar varios filtros alternativos. Por ejemplo, para detectar si la cadena empieza por "B" o "b" se haría:

(WCMATCH "Bloques" "B*,b*") devuelve T

Para detectar si la cadena empieza por un carácter en mayúscula se podría hacer:

(WCMATCH "Bloques" " [A-Z]*” devuelve T

17.3 Modificaciones en comandos de la versión 10.

Existen comandos que sufren alguna modificación en AutoLISP 11. En general, éstas consisten en añadir alguna posibilidad más a las ya exisntes en el comando.

17.3.1 (INITGET...)

La modificación afecta a los códigos de los modos permitidas en la Versión 1Ø (véase el Capitulo) 7, apartado7.l).

Se suprime el código 16 y se añade un nuevo código 64. En el Cuadro 17.2 se muestra la significación de los códigos en AutoLISP V. 11.

Cuadro 17.2

Bits	Modo
1	No admite valores nulos.
2	No admite valor cero.
4	No admite valores negativos.
8	No verifica límites, aunque estén activados.
16	... (no se utiliza) ……
32	Dibuja la línea o rectángulo elásticos, con línea de trazos en vez de continua.
64	Ignora la coordenada Z para la distancia entre dos puntos 3D.

El nuevo código 64 se utiliza para que GETDIST devuelva la distancia 2D entre puntos en 3D, y por tanto sólo tiene sentido ese comando.

De acuerdo con esto, los modos que tienen ahora sentido con los diferentes comandos del tipo GET se detallan en el Cuadro 17.3.

Cuadro 17.3

Comando	Modos que tienen sentido					
	1	2	4	8	32	64
GETINT	SI	SI	SI			
GETREAL	SI	SI	SI			
GETDIST	SI	SI	SI		SI	SI
GETANGLE	SI	SI			SI	
GETORIENT	SI	SI			SI	
GETPOINT	SI			SI	SI	
GETCORNER	SI			SI	SI	
GETKWORD	SI					
GETSTRING						
GETVAR						

17.3.2 (TRANS...)

A este comando se le añade un nuevo código para indicar el nuevo Sistema de Coordenadas del Espacio-Papel.

El Espacio-Papel es una nueva forma de trabajo en AutoCAD V. 11 que permite proyectar cualquier ventana o punto de vista del dibujo sobre una lámina o "papel" en pantalla. De esta forma es posible combinar en esa lámina varias vistas del dibujo, con o sin líneas ocultas, desactivando o inutilizando las capas deseadas en las ventanas que interesen. El trabajo en Espacio-Papel permite desplazar esas ventanas, superponerlas. escalar, copiar y borrar. Una vez dispuestas todas sobre el Espacio-Papel, es posible añadir textos y obtener un archivo de "foto" con la orden de AutoCAD “Sacafoto” o también sacar en trazador (“ploter") ese conjunto de ventanas.

El nuevo concepto de Espacio-Papel permite incluso incorporar modelizaciones de superficies ("shades") a las habituales representaciones en "armazón de alambre".

En contraposición, el "Espacio" de trabajo habitual de AutoCAD, donde sólo una de las ventanas se encuentra activa en cada momento, recibe la denominación de Espacio-Modelo. Es posible cambiar de uno a otro de una forma totalmente interactiva.

Para permitir conversiones de un punto o vector de desplazamiento al Espacio-

Papel, el comando TRANS admite un nuevo código (el 3) que indica el Sistema de Coordenadas del Espacio-Papel.

De esta forma, los códigos admitidos son:

Ø	Sistema de Coordenadas Universal (SCU).
1	Sistema de Coordenadas Personal (SCP) actual.
2	Sistema de Coordenadas Vista actual (SCV).
3	Sistema de Coordenadas (SCV) del Espacio-Papel (sólo es válido en combinación con código 2).

El Sistema de Coordenadas del Espacio-Papel es también un Sistema de Coordenadas de Visualización (SCV), lo mismo que el de la Vista actual. Es un sistema hacia el cual se convierten las imágenes antes de ser visualizadas en pantalla en el Espacio-Papel. Su origen es, por tanto, el centro de la pantalla y el eje Z la línea de visión (perpendicular hacia el centro de la pantalla).

Sólo puede utilizarse cuando se indica en combinación con el código 2. De este modo se pueden convertir puntos o desplazamientos (“proyectarlos") desde su visualización en la ventana activa en el Espacio-Modelo habitual de trabajo hasta el Espacio-Papel- El proceso contrario también es posible.

Por ejemplo:

(TRANS ' (1 2 3) 2 3) podría devolver (-Ø.262448 3.93684 Ø.Ø5571)

En este caso se proyecta el punto de coordenadas 1, 2, 3, desde la visualización actual en el Espacio-Modelo hasta el Espacio-Papel.

Para la explicación completa del comando TRANS, véase el Capítulo 10, apartado 10.6.

17.3.3 Códigos de error.

Aunque la función de error *ERROR* explicada en el Capítulo 15, apartado 15.11, no sufre modificación, existe una nueva Variable de Sistema de AutoCAD V. 11 llamada “ERRNO", que almacena un código de error cada vez que se produce alguno desde un programa en AutoLISP.

El código es un número de 1 a 54, Cada código tiene un significado, y examinando desde un programa en AutoLISP ese código con (GETVAR "errno"} es posible obtener información sobre las causas del error cuando se produce.

La variable "ERRNO" almacena siempre el código del último error producido.

17.3.4 (SSGET...)

En el comando para aceptar conjuntos designados SSGET, cuando se utiliza con el parámetro "X" y se indica la lista de filtros es posible incluir filtros para las cadenas de texto, tal como se ha explicado en el comando WCMATCH (apartado 17.2.3 de este mismo capitulo).

Por ejemplo, para designar un conjunto de entidades con todos los círculos que se encuentran en capas cuyo nombre empieza por "P" se podría hacer:

(SETQ conj (SSGET "X" (LIST (CONS Ø "circle") (CONS 8 "P*"))))

Indicando el filtro de esa manera, SSGET explora la Base de Datos y busca todas las entidades.que sean círculos y que tengan asociado al código 8 un nombre de capa que empiece por la letra “p".

La explicación completa del comando SSGET se encuentra en el Capítulo 13, apartado 13.1.1.

17.4 Nuevos comandos relativos a aplicaciones ADS.

En este apartado se hace referencia a los comandos que permiten cargar y descargar las aplicaciones desarrolladas en Lenguaje C a través del "AutoCAD Development System" ADS. Cada una de estas aplicaciones contendrá una serie de "Subrutinas externas” o "Funciones ADS", como se ha explicado.

17.4.1 (XLOAD <aplic>) Cargar aplicación ADS.

Este comando se utiliza para cargar la aplicación ADS cuyo nombre se indica en <aplic>. Ese nombre es el de un archivo ejecutable que contiene la aplicación y debe ser indicado como una cadena de texto (entre comillas). También podría ser una variable con un valor textual. Si la aplicación se carga, el comando devuelve el nombre. En caso contrario, devuelve un mensaje de error.

Por ejemplo:

(XLOAD "\ \apli\ \ame") podría devolver "\ \apli\ \ame"

si la carga resulta correcta.

Al mismo tiempo que se carga, se verifica si la aplicación ADS es correcta y su compatibilidad con AutoLISP. En caso de detectarse incorrecciones, el proceso de carga quedaría abortado, produciéndose un error. Una vez, cargada una aplicación, no es posible volverla a cargar.

Los archivos de aplicaciones ADS suministrados con AutoCAD tienen la extensión ".EXP". Por ejemplo, "ame.exp" es el archivo ejecutable que contiene el nuevo modelizador de sólidos de AutoCAD V. 11. Una vez cargado ese

archivo, todas las funciones en Lenguaje C contenidas en él quedan incorporadas en la memoria. Esas "Subrutinas externas" o "Funciones ADS" se visualizan, por ejemplo, extrayendo el contenido de la variable ATOMLIST, que contiene todas las funciones cargadas en memoria.

Así, por ejemplo, al cargar la aplicación "ame" aparece una nueva función denominada "c:solarea". Esa función se incorpora como una nueva orden de AutoCAD, cuya utilidad es calcular el arca de un sólido. Para llamarla basta introducir SOLAREA como respuesta a la pregunta de AutoCAD "Orden;".

Habitualmente el programa fuente escrito en Lenguaje C es un archivo de texto con la extensión ".C", por ejemplo "gravity.c". El archivo ejecutable obtendo por compilación de ese programa fuente será en este caso "gravity.exp". Para examinar o modificar los programas habrá que acceder entonces a los archivos de texto con extensión ".C".

17.4.2 (XUNLOAD <aplic>] Descargar aplicación ADS.

Este comando descarga la aplicación ADS cuyo nombre se indica en <aplic>. Este nombre debe ser el mismo que el indicado en XLOAD para cargar esa aplicación. Si se ha indicado un camino ("path") de directorios en XLOAD, es posible omitirlo para descargar con XUNLOAD.

Si la aplicación se descarga, el comando devuelve su nombre. En caso contrario se produce un mensaje de error.

Por ejemplo;

(XUNLOAD "ame") podria devolver "ame"

17.4.3 (ADS) Lista de aplicaciones ADS cargadas.

Este comando devuelve una lista con los nombres de las aplicaciones ADS actualmente cargadas. Cada nombre es una cadena de texto, con indicación del camino ("path") de directorios si es necesario.

Por ejemplo:

(ADS) podría devolver ("ads\ \gravity" "c:\ \acadll\ \ame")

En este caso hay cargadas dos aplicaciones ADS denominadas "gravity" en el directorio "ads" y "ame" en el directorio actual de AutoCAD que se llama "acadll".

17.5 Nuevos comandos de acceso a Base de Datos.

Se estudian en este apartado los nuevos comandos de AutoLISP V. 11 que permiten el trabajo con entidades de dibujo y el acceso a la Base de Datos. Con complementarios de los comandos vistos en el Capítulo 13.

Las nuevas posibilidades se centran en el acceso directo a las entidades incluídas en otras complejas como pololíneas y bloques. También es posible la creación de entidades completamente nuevas mediante el comando ENTMAKE

En la Base de Datos, por otro lado, existen dos nuevas Tablas de Símbolos: DIMSTYLE y APPID.

DIMSTYLE

Es una tabla que contiene una combinación de valores de Variables de Acotación. En AutoCAD V. 11, el usuario puede almacenar varias combinaciones de estos valores de variables con su nombre y después utilizarlos sin nececidad de tener que ir cambiando todos los valores cada vez que quiere cambiar de forma de acotación. Su contenido es simplemente una lista de códigos que representan Variables de Acotación, acompañados del correspondiente valor para cada variable.

APPID

Es un tipo de tabla con un nombre de aplicación ADS. Estas aplicaciones. cuando se utilizan en AutoCAD V. 11, añaden a la Base de Datos una serie de nuevas propiedades especificas de cada aplicación. Estas propiedades constituyen lo que se denomina Extensión de Datos de Entidades ("Extended Entity Data"). Son listas de asociaciones que se añaden alas ya conocidas para cada entidad en Ía Base de Datos. Para diferenciarlas del resto, sus códigos van desde el 1ØØØ hasta el 1Ø71 actualmente.

Así, por ejemplo, al cargar desde AutoCAD o desde un programa en AutoLlSP (con XLOAD) la aplicación "AMElite" (que es la versión simplificada del AME y que se suministra con AutoCAD), se dispone de una aplicación ADS que permite generar sólidos sencillos (lo que se conoce por "primitivas"), como cilindros, esferas, conos, etc. Estos sólidos son tratados como "Referencias de Bloque" en AutoCAD (por ejemplo, al procesarlos con la orden "List"). Pero esas entidades contienen una Extensión de Datos, de forma que es posible, por ejemplo, obtener su area, material de que están hechos, densidad. volumen, etc.

El acceso a esta Extensión de Datos en un programa en AutoLISP puede hacerse llamando a las funciones ADS cargadas con XLOAD. Por ejemplo, para la aplicación "AMElite" podrían ser "c:solarea", "c:sollist", "c:solmat", "c:solvolume", etc. También es posible hacerlo con ENTGET, indicando el nuevo parámetro <listap>, como se explica mas adelante en este mismo capítulo.

Existen dos funciones ADS que permiten especificar un conjunto designado desde un programa AutoLISP. Estas son c:solarea y c:solmassp.

(c:solarea <conj>)

Una vez disponible esta función ADS (cargando AMElite o AME) es posible llamarla con un conjunto designado <conj>. La función examina todas las entidades del conjunto, desechando las que no sean solidos, y devuelve la superficie de todos los sólidos.

```
( XLOAD  "amelite" )
( C:SOLAREA ( SSGET ) ) <se designa un solido>  podría devolver
15Ø.796
```

(c:solmassp <conj>)

Esta función ADS sólo se encuentra disponible en el AME y no en su versión simplificada AMElite. Si se especcifíca la llamada a esa función con un conjunto designado de entidades, examina los sólidos y devuelve una lista con las propiedades más importantes de cada uno.

```
( XLOAD  "ame" )
( C:SOLMASSP ( SSGET ) ) <se designa un solido>
```

La lista devuelta, con el significado de cada uno de los eleementos, podría ser ta siguiente:

```
(       846.416                          Masa
        1Ø7.687                          Volumen
        1.13Ø1                           Error de volumen
        ( 8.Ø 1.Ø Ø.Ø )                  Contorno mín. X,Y,Z
        ( 13.2674 6.3191  5.Ø )          Controno máx. X,Y,Z
        ( 1Ø.6427 3.6762 2.4748 )        Centroide X,Y,Z
        ( Ø.3586 Ø.1157 Ø.1982 )         Error de centroide
        ( 9763.56 35675.23 42ØØ3.43)     Momento de inercia X,Y,Z
        ( 235.67 345.76 321.99 )         Error momento de inercia
        ( 9876.Ø1 7655.51 2167.Ø )       Productos inercia X,Y,Z
        ( 98.76 123.76 1Ø5.Ø6 )          Error productos de inercia
        ( 5.7659 7.8715 6.8821 )         Radios de giro X,Y,Z
        ( 1259.76 2Ø54.78 1549.72 )      Momentos principales I,J,K
        ( Ø.99879 Ø.Ø34567 Ø.ØØ7654 )    Momento I en directorio X,Y,Z
        ( Ø.ØØ7659 Ø.Ø9367 -Ø.99671 )    Momento J en directorio X,Y,Z
        ( -Ø.Ø15674 Ø.99972 Ø.Ø37642 )   Momento K en directorio X,Y,Z
)
```

17.5.1 (NENTSEL [<mens>]) Seleccionar entidad componente de una entidad compuesta, con punto de designación.

Este comando permite acceder a la Base de Datos de una entidad que se encuentre formando parte de una entidad compuesta (polilínea o bloque). La cadena de texto optativa <mens> es el mensaje para la Solicitud de designación de entidad.

El comando funciona de forma análoga a ENTSEL, explicado en el Capítulo 13.

apartado 13.2.3. Cuando la entidad que se designa no forma parte de otra compuesta, NENTSEL devuelve la misma inforrmación que ENTESL: una lista cuyo primer elemento es el nombre de la entidad y su segundo elemento el punto de designación.

Cuando con NENTSEL se designa un componente de una polilíneea, devuelve una lista cuyo primer elemento es el nombre de la subentidad, es decir, el vértice (tipo de entidad "VERTEX") inicial del segmento de polilínea designado. El segundo elemento de la lista sigue siendo el punto de designación.

Por ejemplo:

```
Command: ( NENTSEL "Designar segmento de polilíeca:  " )
Designar segmento de polilínea:  (se designa)
( <Entity name: 6ØØØØØ4e> ( 5.65  6.32 Ø.Ø ) )
```

Cuando con NENTSEL se designa un componente de un bloque, devuelve una lista con cuatro elementos:

El primer elemento es el nombre de la entidad componente del bloque, extraida de la Tabla de Símbolos con la definición de ese bloque.

El segundo elemento es una lista con las coordenadas del punto de designación.

El tercer elemento es una lista que contiene a su vez cuatro listas: es la matriz de transformación del Sistema de Coordenadas Modelo (SCM) al Universal (SCU). El Sistema Modelo (SCM) es aquel al que están referidas todas las coordenadas en la definición del bloque. Su origen es el punto de inserción del bloque. La matriz de transformación permite trasladar las coordenadas de la definición del bloque al Sistema Universal (SCU), para a partir de ahi referirlas al Sistema de Coordenadas más conveniente para el usuario.

El cuarto elemento es una lista con el nombre de entidad que contiene a la designada. Si existen varios bloques anidados, la lista contiene todos los nombres desde el bloque mas interior hasta el más exterior de los que engloban a la entidad designada.

Por ejemplo, la designación de una entidad que forma parte de un bloque, que a su vez se encuentra incluido en otro bloque, podría hacer que NENTSEL devolviera:

```
( <Entity name: 4ØØØØØ9d>
     ( 6.65 5.67 Ø.Ø )
     (   ( 1.Ø  Ø.Ø Ø.Ø )
         ( Ø.Ø 1.Ø Ø.Ø )
         ( Ø.Ø Ø.Ø 1.Ø )
         ( 5.Ø21 4.Ø21 Ø.Ø)
)
(<Entity name: 4ØØØØ1Øe> <Entity name: 6ØØØØ1ba>)
```

)

La excepción a lo dicho son los atributos contenidos en un bloque. Si se designa un atributo, NENTSEL devuelve una lista con sólo dos elementos; el nombre de la entidad de atributo y el punto de designación.

17.5.2 (ENTMAKE <lista>) Construir una entidad en Base de Datos.

Este comando permite añadir una entidad nueva al dibujo, construyendo directamente su lista completa en la Base de Datos. Si la lista introducida es correcta, el comando devuelve esa lista. En caso contrario devuelve nil. Las listas se indican generalmente como literales, con QUOTE.

La lista debe contener todas las sublistas de asociaciones necesarias para definir completamente cada tipo de entidad. Si se omite alguna, se produce un error. Es posible omitir algún dato optativo, y entonces se asume la opción por defecto. Asi, por ejemplo, si no se indica la capa, la entidad construida asume la capa actual.

Una forma cómoda de añadir una nueva cantidad a la Basc de Datos es partir de una entidad ya existente, obtener su lista con ENTGET (explicado en el Capitulo 13, apartado 13.3.1), modificar y añadir lo que sea preciso y crear la nueva entidad con ENTMAKE. Esto evita tener que construir la lista completa desde el programa en AutoLISP.

El tipo de entidad debe ir siempre en inglés (por ejemplo. "circle", "line", "are", etc.) y debe ser el primero o segundo elemento de la lista. Lógicamente, todos los códigos de las sublistas de asociaciones deberán ser correctos.

Para construir una entidad compleja, como Definiciones de Bloques, Referencias de Bloque con Atributos, o Polilíneas, es preciso construir todas las listas necesarias empleando varias veces el comando ENTMAKE: la lista de cabecera o de la entidad principal, las listas con cada subentidad componente y la lista final del tipo "SEQEND" o "ENDBLK" para las definiciones de bloque.

Aunque para explorar todas las entidades contenidas en las definiciones de bloque con ENTNEXT no es necesario buscar un tipo de entidad final (como SEQEND para las polilíneas y atributos), pues al llegar a la última ENTNEXT devuelve nil, a la hora de construir las listas completas de la definición de un bloque es preciso añadir como última lista un tipo de entidad llamado "ENDBLK".

Por ejemplo, para construir un cuadrado como polilínea con cuatro vértices, en la capa actual y con color rojo (número 1) se podría hacer:

```
( ENTMAKE ' (       ( Ø  . "POLYLINE" )
                    ( 62 . 1 )
                    ( 66 . 1 )( 7Ø . 1 )
              )
```

```
)

( ENTMAKE ' ( ( Ø  . "VERTEX" )
        ( 1Ø Ø.Ø  Ø.Ø Ø.Ø ) )
        )
)

( ENTMAKE ' ( ( Ø  . "VERTEX" )
        ( 1Ø Ø.Ø  1Ø.Ø Ø.Ø )
        )
)

( ENTMAKE ' ( ( Ø  . "VERTEX" )
        ( 1Ø 1Ø.Ø 1Ø.Ø Ø.Ø )
        )
)

( ENTMAKE ' ( ( Ø  . "VERTEX" )
        ( 1Ø 1Ø.Ø Ø.Ø Ø.Ø )
        )
)

( ENTMAKE ' ( ( Ø  . "SEQEND" )
        )
)
```

En la cabecera de la polilínea, el código 66 debe ir seguido obligatoriamente del valor 1, que indica que siguen vértices. Para que la polilínea sea cerrada, hay que incluir una lista con código 70 y valor 1.

17.6 Nuevos comandos para extensión de Datos de entidades.

En AutoCAD V. 11, al emplear aplicaciones ADS. como por ejemplo las incluidas en el AME (Avanced Modelling Extensión) que incorpora las posibilidades de edición de sólidos de AutoSOLID dentro de AutoCAD V. 11, en la Base de Datos del dibujo se añade un nuevo tipo de Tabla de Símbolos llamado "APPID". Cada tabla de este tipo contiene el nombre de una aplicación ADS utilizada para intensión (o Ampliación) de Datos en el dibujo actual- De esta forma se diferencian los Datos Ampliados de las entidades, dependiendo de que aplicaciones ADS provienen.

17.6.1 (REGAPP <nom-apl>) Registrar nombre de aplicación.

Este comando registra un nombre de aplicación en el actual dibujo de AutoCAD. Registrando las aplicaciones ADS con un nombre es la mejor manera de agrupar, almacenar o modificar los Datos Ampliados de las entidades. Una aplicación puede registrarse con varios nombres, o con

ninguno. De lo que se trata es de organizar de la manera más conveniente los Datos Ampliados.

Si el registro de la aplicación es correcto. REGAPP devuelve el nombre registrado. En caso contrario (por ejemplo, al especificar un nombre: de aplicación que ya existe) devuelve nil.

Una vez, registrado, el nombre de la aplicación se añade en la Base de Datos como una Tabla de Símbolos del tipo "APPID". El nombre puede contener hasta 31 caracteres, incluyendo los especiales "$" (dólar), "-" (guión) y "_" (barra baja).

El comando ENTGET devuelve la lista con todos los datos almacenados en la Base de Datos para la entidad cuyo nombre se indica. Su funcionamiento se ha explicado ya en el apartado 12.3.1. En AutoLISP 11 se añade la posibilidad de especificar una lista de nombres de aplicaciones registrados con REGAPP en <listap> de acuerdo con el formato:

(ENTGET <nombre-ent> [<listap>])

En este caso la lista devuelta por ENTGET incluye también el código -3, que es el denominado "centinela" o indicador de que la entidad contiene Extensión de Datos para la aplicación o aplicaciones cuyo nombre se ha indicado. Estos Datos Ampliados contienen códigos del 1ØØØ al 1Ø71.

La lista de aplicaciones se indicara generalmente como un literali. Por ejemplo:

```
( REGAPP "brazo" )
( ENTGET ( ENTLAST ) ' ( "brazo" ) )
```

En este caso se registra una aplicación con el nombre "brazo". Se forma una lista con ese nombre y se indica el literal de esa lista en el comando ENTGET.

17.6.2 (XDROOM <nom-ent>) Espacio disponible para Extensión de Datos para una entidad.

Este comando devuelve el espacio de memoria aún disponible para los Datos Ampliados (o Extensión de Datos) de la entidad cuyo nombre se indica en <nom-<ent>. El espacio máximo disponible para cada entidad es de 16,383 octetos. E1 valor devuelto por XDROOM es entonces la diferencia entre este máximo y lo que ocupan los Datos Ampliados ya existentes para la entidad. Por ejemplo:

```
Command: ( SETQ  nom ( entlast ) )
( XDROOM nom ) podría devolver 16264
```

17.6.3 (XDSIZE <lista>) Longitud de lista de Extensión de Datos.

Este comando devuelve la longitud en octetos ("bytes") que la lista indicada ocupa cuando se añade como una Extensión de Datos a una entidad. Es complementario del anterior, XDROOM, y se utiliza para controlar lo que van ocupando en la memoria los Datos Ampliados de una entidad (por ejemplo, un sólido generado con la aplicación AME).

La lista de Extensión de Datos debe contener un nombre de aplicación previamente registrado con REGAPP. Si existen varios nombres de aplicaciones, se forma una lista que englobe a las demás (por ejemplo, con LIST).

17.7 Ejemplo 1: Trazado de linea de tuberías.

Se desarrolla en este programa una aplicación para el trazado en tres dimensiones de una linea de tuberías. El objetivo es presentar un ejemplo de utilización del Módulo de Modelización de Sólidos de AutoCAD V. 11. Sin embargo, y con el fin de que los usuarios que no disponen de este módulo puedan también sacar provecho del programa, se explica en este primer ejemplo una versión para AutoCAD V. 1Ø.

17.7.1 Listado del programa.

```
(DEFUN inic (/ n nent lent)
  (PROMPT
    "Designar lineas de conducción una a una y en orden: \n"
  )
  (SETQ lin (SSGET))
  (SETQ n Ø)
  (SETQ num (SSLENGTH lin))
  (REPEAT num
    (SETQ nent (SSNAME lin n))
    (SETQ lent (ENTGET nent))
    (IF  (/= (CDR (ASSOC Ø lent)) "LIME")
      (PROGN (SSDEL nent lin) (SETQ n (- n 1)))
    )
    (SETQ n (+ n 1))
  )
  (SETQ num (SSLENGTH lin))
  (IF (= num Ø)
    (PROMPT "Ninguna de las entidades son lineas\n")
  )
  (IF (= num 1)
    (PROMPT "Solo una entidad de linea designada\n")
  )
)
(DEFUN dat (/ mens capac)
  (INITGET 7)
  (SETQ rtub (/ (GETREAL "Diámetro exterior de tubería: ") 2))
  (TERPRI)
  (SETQ rcod rtub)
```

```
(SETQ       mens (STRCAT
              "Diámetro exterior del codo<"
              (RTOS (* rtub 2) 2 2)
              ">: "
            )
)
(INITGET 6)
(IF (SETQ rcod (GETREAL mens))
  (SETQ rcod (/ rcod 2))
  (SETQ rcod rtub)
)
(WHILE (< rcod rtub)
  (PROMPT
    "Diámetro de codo debe ser mayor o igual que tuberia\n"
  )
  (INITGET 6)
  (IF  (SETQ rcod (GETREAL mens))
    (SETQ rcod (/ rcod 2))
    (SETQ rcod rtub)
  )
)
(INITGET 7)
(SETQ rpeq (GETREAL "Radio pequeño de curvatura del codo: "))
(TERPRI)
(INITGET 6)
(IF (SETQ tab (GETINT "Precisión de mallas <16>:  "))
  ()
  (SETQ fcab 16)
)
(TERPRI)
(SETVAR "surftabl" tab)
(SETVAR "surftab2" tab)
(SETQ capac (GETVAR "clayer"))
(SETQ capl (STRCAT "c-" capac))
(PROMPT (STRCAT
          "Las curvas de apoyo se situaran en capa "
          capl
        )
)
(TERPRI)
(COMMAND "capa" "cr" capl "des" capl "")
(SETQ ptel (CDR (ASSOC 1Ø (ENTGET (SSMAME lin Ø)))))
(SETQ nptel (CDR (ASSOC -1 (ENTGET (SSNAME lin Ø)))))
(SETQ ptl (TRANS ptel nptel 1))
(SETQ pte2 (CDR (ASSOC 1Ø (ENTGET (SSNAME lin 1)))))
(SETQ npte2 (CDR (ASSOC -1 (ENTGET (SSNAME lin 1)))))
(SETQ pt2 (TRANS pte2, npte2 1))
(SETQ pte3 (CDR (ASSOC 11 (ENTGET (SSNAME lin 1)))))
(SETQ npte3 (CDR (ASSOC -1 (ENTGET (SSNAME lin 1)))))
(SETQ pt3 (TRANS pte3 npte3 1))
```

```
  (SETQ n 2)
  (SETQ prim T)
)
(DEFUN tramo (/     ang alfa tabcod       pri prf cir1 cir2 cir3 cir4 cen          pfeje
              eje)
  (COMMAND "scp" "3" pt1 pt2 pt3)
  (SETQ pt2 (TRANS pte2 npte2 1))
  (SETQ pt3 (TRANS pte3 npte3 1))
  (SETQ ang (ANGLE pt2 pt3))
  (SETQ alfa (-PI ang))
  (SETQ        dis
         (/ (+ rcod rpeq) (/ (sin (/ alfa 2)) (cos (/ alfa 2))) ¿)
        (SETQ ang (/ (* 180 ang) PI))
         (SETQ tabcod (FIX (* tab (/ ang 9Ø))))
        (IF (< tabcod 6)
          (SETQ tabcod 6)
        )
         (COMMAND "scp" "y" "9Ø")
        (SETQ ptl (TRANS pte1 npte1 1))
         (SETQ pt2 (TRANS pte 2 npte2 1))
        (IF prim
          (SETQ pri pt1)
          (SETQ pri (MAPCAR '+ pt1 (LIST Ø Ø disant)))
        )
         (SETQ
           prf (MAPCAR
                 '-pt2
                 (LIST Ø
                       Ø
                       dis
                 )
               )
         )
        (COMMAND "circulo" pri rtub)
         (SETQ cirl (LIST (ENTLAST)
                          (POLAR pri Ø rtub)
                    )
         )
        (IF (OR prim (= rcod rtub))
          ()
          (PROGN (COMMAND "circulo" pri rcod)
                 (SETQ cir2 (LIST (ENTLAST) (POLAR pri Ø rcod)))
          )
        )
         (COMMAND
           "circulo"
           prf
           rtub
         )
        (SETQ cir3 (LIST (ENTLAST) (POLAR prf Ø rtub)))
```

```
 (IF (= rcod rtub)
  ()
  (PROGN (COMMAND
           "circulo"
           prf
           rcod
         )
         (SETQ       cir4
               (LIST
                 (ENTLAST
                 )
                 (POLAR
                   prf
                   Ø
                   rcod
                 )
               )
         )
  )
 )
(SETQcen (POLAR prf (/ PI 2) (+ rcod rpeq)))
 (SETQ
  pfeje (MAPCAR
          '-cen
          (LIST rtub
                Ø
                Ø
          )
        )
 )
(COMMAND "linea" cen pfeje " ")
 (SETQ eje (LIST (ENTLAST) cen))
(IF (OR prim (= rcod rtub))
  ()
  (COMMAND "supregla" cir1 cir2)
)
 (COMMAND "supregla" cir1 cir3)
(IF (= rcod rtub)
  (PROGN (SETVAR "surftab1" tabcod)
         (COMMAND "suprev" cir3 eje " " ang)
  )
  (PROGN (COMMAND "supregla" cir3 cir4)
         (SETVAR "surftab1" tabcod)
         (COMMAND "suprev" cir4 eje " " ang)
  )
)
 (SETVAR "surftab1" tab)
(IF (= rcod rtub)
  (COMMAND "cambprop" cir1 cir3 eje " " "p" "ca" cap1 " ")
  (IF prim
```

```
            (COMMAND "cambprop"        cir1 cir3 cir4 eje " " "p" "ca"          cap1  "
")
            (COMMAND "cambprop"        cir1 cir2 cir3 cir4 eje          " " "p"  "ca"
                     cap1 " ")
          )
        )
         (SETQ prim nil)

 )
 (DEFUN act ()
  (SETQ pte1 pte2
        npte1        npte2
        disant dis
  )
  (SETQ pt1 (TRANS pte1 nptel 1))
  (SETQ pte2        pte3
        npte2        npte3
  )
  (SETQ pt2 (TRANS pte2 npte2 1))
  (SETQ pte3 (CDR (ASSOC 11 (ENTGET (SSNAME lin n)))))
  (SETQ npt3 (CDR (ASSOC -1 (ENTGET (SSNAME lin n)))))
  (SETQ pt3 (TRANS pte3 npte3 1))
  (SETQ n (+ n 1))
 )
 (DEFUN tramof     (/ pri prf cir1 cir3 cir4)
  (SETQ pte1 (TRANS pte1 nptel 1))
  (SETQ pt2 (TRANS pte2 npte2 1))
  (SETQ pt3 (TRANS pte3 npte3 1))
  (COMMAND "scp" "3" pt3 pt2 pt1)
  (COMMAND "scp" "y" "9Ø")
  (SETQ pt3 (TRANS pte3 npte3 1))
  (SETQ pt2 (TRANS pte2 npte2 1))
  (SETQ pri pt3)
  (SETQ prf (MAPCAR '- pt2 (LIST Ø Ø dis)))
  (COMMAND "circulo" pri rtub)
  (SETQ cir1 (LIST (ENTLAST) (POLAR pri Ø rtub)))
  (COMMAND "circulo" prf rtub)
  (SETQ cir3 (LIST (ENTLAST) (POLAR prf Ø rtub)))
  (IF  (= rcod rtub)
    ()
    (PROGN (COMMAND "circulo" prf rcod)
           (SETQ cir4 (LIST (ENTLAST) (POLAR prf Ø rcod)))
    )
  )
  (COMMAND "supregla" cir1 cir3)
  (IF  (= rcod rtub)
    (COMHAND "cambprop" cir1 cir3 " " "p" "ca" cap1 " ")
    (PROGM (COMMAND "supregla" cir3 cir4)
           (COMMAND "cambprop" cir1 cir3 cir4 " " "p" "ca" cap1 " ")
    )
```

```
   )
  )
  (DEFUN c:tubos (/ lin     num    rtub   rcod   rpeq   tab
                    cap1  pte1   pte2   pte3   npte1  npte2  npte3
                    pt1    pt2    pt3    n      prim   dis    disant
                   )
    (inic)
    (IF  (> num 1)

      PROGN
      (SETVAR "gridmode" Ø)
      (SETVAR "biipmode" Ø)
      (SETVAR "cmdecho" Ø)
      (dat)
      (REPEAT (-num 1)
          (tramo)
          (IF (< n num)
            (act)
            (tramof)
          )
      )
    )
    (PROMPT "*No vale*")
  )
  (SETVAR "blipmode" 1)
  (SETVAR "cmdecho" 1)
  (PRIN1)
)
```

17.7.2 Observaciones.

El eje de la 1ínea de tuberías debe estar formado por entidades de línea; dibujadas en el mismo sentido de la conducción. Al utilizar el programa, el usuario debe designar las líneas una a una y en el orden correcto. Esto supone una importante limitación, pero se ha preferido no complicar más el programa (que ya presenta una cierta complejidad) por consideraciones didácticas.

Por otra parte, las explicaciones son mas escuetas, pues se entiende que el programador ha obtenido ya suficiente soltura para la comprensión del mismo, comando además con el apoyo de los gráricos.

17.7.3 Explicación del programa.

Define una nueva orden y cinco funciones de usuario

- **Función c:tubos**

Llama a la función "inic" y establece una condición. Si el usuario ha designado más de una entidad de linea, pueden trazarse los tubos. En caso contrario se visualiza un mensaje: *No vale*.

El trazado de los tubos incluye la llamada a la función "dat" para solicitar todos los datos, y una repetitiva para el trazado de cada tramo de tubo y su correspondiente codo. La función "act" actualiza las variables para dibujar el siguiente tramo, pero si se trata del último, existe una alternativa para llamar directamente a la función "tramof' que lo dibuja.

• Función inic

Solicita la designación de lincas, que son los ejes del trazado de lubos, una a una y en orden. Chequea el conjunto designado para rechazar las entidades que no sean lineas. Si no existen líneas o sólo hay una visualiza mensajes. Al final, el conjunto de "lin" contiene únicamente las entidades de línea.

Variables de entrada: ninguna.

Variables locales:

n Indice para explorar el conjunto designado.
nent Nombre de entidad del conjunto designado.
lent Lista de entidad del conjunto designado.

Variables de salida:

lin Conjunto de entidades con sólo lineas.
num Número de entidades de línea del conjunto.

• Función dat

Va solicitando los datos necesarios. En primer lugar, el diámetro exterior de los tubos; después, el de los codos (ofreciendo como opción por defecto el mismo que para los tubos, y evitando que sea menor que el de los tubos). Las variables almacenan en cada caso el valor de los radios. A continuación se pide el radio pequeño de curvatura del codo. Por último, la precisión del mallado para las superficies (Figura 17.1).

Las curvas de apoyo para el trazado se situarán en una capa cuyo nombre se forma con el de la capa actual y un prefijo "c-".

Se toman los tres primeros puntos de la conducción y se almacenan en el Sistema de Coordenadas de Entidad, que es como se encuentran en la Base de Datos. Se trasladan esos puntos al SCP actual. Se almacenan también los nombres de las entidades a que pertenecen. Por ultimo se inicializa el contador de tramos "n" y se pone a cierto “T” la variable "prim" para controlar que se trata del primer tramo a dibujar.

Variables de entrada:

lin Conjunto de entidades con sólo lineas.

Variables locales:

mens Mensaje de solicitud de de diámetro de codo.
capac Nombre de capa actual.

Variables de salida:

rtub Radio exterior de tubería.
rcod Radio exterior de los codos.
Rpeq Radio pequeño curvatura de codos.
tab Precisión de mallas.
cap1 Nombre de capa para situar curvas apoyo
pte1, pte2, pt3 Puntos en el Sistema de Coordenadas de Entidad (SCE)
npte1, npte2, npte3 Nombres de entidades de línea con puntos
pt1, pt2, pt3 Punto en SCP actual.
n Contador de tramos a dibujar.
prim Variable de control del primer tramo.

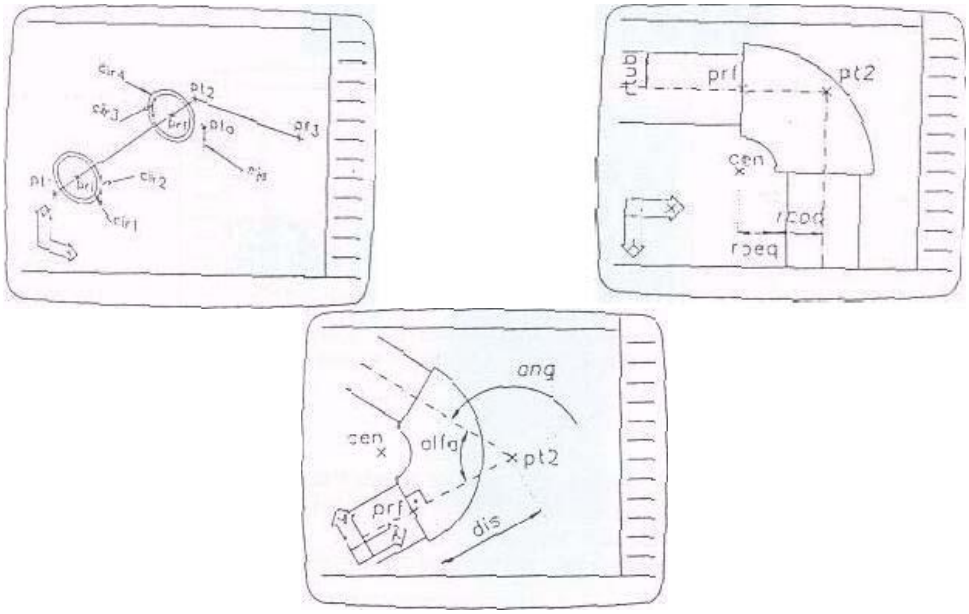


Figura 17.1. Datos necesarios para el trazado de tuberías.

• Función tramo

Se encarga de dibujar cada tramo con su tubo y su codo. Establece el SCP adecuado de acuerdo con el plano formado por las dos líneas que define el tramo. Calcula el ángulo formado por esas líneas en el nuevo SCP y la distancia "dis" del vértice a la que termina el tubo y empieza el codo. Calcula la precisión de mallado "tabcod" del codo, de acuerdo con el ángulo abarcado por el mismo.

Cambia a un nuevo SCP perpendicular al anterior, para trazar los círculos que van a definir las mallas. Los centros de estos circulos serán los puntos en que empieza y termina el tubo "pri" y "prf'. Se trazan los círculos y se forman listas que contengan el nombre de cada uno con un punto de designación (del tipo de las devueltas por ENTSEL). Hay que tener la precaución de trasladar los puntos al SCP actual.

El codo se va a trazar como una superficie de revolución, y esto obliga a

dibujar el eje y almacenarlo también como una lista con nombre y punto de designación.

Una vez trazadas todas las curvas se generan las mallas con "supregla" y "saprev". Se cambian las curvas de apoyo a la capa "cap1" (Figura 17.2).

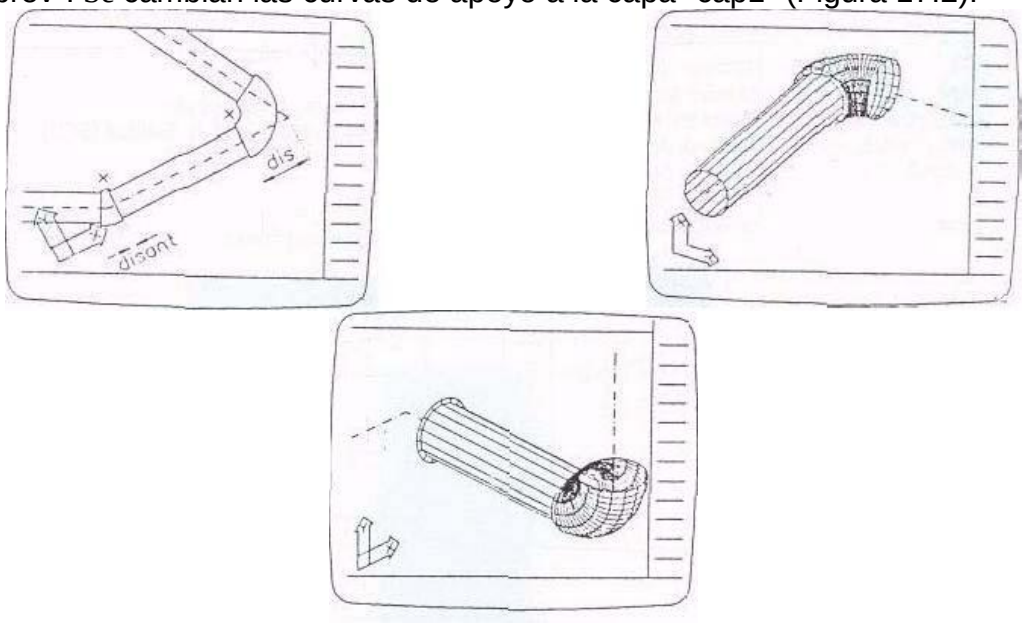


Figura 17.2. Caso en que los tramos no forman ángulo recto, y trazado de las mallas para los dos primeros tramos de una tubería con ángulos rectos.

Las alternativas son para detectar si se trata del primer tramo, en cuyo caso no se dibuja más que un circulo en el punto inicial, o si los radios de tubos y codos son iguales, en cuyo caso sólo se dibuja un círculo en cada lado, evitándose una "supregla".
Una vez recorrido el primer tramo se cambia la variable de control "prim" a nil para el resto de tramos.

Variables de entrada:

Todas las de salida de la función “dat”
disant Distancia inicio de tubo desde vértice.

Variables locales:

ang Ángulo de las dos líneas del tramo.
alfa Ángulo de las dos líneas entre si.
tabcod Precisión de mallas para el codo.
pri Punto inicial del tubo.
prf Punto final del tubo e inicial del codo.
cir1, cir2, cir3, cir4 Listas de designación por un punto de círculos.
cen Centro de curvatura del codo.
pfeje Punto final de trazado de eje.
eje Lista de designación por un punto de eje.

Variables de salida:

las mismas que para la función "dat", con nuevos valores de puntos.
dis Distancia de final de tubo a vértice.

• Función act

Se encarga de actualizar los valores de las variables para empezar el nuevo tramo en la repetitiva. Almacena la distancia "dis" de final del último tramo para utilizarla en "disant" como distancia de principio del nuevo tramo.

Calcula los puntos necesarios para obtener el SCP del plano formado por las dos líneas del nuevo tramo y suma uno al contador "n".

Variables de entrada:
las mismas que para "dat" y "tramo".

Variables locales:
ninguna.

Variables de salida:
las mismas de entrada, con sus valores actualizados.

• Función tramof

Dibuja el tramo final, alterando el orden de los puntos utilizados para formar el SCP adecuado. Dibuja los últimos circulos y obtiene las mallas correspondientes.

Para obtener los puntos necesarios, siempre se parte del SCE de la Base de Datos y de su traslado al SCP actual.

Variables de entrada:
las mismas que para las funciones anteriores.

Variables locales:
pri, prf Puntos inicial y final de último tubo.
cir1, cir3, cir4 Listas de designación por un punto de circulos.

Variables de salida:
ninguna.

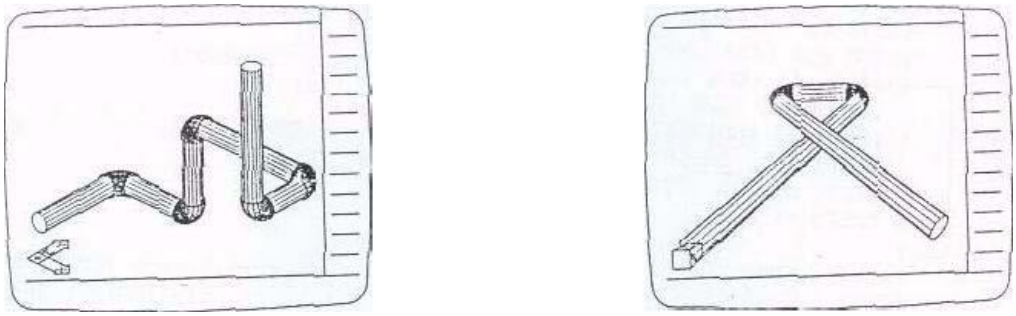


Figura 17.3. Resultado gráfico del trazado de tuberías con y sin ángulos rectos.

17.8 Ejemplo 2: Juntar dos polilíneas 3D en una sola.

Como aplicación de ENTMAKE, nuevo comando de la Versión 11, este programa junta dos polilíneas 3D. Las polilíneas se pueden editar en AutoCAD con la orden “editpol”, pero cuando son en 3D, no existe la opción de juntarlas. Con este programa se consigue eso.

17.8.1 Listado del programa.

```
(DEFUN inic ()
  (SETQ polb (ENTSEL "'Designar 3dpolilinea: "))
  (SETQ polb (CAR polb))
  (WIIILE (/ = (CDR (ASSOC Ø (ENTGET polb))) "POLYLINE")
          (PROMPT "Entidad designada no es polilinea")
          (TERPRI)
          (SETQ polb (ENTSEL "Designar 3dpolilínea: "))
          (SETQ polb (CAR polb))
  )
  (SETQ polj (ENTSEL "Designar 3dpol a juntar: "))
  (SETQ polj (CAR polj))
  (WHILE (/ = (CDR (ASSOC Ø (ENTGET polj))) "POLYLINE")
    (PROMPT "Entidad designada no es polilinea")
    (TERPRI)
    (SETQ polj (ENTSEL "Designar 3dpolilínea: "))
    (SETQ polj (CAR polj))
  )
)
(DEFUN juntar (/ cabec listob lis e1 e2   pb1 pb2        pj1 pj2endb liston)
  (SETQ cabec (ENTGET polb))
  (SETQ el (ENTNEXT polb))
  (SETQ lis (ENTGET el))
  (SETQ listob (LIST lis))
  (SETQ pbl (CDR (ASSOC 1Ø lis)))
  (WHILE (/ = (CDR (ASSOC Ø lis)) "SEQEND")
    (SETQ pb2 (CDR (ASSOC 1Ø lis)))
    (SETQ e2 (ENTNEXT el))
    (SETQ lis (ENTGET e2))
    (SETQ listob (COMS lis listob))
    (SETQ e1 e2)
  )
  (SETQ endb lis)
  (SETQ listob (CDR listob))
  (SETQ el (ENTNEXT polj))
  (SETQ lis (ENTGET el))
  (SETQ listón (LIST lis))
  (SETQ pj1 (CDR (ASSOC 1Ø lis)))
  (WHILE (/ = (COR (ASSOC Ø lis)) "SEQEND")
    (SETQ pj2 (CDR (ASSOC 1Ø lis)))
    (SETQ e2 (ENTNEXT el))
    (SETQ lis (ENTGET e2))
```

```
  (SETQ liston (CONS lis liston))
  (SETQ e1 e2)
 )
 (SETQ liston (CDR liston))
 (COND       ((EQUAL pb2 pj1)
         (PROGN      (SETQ listob (CDR listob))
                 (ENTDEL polj)
                 (EMTDEL polb)
                 (ENTMAKE cabec)
                 (makeb)
                 (makej)
                 (ENTMAKE ende)
         )
        )
        ((EQUAL pb2 pj2)
         (PROGN      (SETQ listob (CDR listob))
                 (SETQ liston (REVERSE liston))
                 (ENTDEL polb)
                 (ENTDEL polj)
                 (ENTMAKE cabec)
                 (makeb)
                 (makej)
                 (ENTMAKE ende)
         )
        )
        ((EQUAL pb1 pj2)
         (PROGN      (SETQ listOn (CDR liston))
                 (ENTDEL polb)
                 (ENTDEL polj)
                 (ENTMAKE cabec)
                 (makej }
                       (makeb)
                       (ENTMAKE ende)
                 )
         )
         ((EQUAL pb1 pj1)
           (PROGN (SETQ liston (CDR liston))
                   (SETQ liston (REVERSE liston))
                   (ENTDEL polb)
                   (ENTDEL polj)
                   (ENTMAKE cabec)
                   (makej)
                   (makeb)
                   (ENTMAKE ende)
           )
         )
         (T (PROMPT "Entidades no se tocan por un extremo"))
        )
 )
 (DEFUN makeb (/ n longl lis)
```

```
    (SETQ longl (LENGTH listob))
    (SETQ n 1)
    (REPEAT longl
      (SETQ lis (NTH (-longl n) listob))
      (ENTMAKE lis)
      (SETQ n (+ n 1))
    )
  )
  (DEFUN makej (/ n long1 lis)
    (SETQ long1 (LENGTH liston))
    (SETQ n 1)
    (REPEAT longl
      (SETQ lis (NTH (- long1 n) liston))
      (ENTMAKE lis)
      (SETQ n (+ n 1))
    )
  )
  (DEFUN c:unepol ()
    (SETVAR "blipmode" Ø*)
    (SETVAR "cmdecho" Ø)
    (inic)
    (juntar)
    (SETVAR "blipmode" 1 *)
    (SETVAR "cmdecho" 1)
    (PRINl)
)
```

17.8.2 Observaciones.

Para no complicar en exceso el progruma, sólo permite juntar dos polilíneas. Se podría generalizar sin demasiado esfuerzo para que el usuario pudiera seleccionar varias polilíneas con una ventana o captura, estableciendo un control para comprobar cuáles son contiguas y cuales no.

En este ejemplo, y como se trata de mostrar e! funcionamiento de ENTMAKE sin alargar en exceso el programa, se piden dos únicas entidades con ENTSEL, y se procede a juntarías si tienen algún punto en común en sus extremos.

17.8.3 Explicación del programa.

Define una nueva orden de AutoCAD y cuatro funciones du usuario. Una de ellas llamará a otras dos.

• Función c:unepol

Llama sucesivamente a las funciones "inic" y "juntr". Como siempre; desactiva marcas auxiliares, eco de órdenes y establece como variables locales todas las que quedan pendientes de las funciones intermedias.

• Función inic

Solicita, en primer lugar, designar una polilínea. Para asegurar que sólo se selecciona una, se emplea el comando ENTSEL. El programa funciona también para polilíneas en 2D. La variable "polb" almacena el nombre de la entidad de cabecera de la polilínea.

A continuación se solicita otra polilínea para juntar. Su nombre se almacena en “polj”. En ambos casos se establece un control, para obligar al usuario a que designe polilíneas y no cualquier otro tipo de entidad.

Variables de entrada:
ninguna.

Variables locales:
ninguna.

Variables de salida:
polb Nmbre de polilínca base.
plj Nombre de polilínea a juntar.

• Función juntar

Su misión es juntar ambas polilíneas, si es posible; es decir, tienen un extremo común. La polilínca resultante de la unión va a tener la cabecera y la lista de "seqend" de la polilínea base. Por eso se almacena en "cabec" la lista de cabecera.

Se extrae en "el" el nombre del primer vértice, y en "lis" la lista corrcspondiente, Se crea una lista total "listob" que va a contener todas las listas necesarias para definir la polilínea. De momento se almacena el primer vértice. Se almacena en "pb1" el punto de ese vértice, que es el inicial de la polilínca base.

Se establece una repetitiva hasta detectar el final de esa polilínea (tipo de entidad "SQEND"). Se van añadiendo las listas de cada vértice (tipo de entidad "VERTEX") en la lista total "listob". La variable "pb2" va almacenando las coordenadas de punto de cada vértice. En el momento en que se detecta "SEQEND", termina la repetitiva y "pb2" sale con el valor del último punto de la polilínea base.

La variable "lis" contiene, al salir del ciclo, la lista de "SEQEND". por lo que se almacena ese valor en "endb". Se elimina esa última lista de "listob" con CDR.

Se realiza la misma operación con la polilínea a juntar. En este caso la lista total con todas las listas de vérlices se llama "liston". El punto inicial de esta polilínea es "pjl" y el final "pj2". No se almacenan ni la cabecera ni la lista de "seqend" de esta polilínea, pues se van a utilizar las de la anterior.

Una vez construidas las listas necesarias, se establece una condicional para ver cuales son los extremos por los que se tocan ambas polilíneas. Dependiendo de ello, se elimina en una de las listas el vértice común para que no se duplique. En caso necesario, se invierten los elementos de una lista para

que al construir la entidad nueva total se encuentren todos los vértices en el orden correcto.

En cada caso se añade a la Base de Datos con ENTMAKE la lista de cabecera "cabec" y se llama a las funciones "makeb" y "makej" en e! orden correcto para añadir sucesivamente todos los vértices. A! final se añade la lista de "SEQEND" almacenada en la variable "endb".

Variables de entrada:

polb Nombre de polilínea base.
polj Nombre de polilínea a juntar.

Variables locales:

cabec Lista con cabecera polilínea.
endb Lista con "seqend" polilínea.
lis Lista con entidad de vértice.
listob Lista total con vértices de polilínea base.
liston Lista total con vértices de polilínea a juntar.
c1 Nombre entidad de vértice.
c2 Nombre entidad de vértice.
pb1 Primer punto polilínea de base.

18 Apéndice A: Lista de comandos de AutoLISP y órdenes AutoCAD.

Cuadro A.1. Comandos de AutoLISP en orden alfabético.

~	DEFUN	ITOA	RTOS
' (QUOTE)	DISTANCE	LAMBDA	s::startup (función)
*	ENTDEL	LAST	SET
error (función)	ENTGET	LENGTH	SETQ
+	ENTLAST	LIST	SETVAR
-	ENTMAKE (V. 11)	LISTP	SIN
/	ENTMOD	LOAD	SQRT
/ =	ENTNEXT	LOG	SSADD
<	ENTSEL	LOGAND	SSDEL
< =	ENTUPD	LOGIOR	SSGET
=	EQ	LSH	SSLENGTH
>	EQUAL	MAPCAR	SSMEMB
> =	EVAL	MAX	SSNAME
1	EXP	MEM	STRCASE
1 +	EXPAND	MEMBER	STRCAT
ABS	EXPT	MENUCMD	STRLEN
ADS (V. H)	FINDDILE	MIN	SUBST
ALLOC	FIX	MINUSP	SUBSTR
AND	FLOAT	NENTSEL (V. 11)	TBLNEXT
ANGLE	FOREACH	NOT	TBLSEARCH
ANGTOS	GC	NTH	TERPRI
APPEND	GCD	NULL	TEXTPAGE (V. 11)
APPLY	GETANGLE	NUMBERP	TEXTSCR
ASCH	GETDIST	OPEN	TRACE
ASSOC	GETENV	OR.	TRANS
ATAN	GETINT	OSNAP	TYPE
ATOF	GETKWORD	POLAR	UNTRACE
ATOI	GETORIENT	PRINI	VER
ATOM	GETPOINT	PRINC	VMON
BOOLE	GETREAL	PRINT	VPORTS
BOUNDP	GETSTRING	PROGN	WCMATCH (V. 11)
CADR	GETVAR	PROMPT	WHILE
CAR	GRAPHSCR	QUOTE	WRITE-CHAR
CDR	GRCLEAR	READ	WRITE-LINE
CHR	GRDRAW	READ-CHAR	XDROOM (V.11)
CLOSE	GRREAD	READ-LINE	XDSIZE (V.11)
COMMAND	GRTEXT	REDRAW	XLOAD (V.11)
COND	HANDENT	REGAPP (V.11)	XUNLOAD (V. 11)
CONS	IF	REM	ZEROP
COS	INIGET	REPEAT	
CVUNIT (V. 11)	INTERS		

Cuadro A.2. Comandos de AutoLISP agrupados según su naturaleza

Definición y carga de funciones de usuario y variables:

DEFUN	Definir función
LAMBDA	Definir función temporal
LOAD	Cargar programas con funciones
SETQ	Crear variables y atribuirles valores
XLOAD (V. 11)	Cargar aplicación ADS
XUNLOAD (V. 11)	Descargar aplicación ADS
ADS (V. 11)	Lista de aplicaciones ADS cargadas

Operaciones de AutoCAD

COMMAND	Llamar a órdenes de AutoCAD
GRAPHSCR	Cambiar a pantalla gráfica
TEXTSCR	Cambiar a pantalla de texto
TEXTPAGE (V. 11)	Cambiar a pantalla de texto borrado
REDRAW	Redibujar
MENUCMD	Llamada a menús de AutoCAD
POLAR	Punto en coordenadas polares
DISTANCE	Distancia entre dos puntos
ANGLE	Angulo definido por dos puntos
INTERS	Intersección de lineas
GETVAR	Extraer valor de variable de AutoCAD
SETVAR	Introducir valor en variable de AutoCAD
OSNAP	Aplicar modo de referencia a entidades
VPORTS	Configuración de ventanas actual
TRANS	Convertir de un SCP a otro

Introducción de datos

GETPOINT	Introducir un punto
GETCORNER	Introducir punto señalando esquina de rectángulo
GETINT	Introducir un número entero
GETREAL	Introducir un numero real
GETANGLE	Introducir ángulo
GETORIENT	Introducir ángulo según origeen y sentido
GETDIST	Introducir una distancia
GETSTRING	Introducir una cadena de texto
GETKWORD	Permitir opciones o palabras clave
INITGET	Modos para los comandos del tipo GET...
GETENV	Valor de variable de entorno de AutoCAD

Operaciones numéricas

+	Sumar
-	Restar
*	Multiplicar
/	Dividir
1 +	Sumar 1

1- Restar 1
SQRT Raíz cuadrada
EXP Exponentc base "e'"
LOG Logaritmo neperiano
EXPT Exponente cualquier base
REM Resto de división
ABS Valor absoluto
FIX Parte entera
MAX Máximo número
MIN Mínimo número GCD Máximo común denominador

Cuadro A.2. comandos de AutoLISP agrupados según su naturateza (continuación)

Operaciones numéricas

PI Número real 3.14159
COS Coseno de ángulo
SIN Seno de ángulo
ATAN Arco tangente de ángulo

Comparaciones numéricas

= Igual
/ = No igual
< Menor
> Mayor
< = Menor o igual
> = Mayor o igual

Operaciones lógicas

AND	Y lógico
OR	O lógico
NOT	NO lógico
EQUAL	Igualdad de evaluaciones
EQ	Identidad de expresiones

Manipulación de cadenas textuales

STRCAT	Concatenar cadenas
STRLEN	Longitud de una cadena
SUBSTR	Subcadena de una cadena
STRCASE	Cambio a mayúsculas o minúsculas
ASCII	Código del primer carácter de una cadena
CHR	Carácter correspondiente a código ASCII
WCMATCH (V.11)	Comparar cadena de texto con filtro

Conversiones

ATOF	Cadena de texto en número real
ATOI	Cadena de texto en entero
ITOA	Número entero en cadena de texto
RTOS	Número real en cadena de texto
FLOAT	Número entero en real
ANGTOS	Ángulo de radianes a cadena de texto
CVUNIT (V.11)	Convertir valor a nuevas unidades

Símbolos y expresiones de AutoLISP

QUOTE	Literal de símbolo o expresión
EVAL	Evaluar símbolo o expresión
SET	Atribuir valor a símbolo no evaluado
READ	Primera expresión de una cadena
VER	Versión de AutoLISP
error	Función de error

Estructuras de programación

IF	Alternativa
COND	Condiciones múltiples
REPEAT	Repetir N veces
WHILE	Repetir según condición
PROGN	Agrupar expresiones consecutivas
TRACE	Marcar funciçon con atributo de rastreo
UNTRACE	Desactivar atributo de rastreo

Cuadro A.2. comandos de AutoLISP agrupados según su naturateza (continuación)

Tratamiento de listas

LIST	Formación de una lista
CAR	Extraer primer elemento de lista
CDR	Extraer segundo elemento y resto de lista
CADR, ...	Segundo elemento de lista….
CONS	Añadir primer elemento a lista
APPEND	Juntar listas
LENGTH	Longitud de lista
LAST	Ultimo elemento de lista
ASSOC	Lista asociada a elemento
MEMBER	Resto de lista a partir de un elemento
SUBST	Sustituir elemento
NTH	Elemento enésimo de lista
FOREACH	Lista invertida
APPLY	Aplicar expresión a lista
MARCAR	Aplicar función a lista
MAPCAR	Aplicar función a elementos sucesivos de listas

Lectura y escritura

PROMPT	Escribir mensaje en pantalla
TERPRI	Terminar escritura
PRIN1	Escribir expresión
PRINC	Escribir expresión ASCII
PRINT	Escribir con interlínea
WRITE-CHAR	Escribir caracter ASCII
WRITE-LINE	Escribir linea
READ-CHAR	Leer carácter
READ-LINE	Leer linea
OPEN	Abrir archivo
CLOSE	Cerrar archivo
FINDFILE	Explorar camino de acceso a archivos

Manipulación de conjuntos designados de entidades

SSGET	Introducir conjunto
SSLENGTH	Número de entidades de conjunto
SSNAME	Nombre de entidad en conjunto
SSADD	Añadir entidad a conjunto
SSDEL	Eliminar entidad de conjunto
SSMEMQ	Ver si entidad está en conjunto

Manipulación de nombres de entidades

ENTNEXT	Nombre de entidad siguiente
ENTLAST	Nombre de la última entidad principal
ENTSEL	Nombre de entidad principal con punto de designación
NENTSEL (V. 11)	Nombre de entidad componente con punto de designación
HANDENT	Nombre de entidad asociada a rótulo

Gestión de datos de entidades

ENTGET	Lista de entidad en Base de Datos
ENTDEL	Borrar o recuperar entidad
ENTMOD	Actualizar lista de entidad en Base de Datos
ENTUPD	Regenerar entidad compuesta
ENTMAKE (V.11)	Construir lista de entidad

Acceso a Tabla de Símbolos

TBLNEXT	Extraer contenido de tabla
TBLSEARCH	Buscar tabla con un nombre dado

Gestión de datos extendidos de entidades

REGAPP (V. 11)	Registrar nombre de aplicación
XDROOM (V. 11)	Espacio disponible para datos extendidos
XDSIZE (V. 11)	Longitud de lista de datos extendidos

Acceso a pantalla gráfica y dispositivos

GRCLEAR	Despejar ventana gráfica
GRDRAW	Dibujar vector virtual
GRTEXT	Escribir en áreas de texto
GRREAD	Lectura directa de datos de entrada

Gestión de memoria

VMON	Paginación virtual de funciones
GC	Recuperación de memoria inutilizada
ALLOC	Tamaño de segmentos en memoria nodal
EXPAND	Número de segmentos memoria nodal
MEM	Estadística de la memoria

Operaciones de chequeo

ATOM	Detectar átomo
MINUSP	Detectar valor numérico negativo
BOUNDP	Detectar valor asociado a símbolo
LISTP	Detectar lista
NULL	Detectar valor asociado nulo
NUMBERP	Detectar valor numérico
ZEROP	Detectar valor numérico igual a Ø
TYPE	Tipo de elemento

Operaciones a nivel binario

~	Negación lógica
LOGAND	Operación Y lógico
LOGIOR	Operación O lógico
BOOLE	Operación Booleana
LSH	Desplazamiento a nivel binario

Cuadro A.3.Órdenes del editor de dibujo de AutoCAD.

Español	Inglés
ACOTA	DIM
ACOTA1	DIM1
ALARGA	EXTEND
APERTURA	APERTURE

ARANDELA	
	DONUT
ARCO	
	ARC
ARCHIVOS	
	FILES
ÁREA	
	AREA
ARRASTRE	
	DRAGMODE
ATRDEF	
	ATIDEF
Español	
	Inglés
ATRED1T	
	ATTEDIT
ATREXT	
	ATTEXT
ATRVIS	
	ATTDISP
AYUDA ./ ”?	
	HELP /(`)?
BASE	
	BASE
BLADISCO	
	WBLOCK
BLOQUE	
	BLOCK
BOCETO	
	SKETCH
BORRA	
	ERASE
CAMBIA	
	CHANGE

CAMBPROP	CHPROP		‘ PANTTEXT	TEXTSCR

CAPA	LAYER		PARTE	BREAK
CARGA	LOAD		PCARA(V. 11)	PFACE
CARGADXB	DXBIN		PLANTA	PLAN
CARGADXF	DXFIN		POL	PLINE
CIRCULO	CIRCLE		POLÍGONO	POLYGON
COLOR	COLOR		PTOVISTA	VPOINT
COPIA	COPY		PUNTO	POINT
CRGIGES	IGESIN		QUITA	QUIT
CHAFLAN	CHAMFER		R	U
DDSCP	DDUCS		REANUDA	RESUME
DESCOMP	EXPLODE		RECORTA	TRIM
DESIGNA	SELECT		RECUPERA	OOPS
DESPLAZA	MOVE		REDIBT	REDRAWALL
DIST	DIST		REDIBUJA	REDRAW
DIVIDE	DIVIDE		REFENT	OSNAP
EDITPOL	PEDIT		REFX (V. 11)	XREF
EDITPOL	PEDIT		REGEN	REGEN
EJES	AXIS		REGENAUTO	REGENAUTO
ELEV	ELEV		REGENT	REGENALL
ELIPSE	ELLIPSE		REJILLA	GRID
EMPALME	FILLET		RELLENA	FILL
ENCUADRE	PAN		RENOMBRA	RENAME
EQDIST	OFFSET		RESVI.STA	VIEWRES
ESCALA	SCALE		RETARDA	DELAY
ESCALATL	LTSCALE		REVISIÓN (V. 11)	AUDIT
ESPACIOM (V. 11)	MSPACE		REVOCA	UNDO
ESPACIOP (V. 11)	PSPACE		ROTULOS	HANDLES
ESTADO	STATUS		RSCRIPT	RSCRIPT
ESTILO	STYI.E		SACAFOTO	MSLIDE
ESTIRA	STRETCH		SALIMPR	PRPLOT
FIN	END		SALTRAZ	PLOT
FORMA	SHAPE		SALVA	SAVE
FORZCOOR	SNAP		SALVADXF	DXFOUT
GIRA	ROTATE		SCP	UCS
GRADUA	MEASURE		SCRIPT	SCRPT
ID	ID		SH	SH
INSERT	INSERT		SHELL	SHELL
INSERTM	MINSERT		SIMBSCP	UCSICON
INVOCA	REDO		SIMETRIA	MIRROR
ISOPLANO	ISOPLANE		SLVIGES	IGESOUT
LIMITES	LIMITS		SOLIDO	SOLID
LIMPIA	PURGE		SOMBRA (V. 11)	SHADE
LINEA	LINE		SOMBREA	HATCH
LIST	LIST		SUPLADOS	EDGESURF
LISTBD	DBLIST		SUPREGLA	RULESURF
LOCTEXTO	QTEXT		SUPREV	REVSURF

MARCAAUX	BLIPMODE		SUPTAB	TABSURF
MATRIZ	ARRAY		TABLERO	TABLET
MENU	MENU		TEXTO	TEXT

MIRAFOTO ‘ MODlVAR OCULTA ORTO PANTGRAF VSLIDE
SETVAR.
HIDE ORTHO GRAPHSCR TEXTODIN TIEMPO TIPOLIN TRAZO UNIDADES
DTEXT TIME LINETYPE TRACE UNITS

Cuadro A.3.Órdenes del editor de dibujo de AutoCAD. (continuación)

Español	Inglés		Español	Inglés
UNIRX (V. 11)	XBIND		ZOOM	ZOOM
VENTANAS	VIEWPORTS		3DCARA	3DFACE
VGCAPA (V. 11)	VPLAYER		3DLINEA	3DLINE
VISTA	VIEW		3DMALLA	3DMESH
VISTADIN	DVIEW		3DPOL	3DPOLY
MULT (V. 11)	MVIEW			

Cuadro A.4. Ordones del editor de acotación de AutoCAD.

Español	Inglés
ACTUALIZA	UPOATE
ALINEADA	ALIGNED
ÁNGULO	ANGULAR
CENTRO	CENTER
CONTINUA	CONTINUE
DIAMETRO	

	DIAMETER
DIRECTRIZ	
	LEADER
ESTADO	
	STATUS
ESTILO	
	STYLE
FIN	
	EXIT
GIRADA.	
	ROTATED
HORIZONTAL	
	HORIZONTAL
INVALIDAR (V. 11)	
	OVERRIDE
LINEABASE	
	RASELINE
Español	
	Inglés
NUEVOTEXTO	
	NEWTEXT
OBLICUA (V. 11)	
	OBLIQUE
ORDINAL (V. 11)	

	ORDINATE
RADIO	
	RADIUS
REDIBUJA	
	REDRAW
RESTABL (V. 11)	
	RESTORE
REVOCA	
	UNDO
SALVAR (V. 11)	
	SAVE
TEDIC (V. 11)	
	TEDIT
TEXTOAINIC	
	HOMETEXT
TROTAR (V. 11)	
	TROTATE
VARIABLES (V. 11)	
	VARIABLES
VERTICAL	
	VERTICAL

APENDICE

19 Apéndice B: Códigos para entidades de dibujo en Base de Datos.

• Línea

((-1 . <Entity name: 6ØØØØØ45>) Nombre de entidad (Ø . "LINE") Tipo de entidad
(8 . "PIEZA") Nombre de capa (1Ø 25.ØØ 75.5Ø Ø.ØØ) Coordenada punto inicial (11 8Ø.ØØ 12Ø.ØØ Ø.ØØ) Coordenada punto final
(6 . "TRAZOS") Nombre tipo de linea Si falta, es "porcapa" Si es Ø, es "porbloque"
(62 . 1) Número de color Si falta, es "porcapa" Si es Ø, es "porbloque"
(38 . 1Ø.ØØ) Valor de elevación Sí falta, es Ø
(39 . 15.ØØ)Valor de altura de objeto

Si falta, es Ø)

• Círculo

((-1 . <Entity name: 6ØØØØØ24>) Nombre de entidad (Ø . "CIRCLE") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø 2Ø.ØØ 5Ø.ØØ Ø.Ø) Coordenada del centro (4Ø.15Ø.ØØ) Valor del radio)

Nombre de tipo de línea, número color, elevación y altura objeto, las mismas que LINEA

• Arco

((-1 . <Entity name: 6ØØØØØ21>) Nombre de entidad
(Ø . "ARC")Tipo de entidad
(8 . “TALADROS) Nombre de capa (1Ø 5.ØØ 5.ØØ Ø.ØØ) Coordenada del centro (4Ø . 15.ØØ) Valor del radio (5Ø . Ø.784354) Angulo inicial en radianes
(51 . 2.549871)Ángulo final en radianes

) Nombre de tipo de lína, número color, elevacióny altura objeto, las mismas que LÍNEA.

• Punto

((-1 . <Entity name: 6ØØØØØ54>) Nombre de entidad (Ø . "POINT") Tipo de entidad
(8 . “Ø”) Nombre de capa (1Ø . 25.ØØ 37.5Ø Ø.ØØ)
Coordenada del punto)

Nombre de tipo de linea, número color, elevación y altura objeto, las mismas que LINEA

• Trazo

(Primer tramo del trazo: (-1 . <Entity name: 6ØØØØØ16>)
Nombre de Entidad (Ø . "TRACE") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø 25.ØØ 25.1Ø Ø.ØØ) Coordenada primera esquina del punto inicial (11 25.ØØ 24.9Ø Ø.ØØ) Coordenada segunda esquina del punto inicial (12 35.1Ø 25.1Ø Ø.ØØ) Coordenada primera esquina del punto final (13 34.9Ø 24.9Ø Ø.ØØ) Coordenada segunda esquina del punto final) (Segundo tramo del trazo: (-1 . <Entity name: 6ØØØØØ21>) Nombre de entidad (Ø . "TRACE") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø 35.1Ø 25.1Ø Ø.ØØ) Coordenada primera esquina del punto inicial. Debe coincidir con punto final del primer tramo. (11 34.9Ø 24.9Ø Ø.ØØ) Coordenada segunda esquina del punto inicial. Debe coincidir con punto final del primer tramo

(12 5Ø1Ø 35.1Ø Ø.ØØ)Coordenada primera esquina del punto final. Deberá coincidir con punto inicial del siguiente tramo
(13 5Ø.9Ø 34.9Ø Ø.ØØ)Coordenada segunda esquina del punto final. Deberá coincidir con punto inicial del siguiente tramo
) ….y así sucesivamente con todos los tramos Nombre de tipo de linea, número color, elevación y altura objeto, las mismas que LINEA

• Sólido

((-1 . <Entity name: 6ØØØØØ75>) Nombre de entidad (Ø . "SOLID") Tipo de entidad
(8 . "BORDE") Nombre de capa (1Ø 1Ø.ØØ 1Ø.ØØ Ø.ØØ) Coordenada primer vértice (11 2Ø.ØØ 1Ø.ØØ Ø.ØØ) Coordenada segundo vértice)
Número color, elevación y altura objeto, las mismas que LINEA

• Texto

((-1 . <Entity name: 6ØØØØØ92>) Nombre de entidad (Ø . "TEXT") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø 5.ØØ 1Ø.ØØ Ø.ØØ) Coordenada punto de inserción (4Ø 3.5Ø) Altura de caracteres
(1 . “Ejemplo de texto) Cadena de texto (5Ø . Ø.ØØ) Ángulo rotación en radianes
(7 . “TSI”)Nombre estilo texto
(41 . 1.ØØ)Factor proporción estilo
(51 . Ø.ØØ)Ángulo inclinación estilo
(71 . Ø) Generación del txto: Ø Normal 2 Reflejado izquierda 4 Cabeza abajo
(72 . Ø) Justificación texto: Ø Izquierda 1 Centrado 2 Derecha 3 Situar 4 Rodear 5 Ajustar

(11 5.ØØ 15.ØØ Ø.ØØ)
Coordenada punto
justificación depende de

que el texto sea centrado,
derecha, etc.
) Número color, elevación y altura objeto, las mismas que LINEA

• Bloque

(
(<Entity name: 6ØØØØ1ac>) Nombre de entidad
- .
1
("INSERT") Tipo de entidad
Ø .

(8 . "Ø")Nombre de capa
(66 . 1) Señal atributos Ø no hay atributos 1 hay atributos
(2 . “RESIS”) Nombre de bloque (1Ø 25.ØØ 25.ØØ Ø.ØØ) Coordenada punto inserción
(41 . 1.5Ø)Escala X de inserción
(42 . Ø.5Ø)Escala Y de inserción
(43 . 1.ØØ) Escala Z de inserción (5Ø . Ø.ØØ) Ángulo rotación radianes (7Ø . Ø) Número columnas MINSERT
(71 . Ø)Número filas MINSERT
(44 . Ø.ØØ)Espaciado columnas
(45 . Ø.Ø) Espaciado filas)

• Atributo

((-1 . <Entity name: 6ØØØØ124>) Nombre de entidad (Ø . "ATTRIB") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø 4.5Ø Ø.ØØ Ø.ØØ) Coordenada punto inserción identficador (4Ø . 3.5Ø) Altura de texto
(2 . “RES”)Nombre identificador
(1 . “5ØØk”)Valor de atributo

(7Ø .) Tipo de atributo: Ø Normal 1 Invisible 2
Constante
4
Verificabl
e
(5Ø . Ø.ØØ)Ángulo rotación de inserción actual
(41 . 1.ØØ)Factor proporción texto
(51 . Ø.ØØ)Ángulo inclinación texto
(7 . “TS1”)Nombre estilo de texto
(71 . Ø)Generación del texto. Ver valores en TEXTO
(72 . 1)Justificación texto. Ver valores en TEXTO
(11 . 6.ØØ Ø.ØØ Ø.ØØ) Coordenada punto justificación

) ... y asi sucesivamente con todos los atributos del bloque hasta encontrar SEQEND

((-1 . <Entity name: 6ØØØØØ85>) Nombre de entidad (Ø . "SEQEND") Tipo de entidad
(8 . "Ø") Nombre de capa (-2 . <Entity name: 6ØØØØ1ac>) Nombre entidad de bloque que contiene atributos)
• Polilínea

((-1 . <Entity name: 6ØØØØØ55>) Nombre de entidad principal (Ø . "POLYLINE") Tipo de entidad
(8 . "Ø") Nombre de capa
(66 . 1) Señal de que siguen vértices

(7Ø . 1) Señal de cierre: Ø Abierta 1 Cerrada
(4Ø . Ø.ØØ) Grosor inicial por defecto
(41 . 1.ØØ) Grosor final por defecto) ((-1 . <Entity name: 6ØØØØØ21>) Nombre entidad de vértice (Ø . "VERTEX") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø 12.ØØ 25.ØØ Ø.ØØ) Coordenada del vértice (4Ø . Ø.ØØ) Grosor inicial en ese vértice
(41 . Ø.ØØ) Grosor final hacia el vértice siguiente
(42 . Ø.ØØ) Factor de curvatura si se trata de poliarco

(7Ø . Ø) Señal adaptación de curva: Ø No curva adaptada 1 Curva adaptada 2 Curva B
(5Ø . Ø.ØØ) Dirección tangente radianes en curva adaptada
) ... y asi .sucesivamente con todos los vértices de la polilínea hasta encontrar SEQEND

((-1 . <Entity name: 6ØØØØ216>) Nombre de entidad (Ø . "SEQEND") Tipo de entidad
(8 . "Ø") Nombre de capa (-2 . <Entity name: 6ØØØØØ55>) Nombre entidad principal que contiene vértices)

Polígono, Elipse, Arandela; Son polilíneas cerradas
3DCARA

((-1 . <Entity name: 6ØØØØ234>) Nombre de entidad (Ø . “3DFACE") Tipo de entidad
(8 . "Ø") Nombre de capa (1Ø Ø.ØØ Ø.ØØ Ø.ØØ) Coordenada primer vértice (11 1Ø.ØØ Ø.ØØ Ø.ØØ) Coordenada segundo vértice (12 1Ø.ØØ 5.ØØ Ø.Ø) Coordenada tercer vértice (13 Ø.ØØ 5.ØØ Ø.ØØ) Coordenada cuarto vértice)

• 3DMALLAS: Son polilíneas con M x N vértices