ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Facultad de Ingeniería en Electricidad

"INTERFASE ENTRE UN GRAFICADOR Y UNA BASE DE DATOS"

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN COMPUTACION

Presentada por:

Marcos Fernando Ruiz Guzmán

Guayaquil - Ecuador

1992

AGRADECIMIENTO: Mi infinita gratitud al Ing. Sixto García, Director de Tesis, por su constante preocupación y excelente disposición a lo largo del desarrollo de la presente Tesis de Grado.

DEDICATORIA:

A cada miembro de mi Familia, por compartir conmigo todo momento y constituir el mejor ejemplo a seguir.

En especial, a mis Padres, por bridarme siempre Su cariño y confianza.

DECLARACION EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestas en esta Tesis, me corresponden exclusivamente; y, el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral"

(Reglamento de Exámenes y Títulos Profesionales)

Marcos Ruiz Guzmán

Subdecano de la Facultad de
Ingeniería en Electricidad
Ing. Jorge Flores Macías

Director de Tesis
Ing. Sixto García Aguilar

Miembro del Tribunal
Ing. Carlos Monsalve

Miembro del Tribunal
Ing. Jaime Puente P.

INDICE

INTRODUCCION

En la actualidad en nuestro país no existen profesionales que se dediquen a diseñar e implementar software. Recursos humanos y técnicos tenemos, no nos cabe la menor duda, pero por qué entonces no producimos software, por qué simplemente nos dedicamos a desarrollar simples sistemas escritos en lenguajes como RPG y COBOL (para los mainframes) o seudo lenguajes como 4GL, Foxpro y tantos otros (para las micros), que tienen como función ayudar a llevar la contabilidad de una empresa. La respuesta tiene lamentablemente características económicas.

No tratamos de desmerecer lo anterior, la computación juega un papel primordial en las empresas, de hecho se está haciendo imprescindible en cualquier actividad comercial por pequeña que sea. Lo que tratamos aquí de decir es que tenemos una puerta cerrada, que muy pocos la abren, la puerta de la investigación y desarrollo de software científico. Y esta palabra "científico" debe entenderse no como algo más allá de nuestro nivel tecnológico y académico, sino como el desarrollo de programas que se salgan del esquema antes mencionado, en el que es cuestión de aprender a programar, coger un manual de ORACLE y sentarnos a hacer una aplicación; por favor!, entonces para qué hemos obtenido el nivel académico que está muy por

encima de esto. Aquellos que se crean capaces deberían investigar e implementar programas de comunicaciones, interfases, aplicaciones a ingeniería civil, a geología, etc., donde se pongan en práctica el bagaje de conocimientos adquiridos y que no deberíamos desperdiciar.

La presente tesis "Interfase entre un Graficador y una Base de Datos" es un primer intento de dar a los usuarios de AutoCAD una poderosa herramienta, una unión con una base de datos que procesa la información en línea.

El primer tema que trataremos será sobre el graficador AutoCAD. Veremos como maneja sus entidades gráficas y luego analizaremos en detalle el AutoLISP, el lenguaje del cual nos valdremos para accesar a la base de datos gráfica del AutoCAD.

En el segundo tema hablaremos del ambiente gráfico Windows, estudiándolo a fondo. Los modos de direccionamiento de los Intel 286 y 386, y los manejadores de memoria son puntos claves a tratar para completar el aprendizaje.

Después daremos un vistazo a la base de datos Foxpro. Veremos las nuevas herramientas que nos ofrece y justificaremos su elección.

A continuación estudiaremos la interfase en sí. Iremos explicando el funcionamiento de cada parte de la interfase, la estructura de sus archivos, los programas escritos en AutoLISP, el proceso de comunicación, etc.

Por último tendremos el manual del usuario, donde veremos paso a paso como y para que utilizar cada función de la interfase, empleando un ejemplo de una urbanización.

CAPITULO I

AUTOCAD

1.1 Herramientas CAD

Si buscáramos en un diccionario de términos informáticos el significado de esta palabra, encontraríamos lo siguiente:
CAD. Computer Aided Design. (Diseño Asistido por Computador). Aplicación del computador para el diseño de productos de consumo e industriales, incluso para el diseño de tarjetas de circuito impreso, chips integrados, y en otros muchos campos.

La idea de aplicar el computador al proceso del diseño industrial surgió por primera vez en la década de los setenta (en el Massachusets Institute of Technology). No obstante, habría de transcurrir una década más hasta que la tecnología del computador le permitiera al diseñador ver una representación gráfica de de su trabajo e interactuar con ella.

En CAD, la representación gráfica del trabajo de un diseñador, puede ser accedida para modificación mediante un digitalizador o un lápiz óptico, exactamente igual que si estuviera frente a un tablero de dibujo. El diseñador puede modificar esa imagen incorporando componentes predibujados y subensambles, y después sacar una copia del dibujo acabado con

un plotter. El computador se convierte entonces, en un sistema de delineación.

Desde el punto de vista del programa, se trata de como almacenar esta imagen de forma tal que se la pueda alterar y manipular. Si fuéramos a realizar el modelo físico de un objeto, utilizaríamos uno de dos métodos básicos: aditivo o sustractivo. El método aditivo es como modelar en arcilla: se va construyendo el objeto pieza tras pieza hasta que llegamos a la forma final. El método sustractivo se parece al que siguen los escultores, que sacan la obra desbrozando la materia.

El software para diseño auxiliado por computador comparte muchas de las características de los paquetes de imágenes generadas por computador: atenuación de curvas, eliminación de elementos ocultos, sombreado, relleno y recoloreado de zonas, por ejemplo. Para formar una curva sólo se requiere la solución repetida de una ecuación simple para una serie de valores. Especificando los puntos de comienzo y final para una línea determinada, y la distancia máxima que alcanzará la curva respecto a dicha línea recta, proporcionamos una solución para la ecuación. A partir de esta solución podemos trabajar a la inversa, para deducir la ecuación en sí misma, y después proceder a resolverla para el resto de la serie de valores, formando así la curva.

Esta capacidad para componer un dibujo a partir de partes estándar de los componentes es el verdadero punto fuerte de los sistemas CAD. Ya no es necesario volver a dibujar los componentes individuales comunes. Una vez que estos se han definido, se puede volver a llamar dicha definición cada vez que se la requiera e incorporarla a nuevos dibujos. Un ejemplo destacable de esto es el uso de computadores para auxiliar el diseño de las futuras generaciones de computadores.

El diseño de un circuito impreso, por ejemplo, es muy complicado, por lo que se requiere la aplicaión de técnicas de optimización para acomodar los componentes y sus pasos de interconexión de la forma más económica posible. El diseñador debe remitirse a menudo al ensayo y error, y es aquí donde los paquetes CAD son especialmente útiles. Todos los componentes individuales se almacenan como imágenes predefinidas y se les llama cuando se necesitan. Resulta muy sencillo probar un diseño determinado en la unidad de visualización de datos, para ver si responde a todas las exigencias, antes de trasladarlo al papel como parte de un dibujo en ejecución. Siguiendo este método se puede esbozar un diseño e incluso probar diferentes soluciones, empleando el tiempo que llevaría completar un boceto.

Los circuitos integrados se diseñan casi exactamente de la misma manera, pero debido a la densidad de componentes y vías de

conexión, se requiere además otra configuración de software: la capacidad para ampliar una parte del dibujo, trabajar en él a escala aumentada y luego devolverlo a su posición dentro del diseño global. Este efecto es hoy una pieza básica del repertorio CAD y ha representado una considerable mejora en cuanto a la eficacia del sistema. Mediante su utilización se puede retener la especificación de un objeto completo en un solo dibujo, y se puede ajustar la escala para satisfacer las necesidades de quien lo mira.

Y no es sólo la escala lo que se puede variar de este modo. Si consideramos el caso de un objeto más complejo (un coche por ejemplo), estamos ante múltiples subsistemas que, juntos, conforman el todo: el sistema eléctrico, el sistema hidráulico, el de escape, la suspensión, etc. Mientras que al diseñador de estética le interesa más el paquete global, a los ingenieros individuales probablemente lo que más les interese sea un solo subsistema. Es muy sencillo hacer que cada subsistema vaya en un color distinto para extraer después del dibujo global, todos los objetos de un color determinado. Esto no equivale a afirmar que el dibujo siempre ha de estar constituido por una mezcla de colores, la codificación se puede suprimir a voluntad cuando no se la considere necesaria.

El avance más sensacional es la capacidad para retener la especificación completa de un objeto (no sólo su forma y su aspecto, sino también información relativa al material en el que está construido, su peso, costo, etc.). Recabar información acerca de la forma y el tamaño del objeto es sólo una función del sistema, que se puede considerar como una base de datos orientada hacia la visualización. Formulando distintas preguntas a dicha base de datos se pueden ordenar pedidos a los proveedores, planificar la fabricación de subensamblajes y componentes, integrar las cadenas de producción para asegurar que los componentes lleguen exactamente a donde y cuando se les necesita, analizar costos y controlar la eficacia de la fabricación, entre otras muchas cosas. Estamos tentados de imaginar que el paso siguiente será un sistema regular de control directo de la fabricación por computador.

1.2 El Graficador AutoCAD

Uno de los programas CAD más poderosos y populares es el AutoCAD. Este paquete, porque realmente es un conjunto de varios programas, es el que eligimos para construir nuestro proyecto por sus características y cualidades. Dentro de sus muchas "virtudes" hablaremos solamente de las más importantes, al menos en relación a nuestro trabajo.

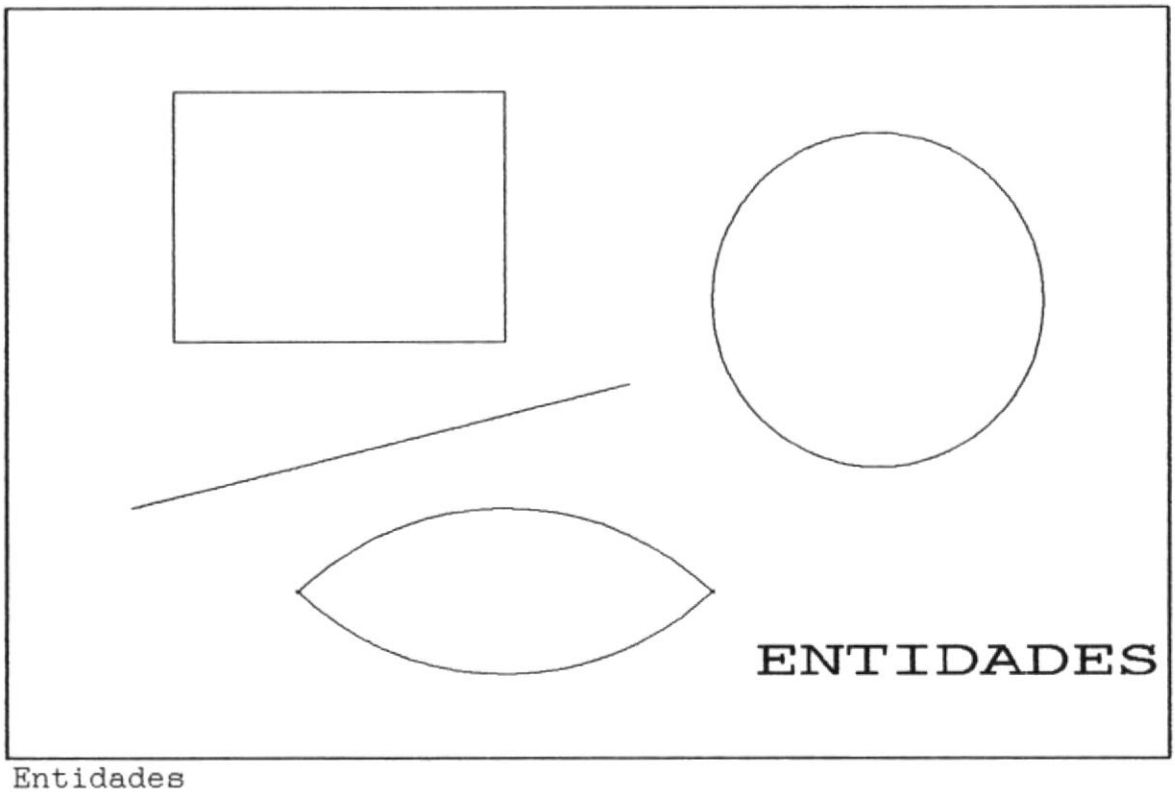
Es necesario aclarar acerca de este capítulo, que el desarrollo de los temas aquí tratados no pretenden ser una guía instructiva o de aprendizaje del AutoCAD o del AutoLISP, sino más bien se trata de dar al lector una idea de todas las herramientas que se usaron para la realización del proyecto.

1.2.1 Entidades

Cuando usamos las herramientas gráficas del AutoCAD para dibujar en la pantalla, estamos creando una entidad gráfica. Un segmento de línea es una entidad, círculos y texto son también entidades. Además de su localización geométrica, cada entidad tiene un color y tipo de línea asociado. AutoCAD llama a estas asociaciones propiedades.

1.2.2 Bloques

AutoCAD hace posible agrupar entidades individuales y tratarlas como un solo objeto. Símbolos y partes gráficas son típicos candidatos para esta clase de agrupaciones. AutoCAD llama a estos grupos de entidades BLOQUES. Nosotros consideramos *partes* como representaciones de objetos reales, dibujados a escala o tamaño total, como un carro o un escritorio. Símbolos son objetos símbolicos, no dibujados a escala, como receptáculos eléctricos, símbolos de soldadura, etc.



Entidades

Si solamente quisiéramos mover o copiar un objeto símbolo, podríamos colectar las entidades individuales en una selección de conjunto y hacer la edición apropiada. Sin embargo, en cuanto los gráficos comienzan a volverse más "populosos", se vuelve más dificultoso conseguir las entidades que queremos en el cojunto de selección. Es mucho más fácil agrupar las entidades en un bloque, dar un nombre al bloque, grabarlo, y hacerlo disponible para un uso posterior en el gráfico o en otros gráficos.

AutoCAD nos permite justamente hacer eso: juntar entidades, dar un nombre de grupo, y operar sobre el grupo como un todo. Además de sus conveniencias, los bloques proveen beneficios

adicionales de edición. Cuando insertamos bloques en nuestro gráfico, podemos modificar su escala y rotación. Típicamente, los usuarios construyen bibliotecas de bloques frecuentemente usados para acelerar la creación del gráfico. Podemos también reemplazar globalmente un bloque con otro, revisando el gráfico entero con un único comando de inserción.

Analicemos ahora los bloques desde el punto de vista de la eficiencia. Cuando graficamos 100 líneas, añadimos 100 entidades al gráfico. Cuando hacemos un bloque conteniendo 100 líneas, AutoCAD crea una *definición de bloque* de 100 entidades. Definiciones de bloques son invisiblemente guardados en el archivo gráfico. Cuando nosotros visiblemente insertamos un bloque en el gráfico, solamente añadimos una nueva entidad, una referencia a la definición del bloque. Así, si insertamos un bloque conteniendo 100 líneas en el gráfico en 12 lugares, sólo tendremos 112 entidades que AutoCAD tendrá que guardar. Al contrario, si nosotros hubiéramos dibujado y copiado 12 grupos de 100 líneas sin bloques, hubiésemos añadido 1200 datos de entidades en el archivo del gráfico.

Los comandos principales con que trabajamos los bloques son: BLOCK e INSERT. Con BLOCK definimos las entidades que van a conformar el bloque y le damos al bloque un nombre. Con INSERT en cambio, insertamos un nuevo bloque del tipo de bloque cuyo

nombre hayamos especificado, lo ubicamos dentro del gráfico, y le damos su escala en x,y (el nombre de bloque que vayamos a insertar ya debe haber sido definido previamente con BLOCK).

1.2.3 Atributos

Al crear una figura completa existe más, que solamente sus entidades gráficas y sus relaciones espaciales. Se podría tener un bloque insertado en el gráfico, pero muchas veces no sólo queremos ver el objeto graficado, sino también alguna información adicional de texto que nos proporcione mayor detalle de la figura, que la complemente. Pensemos en las conveniencias de relacionar entidades con atributos de texto. Podríamos extraer automáticamente costos de materiales, horarios, o podríamos ver la información en forma gráfica. No necesitaríamos tampoco tener todos los datos visibles, podríamos guardarlos invisiblemente hasta que los requiramos en un reporte.

Pero, cómo trabajan los atributos? Nosotros podemos guardar un atributo como un nombre en un bloque de la misma manera que guardamos entidades en un bloque. Asi como creamos primero las entidades antes de incluirlos en un bloque, asi definimos las etiquetas de atributos previo a la creación de un bloque con atributos. La única regla de los atributos es que tienen que ser parte de un bloque.

AutoCAD provee un comando de definición de atributos llamado ATTDEF. Se emplea ATTDEF para crear etiquetas de atributos, definiendo como los atributos de texto van a ser almacenados. Después, se usa el conocido comando BLOCK para agrupar las entidades gráficas y las etiquetas de atributos para formar el bloque. En efecto, introducimos los atributos en la misma definición de bloque.

Una vez formado el bloque de atributos, lo insertamos en el gráfico usando el comando estándar INSERT. Se controla los atributos de pantalla y los atributos de edición con tres comandos adicionales: ATTDISP controla la visibilidad del texto de atributo en el gráfico; ATTEDIT permite hacer cambios al texto de atributo después de haber sido insertado; y ATTEXT extrae información del texto de atributo y del bloque que puede ser usada en un reporte.

1.2.4 Macros y Menúes

Un macro es una palabra abreviada de macro-comandos, es decir, un largo o gran comando. Un ejemplo sería un macro que ejecuta tres veces Control-C:

[^C]^C^C^C

Si colocáramos este macro en el menú de una pantalla y seleccionamos la opción [^C], se ejecutarían tres ^C, como si

uno mismo las hubiese digitado en el teclado. Los macros se construyen escribiendo una secuencia de comandos en un archivo texto.

Los macros se agrupan en archivos de menú. Un menú es un archivo de texto donde cada opción es un macro. El menú estándar del AutoCAD es el ACAD.MNU. Cuando uno crea sus propios comandos usando menúes de macros, uno automatiza sus secuencias de comandos. Para ejemplificar lo dicho, tenemos el siguiente archivo de menú con sus opciones macro:

```
[^C]^C^C^C
[MOVE:     ]^C^C^CMOVE AUTO
[COPY:     ]^C^C^CCOPY AUTO
[STRETCH:]^C^C^CSTRETCH C
[ERASE:    ]^C^C^CERASE AUTO
[CH:LAYER ]^C^C^CSELECT  AU \CHPROP P ;LAYER  (getvar "CLAYER") LAYER
\;
[LAYER:S   ]^C^C^CLAYER S \;
[]
[1-LINE:   ]^C^C^CLINE \\;
[ARC:      ]^C^C^CMULTIPLE ARC
[CIRCLE:   ]^C^C^CCIRCLE
[PLINE:    ]^C^C^CPLINE \W 0 ;
[FILLET:   ]^C^C^CFILLET R 0 MULTIPLE FILLET
[DTEXT:    ]^C^C^CDTEXT
[INSERT:S ]^C^C^CINSERT \SCALE
[]
[VIEW:ALL ]^VIEW R ALL
[ZOOM:P    ]^ZOOM P
[]
[EDIT-MNU ]^C^C^CSHELL \MENU ;
```

Ejemplo de un Menú con Macros

1.3 AutoLISP

AutoLISP es un dialecto del lenguaje LISP que fue derivado del XLISP. AutoLISP coexiste con AutoCAD dentro del programa AutoCAD. Todo lenguaje de programación, incluyendo el AutoLISP, provee algunas herramientas generales de programación. Estas incluyen vías para estructurar el flujo de un programa, herramientas para manipular los datos del programa, y medios para entrada y salida de datos a otros dispositivos.

En adición a estas herramientas generales, AutoLISP tiene herramientas específicas del AutoCAD. Estas herramientas permiten accesar y actualizar las entidades de datos de un gráfico AutoCAD, accesar las tablas, bloques, capas, vistas, estilos y tipos de líneas, y controlar la pantalla gráfica del AutoCAD y el dispositivo de entrada.

Cuáles son los beneficios de utilizar AutoLISP ? AutoLISP es la comunicación directa con AutoCAD. Con él, se tiene acceso a las entidades, tablas de referencia, y a archivos fuera y dentro del AutoCAD. Usando AutoLISP en los menúes de macros, se pueden salvar datos en variables, procesar esos datos, y regresarlos al AutoCAD. Puntos, distancias, y otros valores pueden ser almacenados, calculados, comparados, y usados para

graficar. Se puede controlar el ambiente gráfico a través de las variables del sistema AutoCAD guardando las variables del sistema, cambiando y restaurando los valores. Se puede cambiar la configuración del sistema, trabajar con las entidades gráficas para usarlas en los programas, todo esto de una forma transparente para el usuario.

1.3.1 Variables y Expresiones

Cuando el AutoLISP fue por primera vez introducido dentro del programa AutoCAD, éste no era llamado AutoLISP. Era solamente llamado "Variables y Expresiones" para la construcción de macros.

Variables

Una variable es una etiqueta usada para referir un valor cambiable. En AutoLISP, se puede relacionar un valor a un nombre de variable y usarlo en un comando o dentro de expresiones de AutoLISP para ejecutar cálculos y hacer decisiones lógicas.

De manera similar, el AutoCAD tiene variables que guardan los valores del snap, osnap, ortho o simplemente la posición de un punto; éstas son llamadas variables del sistema AutoCAD (se puede llegar a ellas a través del comando SETVAR).

Se puede accesar a estas variables y crear nuevas variables con nombres propios en AutoLISP. Las variables del sistema y cualquier variable AutoLISP que se cree es de uno de los tres tipos siguientes:

- **STRING´s (cadenas)**

  Tienen valores texto entre comillas dobles para identificar el valor como una cadena de caracteres. Los valores "3.3", "PISO-1", "-1234", y "La pequeña construcción está ..." son todas cadenas.

- **INTEGER´s (enteros)**

  Son números enteros positivos o negativos, sin fracciones, puestos decimales o puntos decimales. AutoCAD a mennudo usa valores de 0 y 1 para indicar el estado de una variable del sistema, como SNAPMODE y ORTHOMODE, apagada (0) o prendida (1). Los enteros deben estar entre -32768 y +32767.

- **REAL´s (reales)**

  Son números positivos o negativos con posición decimal. En AutoLISP no se puede comenzar o terminar un número real con un decimal. Si el valor es menor a 1.0, se debe poner un 0 antes del punto decimal (0.213) o se recibirá un error. Ejemplos de variables del sistema reales son FILLETRAD, LTSCALE y AREA. A diferencia de los enteros, se pueden

hacer reales muy grandes o muy pequeños. AutoLISP genera reales en notación científica cuando estos son muy grandes o pequeños.

- **LIST´s (listas)**

  Consisten de uno o más valores de cualquier tipo de variable agrupado dentro de paréntesis. Los puntos son considerados como listas de AutoLISP. Ejemplos de listas serían: (1.0 2.0 3.0), ("A" "B" "C"), (1.0 "1" 1).

Usando variables del sistema, se puede controlar muchos valores del editor gráfico AutoCAD. Se pueden modificar la mayoría, no todas. Ciertas variables están indicadas como de *sólo lectura*, lo que significa que se puede solamente extraer mas no actualizar sus valores.

Expresiones

Ahora bien, una expresión AutoLISP comienza con un paréntesis abierto y termina con un paréntesis cerrado. Expresiones AutoLISP pueden contener otras expresiones AutoLISP. Se debe anidar cada expresión en su propio par de paréntesis. Una típica expresión AutoLISP tiene la siguiente sintaxis:

(función argumento)

Las reglas a seguir en las expresiones son:

- Cada expresión tiene su par de paréntesis abierto y cerrado.

- Cada expresión tiene un nombre de función. El nombre debe seguir inmediatamente al paréntesis abierto.

- Cada expresión es evaluada (ejecutada) y retorna un resultado. El resultado puede ser nil, o el último valor evaluado.

- Todo en AutoLISP es True (Verdadero) o nil. Si algo no tiene valor, es nil. Si este tiene valor, es True (no-nil).

Funciones y Argumentos

Una función es una subrutina que le dice al AutoLISP que tarea ejecutar. Tareas incluye adición, subtracción, multiplicación o división. Una función podría tener cualquier número de argumentos o ninguno.

Un argumento provee datos a una función. Argumentos pueden ser variables, constantes, u otras funciones. Algunos argumentos son banderas que alteran la acción de la función. Si se define

una función para tomar un argumento, se debe proveer a la función con el valor para el argumento.

Los paréntesis abiertos y cerrados permiten al AutoCAD distinguir entre comandos AutoCAD y expresines AutoLISP. Cada vez que el AutoCAD detecta un paréntesis abierto, este cede la expresión entera al AutoLISP. AutoLISP evalúa la expresión y retorna el resultado al AutoCAD. AutoCAD usa el resultado y continúa.

Usando Funciones Matemáticas del AutoLISP

AutoLISP tiene algunas funciones matemáticas incorporadas. Cuando hablamos sobre las funciones AutoLISP, decimos (+ 1 2) son 3, por ejemplo. En álgebra decimos 1 + 2 igual 3. Este ejemplo de adición tiene dos argumentos, ambos constantes. Argumentos son los datos para la función. Algunas funciones de AutoLISP, como + - * y /, pueden tener cualquier número de argumentos. Algunas funciones requieren un número específico de argumentos, mientras que otros requieren argumentos específicos, pero tienen también parámetros opcionales. A continuación se presenta un lista de funciones incorporadas que se podrían usar en macros:

(/ arg1 arg2 arg3 ...)	ARG1 dividido por ARG2 dividido por ARG3 ...
(* arg1 arg2 arg3 ...)	ARG1 multiplicado por ARG2 multiplicado por ARG3 ...
(+ arg1 arg2 arg3 ...)	ARG1 más ARG2 más ARG3 ...
(- arg1 arg2 arg3 ...)	ARG1 menos ARG2 menos ARG3 ...
(1+ arg)	ARG más 1
(1- arg)	ARG menos 1
(abs arg)	Valor absoluto de ARG
(exp arg)	e elevado a la ARG potencia
(expt base potencia)	BASE a la POTENCIA
(gcd arg1 arg2)	Máximo Común Denominador de ARG1 y ARG2
(log arg)	Logaritmo natural de ARG
(max arg1 arg2 arg3 ...)	Máximo de argumentos
(min arg1 arg2 arg3 ...)	Mínimo de argumentos
(rem arg1 arg2 arg3 ...)	Residuo de ARG1 dividido por ARG2 ...
(sqrt arg)	Raíz cuadrada de ARG

AutoLISP evalúa expresiones al mismo nivel de anidamiento, y de izquierda a derecha. Cuando una expresión anidada es encontrada, la entera expresión anidada es evaluada antes de la siguiente expresión a la derecha. Si todos los argumentos son enteros, los resultados serán enteros y cualquier parte fraccional será eliminada. Si algún argumento es real, el

resultado será real. Tratemos algunas funciones matemáticas y veamos los anidamientos:

```
Comando: (+ 1 2.0)
Lisp retorna: 3.0

Comando: (/ 3 2)
Lisp retorna: 1

Comando: (* 5 (- 7 2))
Lisp retorna: 25
```

Creación de Variables

Creamos variables dando valor a un símbolo. AutoLISP automáticamente asigna el tipo de dato cuando las creamos. Las variables AutoLISP son completamente independientes de las variables del sistema AutoCAD y sus nombres podrían duplicar los nombres de variables del sistema AutoCAD. Cada vez que usamos el nombre de variable o nos referimos al nombre de variable en un macro, programa o expresión, el programa reemplaza el nombre de variable con el más reciente valor asignado a ese nombre.

Las variables pueden ser cualquier combinación imprimible de letras y números excepto aquellas reservadas debido a sus especiales significados en AutoLISP. Existen también nombres de funciones AutoLISP que no deberíamos usar como nombres de variables. La "ATOMLIST" (Lista Atómica) es una variable AutoLISP que guarda todas las funciones definidas y nombres de variables. AutoLISP listará todas las variables usuario-definidas y funciones y sus nombres de funciones si tipeamos !ATOMLIST. El signo de exclamación indica al AutoCAD retornar el valor de la variable AutoLISP que le sigue.

Comando: !ATOMLIST Visualizaremos la lista ATOMLIST.

Setq relaciona un valor guardado con un nombre de variable. En álgebra, nosotros escribimos y=3, mas en AutoLISP entramos (setq y 3). Ambos el = de la expresión algebraica y el setq de la expresión AutoLISP son funciones. Cada función relaciona el valor 3 con la variable y. Los paréntesis abierto y cerrado forman la expresión.

```
Comando: (setq a 1)
Lisp retorna: 1

Comando: (+ (setq a (* a 3)) (+ a 2))
Lisp retorna: 8
```

```
Comando: (+ (+ a 2) (setq a (* a 3)))
Lisp retorna: 14
```

Después de relacionar un valor con un nombre de la variable, podemos usar un signo de exclamación para dar ese valor al AutoCAD. El signo de exclamación identifica la palabra que sigue como símbolo de AutoLISP, generalmente un nombre de variable. Cuando AutoCAD ve el caracter !, este pasa el nombre de variable al AutoLISP. AutoLISP lo interpreta y pasa su valor devuelta al procesador de comandos AutoCAD.

```
Comando: (setq y 2) Asignamos el entero 2 a la variable y.
Lisp retorna: 2

Comando: !y         Mandamos el valor de y al AutoCAD.
Lisp retorna: 2

Comando: (setq x 3)
Lisp retorna: 3

Comando: (+ x y)    Usamos los valores en una expresión de
adi-
Lisp retorna: 5     ción.
```

Creación de Funciones en AutoLISP

Nosotros podemos definir nuestras propias funciones en AutoLISP. **Defun** define una función construyendo una lista estructurada de las sentencias del programa. Nuestras funciones AutoLISP crean un ambiente local. Los datos pasan al ambiente local de la función, las sentencias del programa los usan y manipulan, y luego regresan los datos al ambiente general AutoLISP - AutoCAD.

La forma general de DEFUN es:

```
(defun NOMBRE (ARGs / LOCALs)
   SENTENCIAS DE PROGRAMA...
)
```

Podemos hacer que el NOMBRE de una función tome cualquier nombre que deseemos, con caracteres en mayúsculas y/o minúsculas. Como los nombres de variables, se debe evitar usar los nombres reservados en la lista atómica. Si se usa un nombre reservado, se redefinirá la función original del AutoLISP y ésta no se restablecerá hasta que no se empiece un nuevo gráfico.

Las sentencias de programa son el núcleo de una función. Las sentencias de programa siguen las reglas generales de

evaluación del AutoLISP, de izquierda a derecha y de adentro hacia afuera. Los resultados de la última sentencia evaluada son retornados al ambiente global AutoLISP - AutoCAD.

Argumentos y otras variables usadas dentro de funciones pueden ser *locales* o *globales*. Locales son variables que necesitamos y usamos sólo dentro de la función. Las variables deben ser declaradas (listadas) en los paréntesis siguiendo el nombre de la función en DEFUN para ser local. AutoLISP crea un pequeño ambiente local, el cual almacena los valores de las variables locales. Nosotros usualmente no queremos que las variables de la función sean globales. Los argumentos de una función son siempre locales nos guste o no. Variables locales no tienen valor fuera de su función padre, a menos que hayan sido creadas también fuera de la función. Si una variable tiene el mismo nombre que una variable local fuera de la función, su valor no es afectado por la función y viceversa.

Las variables globales tienen un valor fuera de la función en que fueron creadas, como dentro de la función creadora. Cada variable o función que se cree, global o local, va a la lista atómica durante la actual sesión gráfica. Exceptuando los argumentos, las variables son globales a menos que se las declare locales. Se usa como prefijo estándar, el caracter #, para indicar una variable global.

1.3.2 Accesando la Base de Datos Gráfica

Todos pensamos en AutoCAD como una herramienta de dibujo, pero AutoCAD es también un administrador de base de datos. Cada entidad gráfica existe como un registro de datos en la base de datos gráfica del AutoCAD. Al igual que la mayoría de bases de datos se podría pensar que la base de datos gráfica del AutoCAD almacena registros conteniendo números y nombres. Pero también describe la entidad geométrica, cadenas de texto, atributos, tablas de referencia, y valores del ambiente gráfico de AutoCAD. Como cualquier otra base de datos, AutoCAD ordena, adiciona y elimina, lista y reporta sus registros de información gráfica. El casi completo control que el AutoLISP brinda sobre los datos gráficos del AutoCAD, nos permite obtener grandes beneficios, como modificar, crear, borrar objetos gráficos usando nuestros propios comandos previamente creados. Se puede crear programas más inteligentes, capaces de buscar en el gráfico y seleccionar automáticamente entidades, entre otras muchas ventajas.

Entidades AutoCAD

Cada entidad en un gráfico AutoCAD tiene un único número de identificación como <Entity name: 60000030>. Este nombre de entidad (o ename, abrev. en inglés) es el único medio que se tiene para recuperar los datos de una entidad a través del

AutoLISP. El nombre de la entidad apunta a los datos que definen una entidad en la base de datos gráfica y permite a los programas accesar la información que necesitan.

AutoCAD guarda su base de datos en el mismo orden en el cual se crean las entidades. El nombre de entidad y el orden son mantenidos durante toda la sesión de edición. Entidades eliminadas o borradas permanecen en la base de datos y son simplemente *apagadas*.

Se puede pasar nombres de entidades a cualquier comando AutoCAD que requiera selección de objetos donde la opción Last sea válida. Los comandos AutoCAD aceptan nombres de entidades sin importar que función es usada para recuperarlas. Pero no se puede crear nombres de entidades como se crean otras variables. Un nombre de entidad no es una cadena o un símbolo, es el tipo de datos ename.

Se necesita obtener información como coordenadas, escalas, cadenas de texto, y propiedades, para que se pueda utilizar AutoLISP haciendo cálculos y modificaciones a los datos. La función AutoLISP ENTGET posibilita recobrar estos datos de la entidad. ENTGET lee dentro de la base de AutoCAD y retorna los datos de la entidad que se requerió. Los datos son retornados en una lista asociada, como:

((-1 . <Entity name: 60000018>) (0 . "LINE") (8 . "0") (10 1.0 1.0 0.0) (11 2.0 3.0 4.0) (210 0.0 0.0 1.0))

AutoCAD usa el código DXF (Drawing Exchange Format o Formato de Itercambio Gráfico) para reportar datos de entidad. Un código DXF precede cada tipo de dato, como un punto, capa, nombre, elevación, etc. El código DXF permite a los programas manejar los datos de entidades del AutoCAD sin depender del orden de estos.

Valores Código	**Tipo Dato**	**Uso General**
0-9	Cadenas	Tipo entidad, nombre objeto, texto, manejo.
10-37	Reales	Coordenadas.
38, 39	Reales	Elevación, espesor.
40-59	Reales	Factores de escala, ángulos.
60-79	Enteros	Colores, banderas, contadores, modos.
80-209	No usado	
210, 220, 230	Reales	X,Y,Z dirección de extrusión.
999	Cadenas	Comentarios.
1000-1009	Cadenas	Xdata (datos entidad extendida)
1010-1059	Reales	Xdata
1060-1079	Enteros	Xdata

Los códigos completos DXF son presentados en dos tablas: los Grupos DXF de Entidades y los Grupos DXF de Tablas. Estas

tablas se hayan reproducidas en el apéndice A. Como muestra la siguiente tabla, los códigos DXF se dividen en algunos grupos mayores.

Los siguientes ejercicios presentan las listas y códigos DXF para algunas entidades comunes. Para dejar espacios para los comentarios, se escribió los ejemplos diferente a lo que se vería en la pantalla.

```
Comando: (entget (car (entsel)))
Seleccione objecto:                    Seleccione una entidad línea.
Lisp retorna:
((-1 . <Entity name: 60000019>)        El nombre de entidad.
(0 . "LINE")                           El tipo de entidad.
(8 . "0")                              Su nombre de capa.
(10 1.0 1.0 0.0)                       Primer punto fin.
(11 2.0 3.0 4.0)                       Segundo punto fin.
(210 0.0 0.0 1.0)                      Dirección de extrusión

Comando: (entget (car (entsel)))
Seleccione objeto:                     Escoger un texto.
Lisp retorna:
((-1 . <Entity name: 60000034>)        Nombre de entidad.
(0 . "TEXT")                           Tipo de entidad.
(8 . "0")                              Nombre de capa.
```

(10 2.46276 3.0 0.0)	Punto de alineamiento de texto AutoCAD.
(40 . 0.125)	Altura texto.
(1 . "ALGUN TEXTO")	Cadena de caracteres.
(41 . 1.0)	Factor de anchura.
(50 . 0.0)	Angulo de rotación.
(51 . 0.0)	Angulo de inclinación.
(7 . "STD")	Nombre de estilo.
...	

Entidades de Manejo - Nombres de Entidades Permanentes

Entidades de manejo eliminan el problema de variación de los nombres de entidad. Un manejador es una etiqueta opcional y permanente de identificación asignada a cada entidad en un gráfico. Cada entidad está garantizada de tener una única identificación ID de manejo a lo largo de su vida en el gráfico. Cuando una entidad es borrada, el manejador es retirado para siempre del gráfico. Copias de entidades y desuniones de entidades generan nuevos manejadores. Ningún nombre de manejador es entonces repetido en el mismo gráfico.

Los manejadores de entidad son activados por el comando HANDLES en AutoCAD. Estos pueden ser desactivados en un gráfico, pero sólo por un usuario porque la operación requiere responder

a preguntas. Esto significa que AutoLISP puede activar, pero no deshabilitar, las entidades de manejo. Si los manejadores están activados, la sólo-lectura variable del sistema HANDLES es 1. Los manejadores se presentan en la salida del comando LIST y en la lista de datos de entidad como una cadena con valor hexadecimal en un grupo 5. Los programas no pueden crear o actualizar un manejador, es decir, no se tiene control sobre los nombres de manejo generados por AutoCAD.

Los manejadores son útiles en una variedad de aplicaciones, tales como enlace entre los objetos gráficos y una base de datos externa, agrupación de entidades dentro de un gráfico, o como una simple vía para recordar el nombre de una importante entidad en un gráfico.

Comando: HANDLES Los enciende.

Handles are disabled.

ON/DESTROY: ON

Comando: (entget (car (entsel))) Note la inclusión del grupo 5.

Lisp retorna: ((-1 . <Entity name: 60000068>) (0 . "INSERT") (8 . "0") (5 . "4") (2 . "IL06") (10 4.5 1.5 0.0) (41 . 0.5) (42 . 0.25) (50 . 0.0) (43 . 0.5) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0) (210 0.0 0.0 1.0))

Cuando las entidades de manejo son encendidas, AutoCAD asigna un manejador a cada entidad principal en el gráfico, incluyendo las entidades borradas. Los manejadores son guardados en los datos de cada entidad bajo el código DXF 5. Un manejador es un tipo de datos cadena do sólo-lectura conteniendo un valor hexadecimal. Si se tienen muchas entidades, los números serán cadenas hexadecimales como "1A67B0".

Pero si las entidades ya tienen nombres, por qué necesitamos los manejadores? Como se dijo, los nombres de entidad son reasignados en cada sesión gráfica. Nombres de entidad borradas en la sesión previa son eliminadas. Esta naturaleza transiente hace que los nombres de entidad sean inútiles para relacionar entidades con una base de datos o para cualquier función sesión a sesión.

Los manejadores son permanentes y secuencialmente asignados para la vida del gráfico. Manejadores de entidades borradas simplemente se convierten en inofensivas brechas en la secuencia. Una vez que el manejador es asignado, una base de datos puede depender de él para siempre.

La Tabla de Datos AutoCAD

Se necesita total acceso a la base de datos gráfica para dar a las aplicaciones completo control sobre el AutoCAD. El

triunvirato de una base de datos gráfica consiste de la sección de cabacera donde son guardados los valores de las variables del sistema, la sección de tablas donde son guardadas las definiciones, y la sección de las entidades. Aprendiendo a manipular las definiciones AutoCAD de las capas, tipos de líneas, vistas, estilos de texto, bloques, sistemas de coordenadas y puertos de vistas completará el control y flexibilidad. Estas son todas las cosas, lugares y propiedades nombrados en la base de datos gráfica. AutoCAD llama a las tablas que guardan estas definiciones *tablas de símbolos.*

Aquí, símbolo no significa solamente bloques, inserciones o figuras; significa una *palabra que representa algo.* Por ejemplo, "DASHED" representa un tipo particular de patrón de línea. A diferencia del acceso a las entidades, que era fácilmente verificable observando las entidades en la pantalla, el acceso a las tablas requiere un poco más de estudio. La información gráfica guardada en la tabla de símbolos no es atada a ninguna entidad específica. Estas definiciones de tabla se muestran indirectamente definiendo cómo y dónde aparecen las entidades.

La información definida en las tablas de símbolos AutoCAD LAYER, LTYPE, VIEW, STYLE, BLOCK, UCS, DIMSTYLE Y VPORT pueden ser accesadas por medio del AutoLISP. A pesar que podemos accesar las definiciones de cualquier tabla, no podemos

directamente cambiarlas. Esta limitación se impone debido a que muchas entidades dependen de estas definiciones y una modificación directa podría causar un gran daño al gráfico.

No vamos a encontrar información sobre una específica entidad gráfica en la tabla de símbolos. Lo que vamos a encontrar son datos que definen las llamadas cosas, lugare y propiedades. Los nombres de las tablas de símbolos incluyen:

BLOCK para las llamadas cosas como la definición de bloques (no inserciones de entidades).

VIEW, UCS y VPORT para los llamados lugares como vistas, sistemas de coordenadas y puertos de vistas.

LAYER, LTYPE, STYLE y DIMSTYLE para las llamadas propiedades como capas, tipos de líneas, estilos de texto y estilo de dimensión.

AutoLISP tiene sólo dos funciones que trabajan con todas las tablas de símbolos. Ellas son TBLSEARCH y TBLNEXT. TBLSEARCH busca la tabla especificada y retorna los datos para el específico nombre de símbolo que se suministró. TBLNEXT se posiciona secuencialmente a través de cualquier tabla,

retornando los datos de una entrada por cada paso. Se usa generalmente TBLNEXT para examinar una tabla.

Las funciones de tabla retorna datos en forma similar a una lista de datos de entidad. Como las tablas de entidades, la lista es una secuencia de grupos asociados que usan códigos DXF para identificar el tipo de información en cada grupo. Los Códigos DXF de Tablas de Grupos están reproducidos en en apéndice A.

Comando: (tblsearch "LAYER" "0")	Vamos a obtener la definición de la capa 0.
Lisp retorna: ((0 . "LAYER")	Tipo de tabla.
(2 . "0")	Nombre de capa.
(70 . 64)	Bandera.
(62 . 7)	Color de la capa.
(6 . "CONTINUOUS")	Tipo de línea de la capa.

Comando: (tblnext "LAYER") Avanza a la siguiente capa.

Lisp retorna: ((0 . "LAYER") (2 . "ANN03") (70 . 64) (62 . 6) (6 . "CONTINUOUS"))

1.2.4 Administración de Memoria

A medida que se crean más funciones y progrmas largos, la demanda de la memoria AutoLISP se hace más intensa. Los

programadores están siempre deseosos de computadoras más grandes, rápidas, mejores. AutoLISP, un pequeño pero significante acompañante para AutoCAD, ocupa sólo 128K de memoria en la versión de 640K DOS, unos 83K para el programa en sí y 45K para los programas personales y los datos. Cuarenta y cinco mil bytes de datos definitivamente no llegan muy lejos. En algún punto, se empieza a visualizar los mensajes:

error: insufficient node space
error: insufficient string space

Existen dos vías para administrar la memoria AutoLISP. La primera es la más simple; simplemente encendiendo VMON, la función de *"memory-swapping"* del AutoCAD. La segunda, en Release 10, es usar ExtLISP (Entended AutoLISP).

La función VMON (Virtual Memory ON) le dice al AutoCAD que intercambie funciones AutoLISP dentro y fuera de memoria cada vez que lo necesite.

Desde los inicios de AutoCAD, el programa ha *paginado* los datos del gráfico fuera de la memoria hacia el disco cuando este ha necesitado más memoria. El disco duro ha servido de *memoria virtual*, por lo que los gráficos AutoCAD nunca han sidos limitados por la cantidad de memoria RAM de la computadora. Este

pagineo consume un buen porcentaje de tiempo del AutoCAD, lo cual es una de las razones que los sistemas no-DOS sean más rápidos. La función VMON invoca el mismo pagineo para las funciones definidas AutoLISP. Para encender VMON, sencillamente tipeamos (vmon), como cualquier función del AutoLISP.

Desafortunadamente, una buena administración de memoria no es tan simple como prender VMON y olvidarnos. Un problema es que VMON no nos ayudará con la memoria usada por las variables y cadenas. Valores de variables y cadenas son datos, no programas, y no son paginados. El pagineo VMON es para las funciones definidas solamente, no para sus datos.

Comprendiendo la Memoria AutoLISP

La memoria AutoLISP está dividida en dos áreas especiales: *heap y stack.* Funciones largas y complejas ponen mayor carga en el heap, y funciones recursivas impactan el stack. Heap y stack son usadas como muestra la lista.

La única cosa que VMON libera es la demanda de funciones usuario-definidas, pero esta liberación indirectamente ayuda a tener memoria libre para otros própositos. Esto repercute directamente en los errores de "out of node space" y "insufficient string space". Los nodos, además de ser usados en

las definiciones de función, son la unidad básica de almacenamiento de datos en el AutoLISP, 10 bytes en sistemas DOS y 12 bytes en sistemas no-DOS. Los nodos son agrupados y colocados en segmentos, con un tamaño por omisión de 512 nodos.

HEAP del AutoLISP	**STACK del AutoLISP**
Por omisión 40000 bytes	Por omisión 3000 bytes
Funciones AutoLISP	Argumentos de funciones
Símbolos (nombres de variables)	Resultados parciales
Funciones definidas por usuario	Datos temporales durante
Espacio de cadenas	función
Espacio de nodos	Punteros del AutoLISP
Datos	Contadores de programa
VMON libera	VMON no ayuda

Nosotros podemos ver cuanta memoria está actualmente asignada al AutoLISP por el AutoCAD con la función MEM.

```
Comando: (mem)
Nodes:       2560
Free nodes:  145     Nodos disponibles al momento.
Segmentos:   5       5 x 512 tamaño del segmento = 2560 nodos.
Allocate:    512     Tamaño segmento.
Collections: 24      Número colecciones realizadas.
Swap-ins:    0       Funciones paginadas.
```

```
Page file:    504     Tamaño del archivo de página.
Lisp retorna: nil
```

CAPITULO II
MICROSOFT WINDOWS

Windows es un ambiente gráfico, que introduce nuevas y dinámicas vías para trabajar con el computador personal. Windows no sólo brinda más control sobre el flujo de trabajo, sino que también libera al computador para que pueda operar a todo su poder, impedido por previas restricciones de memoria.

Con Windows, uno encuentra fácil arrancar y trabajar con aplicaciones de software, correr más de una aplicación a la vez, transferir información entre ellas, y organizar y administrar los archivos creados por ellas.

Windows 3.0 puede correr en tres modos: modo real, modo estándar, o modo agrandado. El modo que se use depende del equipo que se tenga y de las aplicaciones de software que se quieran correr en el ambiente Windows.

Debido a que el modo estándar explota características especiales del procesador Intel 80286, este ofrece beneficios que no son disponibles en modo real. De la misma manera, debido a que este toma ventajas de las cualidades especiales del CPU Intel 80386, el modo agrandado 386 provee de beneficios que no

Usando Modo Real

- El kernel de Windows corre en modo real
- Windows task-switches programas DOS
- Gran parte de Windows es sacada cuando un programa DOS corre
- Programas DOS no pueden ser aventanados
- Programas DOS corren en modo real o, usando un DOS extender, en modo protegido
- Programas Windows corren en modo real
- Se puede correr viejos programas Windows 2.x

Usando Modo Estándar

- El kernel de Windows corre en modo protegido 16-bits
- El Windows DOS extender y el DOS Protected Mode Interface (DPMI) están disponibles para programas Windows, pero no para programas DOS
- Windows task-switches programas DOS
- Gran parte de Windows es sacada cuando un programa DOS corre
- Programas DOS no pueden ser aventanados
- Programas DOS corren en modo real o, usando un DOS extender, en modo protegido
- Programas Windows corren en modo protegido 16-bits
- Todos los programas Windows comparten la misma Tabla Local de Descriptores (LDT) en modo protegido

Usando Modo 386 Agrandado

- Porciones del kernel de Windows corren en modo protegido 32-bits
- Manejadores de dispositivos virtuales usan modo protegido 32-bits
- El Windows DOS extender y el DPMI están disponibles para programas Windows y programas DOS
- Programas Windows corren en modo protegido 16-bits
- Programas Windows pueden tener componentes en modo protegido 32-bits
- Programas DOS pueden correr en modo Virtual 86, protegido 16-bits, o protegido 32-bits
- Windows provee de memoria virtual
- Windows multitasks múltiples programas DOS
- Programas DOS pueden correr en plano posterior (background)
- Programas DOS pueden correr en una ventana; Windows queda aún disponible mientras programas DOS están corriendo
- Cada prompt DOS consigue su propia máquina virtual
- Todos los programas Windows comparten la misma Tabla Local de Descriptores (LDT) en modo protegido
- Todos los programas Windows corren en una máquina virtual

son disponibles en el modo estándar o modo real. Las

características de arquitectura de cada modo Windows están sumarizadas en el recuadro.

Viendo el cuadro nos damos cuenta de las muchas diferencias que se hallan en cada modo, pero a más de eso, hemos de resaltar que se hace mención a aspectos de hardware, memoria y modalidades de procesamiento ciertamente poco conocidos. En este capítulo pretendemos más que hablar sobre aspectos descriptivos del Windows, hablar de estos temas fundamentales sobre los que se asienta el Windows para poder así llegar a tener una idea clara del funcionamiento interno de este popular ambiente.

2.1 Memoria Segmentada

En su forma más simple, un segmento en la arquitectura Intel es un bloque de 64K de memoria. En los procesadores 8088, 8086, y 80286, memoria en cantidades mayores a 64K no puede ser direccionada de plano, sino que la memoria debe ser particionada en pequeños segmentos de 64K y luego direccionados un solo segmento a la vez.

Por qué *segmentos*? Intel quiso darle la facultad al 8086 de direccionar un completo megabyte de memoria. Desde que el

número de diferentes direcciones de memoria que uno puede tener es de 2^n, donde *n* es el número de bits que se tiene dado para representar memoria, era necesario construir una máquina que fuera capaz de direccionar 1MB = 2^{20}.

Intel quiso quedarse con sus registros internos de 16 bits para guardar compatibilidad con los anteriores microprocesadores Intel. Esto creaba inmediatamente un serio conflicto: direccionamiento de 16 bits limita a 2^{16}, o 64K de RAM. Por lo que para que ambas metas fueran logradas, Intel optó por dar al 80286 20 líneas de dirección, pero dejar los registros de 16 bits y hacer que cuatro de ellos se usaran como *registros de segmentos* - registros cuyo próposito exclusivo era guardar las direcciones de los segmentos. Usando este esquema, el CPU podría formar una dirección de 20 bits, combinando el contenido de un registro de segmento con un registro de desplazamiento (offset). En otras palabras, el registro de segmento contiene la dirección donde el segmento comienza, y la dirección en el registro de desplazamiento es interpretada como el desplazamiento relativo a la base del segmento.

Por convención, direcciones de memoria son raramente referidas en una forma absoluta. Al contrario, ellas son presentadas en la forma segmento:offset, con los valores de segmento y offset escritos en hexadecimal y separados por dos

puntos. La dirección 12345h, por ejemplo, deberían ser escritas normalmente como 1234:0005. La ventaja de usar esta notación es que claramente se puede ver los valores usados para llegar a la dirección resultante.

Segmentación 286

El mismo concepto de segmentación que se aplica al procesador 8086, también se aplica al 286, 386, o 486 corriendo en modo real. El direccionamiento de memoria es limitado a 1MB, y direcciones 20-bits son formadas de valores de 16 bits.

En ciertos casos, podemos accesar más de 1MB de memoria en modo real cargando el registro de segmento con un valor mayor que F000h y combinándolo con una dirección de offset lo suficientemente alta como para producir una dirección mayor que 100000h. Por ejemplo, la dirección segmento:offset FFFF:1000h referencia actualmente la dirección absoluta 100FF0h, que es claramente superior al límite de 1MB. Puesto que un 8086, con 20 líneas de dirección, no es físicamente capaz de direccionar más de 1MB de RAM, la dirección se vuelca, haciéndose 0000:0FF0h. Pero en los 286, 386 y 486, habilitando selectivamente la línea de dirección A20 (la cual es normalmente usada sólo en modo protegido) permite al CPU alcanzar los primeros 64K de la memoria superior, a pesar que

este corriendo en modo real. Esta región de 64K es conocida como el *Area de Memoria Alta*, o HMA. El manejador prototipo Microsoft de Especificación de Memoria Extendida (XMS) HIMEM.SYS, que será explicado en detalle más adelante, provee un mecanismo para controlar la línea de dirección A20 y para arbitrar accesos al HMA entre procesos competidores.

El 80286 es un microprocesador de 16-bits que contiene 19 registros, 14 de los cuales son de interés para el programador de aplicaciones. Estos registros son: AX, BX, CX, BP, SP, DI, SI, CS, DS, SS, ES, IP y el de banderas. Este conjunto de registros es idéntico a los que se encuentran en los microprocesadores 8088 y 8086. Los otros cinco registros se emplean en el modo protegido y de multitarea, y son

1. Registro de la tabla global de descriptores (GDTR)
2. Registro de la tabla de descriptores de interrupción (IDTR)
3. Registro de la tabla local de descriptor (LDTR)
4. Registro de tareas (TR)
5. Registro de estado de palabras en la máquina (MSW)

El sistema operativo hace uso de estos registros para distinguir entre el modo de dirección real y la operación en modo protegido (registro MSW), para especificar la dirección

del segmento correspondiente a la tarea o programa en ejecución (registro TR), las rutinas de interrupción (IDTR) y todos los programas del sistema (GDTR). En las dos siguientes subsecciones se estudian los medios ambientes de desarrollo tanto en modo protegido como en el modo de dirección real.

El modo de dirección real es el ambiente de operación 80286 de Intel que permite tener compatibilidad con el sotfware desarrollado para el 8088 y el 8086. Todo el mecanismo de direccionamiento hace uso de localidades de direcciones físicas y la multitarea no está disponible. El punto importante es que en este modo de operación todos los programas desarrollados para el 8088 y el 8086 serán ejecutados por el 80286 sin ninguna modificación (en el modo de dirección real). Así mismo, excluyendo algunas instrucciones específicas del 80286, todo el código fuente del Macroensamblador del 80286 en modo de dirección real correrá en los sistemas IBM PC y XT.

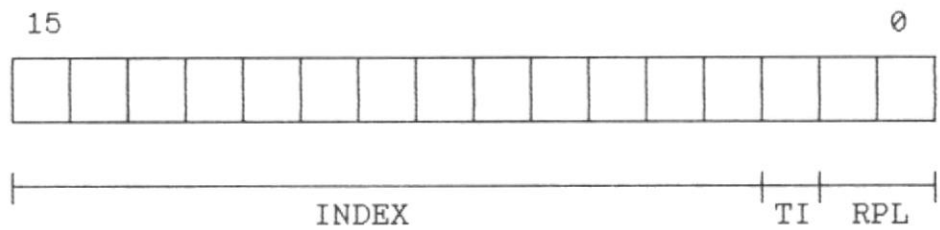
De esta forma, Intel logró completa compatibilidad en software entre los tres circuitos; 8088, 8086 y 80286. Esto no significa que él no tenga avances significativos sobre 8086 y el 8088. Como ya se ha señalado, además de la velocidad, el 80286 tiene caracteristicas nuevas importantes. Sin embargo, para hacer uso de ellas debe emplearse el modo protegido.

Cuando el 80286 opera en el modo de dirección real *calcula*, básicamente, las direcciones de la misma forma que el 8088 y el 8086 utilizando para ellos selectores y desplazamientos. Así mismo, estas direcciones corresponden a direcciones físicas en el sistema. Por una dirección física debe entenderse una localidad de memoria o puerto en el sistema de la microcomputadora que corresponde a una entidad de hardware. Resulta claro que para asignar estas direcciones, el encadenador (*linker*) tiene que considerar la cantidad total de memoria física presente en el sistema del usuario. Para una dirección de 20-bits (segmento más desplamiento), el 80286 en modo real puede tener acceso hasta de un megabyte de memoria. Esta cifra concuerda con la capacidad de direccionamiento encontrada en la PC y XT. Al mencionarse algo específico del 80286 se hace uso del término selector de segmento, ya que éste es clave para las funciones de manejo de memoria del 80286. El selector de segmento tiene una función similar al registro de segmento cuando se emplea el modo real. Afortunadamente, los registros de segmento siempre contienen la dirección correcta de los segmentos, de aquí que tanto DEBUG como el sistema siempre definirán de manera correcta la localidad de memoria física absoluta de una variable o de un puerto de entrada/salida.

Como preludio para un estudio futuro del direccionamiento y distribución de memoria, es importante señalar que las máquinas AT utilizan un bus de 24 bits. Sin embargo, sólo el modo de dirección virtual protegido permite hacer uso de la capacidad de direccionamiento que permite este bus. Con direcciones de 24- bits (un bus efectivo de direcciones de 24 líneas en el AT), el 80286 puede utilizar un espacio de direcciones físicas de 16 megabytes (2^{24}). Por otra parte, en modo de dirección virtual protegido, el mecanismo 80286 para el manejo de memoria puede extender ésta hasta alcanzar un espacio de direcciones virtuales de 1 gigabyte (2^{30}). El 80286 lleva a cabo la transformación entre este espacio virtual y las localidades físicas en el sistema.

En párrafos anteriores se mencionó el hecho de que el 80286, gracias a su canal de 24 líneas de dirección, puede tener acceso hasta 16 megabytes de memoria física y que cuando se emplea la unidad de manejo de memoria el espacio de direcciones crece hasta un gigabyte. Para que esto se lleve a cabo, las instrucciones de manejo de memoria del 80286 se deben usar para obtener un espacio de direcciones virtuales. Cuando el 80286 emplea el mismo modo de direccionamiento que el 8088 y el 8086, la dirección física de 20-bits se obtiene sumando al contenido del segmento, el desplazamiento de cuatro bits a la

izquierda, de un segmento de 16-bits. En modo protegido, el selector de segmento tiene la siguiente forma.



El bit indicador de tabla (TI) define dos espacios separados de direcciones,

0 = direcciones globales

1 = direcciones locales

En un ambiente multitarea,todas las tareas emplean el espacio de direcciones globales mientras que el de direcciones locales sólo es accesible por una tarea. Dado que INDEX es de 13 bits, cada espacio de direcciones se puede subdividir en un máximo de 2^{13} segmentos. El resultado de lo anterior es que INDEX, indexa una tabla residente en memoria que recibe el nombre de tabla de descriptores. Esta última contiene la correspondencia entre la dirección de cada segmento y su localidad en la memoria física. Existe sólo una tabla global de descriptor (GDT) y una o más tablas locales de descriptores (LDT). Cada entrada en la tabla está formada por ocho bytes. El significado de éstos se muestra en la tabla.

	Función
0,1	LIMIT: especifica el tamaño del segmento, hasta 64K
2,3,4	BASE: especifica una dirección base de 24-bits para el comienzo del segmento
5	ACCESS RIGTHS BYTE (byte de derechos de acceso): bit 7 : Protección presente. bit 5-6: Descriptor del nivel de privilegio 0 es el mayor, 3 el menor bit 4: Descriptor de segmento 1: Aplicaciones 0: Sistema bit 3: 1: código 0: datos bit 0-2: Protección de acceso
6,7	Estos bytes son cero para asegurar la compatibilidad con sistemas basados en el 80386.

Para obtener una dirección física a partir de una dirección virtual de 32-bits el procesador selecciona, con base en el estado del bit TI, la GDT o una de las tablas LDT y después multiplica por ocho el contenido del campo INDEX para obtener la ubicación de la tabla de descriptores seleccionada. La dirección física de 24-bits se obtiene al sumar la dirección del segmento de 24-bits (bytes 2,3 y 4) con un desplazamiento de direcciones del segmento de 16-bits. El software del sistema operativo (funciones link y locate) debe asegurar que la dirección se obtenga de manera correcta con el fin de tomar ventaja del uso de una dirección de 24-bits para el segmento base y de un desplazamiento del segmento de 16 bits.

Es importante señalar que el tamaño máximo de cada segemento sigue siendo 64K, lo que no representa problema alguno para emplear módulos independientes en los límites de 64K. En consecuencia, la limitación de 64K del Macroensamblador en el tamaño de los segementos de código no impone, esencialmente, restricciones para el desarrollo de programas.

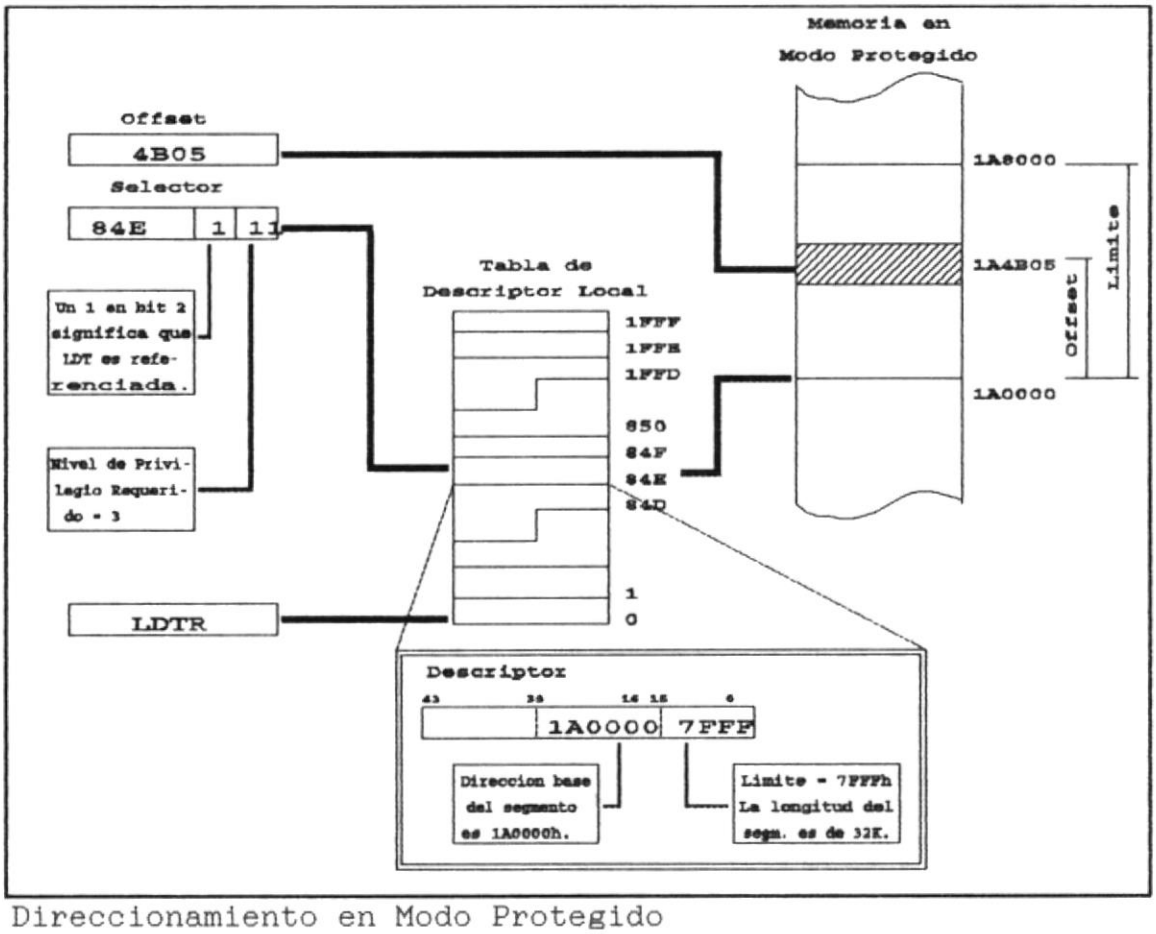
El 80286 tiene características de protección. Estas se encuentran disponibles si se hace uso de los bits 0 a 2 del quinto byte de cada entrada en el stack de descriptores así como de los bits 0 y 1 del selector de segmento. Esta protección es necesaria en ambientes multitarea ya que asegura que el software del sistema tenga su propio espacio y que ningún otro programa pueda invadirlo, hace lo mismo con el software de los diferentes usuarios que en ese momento utilizan el sistema y permite el acceso de datos por niveles de privilegio.

El modo protegido permite hacer llamadas entre diferentes segmentos y proporciona una clase de descriptores de control (ocho bytes) que facilitan que la ejecución de un programa sea transferida de un segmento a otro. Todas las características del modo protegido mencionadas hasta este momento son transparentes para el usuario y su inclusión aquí sirve para

dar una idea de lo que es el ambiente de modo protegido del 80286 tal como puede encontrarse, por ejemplo, en OS/2.

Una de las principales razones por la que los diseñadores de sistemas no utilizaron el modo de dirección virtual protegido sino hasta hace poco, es que el selector de segmento no es tan fácil de manejar en comparación con el direccionamiento del 8088 y el 8086. Antes de cambiar del modo de direcciones físicas a virtuales, se necesita manipular el contenido de INDEX para cambiar el estado del bit indicador de la tabla y el de los bits de protección. Esto, entra en conflicto con el software desarrollado en máquinas basadas en el 8088 ya que es necesario hacer varios ajustes en él (direcciones de segmentos), para poderlo correr en el modo de direcciones virtuales protegido del 80286. Si bien el 80286 puede ejecutar código objeto desarrollado para los microprocesadores 8088 y 8086 (en modo real), en modo protegido la velocidad de ejecución será mucho menor.

La figura diagrama el proceso del 286 para convertir una dirección guardada de la forma selector:offset a una dirección física en la memoia. En este ejemplo, el selector referencia a un descriptor en una tabla de descriptor local, cuya dirección base es guardada en el registro LDTR. El campo índice del selector contiene el valor hexadecimal 84E, el cual apunta al



Direccionamiento en Modo Protegido

descriptor 84. en la LDT. El campo de la dirección base del descriptor contiene el valor de 24 bits de 1A0000h, el cual a su vez nos indica el correspondiente segmento basado en la dirección absoluta 1A0000h, una locación aproximadamente a medio camino entre 1MB y 2MB. El campo límite contiene 7FFFh, indicando que el segmento tiene una longitud de 32K. Finalmente, el valor offset de 4B05h es aplicado relativo a la base del segmento (justo como en el modo real), resultando la dirección absoluta 1A4B05h.

A primera vista, todo esto podría parecer muy confuso. Pero de cerca revela algunas de las ventajas inherentes de estructurar la memoria de esta forma. Teniendo para cada proceso un único LDT, un sistema operativo en modo protegido puede cargar múltiples procesos en la memoria y mantenerlos físicamente aislados uno del otro. La premisa es que un proceso no puede accesar ninguna región de memoria que no este mapeada dentro de su LDT. El sistema operativo puede también prevenir que procesos accesen la GDT corriéndolos a niveles inferiores de privilegio que los de la GDT. Al final, se tiene una sólida base para un ambiente multitarea, donde ningún programa puede corrumpir el sistema sobreescribiendo en memoria ya asignada a otro programa.

Segmentación 80386

La segmentación es marcablemente similar a la segmentación del 286, pero con una particularidad: las capacidades de direccionamiento del 386 son largamente superior a aquellas del 286, proveyendo mayor flexibilidad en la forma en que la memoria en modo protegido puede ser estructurada. De hecho, el 386 fue el primer CPU Intel que barrió con el límite de tamaño de 64K en los segmentos, convirtiéndolo en una atractiva plataforma para los sistemas operativos.

El microprocesador 80386 es diferente del resto de los procesadores de 16-bits (8088,8086 y 80286) ya que el 80386 es un microprocesador de 32-bits. En otras palabras, el 80386 tiene registros y buses de direcciones y datos de 32-bits. Así mismo, el 80386 incorpora en su arquitectura interna todas las funciones de manejo de memoria. En la sección anterior se señaló que buena parte del manejo de memoria(en modo protegido) en el 80286 debe hacerla el software del sistema operativo (o varios circuitos integrados con esta función). En el 80386, el cálculo de las direcciones de los segmentos se lleva a cabo utilizando un buffer de traslación de direcciones(cache) contenido en el propio microprocesador.

Otra diferencia radica en la aritmética. Las CPU de Intel de 16-bits emplean aritmética de una palabra mientras que el 80386 hace uso de aritmética de palabra-doble. El 80386 tiene ocho registros de propósito general: EAX, EBX, ECX, EDX,ESI, EDI, ESP y EBP. El prefijo E sirve para indicar que los registros de 16-bits (AX,BX..) simplemente han sido expandidos a 32-bits. De hecho, la palabra menos significativa de cada uno de estos ochos registros puede emplearse como un registro equivalente de 16-bits, siendo válidas todas las definiciones y palabras reservadas aplicables a los registros de ocho bits. (Estas palabras de 16-bits también pueden subdividirse a su vez en registros de ocho bits, AH, AL, BH, BL,..) Esta

característica implica compatibilidad con el código desarrollado para microprocesadores de 16-bits (8088, 8086 y 80286), pero debe ensamblarse y enlazarse nuevamente.

El apuntador de instrucciones EIP y el registro de banderas (EFLAGS) tienen características de compatibilidad muy similares. Finalmente, existen seis registros de segmento: CS, DS, SS, ES, FS, y GS. Los últimos dos son nuevos y permiten el acceso a segmentos adicionales de datos utilizando sobreescritura (overrides). La longitud de cada uno de estos registros de segmento es de una palabra. Al igual que con el 80286, sólo los programadores de sistemas son los que emplean muchas de las características de la arquitectura del 80386 ya que algunas de sus instrucciones se aplican sólo al software del sistema operativo. Además de los ya mencionados, el 80386 tiene los siguientes registros:

- Registros de manejo de memoria (4): GTDR,LDTR, IDTR, TR(este último se encuentra también en el 80286)
- Registros de control (4): CR0, CR1, CR2, CR3
- Registros de depuración y prueba (8): DR0, DR1, DR2, DR3, DR4, DR5, DR6, DR7

En el modo real, el 80386 ejecuta el código diseñado para ser ejecutado por los procesadores 8086, 8088 y 80286. Desde

el punto de vista del programador, el 80386 emula a los microprocesadores 8086, 8088 y 80286 (este último en modo de direcciones real). El 80386 aparece como un procesador de mayor rapidez con registros de 32-bits y varias instrucciones adicionales.

Es probable que las modificaciones más profundas en el modo protegido del 80386, con respecto al del 80286, sean las que tienen que ver con el manejo de la memoria. El 386 asigna 8 bits adicionales a la dirección base de un segmento, expandiendo el rango de memoria que este puede situar de 16MB a 4GB. Este también asigna 4 bits adicionales al límite, expandiendo el tamaño efectivo del segmento de 64K a 1MB. Pero el tamaño límite de un segmento no se queda allí. Encendiendo el bit de Granularidad en el descriptor del 386 hace que el valor del campo límite sea interpretado como el número de páginas en lugar del número de bytes. Puesto que la longitud de una página es de 4096 bytes (4K), la granularidad de página incrementa el tamaño máximo de segmento 386 a un casi inimaginable 4GB. Una máquina podría no tener nunca semejante memoria, pero si se corre un sistema operativo que virtualiza memoria, como el OS/2, esto no importa: se podrá engañar los programas haciéndolos pensar que existe más memoria instalada.

Cómo accesamos individualmente los desplazamientos dentro de estos gigantescos segmentos? Simple: con registros de 32 bits. Todos los registros generales del 386 son de 32 bits. Desde el punto de vista del programador, las direcciones son todavía construidas con selectores y offsets; pero mientras que el selector es aún de 16 bits, el offset es de 32.

Las funciones de protección y de multitarea involucran también conjuntos de bits en la tabla de descriptores. Estas deben determinarlas el programador de sistemas, lo cual no se expondrá en este capítulo. Ya se ha comentado con suficiencia sobre el modo de direcciones real y el modo protegido para el 80286 y 80386. Esto es para enfatizar la compatibilidad del software desarrollado en máquinas basadas en el modo 80386 por ser considera en la siguiente subsección.

En ambiente multitarea (modo protegido), el 80386 soporta la ejecución de uno o más programas desarrollados ya sea para el 8088 o el 8086. Cuando esto sucede, se dice que el procesador está en modo virtual 8086 y las tareas reciben el nombre de tareas V86, para entrar en este modo, el software del sistema debe activar el bit VM del registro EFLAGS. Una vez hcho esto, el procesador carga los registros de segmento con el formato del 8086 de formación de direcciones y decodifica las instrucciones verificando que no existan violaciones de nivel

de privilegio de entrada/salida (IOPL). Fuera de estas dos facetas del modo V86, el sistema funciona como si estuviese en modo protegido normal. El sistema operativo debe soportar este modo de operación para permitir su configuración en el momento de ejecución (run-time).

2.2 Manejador XMS HIMEM.SYS

Habiendo ya conocido en detalle las capacidades y formas de direccionamiento, y los modos de procesamiento de los Intel 286 y 386, podemos ahora pasar a echar un vistazo al manejo de memoria en Windows, el cual es realizado por el manejador HIMEM.SYS que viene con el ambiente. Este estudio específico a primera vista, nos dará también un enfoque global a lo que hoy en día realizan uno u otro manejador de memoria, en cuanto a administración y organización de ésta se refieren.

HIMEM.SYS es la versión que hace Microsoft del manejador XMS. Windows 3.0 lo usa para tener acceso a la memoria extendida, de forma ordenada, compatible con otros programas que puedan también usar la memoria extendida, aunque no con manejadores del tipo EMS.

Como HIMEM.SYS está presente, se puede, en principio, correr otros procesos de XMS de forma simultánea teniendo a Windows en la memoria, sin tener que arriesgar a Windows ni a los otros programas.

HIMEM.SYS controla al acceso a la memoria extendida (o, para ser más específicos, toda la memoria más allá de los 640k) de la misma forma en que los manejadores de EMS controlan la memoria expandida; brindando un juego de funciones para que los programas usen con llamadas lejanas al punto de entrada del manejador. Usted obtiene el punto de entrada ejecutando una interrupción 2Fh con el valor 4310h en AX, la dirección de 32 bits del punto de entrada del manejador se retorna en los registros ES:BX. Estas funciones de control, que se sumarizan en el cuadro, le permiten a los programas asignar bloques de memoria, copiar y leer datos de los mismos, y liberarlos cuando ya no se necesiten. Para entender las funciones necesita saber de qué forma HIMEM.SYS reconoce 3 tipos de bloques de memoria:

- Bloques Superiores de Memoria (UMB),con direcciones entre los 640k y el 1MB.

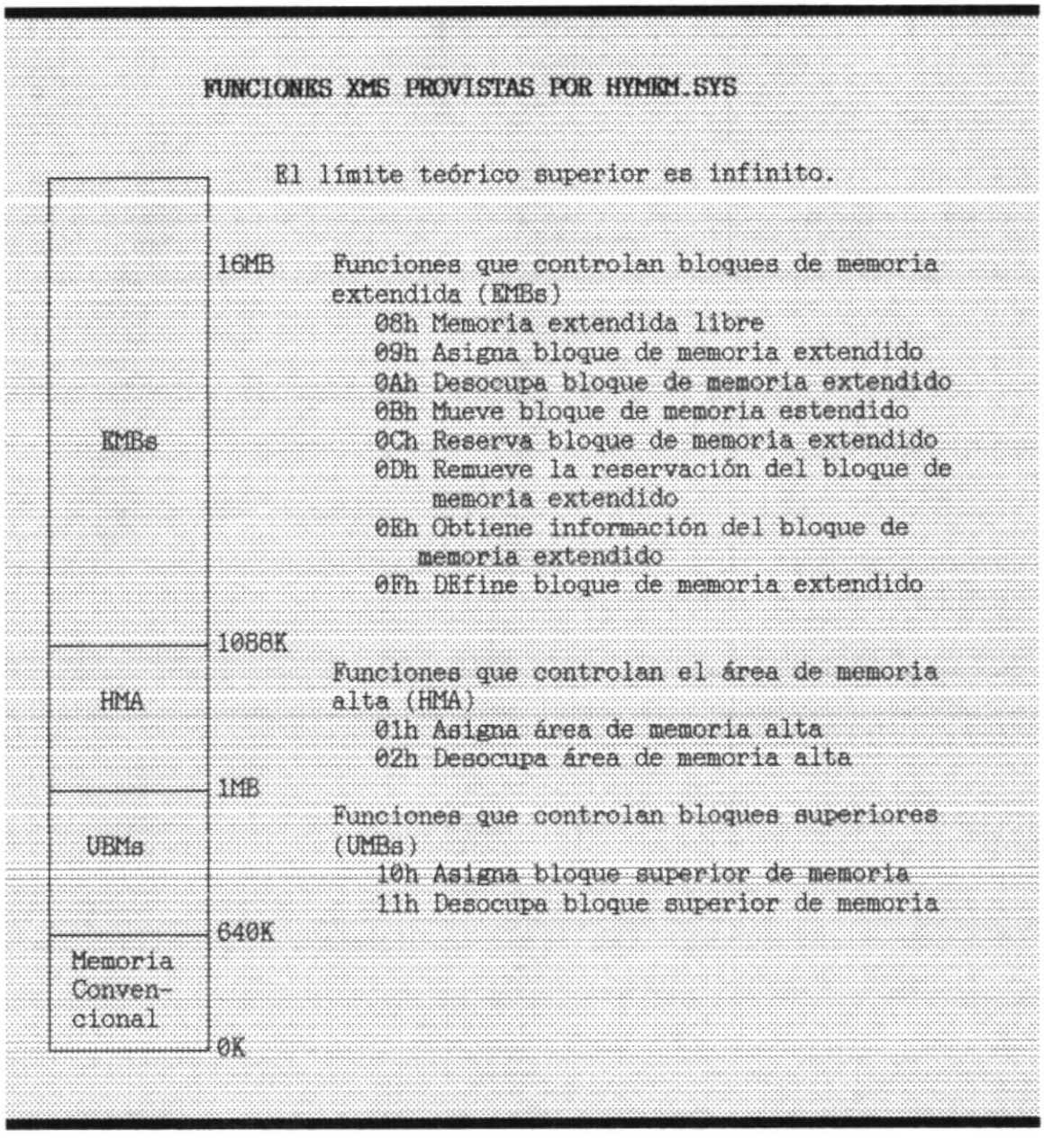
- Area de Memoria Alta(HMA), los primeros 64K de memoria (menos 16 bytes) más allá del 1MB.

- Bloques de Memoria Extendida (EMB) que pueden estar en cualquier parte de la memoria extendida más allá del HMA.

Los Bloques Superiores de memoria (UMB) entre los 640K y el 1MB se reservan para el uso del video y el BIOS del sistema, pero porciones del mismo quedan normalmente sin usarse. HIMEM.SYS se vale de otros servicios, conocidos como proveedores de UMB, para crear Bloques Superiores de Memoria para aprovechar estas regiones que no se usan de la memoria. Cuando estos bloques existen, se puede tener acceso a los mismos en modo real, ya que caen dentro de los límites de direccionado de modo real de la familia 80x86.

Las funciones 10h y 11h administran los accesos a los UMB de manera que dos procesos que aspiren al mismo bloque no se destruirán mutuamente al escribir sobre lo que el otro ya escribió. La función 10h asigna un bloque, la función 11h lo libera. Cuando se asigna un bloque a un programa, queda protegido contra otras aplicaciones que se adhieran a los protocolos XMS, porque el manejador no asignará el mismo bloque a dos programas.

HIMEM.SYS crea Bloques de Memoria Extendida (EMB) en la región de memoria extendida por encima de la marca de 1.088K (1024K más los 64K que ocupa el HMA). Las funciones con



números 08h hasta 0fh controlan el acceso a estos bloques. Por ejemplo, la función 0Bh, Mover bloque de Memoria Extendida, mueve datos desde la mamoria convencional a un EMB,desde un EMB hacia la memoria convencional, o entre dos EMBs. Un programa puede también tener acceso a la memoria extendida directamente

llamando a la función 0Ch para obtener una dirección lineal de 32 bits para el EMB, y simultáneamente reservando su localización en memoria. La función 0Dh es la contraparte de la 0Ch, y se usa para liberar un bloque que se haya reservado previamente.

HIMEM.SYS también provee control independiente del hardware sobre la línea de direcciones A20, que le permite a los procesos que corren en las 286s, 386s, y 486s tener acceso a los primeros 64K de memoria extendida (HMA) en modo real.

El Area de Memoria Alta (HMA) se distingue del resto de la memoria extendida en que una peculiaridad del diseño le permite a las 286s y 386s tener acceso a la misma en modo real. Cuál es el secreto? La activación selectiva de la línea de direcciones A20 de forma que se pueda formar una dirección de memoria de 21 bits. Recuerde que las 8088s y las 8086s tienen 20 líneas de direcciones (con números de A0 hasta A19) que les permite tener acceso hasta 1MB de RAM.

La 80286 y las CPUs superiores tienen más líneas de direcciones para alcanzar memorias aún más altas. Las extras se desactivan normalmente en el modo real de forma que la CPU se comporte como un 8088. Puede tratar de tener acceso a más de 1MB cargando la dirección FFFF:FFFF en un segmento: par de

desplazamiento, pero sino tiene un hardware que lo apoye, la CPU simplemente convertirá la dirección y tendrá acceso al fondo de la memoria, es decir la dirección 0000:FFEF.

Pero si tiene activada la línea A20, la dirección resultante no vivirá al principio. En cambio, llegará más allá de la marca de 1MB y le permitirá a la CPU tener tanto acceso al próximo 1MB de memoria extendida como le permita un registro de desplazamiento de 16 bits: otros 64K, es decir, la región conocida como HMA! Esto es importante porque el cambio de modo real a protegido y viceversa que normalmente se requiere para ganar acceso a la memoria extendida, consume mucho tiempo, especialmente en las máquinas 286, que deben reinicializarse a través del controlador de teclado (un proceso que puede requerir hasta varios milisegundos, una eternidad para una computadora). Aún más, aún un cambio temporal hacia el modo protegido tiene implicaciones más profundas. Un programa no puede manejar las interrupciones normales del hardware cuando se cambia al modo protegido, de forma que deben desactivarse las interrupciones. Sin embargo, esto significa que su programa puede perder ciertas interrrupciones si pasa al modo protegido y regresa.

La línea A20 alivia algo los límites de 1MB de memoria del modo real, y el arbitraje que hace el manejador XMS de la

línea A20 asegura que dos programas que sigan al XMS y que aspiran a la misma memoria no se molestarán mutuamente.

HIMEM.SYS brinda cinco funciones, con números 03h hasta 07h, para manejar la línea A20. Los programas que han pedido y han recibido asignaciones de HMA usan la activación / desactivación *Global*. Los programas que no tienen HMA usan activación / desactivación *Local*. Cada programa es responsable de activar la línea A20 antes de tener acceso al HMA y de restaurar el estado original de la línea (que puede obtenerse con la función 07h) después de tener el acceso.

Se instala HIMEM.SYS con un comando DEVICE= en el CONFIG.SYS. El mismo toma dos parámetros:

- /HMAMIN = n, que especifica la cantidad mínima de memoria (en kilobytes) de HMA que un programa puede pedir. El rango de valores legítimos es de 0 a 63; el valor prefijado es 0.

- /NUMHANDLES = n, que especifica el número máximo de asideros de archivo de bloques de memoria extendida que pueden asignarse en el sistema simultáneamente. El rango válido va desde 0 hasta 128, el valor prefijado es 0.

HMAMIN ayuda a hacer más eficiente el uso del HMA. Siendo algo menor que 64K, solamente puede asignar la HMA como una unidad; no puede dividirse en pequeños pedacitos como el resto de la memoria extendida. Si un programa necesita 4K de memoria de HMA y la pide antes que un programa que necesita 40K lo haga, se negará el acceso al segundo programa a menos que HMAMIN se fije a 5 o un número mayor.

Se conoce con antelación cuánta memoria HMA necesitará un cierto programa, puede hacerse un uso juicioso de la memoria ajustando HMAMIN.

NUMHANDLES, el segundo parámetro, especifica cuánto espacio debe reservar HIMEM.SYS para asideros de archivo internamente. Cada asidero adicional requiere 6 bytes de memoria convencional. Como regla general, se separa espacio para 16 asideros por cada megabyte de memoria extendida en el sistema. Así, puede poner en uso toda la memoria extendida en trozos tan pequeños como de 64K.

Tanto HMAMIN con NUMHANDLES se definen en la versión 2.0 de la especificación XMS, con fecha Agosto 23, 1988. La versión de HIMEM.SYS que viene con Windows 3.0 acepta dos argumentos adicionales en la línea DEVICE=, y estos son /SHADOW : ON ¦ OFF.

Muchas PCs proveen una opción conocida como "copia de ROM a RAM" (shadowing) mediante la cual, el ROM que es notable por ser lento, se copia a RAM y se ejecuta el código desde allí mucho más rápidamente. Por lo general, HIMEM.SYS trata de activar la copia de ROM a RAM en estas PCs para liberar memoria adicional extendida. La opción /SHADOW le deja instruir explícitamente a HIMEM.SYS para que active o desactive la copia. Esta opción no tiene efecto negativo alguno en algunos sistemas, ya que muchas configuraciones de *hardware* no permiten que se active y desactive la copia de ROM a RAM bajo el control del *software*.

Puede usarse HIMEM.SYS para cargar TSRs y manejadores de dispositivos en la memoria alta? No, al menos HIMEM.SYS no puede hacerlo por sí solo.

La especificación XMS no provee funciones de control como las que usan los programas tales como 386MAX y QEMM386 para meter programas en la memoria alta.

La mejor forma de sacar los programas de la memoria convencional es todavía un administrador de memoria de 386 o, en las 286s con memoria expandida, un servicio similar al QRAM de Quarterdeck Systems.

2.3 Inconvenientes del Windows

Aunque actualmente para la mayoría de los usuarios este ambiente gráfico resulta magnífico (de allí su popularidad), vale hacer un análisis, en base a lo visto previamente, de las bien escondidas desventajas (al menos para un simple usuario) que el Windows tiene. Para esto repasaremos lo ocurrido los últimos doce años, las tendencias futuras y el porqué Windows no debe ser visto como la plataforma de los próximos años.

Las Lecciones del Pasado

En algún momento en los próximos años la industria se encontrará cambiando de la arquitectura de 16 bits a la arquitectura de 32 bits. Para las computadoras personales basadas en los procesadores de Intel, esto significa usar los chips 386 y 486.

Para una industria tan joven como esta, es frecuentemente difícil identificar las tendencias a largo plazo. Pero vamos a tratar de perfilar cómo el uso de memoria ha incrementado en los últimos diez años. He notado la siguiente tendencia:

1980 - 64 Kilobytes

1985 - 640 Kilobytes

1990 - 6 Megabytes

En 1980, las máquinas que usaban los procesadores 8080 y Z80 ejecutando CP/M estaban limitadas a 64K de memoria. En el año siguiente, las primeras IBM PCs acomodaban 64K en la tarjeta madre. En 1985, los 640K se habían convertido en algo común para las máquinas del tipo AT. Y en 1990, la mayoría de las máquinas 386 tiene espacio para 6MB en la tarjeta madre, aunque 8MB es probablemente más común.

Los números también representan un nivel cómodo de memoria para ejecutar los programas más avanzados para la PC. En 1985 los problemas de "limitaciones de RAM" se hicieron importantes cuando los usuarios de Lotus 1-2-3 creaban hojas de cálculo grandes que topaban con la barrera de 640K. La especificación de memoria expandida de Lotus/Intel/Microsoft (LIMS) se desarrolló para ayudar a aliviar este problema. Seis megabytes es un nivel cómodo para correr OS/2 1.2 o Windows 3.0, aunque cuatro megabytes parecen suficientes para Windows hoy.

Los números muestran una tendencia exponencial increíble: en los últimos diez años, el uso de memoria se ha incrementado por un factor de diez cada cinco años.

Extrapolando a 1975 obtenemos un resultado de 6K. Esto sería poco tiempo después del nacimiento de la industria de la computadora personal así que ese resultado puede que no sea muy

preciso. Pero, en 1975 una tarjeta de memoria de 4K era algo especial, así que la aproximación es más o menos correcta.

Extrapolando a 1995 obtenemos un número casi increíble de 64MB de memoria (redondeando a la potencia de dos más cercana). Y aunque este número puede parecer increíble, en 1985 la idea de acomodar 6 u 8 megabytes era igualmente increíble.

De 8 a 16 a 32

Las computadoras personales de la última mitad de los 1970s se basaban en microprocesadores de 8 bits, como el Intel 8080. Al principio de los 1980s la industria comenzó a cambiar de una arquitectura de 8 bits a 16 bits. Esta arquitectura de 16 bits ha dominado la industria desde la introducción en 1981 de la IBM PC al presente.

Estamos presenciando ahora una transición hacia las arquitecturas de 32 bits en las computadoras personales, caracterizada por sistemas operativos y aplicaciones que aprovechan el 386 de Intel. La tendencia que vemos es la siguiente:

1970s - 8 bits

1980s - 16 bits

1990s - 32 bits

Un microprocesador de 32 bits tiene las mismas ventajas sobre un procesador de 16 bits que un chip de 16 bits tiene sobre un chip de 8 bits. El incremento en el tamaño de los registros internos le permite al chip manipular segmentos grandes de información más rápidamente.

Si tenemos razón acerca del uso de 64MB de memoria para el año 1995, entonces los procesadores de 32 bits como el 386 serán absolutamente necesarios. El 286 sólo tiene acceso a 16MB de memoria física en el modo protegido, mientras que el 386 tiene 32 líneas de direcciones y puede tener acceso a 4GB de memoria física. Esto es aún un margen suficiente para los 640MB de memoria que se usarían en el año 2000 si la tendencia de memoria continúa.

Incluso si el uso de memoria se detiene en 16MB en los próximos 5 años más o menos, el 386 todavía ofrece ventajas de acceso a memoria. El 286 usa un plan de memoria segmentado que lo hace ineficiente y tosco con bloques de memoria mayores que 64K. Aun hoy, la mayoría de las aplicaciones tienen algunas limitaciones de 64K. El 386 soporta un modelo de memoria continuo, sin segmentos, que simplifica el trabajo con bloques de memoria grandes y mejora el rendimiento.

La mitad de la solución de 32 bits

Ya tenemos el hardware para la computación de 32 bits en las máquinas 386 y 486. Para aprovechar completamente los 32 bits, incluyendo el acceso a memoria de 32 bits, el CPU debe cambiarse al modo protegido. Esto no requiere necesariamente un sistema operativo de 32 bits. Las aplicaciones se pueden desarrollar con varias tecnologías de DOS-Extender (ampliador de DOS) que le permiten correr en el modo protegido de 32 bits y usar el espacio de acceso continuo de 32 bits bajo DOS. Esto puede ser muy transparente para el usuario de la aplicación.

Pero la solución del DOS-Extender no es completamente adecuada. El programa todavía está operando en el modo real de DOS y el BIOS. Las llamadas de funciones de DOS y del BIOS requieren una dirección compuesta de segmentos de 16 bits, a diferencia de las direcciones de 32 bits del modo continuo que las aplicaciones usan para tener acceso al espacio de direcciones. El chip también debe cambiarse al modo real para estas llamadas de funciones.

Además se crean complicaciones cuando tales programas operan en un entorno de multitareas. Esto es así porque las propias aplicaciones, y no un sistema operativo, son las que administran la memoria del sistema y conmutan entre el modo

protegido y el real. Las soluciones para este problema requiere la Interfaz Virtual para Control de Programa (VCPI) o la Interfaz para Modo Protegido de DOS (DPMI), que Microsoft ha desarrollado para Windows 3.0 para corregir las deficiencias de VCPI. Estas interfaces son un tipo de adiciones de software al sistema que ayudan a proteger los programas de DOS-Extender para que no interfieran entre ellos.

No cabe imaginarse el diseño un sistema operativo en el cual el control del modo protegido y la administración de la memoria del sistema se puede manejar desde otro lugar que no sea el núcleo del sistema operativo! Pero DOS se ha hecho tan indispensable y al mismo tiempo tan inadecuado que los remedios como DOS-Extenders y esas interfaces de modos protegidos se consideran,de todas meneras, necesarios.

Una porción de la mejora en rendimiento de un sistema de 32 bits viene del hardware, aún si el software y el sistema operativo todavía están en código de 16 bits. Se obtienen otras mejoras en rendimiento usando aplicaciones de 32 bits. Pero para usar el hardware de 32 bits óptimamente, necesitamos un verdadero sistema operativo de 32 bits con una interfaz de programa de aplicación (API) de 32 bits ejecutando aplicaciones de 32 bits.

Con el aumento en popularidad de las interfaces gráficas, estamos viendo APIs que consisten de cientos de llamadas de funciones. Un API de 16 bits es un cuello de botella, en cuanto al rendimiento, para las aplicaciones de 32 bits. El sistema operativo y las aplicaciones no están ni siquiera hablando el mismo idioma con respecto a la memoria.

El Modo Protegido Incompleto

Windows 3.0 puede aprovechar los 16MB de memoria física, pero su implementación del modo protegido es muy débil e incompleta.

Primero, cuando corre en un 386, Windows 3.0 usa un modo protegido compatible con el 286. No puede usar más de 16MB de memoria física.

Segundo, Windows 3.0 y todas las aplicaciones de Windows comparten la misma Tabla de Descriptor Local (LDT). Esto significa que Windows 3.0 no puede ofrecer una protección adecuada entre las áreas de datos de otros procesos. Un programa de Windows con errores podría corromper las áreas de datos de otros programas de Windows. Si esto pasa alguna vez, puede que lo note o no.

Tercero, Windows 3.0 todavía ejecuta encima de DOS en el modo real. Es fácil escribir un programa de Windows que pase un puntero equivocado a DOS y trabe el sistema. Es fácil imaginarse un error en el programa que haga que esto pase. En entornos de una sola tarea, un programa que traba al sistema quizás destruye un archivo de datos sin guardar. Pero si un programa traba a un entorno completo de multitareas, podría destruir media docena de archivos de datos sin guardar.

Un programa de Windows no puede usar el modelo de memoria de 32 bits exclusivamente, porque el código de 32 bits no puede hacer llamadas a las funciones de Windows o de DOS. El programa tendría que contener por lo menos dos segmentos de código, uno de interfaz con Windows y con DOS, y otro que trabaje con la memoria de 32 bits. Las direcciones que se pasan entre los dos segmentos necesitan convertirse de desplazamientos de 16 bits y segmentos a direcciones continuas de 32 bits.

Y la sección de pila de la memoria del programa (stack) es otro problema. Una aplicación necesitaría mantener dos pilas (una para el código de 16 bits y otra para el código de 32 bits) o dos punteros de pilas (uno de 16 bits y uno de 32 bits). Obviamente, esto requiere un poco de código en ensamblador y un trazado de errores muy delicado. Comparemos esto con la facilidad de escribir código de 32 bits para OS/2

2.0, donde asignar un bloque de memoria de 32 bits puede ser tan simple como una llamada a la función malloc.

El uso de la memoria es fundamental para un programa de computadora. Hay algo seriamente malo cuando los accesos a memoria se hacen difíciles. Todos los programadores que trabajan con los procesadores de Intel entre el 8086 y el 286 han maldecido el límite del segmento 64K. Ahora tenemos la oprtunidad de abandonar los segmentos y cambiar al modelo de memoria continuo de 32 bits. Pero si usar memoria de 32 bits es tan difícil como lo es en Windows 3.0, continuaremos maldiciendo al sistema operativo.

Al promover tanto a Windows 3.0, Microsoft está jugando peligrosamente, pretendiendo que un sistema operativo frágil y escaso como DOS y un sistema de interfaz gráfico de 16 bits como Windows 3.0 sean adecuados para largo plazo.

CAPITULO III

BASE DE DATOS: FOXPRO

La tercera y última herramienta que nos faltaba por estudiar, es la base de datos. De los demás elementos del rompecabezas, se podría decir que este elemento es el que menos brillo le da. Por qué? Porque las bases de datos, y singularmente ésta, son muy conocidas y utilizadas a todo nivel. A pesar de que aquí se hablará de la última versión, que trae muchas y novedosas mejoras, el aporte investigativo-científico al proyecto no es tan remarcable como los anteriores.

Sin embargo, vale la pena echar un vistazo a estas grandes mejoras que el Foxpro 2.0 nos ofrece.

3.1 El Producto FoxPro 2.0

La versión 2.0 como las versiones 1.x se ofrecen para un simple usuario o para multiusuarios. Sin embargo, el software de Fox ahora provee dos versiones en cada paquete:la standard y la extendida. De este modo el paquete para un simple usuario consiste de programas FoxPro (standard) y FoxProX (extendidos);

mientras que para multiusuarios o paquetes LAN consiste de programas FoxProL (standard) y FoxProLX (extendidos).

La versión estandar puede ser usada en una típica máquina AT con una memoria de 640K. La versión extendida está diseñada para máquinas que tienen memoria extendida y cuyo procesador sea 80386 de Intel. La versión extendida deberá ser usada cuando se procesan bases de datos cuyos números de registros sobrepasan el millón; o cuando se necesitan aproximadamente 65000 variables de memoria, 65000 elementos por arreglo y 125 áreas de trabajo. La versión extendida usa todos las ventajas de la memoria extendida, permitiendo así no tener un límite real sobre el número de indices, ventanas y browse en sesiones. La máxima longitud de un string en la versión extendida es 2 gigabytes.

FoxPro 2.0 tiene una memoria base de cerca de 280K RAM, considerablemente menor que la versión 1.x. Consecuentemente las aplicaciones de la versión 1.x tienen problemas para ejecutarse sobre redes y trabajan mucho mejor bajo la versión 2.0. Cuando se invoca a FoxPro 1.X siempre empieza creando un archivo de 400K que fue automáticamente borrado cuando el programa fue terminado normalmente. Esto representa un problema para los administradores de redes ya que significa proveer suficiente espacio en el disco para cada usuario de FoxPro. La

versión 2.0 no requiere los 400K para el archivo de inicio, por lo tanto ésta puede ser iniciada al igual que si se trabaja con un drive de disco flexible de 360K y el path de la versión 2.0 sobre el disco duro.

La versión 2.0 usa un nuevo esquema de manejador de memoria "cargador de segmento", en lugar de tres pequeñas áreas de overlay de la versión 1.X. Ya que la versión 2.0 evita la degradación en la perfomance de los resultados haciendo swap in y swap out en las áreas de overlay, las aplicaciones de FoxPro 2.0 se ejecutan más rápido particularmente sobre una red.

3.1.1 Arquitectura Abierta

FoxPro 2.0 adopta una filosofía de arquitectura abierta que es reflejo en gran parte de su nueva imagen. Por ejemplo, los desarrolladores tienen acceso al FoxPro como instrumento a través de la rutina externa APIs (Application Program Interface). El producto también tiene nuevos comandos y funciones que permiten a los desarrolladores la manipulación de su sistema de ventanas y menús en orden que permiten ver cómo el FoxPro crea aplicaciones. Mientras que la versión 1.X tiene archivos inescrutables en formato binario, la versión 2.0 tiene archivos en el familiar formato .dbf . Por ejemplo FoxPro 1.X

tiene el archivo de ayuda como una base de datos, permitiéndole crear a usted fácilmente archivos de ayuda para sus aplicaciones. Ahora FoxPro 2.0 tiene reportes, menús y formatos de pantalla en archivos .dbf que es muy conveniente ya que se pueden manipular como cualquier otra base de datos.

3.1.2 Tecnología Rushmore

FoxPro 2.0 usa una técnica de acceso de datos llamada "Rushmore", la cual permite que registros de base de datos sean recuperados con una velocidad comparable a la de los mainframes. Rushmore permite a FoxPro tener aplicaciones que manejen base de datos con un millón de registros confortablemente. Esto depende de la expresión FOR para búsqueda en base de datos. Para tomar ventaja de Rushmore, las bases de datos deben estar indexadas y la sentencia FOR debe contener una básica expresión optimizable (BOE). Una BOE puede tomar la siguiente forma:

<expr indice> <operador booleano> <expr constante>

donde <expr indice> debe contener la expresión con la cual el indice fue creado. La expresión indice no puede contener un alias. El indice puede ser regular (.idx) o un indice compacto. Si la base de datos contiene los siguientes campos: Nombre, Dirección, Ciudad, Provincia y Comentarios; y ésta es indexada

sobre los primeros cuatro campos, las siguientes expresiones son BOEs:

```
Nombre = "Ruiz"
Provincia <> "Guayas"
UPPER (Ciudad) = "MANTA"
```

Las siguientes expresiones no son BOEs:

```
UPPER (Comentarios) = "NOTAS"          && El campo Comentarios no está
                                          indexado.
"Manta" $ Provincia                    && El operador $ no es booleano.
SUBSTR (A ->Direccion, 1, 4) = "Casa"  && Contiene un alias.
```

Las expresiones BOEs pueden ser combinadas con otras usando los operadores lógicos AND, OR o NOT para formar sentencias FOR complejas que son optimizables. BOEs pueden ser combinadas con no BOEs usando el operador AND para producir expresiones parcialmente optimizables.

La tecnología Rushmore es automáticamente usada en el nuevo comando SQL SELECT. FoxPro desabilita Rushmore siempre que una sentencia WHILE es incluída en un comando, por ejemplo LIST, SCAN y BROWSE; de otro modo se usa Rushmore. FoxPro estandar no puede usar Rushmore cuando el número total de

registros en la base de datos abierta exceda al medio millón de registros; en estos casos se deberá usar la versión extendida.

Se puede desabilitar Rushmore de dos maneras; la primera usando la sentencia NOOPTIMIZE de un comando que puede usar la expresión FOR, y la segunda usando el comando SET OPTIMIZE OFF. El comando SET OPTIMIZE ON habilita Rushmore. Usted querrá deshabilitar Rushmore siempre que el potencial optimizable del comando modifique el indice clave en una sentencia FOR. En tal situación si usted no deshabilita Rushmore, éste no tendrá la información más actual de la base de datos.

3.1.3 Compilador y Manejador de Proyectos

FoxPro 2.0 tiene un compilador .exe haciendo que permite que se puedan distribuir aplicaciones sin empaquetar con el módulo familiar Runtime de FoxPro. El compilador es realmente justo una facilidad del manejador de proyectos del FoxPro, el cual es similar al de los lenguajes de programación convencionales como el C. Un archivo proyecto en FoxPro es un archivo especial que mantiene el trayecto de todos los programas y archivos de datos requeridos por una aplicación, incluyendo programas, pantallas, menus, librerías, reportes,

etiquetas, queries, formato de archivos, y todas las conexiones y dependencias entre los archivos.

Cuando FoxPro compila una aplicación, el archivo proyecto ayuda que todos los módulos compilados en la aplicación estén basados en el más reciente código fuente. La aplicación creada de un archivo proyecto incluye código generado de pantallas y menus.

El archivo proyecto es así mismo una base de datos FoxPro (.pjx) con un archivo memo asociado (.pjt). Usted normalmente crea un proyecto interactuando a través de diálogos invocados por el uso del comando CREATE/MODIFY PROJECT. Alternativamente usted puede crear un proyecto desde la línea de comandos o en un programa usando el siguiente comando:

BUILD PROJECT <prjfile> [FROM <mainpgm> [,<pgm> ¦ <lib>[,...]]]

FoxPro crea el proyecto procesando uno o más programas y archivos de librerias, y por búsquedas de referencias a otros programas y librerias, lo cual FoxPro procesa satisfactoriamente.

Puede usar el nuevo comando EXTERNAL para incluir archivos y para determinar referencias no definidas en un proyecto

creado por el manejador de proyectos. El comando EXTERNAL es también usado para alertar al manejador de proyectos de un nombre de archivo contenido en una expresión o macro como válidas para arreglos de nombres que están creados en otros procedimientos o UDF (User Defined Functions).

Con FoxPro y su rutina externa APIs, usted puede escribir sus propias funciones y extensiones para el lenguaje y tenerlas congregadas dentro de una librería externa. El nuevo comando SET LIBRARY TO <lib> realiza las funciones en la librería externa aprovechando esta ventaja del FoxPro para la aplicación.
Las funciones externas destinadas para inclusión en la librería pueden ser escritas en C o en Assembler y entonces compiladas en archivos objetos.

Después de crear un proyecto en FoxPro, usted tiene la opción de crear una aplicación (.fxp) para distribución runtime, similar a la que ofrece FoxPro 1.x; o usted puede crear una standalone .exe que no requiere el módulo Runtime de FoxPro.

Para crear el .exe requiere el comando BUILD EXE <exefile> FROM <prjfile>. El comando complementario BUILD APP crea la aplicación en la forma .fxp .

3.1.4 Las Rutinas Externas APIs

La Interfase para Programas de Aplicación (API: Application Program Interface) del Foxpro le permite extender las capacidades del lenguaje y su interfase de usuario grandemente. Con esto se pueden ejecutar operaciones avanzadas, incluyendo manejador de eventos de FoxPro, procesar sentencias de FoxPro, y manipulación de memoria, de archivos memos, base de datos y ventanas.

La libreria API contiene un extenso conjunto de rutinas, entre las cuales están:

- Creación, eliminación, y cambio de valores de las memvars.
- El poder de FoxPro para optimizar archivos de entrada/salida.
- Acceso a la base de datos, incluyendo retorno de listados de registros de la última versión leída del disco.
- Permite manejadores de eventos para interceptar eventos FoxPro.
- Control de ventanas y flujo de salidas.
- Creación o eliminación de menúes; o cambio del sistema de menúes para quizás crear un nuevo conjunto de accesorios al disco.
- Intercepción y reportes de condición de errores.

Rutinas externas reciben un parámetro de FoxPro que es un puntero para el bloque que contiene el número de parámetros suministrados y dichos parámetros. Los programas pueden pasar parámetros por referencia o por valor, los programas pueden retornar resultados al FoxPro usando una variedad de rutinas API, incluyendo una llamada _RetVal(), la cual puede ser usada para retornar un string con nulos o para retornar cualquier tipo de datos excepto el tipo memo.

Para un usuario de FoxPro una rutina externa se ve como una función creada en FoxPro o como un comando, excepto que éste no puede ser abreviado a cuatro caracteres.

Qué pasa si una rutina externa tiene el mismo nombre que una función creada en FoxPro, o que un arreglo de memoria, o que una función de definición de usuario (UDF) ? La prioridad de evaluación entre función o rutina es tal que una rutina externa tiene prioridad sobre un nombre similar de UDF, pero ésta no tiene prioridad sobre una función creada o un arreglo.

Las rutinas externas pueden ser codificadas solamente en Assembler 8086 o en Watcom C, y están sujetas a estrictas reglas de procedimientos en cuanto a protección de memoria. Las rutinas son purgables o bloqueadas, dependiendo de si están localizadas en el área de overlay o en el segmento raíz.

3.1.5 SQL SELECT y RQBE

Otra nueva faceta de FoxPro 2.0 es el comando SQL SELECT, el cual no debe ser confundido con el comando SELECT <workarea> usado para recuperar datos de una o más bases de datos. SQL SELECT es un comando muy poderoso, ya que le permite recuperar datos sin que se requiera conocer la base de datos en el área de trabajo que contiene el dato. Un comando SQL SELECT reemplaza a una serie completa de comandos en FoxPro o procedimientos para reunir, ordenar y mostrar la información deseada.

Al contrario que dBASE IV, no se necesita primero ejecutar SET SQL ON para usar SQL SELECT. Además de esto si no se quiere formular el comando Select completamente, se puede desde la ventana de comandos o desde el programa, recuperar el mismo conjunto de datos interactivamente usando la nueva facilidad de FoxPro: RQBE (Relational Query-By-Example). Con RQBE se define el dato que se quiere desde el prompt, picklists y cuadros de chequeo. FoxPro entonces formula el apropiado Select query y lo ejecuta. Se pueden grabar queries a un archivo del tipo .qpr, y se pueden direccionar los resultados del query hacia un archivo, impresora o hacia una tabla.

3.2 Justificación

Realmemte se pueden encontrar muchas de las justificaciones para la selección de la base de datos Foxpro de las que hablaremos en la sección anterior. Con sobra de merecimientos el Foxpro en estos últimos años ha demostrado que está en la vanguardia de las bases de datos, habiendo inclusive desplazado al otrora famoso Dbase.

El Foxpro es de sencillo manejo, posee un seudo lenguaje extremadamente eficiente y práctico. Sus cualidades en cuanto a accesos a las bases de datos, generación de reportes, indexamiento de claves, manejo de archivos, etc. son muy buenas.

Pero además de esto, se escogió trabajar con Foxpro 2.0 por dos razones fundamentales: El hecho de ya poder crear archivos .EXE de las aplicaciones y el de poder usar comandos SQL. Dos motivos más que suficientes, que dejaban a un lado los demás competidores.

Informix es muy buena para sistemas en Unix, pero para sistemas DOS tenemos nuestras dudas. Por experiencias anteriores hemos comprobado que el Informix consume muchos recursos de la máquina, comenzando con la infinidad de archivos

que instala para administrar las bases, y que crea cada vez que se crea una nueva. En DOS se hace una eternidad esperar a que termine de compilar cada módulo y se podría decir que es un ambiente muy poco amigable (ni siquiera tiene un depurador). Encontramos además un obstáculo infranqueble, que para Informix se necesita cargar primeramente el manejador de SQL el STARTSQL, el cual ocupa mucho espacio en memoria. Pensamos que esto podría traer consigo problemas al momento de la ejecución de todo el sistema.

Existen muchas otras bases de datos en el mercado, unas enormemente poderosas (como ORACLE), otras sin las herramientas necesarias, pero en general, que no se ajustaban a las necesidades del sistema, o muy complejas (demandaban muchos recursos) o muy sencillas.

Se puede concluir entonces que Foxpro era la base de datos más adecuada para nuestro proyecto.

CAPITULO IV
INTERFASE GRAFICADOR – BASE DE DATOS

En este capítulo nos dedicaremos a explicar la construcción de la interfase entre el graficador AutoCAD y la base de datos Foxproln. Aquí pondremos en práctica toda la teoría vista an los anteriores capítulos. Expondremos de una forma sencilla pero completa las herramientas y los procesos involucrados en el funcionamiento de la interfase.

4.1 Definición

Empecemos definiendo lo que es la interfase y qué es lo que debe hacer. La interfase es un conjunto de programas que nos permiten comunicarnos desde el graficador, en este caso el AutoCAD, con una base de datos, Foxproln. El enlace es transparente para el usuario, de tal forma que, trabajando normalmente desde el graficador el usuario se comunica directamente con la base de datos, como si se tratase de otro comando o rutina propia del graficador.

La interfase resultante es sencilla de manejar, muy flexible; tiene todas las funciones básicas que corresponden al

manejo de registros (añadir, modificar, eliminar, etc.) e interactúa con el AutoCAD eficazmente.

4.2 Requerimientos de la Interfase

Debemos seguir con las necesidades de hardware y software de la interfase, porque obviamante es lo primero que debemos tener en cuenta si queremos instalar cualquier programa.

Aquí simplemente nombraremos estas necesidades y posteriormente iremos analizándolas y justificándolas como corresponde.

Las necesidades de hardware son las siguientes:

- Un computador 386 con coprocesador matemático
- 4MB de memoria instalada (mínimo)
- 4MB de espacio libre en el disco duro (mínimo)
- Un mouse o ratón

Las necesidades de software son:

- Windows 3.0
- AutoCAD 10
- Foxpro 2.0

Un requerimiento adicional sería el de habilitar un disco virtual en la memoria para acelerar los procesos de comunicación de la interfase. Es suficiente definir este disco de unos 10 Kbytes.

4.3 Bosquejo del funcionamiento

La interfase está constituida básicamente de tres partes:

- El conjunto de programas AutoLISP que dentro del AutoCAD manejan los datos de las entidades y los colocan en el medio de enlace.
- El medio de enlace, que en sí son un conjunto de archivos con tareas específicas.
- El programa escrito en el seudo lenguaje del Foxpro que accesa a las bases de datos con la información leída en el medio de enlace.

El conjunto de programas AutoLISP incluyen una serie de rutinas para el manejo de memoria, para la creación y captación de la entidad gráfica como bloque de datos (un bloque de datos es aquella entidad gráfica en AutoCAD que tiene relacionada información en una base de datos), para el control de errores, para la introducción de datos, para hacer consultas, etc.

El medio de enlace, los archivos, unos tienen la función de banderas, es decir, sirven para comunicar a cualquiera de los dos programas cuando pueden escribir o leer los datos pasados por el otro programa; los otros tienen la función, obviamente, de transportar estos datos. Los archivos "banderas" además están sincronizados para evitar que los dos programas intenten accesar los archivos de datos al mismo tiempo. Por otro lado, están los archivos de descripción, los cuales son sólo leídos por los programas AutoLISP para obtener información de las bases de datos creadas.

El programa Foxpro (de ahora en adelante lo denominaremos así) lee los parámetros recibidos en el archivo de datos y ejecuta una acción determinada. Existen rutinas que van desde la más complejas como, la creación de una base dentro del mismo programa, la construcción de una selección de registros (query), la reorganización total de una base, hasta las más sencillas como la adición, la modificación o eliminación de un registro.

Pero como conseguimos la transparencia al usuario con dos programas que corren en DOS? Pues bien, la solución la vimos en los requerimientos: Windows. Sí, Windows 3.0 nos da la posibilidad de correr dos procesos DOS al mismo tiempo. Colocando al AutoCAD como proceso principal en la pantalla y al

programa Foxpro ejecutándose en la parte posterior (nunca lo vemos), hace que siempre parezca que sólo estamos trabajando dentro del AutoCAD, consiguiendo un efecto de "transparencia total" (de uso y de visualización) al usuario.

Por lo anterior, el requerimiento de memoria de 4MB se hace imprescindible, se debe cargar Windows y correr los dos procesos, todo lo cual consume muchos recursos. Como vimos en un capítulo previo, Windows a su vez, debe ser arrancado en la modalidad de 386 agrandado (enhanced mode), para que soporte multitarea de procesos DOS.

Analizaremos más adelante en detalle cada parte de la interfase.

4.4 Estructuras de Bases y Archivos

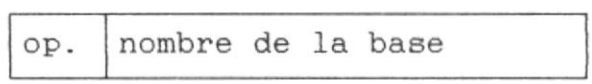
Los programas de la interfase se valen de bases de datos y archivos predeterminados para controlar, administrar y describir el ambiente del sistema.

Para el traspaso y coordinación de datos entre el AutoCAD y el Foxpro se usan un total de cinco archivos: inter01.txt,

inter02.txt, inter03.txt, $_acad.txt y $_fpro.txt. Los cinco tienen tareas fijas, explicadas a continuación:

- inter01.txt: Es el archivo bandera que le indica al programa Foxpro que existen datos para ser procesados. El archivo es creado cada vez que el programa AutoLISP requiere comunicarse con el programa Foxpro. Este lee los datos y se encarga de borrarlo. Físicamente es un archivo con 0 bytes en el directorio.

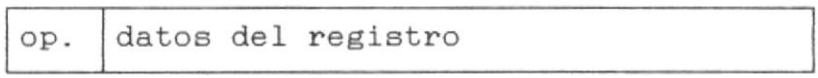
- inter02.txt: Es un archivo que le indica al programa Foxpro que operación a nivel de base de datos realizar y que base usar. Cuando el programa Foxpro detecta el archivo inter01.txt, lo primero que hace es leer el inter02.txt para cargar esta información. El archivo es reescrito si cambiamos de base o de operación. La estructura del archivo es la siguiente:



Operación: 0 = Crear
1 = Usar
2 = Eliminar
3 = Reorganizar
9 = Terminar

- inter03.txt: Es el archivo bandera que le indica al programa AutoLISP que el programa Foxpro le ha retornado la información consultada. El archivo es creado cada vez que el programa Foxpro ha procesado los datos y necesita darle una "respuesta" al programa AutoLISP. Este lee la información y se encarga de borrarlo. Físicamente es un archivo con 0 bytes en el directorio.

- $_acad.txt: Este archivo corresponde a los llamados archivos de datos. Es el que lleva los datos en forma de registro de una entidad ingresados en el ambiente AutoCAD al registro en sí de una base de datos,es decir, el enlace AutoCAD -> Foxpro . Su primer byte nos indica la operación que a nivel de registro se quiere hacer y en los restantes están los datos, cuya longitud dependerá del total de bytes por registro de la base a usar. En el programa AutoLISP se crea este archivo primero, y después se crea el inter01.txt, para que lo reciba el programa Foxpro. El $_acad.txt se reescribe las veces necesarias y sólo se lo borra al terminar la sesión de AutoCAD. El esquema del archivo es:



Operación: 0 = Consultar
1 = Añadir
2 = Modificar
3 = Eliminar
4 = Localizar

- $_fpro.txt: Es el segundo archivo de datos y hace la comunicación Foxpro -> AutoCAD, es decir, la información accesada por el programa Foxpro es devuelta al AutoCAD en este archivo. El $_fpro.txt se sincroniza junto con el inter03.txt para indicar al programa AutoLISP que ya puede leer la "contestación". La estructura del archivo sería la misma de un registro de la base de datos previamente accesada, por lo que dependería de que bloque de datos estemos usando.

Además de estos primeros cinco archivos, encontramos otros que tienen la misión de informar y controlar al sistema de enlace y los objetos por él utilizados. Así tenemos una base de datos bases.dbf para guardar información de todos los bloques, archivos de definiciones de bloques y archivos de errores.

- bases.dbf: Es un archivo de base de datos donde se almacena la información relacionada con el nombre de cada bloque de datos creado por el sistema, su gráfico y su base de datos asociados, y su fecha de creación. Por medio de esta base, el programa Foxpro averigua que base de datos usar, y también con ella se genera un archivo, el $_bases.df, para que el programa AutoLISP lo accese y consulte directamente. Los dos archivos deben ser permanentes para el sistema; nunca deben ser borrados. La estructura de bases.dbf es:

Campo	Tipo	Long	Dec
NOMBRE	Carac.	8	0
DETALLE	Carac.	30	0
BASE	Carac.	8	0
BLOQUE	Carac.	8	0
FECHA	Fecha	8	0

- ??????.df: Los archivos .df son los que tienen la definición de la estructura de la base de datos asociada al bloque de datos que se quiere usar. En ellos se encuentra una descripción detallada de los campos y sus longitudes. Debe existir un archivo de definiciones por cada bloque de datos en el sistema. Se crean al hacer un nuevo bloque dedatos y debenser también archivos permanentes.

- error.txt: Es el archivo primario de errores del sistema de enlace. Cada vez que se hace una comunicación entre los dos programas el AutoCAD lee este archivo para verificar que no haya habido errores. Está compuesto de dos registros, el primero es un número 0 o 1; 0 todo está bien, 1 hubo un error. Si es 1 entonces existe otro registro con un mensaje describiendo el error. Los errores que se manejan aquí son errores controlados, que no afectan la integridad del sistema. El archivo es automáticamente borrado al salir de la sesión.

- $_error.txt: Cuando por alguna razón se produce un error severo en el programa Foxpro y se interrumpe la comunicación, se genera este archivo. En él hallaremos una descripción detallada del error que no pudo ser controlado, el mensaje y el código del error, la línea de ejecución en que se cayó y el nombre del procedimiento o subrutina.

De los archivos mencionados, cuatro son creados en un disco virtual de memoria. Estos son: inter01.txt, inter02.txt, inter03.txt y error.txt. Con ello se mejora la rapidez de

acceso a los archivos y un mayor perfomance de los programas. Se podría también poner los demás archivos en el disco virtual, pero para ello necesitaríamos más memoria que los 4MB que pusimos como base.

4.5 Los Programas AutoLISP

Esta parte es sin duda la más compleja y larga de la interfase. En ella se encuentra el código que hace posible la delicada relación usuario - programa; y es que en el AutoCAD por medio del AutoLISP se introduce y se presenta toda la información hacia y desde la base de datos, es decir, hallamos en estos programas la interfase del usuario.

Esto ya de por sí, representaba una gran complicación, porque para los que conocemos AutoLISP sabemos que este lenguaje no provee al programador de mandatos que permitan obtener un completo control de la pantalla y entrada de datos como ocurre con todos los lenguajes de bases de datos de la actual generación. El que se haya preguntado alguna vez porqué es que los menúes de AutoCAD son tan sencillos, siendo AutoCAD un programa tan avanzado, tiene ya la respuesta. Las pantallas que se crean para la introducción de datos, la validación en la entrada, han sido construidos en la medida de lo posible,

llevando también una concordancia con la interfase de usuario que usa el AutoCAD (sencilla), con eficacia y rapidez.

Debemos también acordarnos que el AutoLISP no posee mucho espacio de memoria para los programas, algo así como unos 45 Kbytes, lo que nos imponía una restricción insalvable. Podía entonces usarse la opción de memoria virtual (VMON) del AutoLISP pero se perdería rapidez de ejecución al tener que recuperar una función posiblemente del disco, y esta última característica mencionada, la rapidez de ejecución, es de vital importancia cuando se trata de un lenguaje que es interpretado, no compilado. Se decidió no utilizar el VMON, por lo que se optó por optimizar el código de las rutinas y administrar las cargas a memoria de los procedimientos por un programa específicamente diseñado, un manejador de memoria.

Otro aspecto fundamental de estos programas es la relación que consiguen entre la entidad gráfica y la base de datos. Es algo discutido previamente, que toda entidad tiene un nombre (ename) y tiene una lista de atributos asociada que lo definen. El código DXF 5 es la clave con la que relacionamos la entidad gráfica con los datos en un registro. Este es un número único hexadecimal que se asigna a cada entidad principal automáticamente cuando se la crea en el AutoCAD. Los programas

AutoLISP accesan este número y lo envían como campo clave junto con los demás campos del registro a la base de datos.

Los programas AutoLISP importantes son:

Operaciones a nivel de base de datos

- USAR.LSP: Que nos permite seleccionar el bloque de datos con el que vamos a trabajar.
- CREAR.LSP: Es el programa con el que creamos los bloque de datos, especificando los archivos asociados y su estructura (campos).
- ELIM.LSP: Con él borramos los archivos y registros de un bloque de datos.
- REORG.LSP: Es un programa que corrige las diferencias de correspondencia en la base de datos con las entidades en el gráfico.

Operaciones a nivel de registros

- CONSUL.LSP: Para hacer consultas a la base de datos del bloque que se estuviera usando.
- ANADIR.LSP: Como su nombre lo indica, sirve para añadir registros a la base.
- MODIF.LSP: Lee un registro especificado y lo lleva al AutoCAD para modificarlo o eliminarlo de la base.
- QUERY: Tal vez el programa más poderoso (aunque realmente son un conjunto de mini-programas)

porque nos permite armar una consulta al instante.

Además tenemos los programas del sistema

- REDEFS.LSP: Que maneja funciones para controlar y verificar la integridad del sistema.
- MEMORY.LSP: Es el administrador de memoria dentro del AutoLISP. Se encarga de cargar o sacar de memoria los programas.

Variable	Función	Tipo
#fnom	Nombre del bloque de datos	Caracter
#bnom	Nombre del gráfico asociado	Caracter
#cv	Número de [1]campos visibles	Entero
#lcv	Longitud de campos visibles	Lista de enteros
#ncv	Nombres de campos visibles	Lista de carac.
#ecv	Enunciados de campos visible	Lista de carac.
#camp	Nombres de campos del registro	Lista de carac.
#cnom	Enunciados de los campos	Lista de carac.
#ctip	Tipo de los campos	Lista de carac.
#clon	Longitud de los campos	Lista de enteros
#cdec	Decimales en los campos	Lista de enteros
#lreg	Longitud del registro	Entero
#lista	Lista de entidades seleccionadas	Lista de entid.
#xr	Factor de escala en x	Real
#yr	Factor de escala en y	Real

En el sistema existen valores globales, a los cuales accesan todos los programas para obtener determinado tipo de información. Los nombres de las variables globales, sus

---

[1] Aquí campos visibles representan los atributos del bloque gráfico asociado. En realidad no son campos de ninguna base, simplemente se los considera porque estamos tomando al bloque de datos como una sola y única entidad formada por un bloque gráfico y un archivo de base de datos.

funciones en el sistema y sus tipos de datos se presentan en la lista anterior.

A continuación explicaremos detalladamente cada programa del sistema de enlace.

USAR.LSP

Cuando vamos a trabajar en el gráfico AutoCAD y necesitamos de un bloque de datos, lo primero que debemos hacer es indicar que vamos a usar ese bloque datos. LLamando a este programa nos preguntará por el nombre del bloque de datos que queremos usar y después por sus factores de escala en x y. El programa entonces cargará las definiciones de dicho bloque de datos: el nombre del bloque gráfico asociado, el nombre de la base de datos asociada y todo el detalle de los campos de esa base; a partir del archivo de definiciones que corresponda al bloque (tiene el mismo nombre que el bloque datos). Si no llamamos a esta rutina e indicamos que bloque de datos deseamos usar, no podremos trabajar con las operaciones a nivel de registro como consultar, añadir, modificar, etc.

Todas los valores globales son asignados en este programa, excepto el valor #lista. Si observamos la tabla anterior, todos los demás valores son definiciones del bloque, sus campos, la longitud del registro, su bloque gráfico, etc.

El programa verifica que exista el archivo de definiciones para poder leer la descripción del bloque. Luego, procede a cargar el bloque gráfico a la tabla de entidades y la accesa con el fin de obtener la información respecto a los campos visibles del bloque, o lo que es lo mismo, sus atributos.

El archivo inter02.txt es entonces generado para comunicarle al programa Foxpro que base de datos usar, al momento de realizar alguna operación. También tenemos que una opción del programa nos presenta un listado con todos los bloques de datos definidos en el directorio.

CREAR.LSP

La flexibilidad del sistema viene dada en gran parte por este programa, cuya virtud es que nos permite crear el archivo de base de datos, especificando su estructura, interactivamente y desde el AutoCAD mismo, sin salirnos a la base de datos.

Primero definimos el bloque de datos en lo que tiene que ver con el nombre del mismo y con el gráfico y el archivo de base de datos asociados. Datos adicionales son: la fecha de creación y un comentario descriptivo. Estos datos son en sí los que conforman el registro del archivo de bases de datos bases.dbf, que es el que lleva la información de todo bloque de datos creado (visto anteriormente). El programa verifica si el

archivo de base de datos existe, si es así le avisa al usuario que usará un mismo archivo para más de un bloque, sino se procede a presentar las pantallas para la definición de campos. Aquí tenemos otro ejemplo de la flexibildad del sistema, permitiéndonos tener varios bloques gráficos asociados al mismo archivo de base de datos, i.e., con el mismo tipo de información.

El programa escribe dos archivos: el inter02.txt y el archivo de definiciones del nuevo bloque de datos. En el primero encontraremos los datos para la base bases.dbf y en el segundo la estructura del archivo de base de datos. Una vez creados los dos, se crea el inter01.txt para enlazar con el programa Foxpro. El programa queda esperando en un lazo por el archivo inter03.txt hasta que el Foxpro lo genere. Cuando ve que ya ha sido colocado en el directorio, lee el archivo de errores error.txt para comprobar la terminación exitosa de la operación.

ELIM.LSP

Este procedimiento no necesita de mucha explicación. Sirve para eliminar o borrar un archivo de base de datos y sus archivos relacionados, obvio. Lo que cabe notar es de que si es un bloque de datos con un archivo de base de datos compartido, solamente se eliminarán los registros de dicho bloque de datos.

Se da la opción de poder mantener las entidades gráficas en el AutoCAD, ya no como bloques de datos, sino como simples bloques, o de borrarlas.

REORG.LSP

El presente programa o procedimiento es uno de los más útiles en el sistema, ya que nos permite corregir errores de correspondencia entre lo que serían la base gráfica del AutoCAD con sus bloques y el archivo de base de datos con sus registros. Podrían darse muchos casos para que ocurriera algún desfase en cualquiera de los dos sentidos: que exista un bloque en el gráfico y no su registro en la base, o viceversa; para empezar podríamos hablar de un corte de luz, un bug en Windows, una salida de edición en AutoCAD sin salvar, un borrado incorrecto de bloque, etc., tantos casos por los que podrían quedarse en el aire ciertos bloques y registros.

La reorganización de un bloque de datos consiste de dos partes. La primera que va en sentido Foxpro -> AutoCAD, recoge todas las claves de los registros y los lleva al AutoCAD para verificar que existen entidades relacionadas con las claves. Las claves que no encuentran correspondencia son agrupadas y devueltas al programa Foxpro. En él son borrados los registros de estas claves y termina allí la primera parte.

En la segunda parte se realiza el proceso inverso, AutoCAD –> Foxpro, donde un archivo de claves de entidades es generado y leido por el programa Foxpro. Este busca cada clave en los registros de la base del bloque de datos y determina cuales no tienen registro. Le devuelve al AutoCAD estas claves y el programa se encarga de señalar al usuario qué bloques no tienen su respectivo registro de datos, quedando en libertad de decidir si borrar estos bloques o volverlos a ingresar a la base. Estas claves de entidades o bloques en el aire se agrupan en una variable global #lista para no perderlas después de finalizado el proceso.

Este procedimiento es el que más utiliza intercambio de archivos. Se da una gran comunicación entre los dos programas del sistema, mandando y recibiendo los archivos de transferencia $_acad.txt y $_fpro.txt sucesivamente, coordinándose con los archivos banderas inter01.txt e inter03.txt.

A medida que se desarrolla el proceso se envían mensajes al usuario para que se mantenga atento (no piense que se ha inhibido la máquina) y especialmente se le indica la cantidad de registros no encontrados y de bloques sin datos.

CONSUL.LSP

Esta rutina se encarga de realizar las consultas de bloques individualmente, presentando la información de los campos de su registro en la misma pantalla gráfica del AutoCAD. El programa usa dos subrutinas muy interesantes, que contribuyen a dar mayor realce al sistema, *uentsl* y *cambio*.

Lo primero que nos pide el programa es señalar en el gráfico de que bloque de datos queremos ver los datos, y aquí es donde llama a la función uentsl. Esta función inicia una búsqueda dinámica del bloque, porque no es necesario apuntar exactamente con el mouse sobre la entidad que se quiere coger (algo muchas veces molestoso), sino simplemente posicionarse cerca del objeto y automáticamente la función empezará a buscar en todas direcciones hasta encontrarse con el objeto menos distante (se podría asemejar a un cuadrado agrandándose paulatinamente).

```
(T (setq bpt (getvar "snapunit");          inp punto inicial
         pt1 (mapcar '+ inp bpt);          pt1 esquina sup. der.
         pt2 (mapcar '- inp bpt);          pt2 esquina inf. izq.
         conj (ssget "C" pt1 pt2));        Selecciona entidades den-
   (while (not conj);                      tro de pt1-pt2.
      (setq pt1 (mapcar '+ pt1 bpt);       Sino encuentra,incrementa
            pt2 (mapcar '- pt2 bpt);       pt1 y pt2
            conj (ssget "C" pt1 pt2)))
   (ssname conj 0);                        Regresa la 1ra. entidad
);T
```

UENTSL.LSP

La segunda subrutina entra a funcionar cuando la entidad seleccionada no coincide con el bloque definido en ese momento, esto es, si estuviéramos trabajando con el bloque de datos A (previamente seleccionado en el programa USAR.LSP) y seleccionamos un bloque de datos B para consultar; el programa se da cuenta de eso y manda a ejecutar la subrutina cambio, que hace precisamente eso, cambia la definición del bloque anterior por la del nuevo seleccionado, sin que tenga que intervenir el usuario. CONSUL.LSP es la única función que permite cambiar de bloque de datos sin tener que recurrir a la función específica para eso, USAR.LSP.

La presentación de los datos del bloque se la realiza dividiendo la pantalla en dos horizontalmente, quedando 2/3 de ésta para el gráfico y el 1/3 restante en la parte de abajo de la pantalla para los datos. El resultado final es muy satisfactorio, a pesar del tiempo que se toma en regenerar el

gráfico. El programa utiliza una capa (layer) CONSU99 para mostrar la información solicitada, manteniendo aislado este texto de los objetos del gráfico ubicados en los demás layers.

```
while (not (findfile "d:inter03.txt")); Lazo de espera
(error);    Rutina para verificar posibles errores
(setq file (open "$_fpro.txt" "r"); Abrimos el archivo para leer
     reg (read-line file));         el registro
(close file)
(setq cnt0 (length #cnom);    Asignamos la cantidad de campos a
     cnt1 0);                 la variable local cnt0
(textscr)
(panta "CONSULTA DE DATOS")
(goto 4 1)
(princ "Visualizando el registro...")
(command "del" "d:inter03.txt");    Borramos el archivo bandera
(graphscr)
(setq cnt1 0)
(command "text" "0.01,-0.030" "");  Nos posicionamos y comenzamos a
(while (< cnt1 cnt0);               escribir los datos en la panta-
  (setq e02(nth cnt1 #cnom);        lla, un campo en cada linea
        e02(strcat (if (< (1+ cnt1) 10) " " "")
           (itoa (1+ cnt1)) ". " e02)
        e03(atoi (nth cnt1 #clon))
        e04 (substr reg 1 e03)
        reg (substr reg (1+ e03))
        l1 (strcat e02 ": [" e04 "]"))
  (command l1 "text" "")
  (setq cnt1(1+ cnt1))
)
(command nil)
```

CONSUL.LSP

El programa después de enviar la consulta queda espendo en un lazo hasta que aparezca el inter03.txt, verifica que haya sido encontrado el registro y luego lee la respuesta del $_fpro.txt para escribirla en pantalla.

ANADIR.LSP

Como su nombre lo dice, ANADIR.LSP es el programa que añade nuevos registros al archivo de base de un bloque de datos e inserta el bloque en el gráfico.

La virtud de esta rutina es que para insertar el nuevo bloque funciona de la misma forma como si estuviéramos trabajando con el comando INSERT del AutoCAD. Dados su ubicación y su ángulo de rotación, cambiamos a la pantalla texto para empezar a introducir los datos del bloque. Por las limitaciones del AutoLISP ya explicadas, la pantalla de edición es modesta, pero cumple su cometido básico. La particularidad que tiene es que en los campos a ingresar, también vamos a encontrar los llamados campos visibles del bloque (los atributos). El concepto de bloque de datos es de esta manera reafirmado bajo la acción de la introducción de una nueva entidad gráfica y un nuevo registro, como si fueran una sola cosa.

Como los anteriores, generamos el $_acad.txt y el inter01.txt para iniciar el enlace, y luego nos quedamos esperando por el inter03.txt para verificar que todo haya salido bien.

MODIF.LSP

El procedimiento MODIF.LSP está hecho para trabajar con el registro de datos del bloque seleccionado. Se puede editar los campos o borrar el registro de la base, e incluso se lo puede usar simplemente para consultar los datos, ya que debido a que presenta los datos en la pantalla texto del AutoCAD, no es necesario regenerar el gráfico, como en CONSUL.LSP, para mostrar la información y consecuentemente es mucho más rápido. Pero a su vez, este no tiene la poderosa función que le permite al CONSUL.LSP cambiar de bloque de datos sencillamente seleccionándolo.

Al igual que CONSUL.LSP, el MODIF.LSP utiliza la subrutina uentsl para seleccionar un bloque fácilmente. Si uentsl regresara una entidad no válida la operación se cancelaría.

El protocolo de comunicación que se usa con el programa Foxpro no es la excepción para este programa, ni lo es el manejo de errores.

QUERY

QUERY no es específicamente un programa, es el nombre que a un conjunto de programas le damos, porque todos precisamente construyen la consulta. Los programas que hacen esto son:

- CAMPOS.LSP: A través de él vemos todos los campos de la base del bloque y seleccionamos aquél que queramos usar en la consulta.
- OPERAD.LSP: Con OPERAD.LSP definimos el operador relacional que la expresión va a utilizar en la consulta y además introducimos el valor a ser comparado.
- RELACI.LSP: Es el programa que maneja los operadores lógicos AND y OR para unir más de una expresión en la consulta.
- LOCALI.LSP: Envía la consulta al programa Foxpro y espera por la contestación, esto es, con los registros que cumplen con la condición de la consulta. Estos son devueltos en $_fpro.txt y al recibirlos, el programa los almacena en la variable global #lista y manda un mensaje al usuario diciéndole cuántos bloques fueron hallados.

El resultado final de QUERY es que nos señala en el gráfico cuáles son los bloques seleccionados de la consulta. Estos bloques quedan guardados en #lista hasta que se ejecute otra consulta. Por medio de la función SIGUE.LSP nos movemos dentro del gráfico automáticamente y nos da un acercamiento total de cada uno de los bloques.

```
(defun sigue ( / pt1 pt2 per ang num0 )
  (if #lista (progn
    (command "zoom" "1")
    (setq pt1 (getvar "viewctr");          pt1 punto centro de la vista actual.
          per (car #lista);                Cogemos el primer bloque y cambiamos la lista, colo-
          #lista (cdr #lista);             cando el bloque al final.
          #lista (append #lista (list per))
          pt2 (dxf 10 (entget (handent per)));    pt2 punto de inserción del bloque.
          ang (angle pt2 pt1);             Calculamos el ángulo y la distancia del des-
          num0 (/ (distance pt1 pt2) 2));  plazamiento.
    (command "pan" pt1 (polar pt1 ang num0);      Empezamos a movernos hacia el bloque.
      "pan" (getvar "viewctr")
      (polar (getvar "viewctr") ang num0))
    (command "zoom" "2" "zoom" "3" "zoom" "4" "zoom" "5");  Hacemos un acercamiento de x5.
    (redraw (handent per) 3))
    (prompt "\nNo existen entidades seleccionadas.")
  )
)
```

SIGUE.LSP

REDEFS.LSP

Es el nombre de un programa especial, que se carga al iniciar la sesión de edición en el AutoCAD. Tiene dos funciones claves para la integridad del enlace: *control* y *error*.

La función control verifica que no se encuentre el archivo inter03.txt en el directorio, si lo encuentra lo borra inmediatamente. Esta función se la coloca al inicio de cada procedimiento de enlace en el sistema. El hecho de que exista ese archivo al momento de correr un procedimiento, nos indica que hubo un problema con la ejecución del último, no necesariamente grave (por ejemplo, pudimos simplemente haber

interrumpido el procedimiento con un ^C), pero que sí influye mucho en la corrida del siguiente. Obviamente, si inter03.txt es nuestro archivo bandera y lo tenemos "prendido", cuando comencemos el enlace, este terminará enseguida, creyendo que la respuesta ya ha sido devuelta.

La otra función importante es la función error, que como dice su nombre, se encarga de verificar si ha habido algún error en el enlace. Cuando se ejecuta un procedimiento y se regresa la respuesta, se ha generado un archivo, error.txt, en el programa Foxpro. En él encontramos los posibles errores, un código y un mensaje, que son leídos en este programa y según el código cancela o no la operación. La creación de este archivo se encuentra garantizada, asi se trate de un error severo, porque está asociado a una rutina de errores en el programa Foxpro.

Los mensajes de error que se dan son:

Código	Mensaje de Error
0	Todo correcto (no se muestra).
1	Archivo no existe.
2	No es un archivo de base de datos.
3	No se puede crear archivo.
4	No se puede abrir archivo.
5	Archivo de índices no encontrado.
6	Registro no encontrado.

Nótese que estos errores sólo corresponden al enlace; los programas AutoLISP manejan además otros errores como de comprobación de datos, de entidades, de archivos, etc.

MEMORY.LSP

Este programa, que se carga desde el inicio mismo de la sesión, es el que administra la poca memoria que el AutoLISP nos deja a los programadores para las aplicaciones.

La idea principal de esta función es la de controlar la carga y descarga de los demás programas de memoria para no dejar que el AutoCAD manejara esto. Por qué? Bueno, porque existía un grave problema. El problema consistía en que el total de la aplicación excedía en capacidad de memoria al AutoLISP, se quedaba sin espacio para más variables, nodos, etc.; si usábamos la función de memoria virtual VMON, el AutoLISP mandaba a memoria virtual las funciones que no cabían. Esto traía como consecuencia que si llamábamos precisamente a una de estas funciones, la ejecución de ésta se volvía inadmisiblemente lenta. Recordemos de nuevo, que el lenguaje AutoLISP no es compilable, más lento todavía!

Debido a lo anterior se decidió construir esta función. Pero, cómo maneja la memoria? La respuesta es algo compleja. Para empezar, explicaremos la función del AutoLISP *eval*. Eval

evalúa cualquier función del AutoLISP definida en su lista atómica con sus argumentos. Supongamos que tenemos una función x, si pusiéramos (eval x) sin argumentos, lo que nos aparecería es todo el código de esa función. La ventaja que nos proporciona esta función, si no se han dado cuenta, es que podríamos construirnos una cadena de caracteres con el nombre de cualquier función y sus argumentos y ejecutarla dentro de un programa.

Por otro lado tenemos dos variables globales, de las cuales no habíamos hablado antes, que controlan ciertos parámetros. La primera es la variable #flist, que guarda los nombres de las funciones que han sido cargadas y que permanecen en memoria. La segunda es la variable #func(i), que guarda en #func0 la primera función (su código), en #func1 la segunda, y así sucesivamente.

Digamos ahora que la memoria la dividimos en tres partes, una parte grande, una mediana y una pequeña, numeradas 0 1 y 2 respectivamente, para que al momento de cargar una función, en base al tamaño de su código, la coloquemos en una de estas tres "piscinas". Si la piscina que le tocaría a un programa estuviese llena con otro programa, este último sería inmediatamente sacado. Con esto conseguiríamos mantener siempre las funciones en memoria real, y no sobrepasarnos del límite

(para que no se nos vayan a memoria virtual). La modularización y el ajuste de los programas a este esquema sería algo fundamental para trabaje bien esta teoría.

Todos los conceptos previos se encuentran reunidos en la función MEMORY.LSP. Tenemos la variable #flist para guardar registro de las funciones que se hallan en la memoria y si mandamos a ejecutar otra función, lo primero que hacemos es verificar que esta no se haye ya cargada en memoria por medio de esta variable, sino está, procedemos a cargarla y reemplazarla por la función que haya estado en la piscina. Y en que variable la cargamos, pues en #func0,#func1 o #func2, dependiendo de la piscina; y finalmente la ejecutamos con la función eval.

```
(setq #flist '((nil 0) (nil 1) (nil 2))); Inicializamos la variable
                                          #flist y construimos la
                                          lista.
...
(if (assoc fname #flist);     Verificamos si existe la función
                              fname en #flist.
...
(setq proxy (cons (read (strcat "#func" (itoa index)))args));
                        Construimos la variable proxy para ser
                        evaluada.
...
(set alias nil);              Eliminamos la función anterior poniéndola a
                        nil.
...
(eval proxy);     Al final ejecutamos la función.
```

Partes de MEMORY.LSP

El formato de la función acepta la piscina, el nombre de la función y una lista de sus argumentos:

(memory pisc "función" '(arg arg arg))

Por lo que si quisiéramos ejecutar el programa CONSUL.LSP tendríamos que poner la sentencia

(memory 0 "consul" nil)

Con esto terminamos el análisis de los programas y procedimientos principales que conforman el sistema de enlace en lo que al AutoLISP se refiere. Toca el turno al programa Foxpro y sus rutinas.

4.6 El Programa FOXPRO

El programa Foxpro es un conjunto de procedimientos que se encargan de "decodificar" la operación requerida en el AutoCAD y mandarle una respuesta. Así como teníamos un programa para cada operación en el AutoLISP, así también tenemos aquí un procedimiento que se encarga de realizar las distintas operaciones con la base de datos.

A continuación damos una lista detallada de los procedimientos y sus funciones, poniendo solamente énfasis en aquellos que lo ameriten:

- ENLACE: Es el procedimiento inicial y por lo tanto "padre" de todos los demás. Se encarga de preparar el ambiente de la base de datos y se queda en un lazo esperando por el achivo bandera inter01.txt. Cuando lo "ve" en el directorio, busca en el archivo inter02.txt el tipo de operación a realizar y con que bloque de datos trabajar. Con el nombre del bloque como clave, accesa el archivo de base de datos bases.dbf y obtiene el nombre del archivo de base de datos asociado al bloque. Después de obtenida esta información básica, pasamos a ejecutar el procedimiento acorde a la operación solicitada.

Terminada la operación, el procedimiento borra el archivo bandera y vuelve a esperar por el siguiente requerimiento.

- CREAR: Esta rutina crea el archivo de base de datos de un bloque dado. Lo que se debe notar aquí, es que para formar la estructura del nuevo archivo nos valemos del archivo de definiciones creado en el programa AutoLISP CREAR.LSP. Si el bloque es asociado a una base existente, se copian las definiciones previas.

```
do while (.not.file("d:\inter01.txt"))          Esperamos por el
enddo                                          archivo bandera.
arch01=fopen("d:\inter02.txt",10)          Leemos el inter02.txt.
linea=fgets(arch01)
=fclose(arch01)
s_basoper=subs(linea,1,1)     Obtenemos el código de la operación
s_base=f_nombase(trim(subs(linea,2,8)))          y la base del blq.
do case
case s_basoper='0'
   do crear with subs(linea,2)
case s_basoper='1'
   do usar
case s_basoper='2'
   do belim
case s_basoper='3'
   do reorg
case s_basoper='9'          9 es el código de la operación para
   dele file d:\inter01.txt          terminar el programa.
   dele file d:\inter02.txt
   close all
   quit
endc
```

PROC.ENLACE

- USAR: De igual forma que ENLACE, USAR lee el archivo $_acad.txt y de él obtiene dos datos: el código de la operación a nivel de registro a realizar y el registro en sí, con los datos a guardar en los campos de la base. El arreglo s_campos es donde almacenamos el nombre y las definiciones de cada campo de la base. En s_regoper la operación de registro y en s_reg los datos para el registro.

- BELIM: Sencillamente borra la base de un bloque de datos.

- REORG: Es el procedimiento que se une con REORG.LSP para corregir las incongruencias del archivo de base de datos de un bloque. Los registros de la base que no encontrasen su respectiva entidad gráfica son eliminados de la base.

- BUSCAR: Es la rutina que funciona a la par con el programa CONSUL.LSP para obtener la consulta de un bloque en el gráfico. Lo único que se recibe es la clave del bloque para accesar a la base y devolver la información.

- ANADIR: Coge los datos en la variable s_reg, añade un nuevo registro a la base y copia el contenido de s_reg a sus correspondientes campos.

- MODIF: Lo mismo que su análogo en AutoLISP, se complementa para la modificación de un registro.

- RELIM: Realiza el borrado de un registro.

- LOCALIZAR: Esta rutina nos da la posibilidad de usar una de las bondades de esta base de datos Foxpro, la sentencia SQL SELECT, que nos permite seleccionar un conjunto de registros en base a una condición. El resultado de la consulta se lo genera en el archivo $_fpro.txt. El código de el procedimiento se muestra a continuación.

```
procedure localizar
filtro=s_reg              Damos a filtro la condición de la consulta.
? filtro
if 'LIKE' $ filtro
   filtro=strtran(filtro,"*","%")
   filtro=strtran(filtro,"?","_")
endi
if len(filtro)>0
? s_base
select entidad from (s_base) to $_fpro.txt;
   plain where &filtro
endi
do okay              Rutina para indicar el éxito de la operación.
do enviar            Creamos el archivo bandera inter03.txt.
retu
```

PROC.LOCALIZAR

- ERRHAND: Este es el procedimiento de control de errores para todo el programa Foxpro. Si por algún acaso, llegara a ocurrir un error durante la ejecución del programa, inmediatamente esta rutina toma el control, analiza el error y dependiendo de esto procede. Si el error se halla dentro de la lista errores (cuando hablamos del archivo error.txt), se envía el mensaje al AutoCAD y se cancela la ejecución del proceso. Si el error es severo, se crea otro archivo, el $_error.txt, para dar información detallada del error. Pero sea cual sea el error, este procedimiento una vez tomada las acciones descritas, regresará al procedimiento "padre", ENLACE, con lo que nos aseguramos la continuidad e integridad del sistema.

Se podría acotar que todos estos procedimientos son cortos y simples, debido a que el programa Foxpro no se comunica directamente con el usuario; no necesitamos de pantallas, ni de validaciones, y en sí, el seudolenguaje del Foxpro es muy eficiente y práctico. Con una sencilla instrucción podemos cargar toda la estructura de una base a un arreglo, por ejemplo; cosas como estas facilitaban el proceso.

El siguiente capítulo estará dedicado a explicar el manejo del sistema, partiendo desde cero y desarrollando como práctica un proyecto de urbanización de terrenos.

CAPITULO V
MANUAL DEL USUARIO

En este capítulo daremos a conocer al lector la forma de usar el sistema de enlace. A medida que vayamos explicando paso a paso las opciones del sistema, iremos desarrollando un ejemplo específico, construiremos una base de datos para almacenar los datos de los terrenos de una urbanización.

Debemos hacer notar al lector, que este manual es para personas con experiencia en el manejo de AutoCAD y de Windows. Es indispensable este requisito por cuanto vamos a explicar únicamente el uso y las opciones de la interfase, sin adentrarnos en lo que serían temas propios del AutoCAD o Windows.

5.1 Instalación

El sistema deberá tener los requerimientos de hardware y de software que fueron mencionados y analizados en el capítulo anterior. Para poder instalar el paquete debe Ud. disponer de unos 2MB de espacio libre en disco. Esto es suficiente para albergar los componentes del sistema.

5.2 Cómo arrancar el sistema

Para arrancar el sistema debe entrar al ambiente Windows normalmente con WIN. Dentro de Windows corra el accesorio RECORDER y seleccione de allí el archivo de macros MACRUIZ.REC. El macro Control-F3 es el macro definido para el arranque automático del sistema. Presione esta combinación y el sistema empezará a cargarse.

Aparecerá una lista de mensajes hasta que finalmente Ud. verá el menú corriente del AutoCad. Escoja la opción, según vaya a trabajar en un archivo ya creado o si va a crear uno nuevo. Cualquiera sea el caso, Ud. puede comprobar que la interfase ha arrancado bien, yendo al Program Manager del Windows (con Alt-Tab para cambiar de procesos) y verificando que al momento deben estar dos procesos DOS corriendo llamados: AutoCad y Enlace.

PRECAUCION: Por ningún motivo vaya Ud. a cancelar el proceso Enlace durante la sesión de trabajo. Este proceso debe mantenerse corriendo y terminará automáticamente cuando salga de la sesión del AutoCad.

NOTA: Dependiendo de la cantidad de memoria que disponga su computador, Ud. podrá ejecutar más procesos a la vez o no.

Aconsejamos el uso de un disco virtual para ciertos archivos temporales, para obtener una mayor rapidez y eficiencia del sistema. El tamaño del disco virtual se recomienda de unos 10Kbytes y su denominación la letra D:.

Verifique el espacio en disco y proceda a copiar los archivos del diskette #1 fuente al directorio de trabajo del AutoCad.

```
COPY A:*.* C:\ACAD
```

Si no tiene un directorio de trabajo para los archivos del AutoCad entonces copielos directamente al directorio de este. Lo importante es que el sistema de enlace debe estar en el mismo directorio donde esten o vayan a estar los archivos gráficos del AutoCad.

El siguiente paso es copiar el archivo MACRUIZ.REC al directorio del Windows del diskette #2.

```
COPY A:MACRUIZ.REC C:\WINDOWS
```

Si no ha obtenido ningún error, la instalación está completada y estará listo para correr el sistema.

Si llegara a quedarse sin memoria, lo mejor sería que termine con el sistema de la interfase (los procesos AutoCad y Enlace) y los demás procesos para que después vuelva a correr la interfase pero sola.

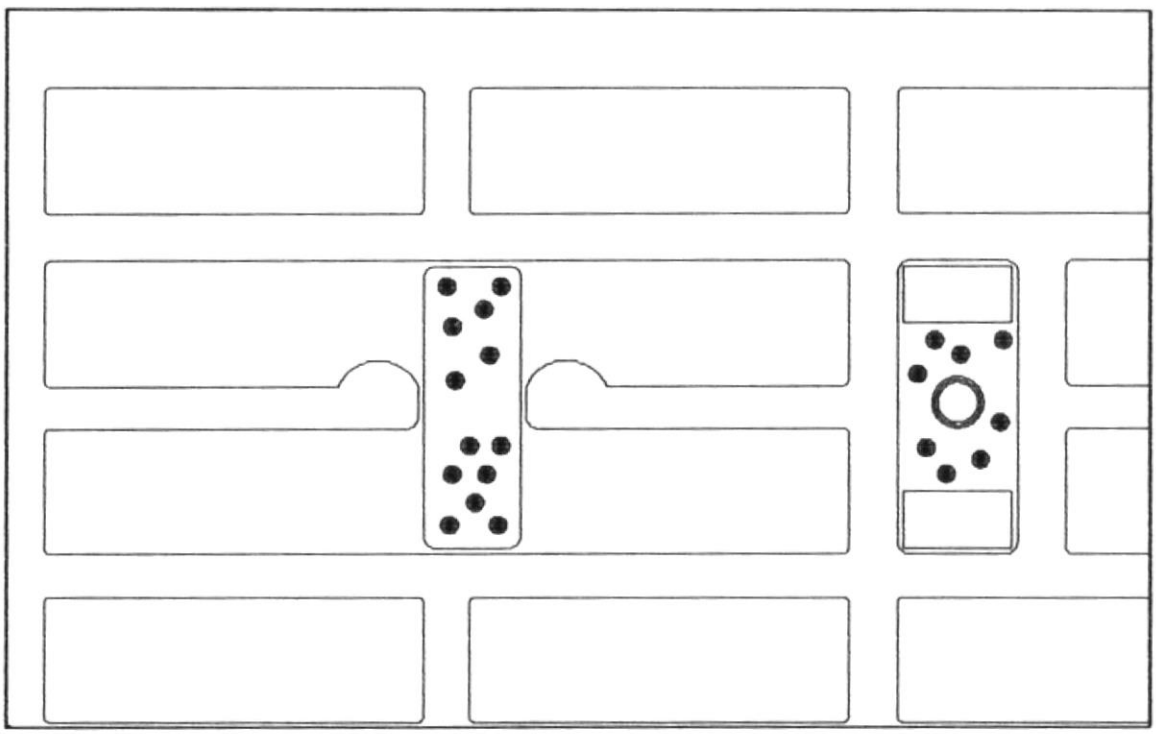
5.3 Conociendo los Bloques de Datos

Estando ya dentro de la sesión de edición del AutoCad comenzamos a diseñar nuestro ejemplo guía. Vamos a dibujar una pequeña urbanización y le vamos a relacionar cada terreno con los datos de su dueño.

La gráfica siguiente nos muestra la construcción de las manzanas para lo cual hemos usado los comandos del AutoCAD como LINE, FILLET, etc. Hemos incluso añadido parques y árboles al dibujo.

NOTA: El usuario debe haber configurado su hoja de trabajo, puesto la escala, los límites, las unidades y todo lo que concierne a su ambiente de trabajo.

Toca ahora dibujar los terrenos y lo que tenemos que hacer es clasificarlos, tener terrenos de varios tamaños, e irlos introduciendo al gráfico. Para esto conviene hacer los terrenos



Comenzando el Diseño de la Urbanización

bloques, lo que sin duda facilitará su manejo. Con el comando BLOCK definimos el bloque, con WBLOCK creamos un archivo del bloque, y con INSERT lo llamamos para insertarlo. Además podemos darle atributos al bloque con ATTDEF, como en este caso le pusimos el número del terreno a los bloques.

Hasta aquí, nada nuevo, estamos simplemente graficando con el AutoCAD. Ud. se preguntará, qué pasaría si quisiera tener información de los dueños de los terrenos, y poder hacer consultas, búsquedas o generar reportes en base a los datos?

Con el AutoCAD no podemos obtener esta clase de información. Pero sí con la interfase adicionada al AutoCAD. Si se fija Ud. bien, existen dos opciones del menú que le deben ser extrañas: Datablock y Register. Estas opciones nuevas contienen funciones que sirven precisamente para llevar a cabo lo antes mencionado.

Cómo? Pues definiendo un bloque de datos. El bloque de datos es en sí un concepto nuevo. Un bloque de datos está compuesto por dos partes principalmente: el bloque gráfico y su archivo de base de datos. Para ponerlo de una forma clara, en el ejemplo tenemos los terrenos como los bloques gráficos, lo que les faltaría para llegar a ser bloques de datos es su correspondiente archivo de base de datos.

5.4 Las funciones de Datablock

La función CREATE

Esta función la encontramos en la opción del menú Datablock y como es obvio crea un bloque de datos. Lo primero que debe ingresar es el nombre con el que va a identificar al bloque de datos. Este nombre debe ser único, de una longitud de 8 caracteres. Si existe un bloque con el mismo nombre, la

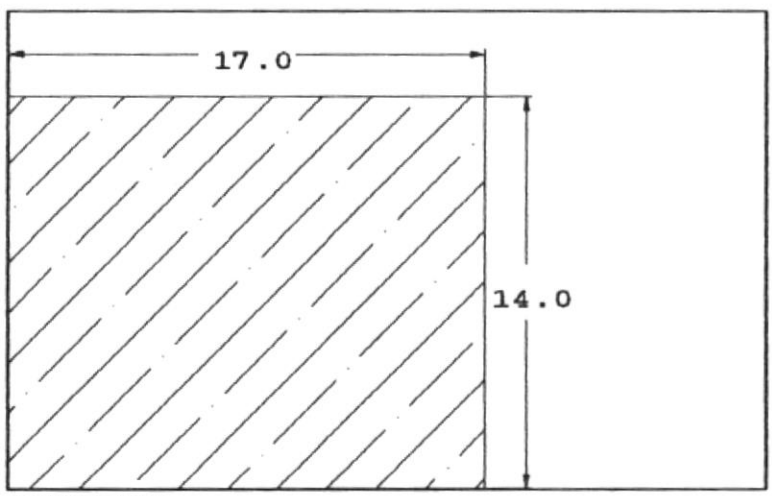
función no le permitirá crearlo. Después le pedirá por un texto descriptivo del bloque. Luego, debe Ud. introducir el nombre del bloque gráfico y el nombre del archivo de base de datos a relacionar con el nuevo bloque de datos.

Esto significa, que Ud. puede asociar varios bloques de datos al mismo archivo de base de datos o al mismo bloque gráfico, con lo que se adquiere versatilidad. Es muy importante analizar esto, ya que en nuestro ejemplo, tenemos varios tipos de terrenos, pero todos tienen la misma clase de información, por lo que todos deben accesar al mismo archivo de base de datos.

Tenemos en la urbanización 5 tipos de terrenos: terreno1, terreno2, terreno3, terreno4, y terreno5; y cada uno relacionado al mismo archivo: terrenos.dbf.

En la figura que sigue tenemos el gráfico correspondiente a uno de los terrenos. Se trata del terreno2, que es un terreno esquinero. Dijimos hace un rato que Ud. asocia un bloque gráfico y un archivo de base de datos a un bloque de datos, pero además puede Ud. relacionar un tercer elemento, un archivo tipo .sld (slide), con la idea de mostrar el bloque gráfico con más detalle. En la figura del terreno2 nótese que tiene las dimensiones del terreno. No sería adecuado introducir en

nuestra urbanización a todos los terrenos con sus medidas (los terrenos uno junto al otro, se montarían las cotas, o sería redundante tener a cada uno de los terrenos con medidas a lo largo de todo el gráfico!).



TERRENO2

En nuestro ejemplo, el terreno2 tiene relacionado un bloque, el bloque gráfico ter2-blq, que es el mismo terreno2 pero sin cotas. Podría Ud. tener también un bloque gráfico con una figura en dos dimensiones para trabajar con él, y por otro lado la misma figura en tres dimensiones para visualizarla mejor, esto es, depende de Ud. como quiera usar este tercer elemento.

El nombre de este archivo es el mismo nombre que le damos al bloque de datos. El programa buscará por un archivo llamado

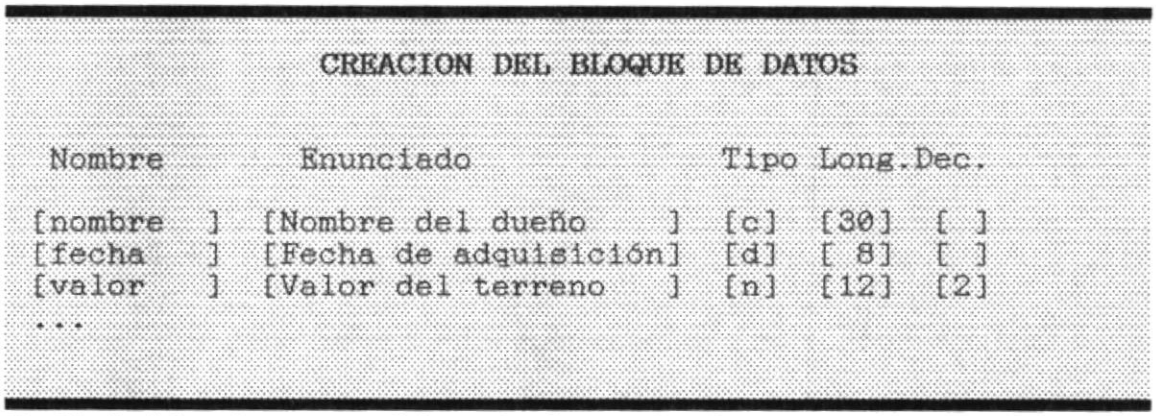
"nombre-bloque datos.sld", si no lo encuentra simplemente no lo muestra.

Pasemos a definir el archivo de base de datos del bloque de datos. Esto le toca hacer si la base a la que relaciona no está ya creada. Lo que tiene que tomar en cuenta es que el número de campos del registro más el número de atributos del bloque gráfico no deben ser mayor a 15; si tenemos que al bloque ter2-blq le definimos un atributo, entonces son 14 el número máximo de campos del registro.

Típicamente los tipos de datos que Ud. puede optar son tres: caracter, numérico y fecha. El tipo de datos caracter puede tener hasta 55 caracteres (bytes) de longitud, el numérico hasta de 20 cifras (incluidos 5 decimales) y el tipo fecha siempre es de 8 bytes de longitud.

La longitud del nombre del campo es de 10 caracteres como máximo. Se puede usar cualquier combinación de caracteres válidos para los nombres (A...Z,0...9,_,).

Ud. introduce también un texto que es el texto que le aparecerá cuando se visualice el campo, ya sea para ingresar, consultar, modificar, etc., en otras palabras es el enunciado del campo.



Definición de un archivo de base de datos

Cuando haya terminado de especificar los campos, debe esperar unos segundos hasta que esté creado el bloque de datos.

NOTA: Si la creación del bloque de datos no terminase correctamente, el AutoCAD le enviará un mensaje indicándole que no pudo crearse el bloque.

La estructura del archivo de base de datos asociado con nuestros terrenos se muestra en la página siguiente.

La función INITIALIZE

Esta función es primordial para que Ud. pueda empezar a trabajar con los bloques de datos. INITIALIZE configura el archivo gráfico del AutoCAD en el que se va a editar el dibujo. Ud. debe presionar con el mouse sobre esta función para que se ejecute, antes de hacer cualquier operación con los bloques de datos.

CAMPO	ENUNCIADO	TIPO	LONG.	DEC.
Propietar	Propietario	c	30	
Fecha	Fecha de Adquisicion	d	8	
Ubicacion	Ubicacion	c	30	
Caracter	Caracteristicas	c	10	
Avaluo	Avaluo Catastral	n	11	2
Valor	Valor Estimado	n	9	0
Comentario	Comentarios	c	20	

Archivo de Definición

Unicamente es necesario accionar INITIALIZE una sola vez para toda la creación y edición del proyecto, porque cuando se graba el archivo la configuración queda también en él.

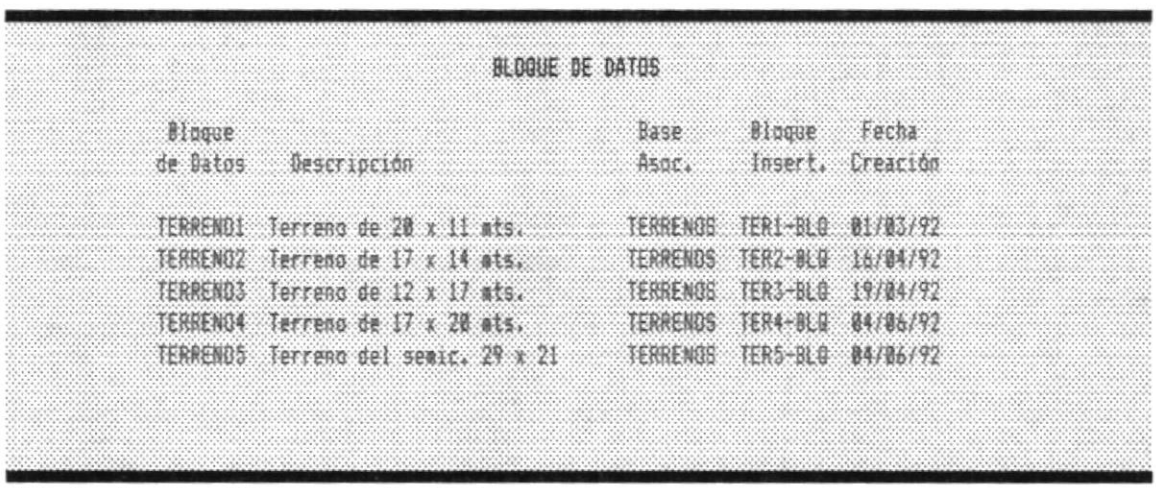
La función USE

Cuando Ud. quiere trabajar con un bloque de datos, lo primero que debe hacer es decirle al AutoCAD cuál bloque va a usar. Con USE Ud. le dice al sistema que se prepare para operar con un bloque de datos específico, de lo contrario AutoCAD no sabrá que bloque utilizar y no lo dejará entrar en las opciones de Register.

Ud. puede crear muchos bloques de datos, tener su biblioteca de bloques, pero cómo recordará las relaciones de cada uno y hasta los mismos nombres de los bloques?

Bueno, USE guarda una opción que es "?", si Ud. digita un "?" cuando USE le pide por un bloque de datos, se le presentará un listado con todos los bloques definidos por Ud. y sus respectivos archivos, el bloque gráfico y el archivo de base de datos, y la fecha de creación de cada uno.

A continuación la lista que vemos en el proyecto, con todos los terrenos y sus archivo relacionados.



BLOQUE DE DATOS

Bloque de Datos	Descripción	Base Asoc.	Bloque Insert.	Fecha Creación
TERRENO1	Terreno de 20 x 11 mts.	TERRENOS	TER1-BLQ	01/03/92
TERRENO2	Terreno de 17 x 14 mts.	TERRENOS	TER2-BLQ	16/04/92
TERRENO3	Terreno de 12 x 17 mts.	TERRENOS	TER3-BLQ	19/04/92
TERRENO4	Terreno de 17 x 20 mts.	TERRENOS	TER4-BLQ	04/06/92
TERRENO5	Terreno del semic. 29 x 21	TERRENOS	TER5-BLQ	04/06/92

Listado de Bloques de Datos

En la especificación del bloque de datos en USE también tiene que entrar los factores de escala del bloque gráfico en X y Y, es decir que si quisiera modificar el tamaño del bloque para el dibujo, lo puede hacer aquí y tendrá el bloque reescalado temporalmente.

La función DROP

La función DROP elimina, borra o deshace un bloque de datos. Si Ud. ya no requiere más trabajar con un bloque de datos o simplemente desea cambiarlo y crear otro, entonces Ud. puede eliminarlo y borrar así, tanto los archivos, como los bloques gráficos, si estos se hallasen en el gráfico.

DROP confirma su intención de deshacer un bloque y también le brinda la opción de borrar las entidades gráficas (los bloques) en la sesión de edición del archivo.

La función REORGANIZE DATA

Ud. a la hora de trabajar con los bloques de datos, se le podrían presentar problemas inherentes a todo sistema de computación, esto es, por algún motivo que se inhiba la máquina, que se corte el sumnistro eléctrico, etc.; esto podría acarrear problemas de inconsistencia de los datos: tener la entidad gráfica en el dibujo pero no su correspondiente registro de datos.

También podría Ud. tener inconsistencia de datos en el otro sentido, teniendo los datos pero no la entidad gráfica.

NOTA: Ud. debe trabajar con los bloques de datos con las operaciones específicas para ellos para prevenir las inconsistencias.

Ud. únicamente selecciona el bloque de datos a reorganizar y el proceso comenzará su ejecución. Durante el proceso, Ud. verá mensajes indicativos, correspondientes a las acciones que enese momento realiza el proceso.

Al finalizar, enviará un mensaje señalándole cuántos bloques gráficos no tienen datos. Ud. podrá saber qué bloques son con la opción NEXT ENTITY. Esta opción le irá mostrando uno a uno los bloques que resultaron sin datos en el proceso de reorganización. Ud. podrá decidir qué hacer con ellos, si borrarlos o volverlos a ingresar.

La función VIEW OBJECT

VIEW OBJ. le permite visualizar el archivo de detalle .sld relacionado con el bloque de datos corriente.

Si no hubiese este archivo la función sencillamente no realiza nada.

Al ser un archivo de slide, se muestra en la pantalla sobreponiéndose sobre el gráfico pero sin alterarlo. Para volver al gráfico, debe ejecutar un simple REDRAW y la figura desaparecerá.

5.5 Funciones de Register

Las funciones de Register le permiten trabajar con los datos del bloque de datos. Siempre debe estar un bloque de datos definido para poder entrar y correr una de estas funciones.

La función APPEND

APPEND adiciona un bloque de datos al gráfico. Al seleccionar esta función, lo primero que obtiene es el bloque gráfico listo para que lo coloque en el sitio que quiera (semejante al INSERT); luego, Ud. puede ingresar también el ángulo de rotación del bloque.

Una vez colacada la figura en el gráfico, se le presenta una pantalla para introducir los datos del bloque. Note Ud. que alguno de los campos puede tener un asterisco al lado del número del orden. Esto significa que ese campo no es realmente

un campo, sino un atributo del bloque, se recuerda? Cuando Ud. definió su bloque gráfico, pudo haberlo creado con atributos. Estos atributos son leídos por el programa para que los ingrese junto con los campos como datos.

Las definiciones de los atributos de bloque son tomadas por el programa de la siguiente forma. La etiqueta del atributo es el nombre del campo. El enunciado del atributo es el enunciado del campo, y del valor por omisión se calcula la longitud del campo. El tipo de todo atributo en el AutoCAD es caracter.

```
Command: ATTDEF
Attribute modes -- Invisible:N Constant:N Verify:N Preset:N
Enter (ICVP) to change, RETURN when done:
Attribute tag: VILLA
Attribute prompt: Número de villa
Default attribute value: --
...
```

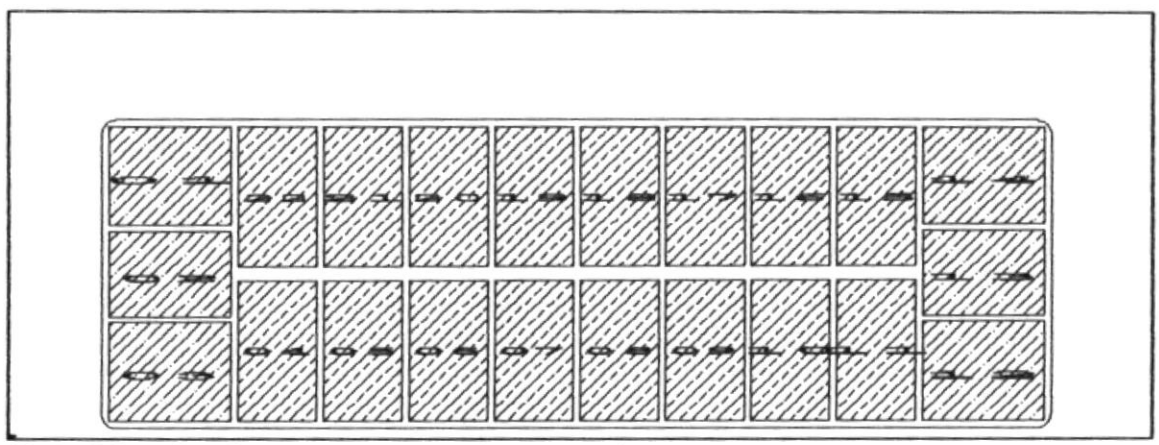
Definición de atributos

En el ejemplo de arriba, definimos un atributo villa, con un enunciado y un valor por omisión. De estos parámetros se

vale el programa. Como ve, el valor por omisión tiene "--", esto significa que el campo está siendo definido de longitud 2. Si no se pusiera ningún valor el programa asume 20.

En nuestro proyecto todos los terrenos tienen definidos un atributo, que nos indica en el gráfico el número de terreno correspondiente en la manzana.

Ud. puede después de ingresar los datos, modificarlos, presionando "m", cancelar toda la inserción con una "c", o aceptarlos e ingresarlos al archivo con una "i".



Detalle de los terrenos insertados

La función DISPLAY

DISPLAY es una las funciones más útiles para Ud. Con ella Ud. visualiza la información de un bloque de datos. Simplemente posicionándose sobre la entidad y presionando el botón del ratón, la pantalla gráfica se dividirá en dos y Ud. verá la información abajo y el bloque seleccionado arriba.

Ud. disfrutará con esta función el poder cambiar de definición de bloque de datos directamente (sin usar USE). Si la función se da cuenta que Ud. ha escogido un bloque diferente al definido en ese momento, ésta automáticamente procederá a cambiar de bloque de datos.

Justamente para complementar esta función se creó la función NORMAL SCREEN. Esta le permite a Ud. regresar la pantalla a su estado normal terminada la consulta.

La función WORK WITH...

Bueno, si Ud. tiene hasta aquí funciones para añadir y consultar, las que faltan las encontrará en WORK WITH... Con ella, obviamente Ud. podrá modificar o borrar los datos del bloque de datos. Semejante a DISPLAY, Ud. lo único que hace es señalar con que bloque va a trabajar y la información

aparecerá, pero ya no en la pantalla con el gráfico, sino en la pantalla posterior de texto.

La otra limitación que hallará en WORK WITH es que tiene obligatoriamente que escoger un bloque de tipo igual al que se halle definido en ese momento. WORK WITH no cambia automáticamente la definición de bloque como USE.

La función QUERY

Si Ud. quisiera construir una consulta para obtener un conjunto de terrenos con una particularidad en común, entonces Ud. tiene la perfecta solución a su problema: el QUERY.

QUERY le posibilita a Ud. generar una consulta guiándolo a través de varias pantallas. Al presionar en QUERY se le abrirá el menú de QUERY, en donde podrá ir seleccionando en orden los campos, los operadores relacionales, los lógicos, para ir paso a paso construyendo la condición.

En FIELDS Ud. ve todos los campos del bloque en uso y selecciona uno de ellos. Después, en RELATIONALS Ud. escoge el operador relacional e ingresa el dato a comparar. Ud. puede consultar con una condición simple o haciendo una condición

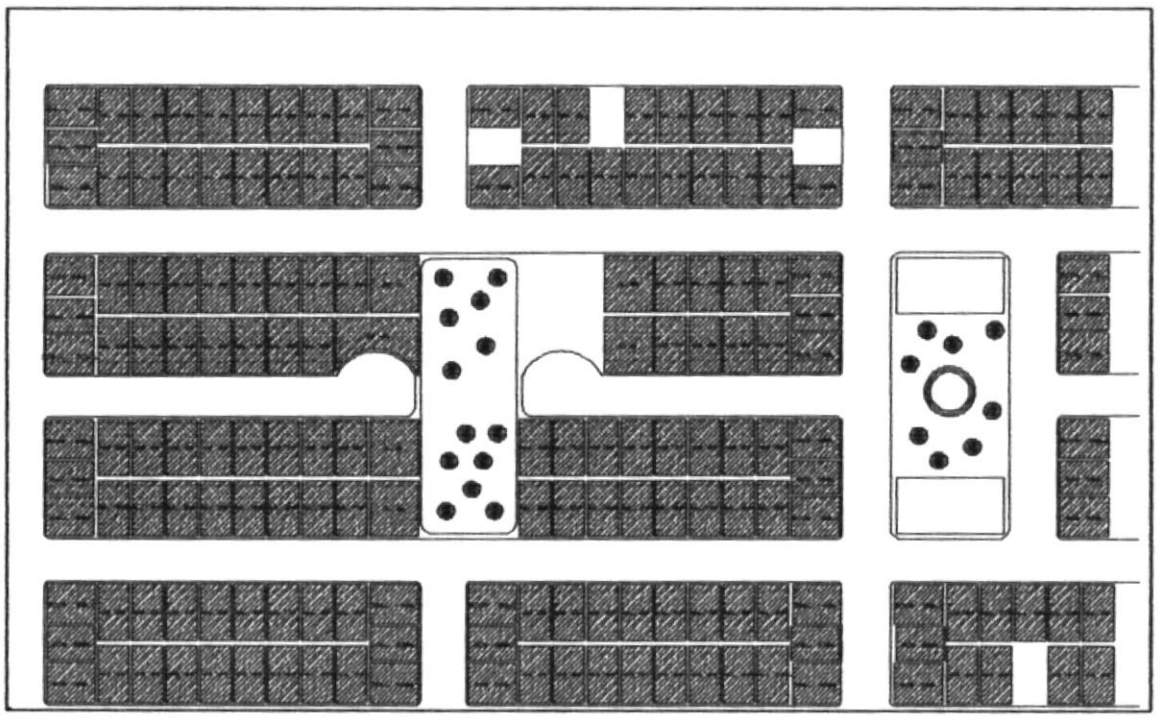
compuesta, que sería la unión de condiciones simples con operadores lógicos. Estos los encuentra en la opción LOGICALS.

Cuando tiene ya construida la condición, se puede ahora sí ejecutarla con GO. Go inicia la búsqueda, hace la selección y envía un mensaje indicando cuántos bloques han cumplido con la condición. El programa para señalar estos bloques los redibuja con líneas punteadas, de tal manera que al finalizar la selección y si tiene escalado todo el dibujo en la pantalla, verá los bloques escogidos.

Al igual que en REORG. DATA, los bloques quedan marcados, pudiendo ir a verlos uno por uno con la opción NEXT ENTITY.

Si en cualquier momento de la generación de la condición Ud. desea ver como va, Ud. tiene la opción LOOK AT, que le muestra la condición en ese instante.

En general, se utilizó cada una de las funciones antes explicadas para llegar a construir el diseño de la urbanización con los diferentes tipos de bloques de datos. Al final lo que se obtuvo fue un sistema de información de los terrenos de una urbanización y sus propietarios.



La Urbanización terminada

CONCLUSIONES Y RECOMENDACIONES

El presente trabajo tiene como resultado práctico una interfase que comunica al graficador AutoCAD con archivos de base de datos contruidos por la base de datos Foxpro. El que tenga una idea clara de las aplicaciones que involucran cada paquete, sabrá que esta unión puede ser calificada como "más que provechosa" para los usuarios de AutoCAD.

Y es que si analizáramos el sin número de aplicaciones que se podrían dar de esta interfase fácilmente, llegaríamos a la conclusión de que, no sólo se han adicionado unos comandos al AutoCAD para expandir sus habilidades, sino que se ha construido la parte que le faltaba al AutoCAD para convertirlo, a todas luces, en un paquete completo para el usuario.

Con esta "nueva parte" del AutoCAD, el usuario común será capaz de guardar toda clase de información referente a su diseño gráfico y podrá consultarla, modificarla, seleccionarla; ya no se limitará el diseño a trabajar con las figuras gráficas, sino que a la par, trabajará con los datos de esas figuras.

La información almacenada en los archivos de base de datos y manejada por la base de datos Foxpro, como se ha estudiado previamente, reúne las siguientes características entre otras:

- Los registros de los archivos de las bases de datos son aleatoriamente accesados.

- La estructura de los archivos que guardan la información no es rígida, esto es, el usuario determina la estructura de todo archivo de base de datos al crearlo.

- La información puede ser en cualquier momento fácilmente accesada y generar toda clase de reportes por medio del ambiente Foxpro.

Por otro lado, la interfase tiene algunas limitaciones e inconvenientes. Primero, la dependencia de un tercer paquete de software, el ambiente gráfico Windows, es una limitante desde todo punto de vista si nos ponemos a pensar en que necesitamos de él para poder correr la interfase. De otro modo, es imposible correr los dos programas al mismo tiempo, al menos en DOS.

Segundo, como consecuencia del Windows, se requiere una máquina de mínimo 4MB de memoria física instalada, para que se

pueda trabajar en modo agrandado de Windows (enhanced mode) y correr AutoCAD y la interfase al mismo tiempo sin quedarnos con espacio insuficiente de memoria.

Tenemos también la limitante de la velocidad de la máquina. Para que la interfase tenga un tiempo de respuesta aceptable a lo menos debe correrse el sistema en un hardware de 25Mhz. Lógicamente, a medida que aumentemos la velocidad de la máquina, obtendremos mejores resultados.

Son muchas las mejoras que se pueden implementar a partir de la actual interfase. Para empezar podríamos mencionar el trabajar con un disco virtual de memoria, tanto para los archivos temporales que crea Windows, como para el archivo temporal que usa el AutoCAD en la sesión de edición. Redirigiendo estos archivos a una memoria física incrementaría en mucho la velocidad de ambas aplicaciones. Estimamos que llegar a tener 2MB más de memoria sería muy beneficioso y nos ahorraría la molestia de usar la engorrosa memoria virtual del AutoCAD.

En nuestra interfase hemos utilizado el AutoCAD versión 10, pero la misma mejoraría enormemente si hubiésemos podido implementarla en la última versión del AutoCAD. El nuevo AutoCAD tiene un ambiente desarrollador de aplicaciones en C, y

creo todos nos podemos imaginar lo que esto significa. Además, ofrece muchas ventajas que harían muchos de los procedimientos actuales de la interfase menos rígidos y más eficientes.

Si nos ponemos de cara al futuro de la interfase, nos encontramos con varios caminos a escoger, varias alternativas que se dan para que se siga experimentando e investigando con la interfase. Por mencionar algunos caminos, tenemos por ejemplo, tratar de instalar la interfase en una red, adicionarle funciones especifícas para alguna aplicación a la interfase, como generación de reportes, cálculos de presupuestos, proyecciones, etc.

Si bien no quedamos del todo satisfechos en cuanto a innovación tecnológica se refiere, creemos que hemos cumplido con las metas que nos impusimos al iniciar este proyecto, y aspiramos que en un futuro no muy lejano, otros estudiantes, tomen la posta de este proyecto, que los que vengan sigan abriendo la puerta correcta y continúen en la búsqueda de una excelencia académica.

APENDICE

A. GUIA DE FUNCIONES DE AUTOLISP.-

(+ *número número ...*)

Retorna la suma de todos los números, ya sea como entero o real dependiendo del valor.

(- *número número ...*)

Retorna la diferencia del primer número menos la suma de los números restantes. Un entero o un real es retornado, dependiendo del valor.

(* *número número ...*)

Retorna el producto de todos los números. Un entero o un real es retornado, dependiendo del valor.

(/ *número número ...*)

Retorna el cociente del primer número dividido para el producto de los números restantes. Un entero o un real es retornado, dependiendo del valor. Si los dos números son enteros, el residuo es eliminado.

(= *átomo átomo ...*)

Retorna T si los átomos son numéricamente iguales. En caso contrario retorna NULO. Sólo son válidos números y strings.

(/= *átomo átomo ...*)

Retorna el número decrementado en 1.

(abs *número*)

Retorna el valor absoluto de un entero o real.

(ads)

Retorna una lista con los nombres y paths de las aplicaciones ADS cargadas. Si ninguna está cargada, retorna NULO.

(alloc *número*)

Establece el tamaño del segmento (grupos de nodos) al valor del número y retorna el tamaño anterior.

(and *expresión ...*)

Retorna T si todas las expresiones son verdaderas. En caso contrario retorna NULO y cesa la evaluación en la primera expresión NULA encontrada.

(angle *punto punto*)

Retorna el ángulo en radianes desde el eje X, en dirección contraria al movimiento de las manecillas del reloj, hasta una línea entre los dos puntos.

(angtos *ángulo modo precisión*)

Retorna un string con la conversión del valor de un ángulo, de radianes a las unidades especificadas por el modo. De no especificarse los valores opcionales de modo y precisión, se toman por omisión sus valores corrientes.

(append *lista ...*)

Retorna una única lista construída a partir de varias listas.

(**apply** ***función lista***)

Aplica una función a los argumentos presentados por la lista. Generalmente, la función y la lista son escritos entre comillas, de manera que sus contenidos no sean evaluados.

(**ascii** ***string***)

Retorna el primer caracter del string como un código ASCII entero.

(**assoc** ***elemento lista***)

Retorna la lista, de una lista de listas, que contiene el elemento clave como un primer elemento.

(**atan** ***número*** *número*)

Retorna el arcotangente del número1, de -PI a PI. Si existe un número2, se retorna el arcotangente de número1/número2. Si número2 es 0, se retorna -PI/2 o +PI/2 radianes (-90 o +90 grados), dependiendo del signo del número1.

(**atof** ***string***)

Retorna el valor real de un string.

(**atoi** ***string***)

Retorna el valor entero de un string.

(**atom** ***elemento***)

Retorna T si el elemento no es una lista. En caso contrario retorna NULO.

(**boole** ***función entero entero ...***)

Retorna una de las 16 posibles operaciones lógicas, en base al valor de la función sobre los enteros.

(**boundp** ***átomo***)

Retorna T si el átomo es limitado a un valor. En caso contrario retorna NULO.

(**cadr** ***lista***)

Retorna el segundo elemento de la lista. Use CADR para extraer la coordenada Y de un punto.

(**car** ***lista***)

Retorna el primer elemento de la lista. Use CAR para extraer la coordenada X de un punto.

(**cdr** ***lista***)

Retorna una lista sin su primer elemento.

(**c????r** ***lista***)

Retorna un elemento o lista definido por la combinación de los caracteres 'a' y 'd' en la expresión. Ejs: CAADR, CDDR, CADAR, etc.

(**chr** ***entero***)

Retorna el string convertido de un código entero ASCII.

(**close** ***file-desc***)

Cierra el archivo representado por la variable file-desc.

(command *argumento ...*)

Manda sus argumentos como entrada a AutoCAD. Strings y números son tomados como entrada literal; otros argumentos mandan el valor retornado por sus expresiones, como entrada a AutoCAD. La función COMMAND por sí sola ejecuta un retorno, (COMMAND nil) ejecuta un <^C>. El símbolo PAUSE (variable con valor '\'), usada como una función COMMAND, realiza una pausa sobre la función COMMAND y permite una entrada por parte del usuario.

(cond *test expresión ...*) ...)

Evalúa la(s) expresion(es) del primer test no NULO. Cualquier número de listas (text expresión ...) son revizadas. El valor de la última expresión evaluada es retornado. COND cesa su evaluación después de encontrar un test NULO o después de completar la lista de tests.

(cons *elemento elemento*)

Retorna una nueva lista con el primer elemento como el nuevo primer elemento de la lista, si el segundo elemento es una lista. Si el segundo elemento no es una lista, se retorna un par en la forma (elemento.elemento).

(cons *elemento lista*)

Añade el elemento como el primer elemento de la lista. Retorna la nueva lista.

(cos *ángulo*)

Retorna el coseno del ángulo en radianes.

(**cvunit** ***número de-unidad a-unidad***)

Retorna el real en el sistema de unidades a-unidad, convertido del número en el sistema de unidades de-unidad. De-unidad y a-unidad son strings correspondientes a las definiciones del archivo ACAD.UNT.

(**defun** ***nombre (argumento ... / local ...) expresión ...***)

Crea una función con el nombre dado. La lista de argumentos puede ofrecer variables a ser pasadas a la función. La lista de variables seguidas a la barra son variables locales a la función y no afectarán otras funciones. La función evaluará los comandos del programa y retornará el resultado de la última expresión evaluada. Usando como prefijo C: al nombre de la función creará un comando LISP que actúa como un comando estándard de AutoCAD. Definir una función S:: STARTUP en el archivo ACAD.LSP creará una una función de ejecución.

(**distance** ***punto punto***)

Retorna la distancia entre puntos en 3D o 2D.

(**entdel** ***ename***)

Elimina o recupera el ename dependiendo en su status en la sesión corriente de edición. Retorna el ename.

(**entget** ***ename*** *applist*)

Retorna una lista de datos describiendo la entidad especificada por el nombre de la entidad. Applist es una

lista de aplicaciones para las cuales se retornan datos extendidos de las entidades, así como los datos normales de las entidades.

(**entlast**)

Retorna el último nombre de entidad no eliminada de la Base de Datos.

(**entmake** *edata*)

Para una lista válida de edata, crea una nueva entidad y retorna la lista.

(**entmode** *lista*)

Modifica una entidad de la Base de Datos con una nueva lista de descripción de datos de la entidad y retorna la nueva lista de datos de la entidad. Las entidades son inmediatamente regeneradas en la pantalla con los nuevos datos, excepto para entidades muy complejas que requieren el uso de la función ENTUPD.

(**entnext** *ename*)

Retorna el primer nombre de entidad no eliminada de la Base de Datos. Si se especifica un ename, el nombre de entidad no eliminada, seguido al ename, es retornado.

(**entsel** *prompt*)

Retorna una lista conteniendo el nombre de entidad y las coordenadas del punto usado para tomar la entidad. Prompt provee instrucciones específicas para la selección de la entidad.

(**entupd** ***ename***)

Permite modificación selectiva de los vértices de un polígono y de los atributos de bloque de los nombres de entidad, después de que ENTMODs fue ejecutado.

(**eq** ***variable variable***)

Retorna T si la primera variable tiene el mismo límite que la segunda variable. En caso contrario retorna NULO.

(**equal** ***expresión expresión*** *precisión*)

Retorna T si la primera expresión es igual a la segunda (evaluadas al mismo valor). En caso contrario retorna NULO. La precisión opcional determina con qué precisión los dos números deben ser considerados iguales.

(***error*** ***string***)

Una función, definida por el usuario, para errores. String contiene el mensaje que describe el error.

(**eval** ***expresión***)

Retorna el resultado de evaluar la expresión.

(**exp** ***número***)

Retorna e elevado a la potencia del número.

(**expand**)

Intenta asignar un número de nodos desde el espacio de heap, al espacio de stack y retorna el número requerido.

(**expt** ***base potencia***)

Retorna el número base elevado a la potencia especificada. El valor retornado es un entero o un real dependiendo de los valores de la base y la potencia.

(**findfile** ***nombre-archivo***)

Retorna el nombre de archivo con su path, si el archivo es hallado. En caso contrario retorna NULO. Se busca únicamente en el directorio especificado, si un path es añadido al nombre-archivo. Si no se añade path, se busca en el path de la librería AutoCAD. No se permiten comodines.

(**fix** ***número***)

Retorna en entero del número y elimina el residuo.

(**float** ***número***)

Retorna el valor real de número.

(**foreach** ***símbolo lista expresión ...***)

Evalua la expresión, sustituyendo cada elemento en la lista por el símbolo de un lazo de la expresión. El símbolo es una variable alias, que temporalmente toma el valor de cada elemento de la lista de un lazo. La expresión en el lazo se debe referir al elemento corriente de la lista del alias. El valor de símbolo es local para FOREACH.

(**gc**)

Recoge los nodos "basura", no relacionados más a símbolos, y los añade a un espacio de nodos libres.

(**gcd** ***entero entero***)

Retorna el máximo común denominador de dos enteros.

(**getangle** *punto prompt*)

Retorna un ángulo en radianes. Un valor es ingresado por el usuario o determinado por dos puntos ingresados por el usuario. El ángulo es medido desde el eje X, en sentido contrario a las manecillas del reloj, a menos que el ángulo sea invertido por el comando UNITS. Un punto opcional especifica el punto base de una línea. Prompt provee específicas instrucciones para determinados valores de puntos. Use GETANGLE para rotación.

(**getcorner** ***punto*** *prompt*)

Retorna un punto seleccionado como la segunda esquina de una ventana de AutoCAD. Prompt provee específicas instrucciones para la selección de un punto deseado.

(**getdist** *punto prompt*)

Retorna una distancia ingresada por el usuario o una distancia calculada entre dos puntos ingresados por el usuario. Punto especifica un punto base para una línea. Prompt provee específicas instrucciones para la selección de un punto deseado.

(**getnev** ***nombre***)

Retorna el valor string de la variable del sistema operativo especificada por el argumento nombre. Si no se la encuentra, se retorna NULO.

(**getint** *prompt*)

Retorna el entero ingresado por el usuario. La entrada debe encontrarse entre -32768 y +32768. Prompt provee instrucciones específicas de entrada.

(**getkword** *prompt*)

Retorna un string que coincide con la clave ingresada por el usuario. Las claves son especificadas en la función (initget). Prompt provee instrucciones específicas de entrada.

(**getorient** *punto prompt*)

Retorna un ángulo en radianes como entrada del usuario o calculado por dos puntos ingresados por el usuario. El ángulo es medido desde la base de 0 grados en la dirección especificada por UNITS. El punto opcional especifica el punto base para una línea. Prompt provee instrucciones específicas para determinados valores de puntos. Use GETORIENT para orientación.

(**getpoint** *punto prompt*)

Retorna un punto. Punto especifica el punto base para una línea. Prompt provee instrucciones específicas para la selección de un punto deseado.

(**getreal** *prompt*)

Retorna un real ingresado por el usuario. Prompt provee específicas instrucciones de entrada.

(**getstring** *flag prompt*)

Retorna un string de hasta 132 caracteres ingresados por el usuario. Si Flag es NULO u omitido, los espacios no son permitidos en el string y actuarán como un <RETURN> y final del ingreso. Prompt provee específicas instrucciones de entrada.

(**getvar** ***sysvar***)

Retorna el valor especificado por sysvar, un string que contiene el nombre de una variable del sistema.

(**graphscr**)

Pasa de un modo texto a modo gráfico, en sistemas de una sóla pantalla. Retorna NULO.

(**grclear**)

Limpia temporalmente la ventana (viewport) corriente, dentro de modo gráfico. Volver a dibujar refrescará la pantalla.

(**grdraw** ***punto punto color*** *modo*)

Dibuja un vector entre dos puntos, en el color especificado por un entero. Un argumento negativo de color, complementa el color en que se sobredibuja. (OOJJOO)
Mediante el argumento de modo se puede utilizar ´highlight´ la línea, o entrecortarla con guiones.

(**grread** *track*)

Lee el dispositivo de entrada directamente. Si está presente el argumento track, éste retorna la localidad del punto corriente del dispositivo (mouse o cursor digital).

(**grtext** *cuadro texto modo*)

Escribe un string en la porción de texto de la pantalla gráfica especificada por el número del cuadro. El argumento modo permite 'highlight' el cuadro del texto. Un número de cuadro de -1, escribe sobre la línea de status, -2 escribe la línea de status de las coordenadas y 0, 1, 2, 3, etc escribe en las etiquetas del menú. 0 representa la etiqueta superior.

(**handent** ***handle***)

Retorna el nombre de entidad correspondiente al nombre de manejo correspondiente, si los manejos son permitidos.

(**if** ***test expresión*** *expresión*)

Si test no es NULO, la primera expresión es evaluada. Si test es NULO, la segunda expresión opcional es evaluada. La función retorna el valor de la expresión evaluada.

(**initget** *bits string*)

Establece opciones para las funciones GETxxx.

1	Entrada nula no es permitida.
2	Valores cero no son permitidos.
3	Valores negativos no son permitidos.
8	No chequear límites.
16	Retornar puntos en 3D.
32	Usar líneas entrecortadas para borrado.

String define una lista de palabras claves aceptables como entrada a GETxxx.

(**inters** ***punto punto punto punto*** *flag*)

Retorna el valor de un punto en la intersección de una línea entre los primeros dos puntos y una línea entre los segundos dos puntos. Si flag es no NULO, las líneas son proyectadas infinitamente para calcular la intersección.

(**itoa** ***entero***)

Retorna un entero convertido en string.

(**lambda** ***argumento expresion ...***)

Define una función, supliendo los argumentos a las expresiones a evaluarse.

(**last** ***lista***)

Retorna el último elemento de una lista.

(**length** ***lista***)

Retorna el número de elementos de una lista.

(**list** ***expresión ...***)

Retorna una lista construída de las expresiones presentes.

(**listp** ***elemento***)

Retorna T si el elemento es una lista. En caso contrario retorna NULO.

(**load** ***nombre-archivo*** *expresión*)

Carga el archivo AutoLISP especificado por nombre-archivo. La expresión opcional es evaluada se haya o no realizado

la carga exitosamente. Se retorna el resultado de la evaluación sólo si la carga falla. De esta manera se puede chequear el éxito o fracaso de la carga.

(**log** *número*)

Retorna el logaritmo natural del número real.

(**logand** *entero entero ...*)

Retorna un entero, resultado de la operación lógica AND (a nivel de bits) entre dos o más números.

(**logior** *entero entero ...*)

Retorna un entero, resultado de la operación lógica OR (a nivel de bits) entre dos o más números.

(**lsh** *número número-bits*)

Retorna un entero, resultado del desplazamiento lógico de los bits de un número, tantas veces como número-bits lo determine.

(**mapcar** *función lista ...*)

Ejecuta secuencialmente una función lisp en cada conjunto de elementos en una o más listas de argumentos.

(**max** *número número ...*)

Retorna el valor máximo de una serie de números.

(**mem**)

Reporta el uso de la memoria de AutoLISP.

(**member** *elemento lista*)

Si elemento es encontrado en la lista, retorna el residuo de la lista empezando por elemento. En caso contrario retorna NULO.

(**menucmd** ***string***)

Carga y presenta la página de menú especificada por el string. String debe incluir el código del dispositivo del menú y el nombre de la página. Ej: "S=NAME" para la página de dispositivo, llamada NAME.

(**min** ***número número ...***)

Retorna el valor mínimo de una serie de números.

(**minusp** ***número***)

Retorna T si el número es negativo. En caso contrario retorna NULO.

(**not** ***elemento***)

Toma un argumento simple y retorna el valor opuesto.

(**null** ***elemento***)

Retorna T si el elemento está limitado por un valor NULO. En caso contrario retorna NULO. Se usa típicamente para listas.

(**numberp** ***elemento***)

Retorna T si el elemento es un número. En caso contrario retorna NULO.

pause

Esta constante es usada en la función command, para esperar por una entrada por parte del usuario.

pi

Está constante está aproximada a 3.1415926.

(polar ***punto ángulo distancia*** **)**

Retorna un punto calculado en el ángulo y distancia desde un punto base.

(prompt ***string*** **)**

Presenta un string en la pantalla. Retorna NULO.

(quote ***expresión*** **)** o ***'expresión***

Retorna la expresión sin ser evaluada.

(read-line ***archivo-desc*** **)**

Retorna un string tipeado en el teclado o leído desde un archivo opcional.

(rem ***número número ...*** **)**

Retorna el residuo del primer número dividido para el producto del resto de números.

(repeat ***número expresión ...*** **)**

Evalúa cada expresión el número de veces especificado. El número debe ser un entero.

(reverse ***lista*** **)**

Retorna una lista de elemento invertidos del orden original.

(setq ***símbolo expresión*** *símbol1 expres1* **)]**

Asigna a símbolo el valor de la expresión.

(setvar ***sysvar valor*** **)**

Asigna a la variable del sistema AutoCAD especificada por sysvar, el valor presente.

(**sin** *ángulo*)

Retorna el seno del ángulo en radianes.

(**sqrt** *número*)

Retorna la raíz cuadrada del número real.

(**ssdel** *ename selection-set*)

Elimina ename del conjunto de selección y retorna el conjunto de selección.

(**sslength** *selection-set*)

Retorna el número de entidades en un conjunto de selección.

(**ssmemb** *ename selection-set*)

Retorna el ename del nombre de la entidad, si ésta está en el conjunto de selección. En caso contrario retorna NULO.

(**strcase** *string flag*)

Retorna el string convertido a mayúscula, a menos que la flag sea T. En ese caso se convierte el string a minúscula.

(**strcat** *string string ...*)

Retorna un string producto de la combinación de todos los strings.

(**strlen** *string*)

Retorna el número de caracteres de un string.

(**subst** *elemento elemento lista*)

Sustituye el primer elemento que encuentra en la lista por el segundo elemento.

(substr *string inicio* *longitud*)

Retorna una cadena de caracteres correspondientes al inicio de string hasta longitud de caracteres.

T

Símbolo constante que es usa como el valor lódigo de ´verdadero´.

(tblnext *tname* *flag*)

Retorna una lista de descripción de datos para el nombre de tabla especificado en tname.

(tblsearch *tname símbolo* *flag*)

Retorna una lista de descripción de datos para el nombre de tabla tname y el símbolo.

(terpri)

Imprime un ´newline´ en la pantalla y retorna NULO.

(textscr)

Cambia de modo gráfico a modo texto, en sistemas de una única pantalla. Retorna NULO.

(trace *función* ...)

Para depurar una función.

(type *elemento*)

Regresa el tipo de dato del elemento.

(untrace *función* ...)

Remueve la función de seguimiento, de las funciones especificadas.

(**ver**)

Retorna un string con la versión corriente de AutoLISP.

(**vmon**)

Enciende la memoria virtual del AutoCAD.

(**vports**)

Retorna una lista con las divisiones de la pantalla actuales.

(**while** ***test expresión ...***)

Evalúa la expresión mientras test sea verdadero.

(**write-line** ***string*** *archivo-desc*)

Escribe un string en la pantalla o a un archivo opcional.

(**zerop** ***número***)

Retorna T si el nombre es igual a cero. En caso contrario retorna NULO.

BIBLIOGRAFIA

D.Haker y H.Rice, Inside AutoCAD (EEUU: New Riders Publishing, 1989), pp. 1-578.

P.L.Olympia y Kathy Cea, Developing Foxpro 2.0 Applications (EEUU: Addison-Wesley, 1991), pp. 1-17.

J.Smith y R.Gesner, Maximizing AutoCAD: Inside AutoLISP, (EEUU: New Riders Publishing, 1991), pp. 1-614.

J.Godfrey, Lenguaje Ensamblador para Microcomputadoras IBM, (EEUU: Prentice Hall, 1991), pp. 20-63.

C.Petzold, "Computación en los 90", Revista PC Magazine, Vol.2, No.3 (Marzo 1991), pp. 95-97.

A.Schulman, "Windows 3.0", Revista PC Magazine, Vol.10, No.11, (Junio 1991), pp. 352-358.