UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DESARROLLO DE UN MANEJADOR DE OBJETOS GRAFICOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

Griselda Gutiérrez Izquierdo

Asesor : M. en I. Luis Alvarez Icaza

México, D. F. 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

ÍNDICE

CAPITULO 1

INTRODUCCIÓN

Con el objeto de facilitar la interpretación de la información sobre el estado de un sistema a través de iconos o gráficos se han realizado grandes avances en la representación gráfica de información pues es sabido que el usuario puede entender con mayor rapidez y facilidad información en forma gráfica que caracteres alfanuméricos.

Por lo anterior el presente trabajo tiene como objetivo desarrollar técnicas por medio de las cuales se puedan representar y manipular objetos gráficos planos; se pretende en particular que un objeto gráfico con combinaciones arbitrarias de formas, figuras, texturas y colores pueda crearse, modificarse, almacenarse y recuperarse.

Además se consideró conveniente dotar al sistema de capacidad para manejar colecciones agrupadas de objetos gráficos definiéndolos como un sólo objeto gráfico de tal forma que todas las operaciones de manipulación predefinidas para un objeto gráfico único se puedan aplicar a estas colecciones.

En particular el manejador ha sido empleado para mostrar de manera gráfica en el monitor de una computadora personal los resultados de la simulación del acueducto del Sistema Cutzamala [ref. 1].

Existen en el mercado paquetes de graficación mucho más versátiles que éste, sin embargo, la particularidad del paquete de objetos gráficos desarrollado permite modificar el estado de los objetos a través del tiempo de una manera muy simple.

El paquete se desarrolló en el lenguaje C de Microsoft con la ayuda de una librería de funciones gráficas llamada Metawindows.

Debido al manejo de objetos gráficos se requiere que la PC disponga de un monitor con una resolución de al menos 640 por 480 pixeles.

El contenido del siguiente reporte se ha dividido en los siguientes capítulos:

En el capítulo dos se presentan las definiciones básicas que se utilizaron para el desarrollo del manejador de objetos gráficos incluyendo restricciones establecidas y los sistemas de referencia.

El capítulo tres presenta la metodología de dibujo y describe la forma de construcción y almacenamiento en memoria de los objetos gráficos.

El capítulo cuatro explica la manera en que los objetos gráficos pueden ser almacenados y recuperados de archivos en disco.

El quinto capítulo describe los algoritmos básicos del manejador de objetos gráficos.

El capítulo seis presenta como ejemplo del manejador de objetos gráficos, un editor el cual permite desarrollar esquemas generales de sistemas de manera muy simple. Con

base en este editor se describe especificamente el editor del Sistema Hidráulico de Cutzamala. Las conclusiones del presente trabajo se mencionan en el capítulo siete.

Los apéndices contiene las librerías y listados de los programas del manejador, junto se incluyen los archivos de datos que utiliza el editor de objetos gráficos para el Sistema Cutzamala.

CAPITULO 2

CONCEPTOS BÁSICOS

DEFINICIONES

Un ***objeto gráfico*** se considera como la representación mímica de un objeto en pantalla. Los objetos gráficos que se consideran en el trabajo tiene ciertas características que a continuación se mencionarán.

Los objetos gráficos que se definen son planos y están compuestos de líneas, arcos, etc. unidos de tal manera que representan una figura en especial además estos objetos gráficos se pueden colorear y sombrear.

Cada objeto gráfico tiene asociada una coordenada (x,y) que indica la posición de referencia en la pantalla en la que éste se debe dibujar, la definición de estas coordenadas en cada objeto es seleccionada por el programador y no corresponde necesariamente al centro del objeto, el objeto gráfico también se encuentra relacionado con cuerdas de caracteres que servirán como identificadores del mismo.

Un objeto gráfico puede estar desplegado o no en pantalla lo cual queda especificado con un valor inherente a él.

La unión o colección de varios objetos gráficos se define por medio de ligas secuenciales que unen a cada objeto de la colección con uno anterior y otro posterior (liga padre y liga hijo respectivamente). Así la lista formada por medio de las ligas padre e hijo constituye una ***colección de objetos gráficos*** y puede ser manipulada como un solo objeto gráfico, con características exclusivas de la colección. Por lo tanto se observa que cada objeto gráfico estará relacionado con sus ligas padre e hijo.

SISTEMA DE COORDENADAS

Se cuenta con tres sistemas de coordenadas: globales, virtuales y locales; de los cuales el utilizado en el manejador de objetos gráficos es el de coordenadas virtuales.

El sistema de coordenadas globales se define como aquel que considera el área de dibujo exclusivamente en función del rango de valores que proporciona la resolución de la pantalla que se está manejando, por ejemplo, en el caso de un monitor VGA se tienen valores de 0 a 640 para x y de 0 a 480 para y, teniendo el origen en la esquina superior izquierda. Los valores de x van aumentando de izquierda a derecha y los de y de arriba hacia abajo.

El sistema de coordenas locales se define a través de una translación del origen del sistema de coordenadas globales sobre el sistema global, por lo que tiene el mismo rango de valores posibles para x e y, es decir, de 0 a 640 y de 0 a 480 en el caso de un monitor VGA.

El sistema de coordenadas virtuales a diferencia de los dos anteriores, permite establecer arbitrariamente los rangos de valores para x e y. Por esto permite modificar la resolución del monitor virtualmente.

CAPITULO 3

METODOLOGÍA DE DIBUJO

ESTRUCTURAS DE DATOS

Como ya se mencionó con anterioridad, uno de los objetivos del paquete es buscar la manera de manipular los objetos gráficos, esto implica disponer de una técnica eficiente de almacenar y recuperar un objeto gráfico en y de memoria, respectivamente con cierta facilidad y rapidez. A continuación se describe la forma en que se propone resolver este problema.

Los objetos gráficos tienen características geométricas inherentes a ellos, por ejemplo la posición establecida por un punto (x,y), diámetros, bases, alturas, ángulos de rotación etc. de tal forma que cada elemento gráfico se puede identificar según los parámetros que lo conforman.

Por otro lado en el lenguaje de programación C, existe el concepto de *estructuras*, que se refiere a una colección de una o más variables, de tipos diferentes, agrupadas bajo un solo nombre, el manejo de estas *estructuras* permite tratar variables relacionadas como una unidad en lugar de como entidades separadas. Además variables que forma parte de la estructura se les llama miembros o campos.

Por lo tanto una forma eficiente de representar a un objeto gráfico es por medio de estructuras dotadas con las características del lenguaje C.

El siguiente ejemplo muestra una estructura en lenguaje C para un objeto gráfico arbitrario.

```
typedef struct _algo
{short X;
 short Y;
 short BORRA;
 short PAR1;
 short PAR2;
       .
       .
       .
 short PARN;
 char  NOM[8];
 char  NUM[8];
}algo;
```

En este ejemplo los primeros dos campos corresponden a las coordenadas **X** e **Y** en donde se debe dibujar el objeto, en coordenadas virtuales, el campo llamado BORRA indica si el objeto se debe dibujar o borrar en pantalla. Los demás de estos campos de la estructura son variables y dependen del objeto gráfico en particular. Algunos de los campos que se mencionaron anteriormente son comunes a todos los distintos tipos de estructuras que existen en el manejador.

Este tipo de manejo de información permite utilizar varios objetos gráficos con un tipo de estructura para cada uno de ellos; con lo que se facilita el manejo de varios objetos de un mismo tipo. Así cada objeto gráfico tendrá un tipo de estructura determinado. Además cada objeto gráfico esta dotado de un número de figura que determina el lugar que ocupa en una lista de objetos gráficos con el mismo tipo de estructura.

Como ya se mencionó anteriormente el paquete debe además de manejar objetos gráficos aislados, permitir manipular colecciones de ellos. Esto se puede lograr agregando a cada estructura campos para guardar información útil sobre las colecciones o unión de objetos. En particular se emplean dos campos tipo entero que definen el tipo de estructura y el número de figura del objeto gráfico que sigue (hijo), otros dos campos se utilizan uno de ellos como la dirección del objeto siguiente (hijo) y el otro como la dirección del objeto que le antecede (padre).

Como resulta la forma en que la información de cada objeto se almacena fue posible obtener una generalización en el ordenamiento de los campos de las estructuras, esto es todas las estructuras de los diferentes objetos gráficos tienen primero dos campos de tipo apuntador a enteros que contienen la liga al objeto padre y al objeto hijo respectivamente; luego dos campos enteros para las coordenadas (X,Y) que representan su posición en la pantalla, un campo entero BORRA que indica si esa figura se tiene que dibujar o borrar, y a continuación dos campos enteros que proporcionan información necesaria para calcular la dirección del objeto hijo. Además la estructura contiene un N número definido de enteros en donde los parámetros del objeto en particular son especificados. Como campos terminales de la estructura se seleccionaron arreglos tipo caracter(Char) que almacenan el identificador de cada figura, como por ejemplo su nombre.

Este ordenamiento de enteros y de cuerdas dentro de las estructuras permite entonces localizar una variable entera de una estructura por medio de la posición del entero en el ordenamiento y una cuerda por medio del número total de enteros definidos en la estructura y la posición de la cuerda en el ordenamiento. Además la dirección de una figura, o su estructura, se obtiene al multiplicar el tamaño en octetos de la estructura por el número del elemento definido en la lista de objetos de este tipo de estructura más la dirección de la estructura del primer elemento de la lista.

En el apéndice B se muestran ejemplos de estructuras representando figuras de un sistema hidráulico como son: bombas, tuberías, tanques etc.

CONSTRUCCIÓN DE FUNCIONES

Tomando en cuenta la representación en memoria de los objetos gráficos se decidió definir una función para cada objeto gráfico que se dibuja.

Obviamente se permite anidamiento así que una función pueden utilizar otras previamente definidas. El grupo de funciones gráficas a partir de las cuales se pueden representar figuras más complicadas se denotan primitivas; y ejemplo de ellas son las que dibujan líneas, círculos, rectángulos, etc.

Debido a que la librería de gráficos en C contiene pocas funciones primitivas y están restringidas, se buscó una librería más extensa que se adaptara mejor a las necesidades descritas del manejo.

El código elegido fue un paquete gráfico METAWINDOWS (ref. 2). Este paquete posee un conjunto amplio de funciones, además de permitir realizar aplicaciones de ventanas múltiples mediante el manejo de puertos(ports), con gran rapidez.

Algunas de las propiedades de Metawindows, son las siguientes:

-Tiene distintos estilos de líneas, patrones de llenado y estilos de cursor los cuales pueden ser predefinidos o definidos por el usuario.

-Soporta tres sistemas de referencia: global, local y virtual.

-Maneja textos en forma gráfica a través de caracteres mapeados bit a bit (bitmap) o por dibujo de líneas.

-Proporciona distintos juegos de caracteres, normal, resaltada, itálica, subrayada, tachada, etcétera.

Del amplio conjunto de funciones que posee se tomaron las funciones primitivas que dibujan líneas, rectángulos, círculos, elipses, arcos y polígonos, además de las que permiten el manejo de textos gráficos y de rellenados.

El patrón general que gobiernan las funciones de dibujo es el siguiente: a partir de la estructura asociada a cada objeto gráfico en memoria, las funciones con base en los parámetros de la misma dibujan por medio de las funciones primitivas el objeto gráfico deseado. Además debe de existir una función de dibujo por cada objeto gráfico cuyo único parámetro es un apuntador al tipo de estructura apropiado.

En el apéndice B se pueden observar algunos ejemplos de estas funciones de recuperación para el caso del Sistema Hidráulico de Cutzamala.

INCORPORACIÓN A UN PROGRAMA

Con el fin de integrar las técnicas de almacenamiento y recuperación de objetos gráficos en un programa manejador que permita tratar con esquemas generales con N número de objetos de distintos tipos de estructuras se establecieron las siguientes reglas:

1.- Un esquema general puede tener N objetos distinto cada uno respectivamente con su tipo de estructura y M número de objetos del mismo tipo de estructura.

2.- Es necesario definir un arreglo para cada tipo de estructura existente en el esquema (N arreglos), y en donde cada arreglo contiene además tantos elementos como objetos similares de su tipo de estructura existen (M objetos distintos). Cada arreglo almacenará la información de los objetos que componen al esquema de acuerdo a la siguiente definición:

obj objeto[NMAXOBJ];

donde el tipo de estructura es *obj* que guarda los parámetros del objeto gráfico y NMAXOBJ es el rango máximo de elementos de este arreglo lo cual representa el número máximo de objetos con tipo de estructura obj que pueden existir en el esquema.

En el apéndice B se puede observar la definición de los arreglos de éste tipo, empleados en un sistema hidráulico.

3.- Se deben definir apuntadores a funciones con el fin de utilizar las funciones de dibujo de una manera eficiente .

La regla 3 es fácilmente implementable en C ya que una función por sí sola no es una variable, pero es posible definir apuntadores a funciones, que pueden asignarse, ser colocados en arreglos, o utilizados como parámetros de funciones.

Por ejemplo la definición siguiente:

*void (fun_objeto *)(void *);*

se refiere a que *fun_objeto* es un apuntador a una función que tiene un argumento tipo *void* * el cual no regresa ninguna variable.

De las dos consideraciones anteriores, se puede concluir que dentro del manejador de objetos gráficos si se desea manejar un objeto determinado NFIG con un tipo de estructura EST basta con calcular el apuntador a su estructura correspondiente y mandar llamar a la función respectiva por medio del apuntador a dicha función.

Por ejemplo, supóngase que se desea dibujar la figura *m* del tipo de estructura *obj* entonces el calculo del apuntador a su estructura correspondiente se realiza por medio de:

ap_obj=objeto[m];

donde *objeto[m]* es la dirección de la estructura que corresponde el objeto *m*.

Para mandar llamar a la función de recuperación de un objeto se debe escribir: *fun_obj(ap_obj);*

Con el fin de lograr un manejo mucho más eficiente de los apuntadores cuando existen muchos tipos de objetos es conveniente manejar arreglos de apuntadores a las funciones y a las estructuras.

En un caso general en donde se definió un arreglo de NEST apuntadores a arreglos de estructuras, este arreglo contiene los apuntadores a los arreglos de estructuras existentes dentro del esquema general.

La definición fue de la siguiente manera:

*void *ap_est[NEST];*

lo cual indica que ap_est es un arreglo de apuntadores a variables tipo void, que serán arreglos de estructuras.

Se definió también un arreglo de apuntadores a funciones, el cual contiene los apuntadores a las funciones de dibujo de cada una de las estructuras existentes. La siguiente expresión define este arreglo:

*void (*ap_fig[NEST])(void *);*

el cual es un arreglo de apuntadores a funciones que tienen un argumento tipo apuntador a void (void *) y que no regresan nada por ser de tipo void.

El definir los dos arreglos anteriores fue de gran utilidad pues permitió hacer llamadas a distintas funciones para recuperar distintas estructuras variando únicamente los índices respectivos. De esta forma para dibujar una figura basta con saber el tipo de estructura que es (EST) y que lugar ocupa en su arreglo correspondiente, por ejemplo supóngase que se desea dibujar el *m-ésimo* objeto de tipo de estructura *n* donde EST=n; La llamada a la función de dibujo será de la siguiente forma:

*ap_fig[EST]((void *)apunt);*

donde EST se refiere al apuntador a la función que dibuja el tipo de objeto y *apunt* es el apuntador a la estructura correspondiente, y que ocupa el lugar *m-ésimo* en el arreglo de estructuras tipo EST; en este caso el forzamiento de tipo(cast) que antecede a *apunt* hace que éste sea del tipo *void* *, apunt se puede calcular de la siguiente manera:

*apunt=(int *)ap_est[EST]+tam_est[EST]*m;*

donde *ap_est[EST]* es el apuntador al tipo de estructura que se desea en este caso al tipo EST, *tam_est[]* es un arreglo de enteros que contiene el tamaño de cada estructura en bytes, Esto implica que *tam_est[EST]* representa al tamaño de la estructura EST, y *m* es el número de figura dentro del arreglo de estructuras tipo EST.

4.- Es necesario guardar en memoria la información de cada uno de los conjuntos de objetos gráficos de acuerdo a la organización de arreglos mencionada anteriormente, siendo necesario inicializar dichos arreglos con valores numéricos que se pueden obtener de distintas maneras, por ejemplo, de archivos de datos.

MANEJO DE SECUENCIAS

Como ya se ha mencionado repetidamente en el diseño del manejador se buscó trabajar la colección de objetos gráficos de tal manera que se pudieran manipular como objetos gráficos únicos.

La estructura de datos definida para el manejo de colecciones de objetos gráficos, es similar a un lista doblemente ligada. De nueva cuenta cada objeto gráfico esta unido por medio de dos ligas -liga padre (LIGAP) y liga hijo (LIGAH)- a dos objetos diferentes (objeto padre e hijo respectivamente); donde el *objeto padre es* aquel que antecede dentro de la lista al objeto gráfico del que se trata, y el *objeto hijo* es el que le precede.

Para el manejo de la colección de objetos cada estructura de los objetos gráficos cuenta con cuatro campos específicos para este fin, los dos primeros son los apuntadores a las direcciones de los objetos gráficos padre e hijo respectivamente (LIGAP y LIGAH), los dos últimos contienen el tipo de estructura y el número de objeto en el arreglo de estructura del tipo de estructura del objeto hijo; de tal manera que por medio de ellos es posible calcular la dirección de la estructura del objeto hijo (LIGAH) unido a cada objeto; una vez calculadas estas direcciones se pueden obtener las direcciones a los objetos padres por medio de un ciclo que recorra desde el primer objeto de la colección y le asigne LIGAP igual a *null* hasta el último cuya LIGAH es *null* y le asigne como LIGAP la dirección de la estructura del objeto anterior. De lo se puede deducir que cada colección tiene un objeto que la inicializa (padre de la colección), cuyo valor de la liga padre (LIGAP) es *null;*

Con objeto de poder manipular la colección de objetos de manera versátil, se propusieron tres formas para manejar dicha colección:

1)Manejar objetos aislados que pertenezcan a alguna colección, esto se efectúa por medio de la dirección de las estructura del objeto del que se trata que es calculada usando su tipo de estructura y su número de objeto dentro del arreglo de estructura correspondiente.

2)Manejar todos los objetos de la colección lo cual se realiza por medio de un barrido desde el objeto padre de la colección hasta el último objeto de la misma cuya LIGAH es *null.*

3)Manejar sólo los objetos hijos de una estructura seleccionada, es decir, se trabaja a partir de ésta estructura hasta aquella estructura de la colección cuya LIGAH es *null,* esto se efectúa realizando un barrido desde el objeto seleccionado hasta el último de la colección.

CAPITULO 4

LECTURA Y ESCRITURA DE DATOS DE LOS OBJETOS GRÁFICOS

En los capítulos anteriores se ha descrito la manera en que los objetos gráficos se representan en memoria y como pueden dibujarse o modificarse por medio de estructuras. Sin embargo, no se ha tratado la forma en que la información puede almacenarse o recuperarse en memoria secundaria. El objetivo de este capítulo es describir el uso de archivos de datos por medio de los cuales se pueden almacenar o recuperar los datos de las estructuras.

El espacio en memoria reservado para guardar la información de un esquema general se encuentra organizado en forma de estructuras diferentes, existen N objetos de un mismo tipo de estructura. El primer punto importante para recuperar un esquema general de disco, es el saber cuantos objetos de cada tipo de estructura existen en él, por lo que de un archivo de datos (NUM_FIG) se debe tomar esta información.

Cada estructura que represente a un objeto gráfico de un tipo de estructura determinado estará compuesta por dos campos apuntadores a las figuras hijo y padre,

más E número de enteros dependiendo de la estructura, más C número de cuerdas que servirán como identificadores.

Para que el manejador de objetos gráficos pueda reservar la memoria necesaria para guardar el esquema general es necesario saber el número de enteros y cuerdas que tiene cada estructura, estos valores también se leen de un archivo de datos (NUM_PAR) el cual indica para cada estructura el número de enteros y de cuerdas que la conforman.

Es importante hacer notar que los datos de cada estructura se leen por separado y no todos juntos debido a que se buscó dentro del manejador la mayor generalización posible y que las estructuras no son necesariamente del mismo número de cuerdas y de enteros.

Para almacenar en disco los valores que conforman los objetos de el esquema general, el manejador de objetos gráficos sigue la siguiente estructura:

--Se crea un archivo de datos particular para cada tipo de estructura que existe dentro del esquema general, el cual contiene los parámetros enteros y las cuerdas de cada figura de ese tipo de estructura. El nombre de estos archivos varia según el esquema que se desee tratar, por lo que es necesario que el manejador de objetos gráficos lea de un archivo de datos el nombre de éstos (nom_arch).

Una vez obtenidos los nombres de los archivos de datos donde se encuentran los valores de los parámetros para cada objeto gráfico, es necesario leer estos datos del archivo e ir llenando las localidades de memoria reservadas. Esto con el fin de tener residentes en memoria los parámetros necesarios para recuperar y modificar un objeto gráfico, disminuyendo así el tiempo de acceso..

Es conveniente hacer notar que la organización de estructuras reservada en memoria es de gran utilidad porque permite no accesar archivos en disco para la recuperación y

modificación de objetos, lo cual redunda en una disminución considerable del tiempo de manejo.

La manipulación de colecciones en el manejador también se realiza con la ayuda de algunos archivos de datos.

Un esquema general puede tener N numero de colecciones posibles donde cada colección tiene un objeto padre que es de un tipo de estructura definido (EST) y ocupa un lugar NFIG dentro de su arreglo de estructura correspondiente; estos datos se obtiene directamente de un archivo en disco (coleccion.dat).

Cuando la tarea del manejador de objetos gráficos es recuperar y modificar los parámetros de los objetos toda la información se lee de archivos en disco y se realizan las modificaciones en memoria y no sobre dichos archivos. Por tanto una vez que el usuario ha terminado de utilizar el manejador para trabajar con esquema general es necesario guardar los cambios realizados en memoria; es por ésto que se implantó una fase de escritura en la que toda la información residente en memoria se guarda en los archivos de información inicial.

Para la lectura y escritura de la información se programaron algunas subrutinas que a continuación se mencionan.

Una subrutina que lee N enteros de un archivo de datos *(lee_ent(char *nombre, int *dato, int num_dat, int modo))*; esta subrutina tiene como argumentos un apuntador a caracter que es el nombre del archivo que se va a leer, un apuntador a entero que es la dirección a partir de la cual los enteros que leídos se guardan, un entero que representa el número de enteros a leer y por último un entero (modo) que indica la acción a ejecutar sobre el archivo: modo>0 abrir y leer, modo<0 leer y cierra y modo=0 únicamente lectura.

La manera en que esta subrutina se ejecuta es sencilla. Primeramente se trabaja con el archivo según el modo indicado, si se va a leer se va tomando entero por entero hasta que se terminan de leer los N enteros establecidos de tal forma que por medio de esta función se pueden leer diferentes cantidades de enteros de diferentes archivos, lo cual se acopla a las necesidades del manejador, pues para cada objeto gráfico se tienen diferentes enteros a leer.

También se cuenta con una subrutina para escribir N enteros sobre un archivo *(esc_ent(char *nombre, int *dato, int num_dat, int modo))* la cual cuenta con los mismos argumentos que la subrutina anterior, y escribe enteros de memoria a un archivo. Esta subrutina después de cada entero se escribe un espacio en blanco y los modos de trabajar con el archivo son: apertura y escritura (modo>0), escritura y cierre (modo<0), únicamente escritura (modo=0) .

Se tiene una subrutina de lectura de cuerdas *(lee_cuerda(char *nombre, char *dato, int num_dat, int ancho, int modo))* la cual lee N cuerdas de M número de caracteres; esta subrutina tiene como argumentos a un apuntador a caracter el cual indica el nombre del archivo del que se va a leer, un apuntador a caracter que representa la dirección apartir de la cual se van a guardar los caracteres de las cuerdas leídas, un entero que representa el número de cuerdas que se van a leer, un entero que representa el número de caracteres que deben tener las cuerdas, y un entero (modo) que indica como se va a manejar el archivo si se va a abrir y leer (modo>0), cerrar y leer (modo<0) o únicamente leer (modo=0). Esta subrutina lee del archivo caracter por caracter y lo va colocando en memoria hasta que alcanza el número máximo de caracteres para cada cuerda o hasta que se ha encontrado un fin de línea el cual indica que la cuerda ha finalizado.

Existe también una subrutina diseñada para escribir N cuerdas sobre un archivo *(esc_cuerda(char *nombre, char *dato, int num_dat, int ancho,int modo))*, los argumentos y el procedimiento son similares al de la subrutina de lectura de cuerdas, solo que en esta en vez de leer de un archivo se escribe sobre el.

Es importante hacer notar que las funciones anteriores permiten manejar archivos en los que se tienen tanto enteros como cuerdas, por lo que los formatos de archivos que se pueden manipular son los siguientes.

1)Archivos únicamente de enteros, los cuales están separados por caracteres que C identifica como caracteres separadores como son el espacio en blanco o el de fin de línea.

2)Archivos únicamente de cuerdas, las cuales deben de finalizar con un fin de línea.

3)Archivos de enteros y cuerdas en los que los enteros pueden estar separados por espacios en blanco o de fin de línea y las cuerdas con el caracter fin de línea.

Los archivos que manipula el manejador de objetos gráficos pueden ser creados en cualquier editor de texto por ejemplo el de Turbo Pascal.

CAPITULO 5

IMPLANTACIÓN

En los capítulos anteriores se realizó una descripción de los principios en los que se basa el manejador de objetos gráficos; A continuación se describe el algoritmo seguido para implantar estos principios en el manejador. La implantación se desarrollo en el lenguaje C de Microsoft en conjunto con el paquete de Metawindows.

Este algoritmo puede utilizarse en distintas aplicaciones prácticas del manejador, tal es el caso del despliegue de esquemas generales o de la edición de los mismos. En capítulos posteriores se tratará una aplicación específica.

El algoritmo es el siguiente:

1) Definir el número máximo de objetos para cada tipo de estructura existente en el esquema.

2) Definir las constantes generales que se utilizan dentro del programa.

3) Definir las variables globales.

4) Definir los apuntadores a enteros, caracteres y estructuras *(*apent, *apent, *apcuer, * ap_auxant, *ap_auxpos etc.)* que ayudan al cálculo de las direcciones de las estructuras, de las direcciones de las estructuras de los objetos padres, de las direcciones del primer entero y de la primera cuerda de cada estructura.

5) Definir un arreglo de enteros *(nent_fig[NEST])* con tantos elementos como tipo de estructuras existen en el esquema. Este arreglo sirve para guardar el número de campos tipo entero que tiene cada estructura. Los datos que se almacenarán en este arreglo se leen de un archivo .

6) Definir un arreglo de enteros*(ncuer_fig[NEST])* semejante al arreglo anterior para almacenar el número de cuerdas que tiene cada estructura. Los datos para este arreglo se leen de un archivo .

7) Definir un arreglo de enteros *(n_fig[NEST])* donde se van a almacenar el número de objetos que existen para cada tipo de estructura. Estos datos se leen de archivos.

8) Definir un arreglo de enteros *(fig_ini[NCOLMAX])* con tantos elementos como colecciones dentro del esquema existan, el cual va a almacenar el tipo de estructura que tiene el objeto padre de cada colección.

9) Definir un arreglo de enteros *(tam_est[NEST])* que va a contener el tamaño de cada tipo de estructura.

10) Definir un arreglo de apuntadores a enteros *(*dir_ini[NCOLMAX])* que contendrá las direcciones de las estructuras de los objetos padres de cada colección; existen tantos miembros del arreglo como colecciones hay en el esquema general (NCOLMAX).

11) Definir un arreglo de cadenas de caracteres *nom_dat[NEST][LNOM]* para guardar los nombres de los archivos de datos que contienen los parámetros de los objetos del esquema general.

12) Definir todos los arreglos de estructuras; existen tantos arreglos como tipos de estructuras hay en el esquema general, cada arreglo tendrá tantos elementos como lo indique el número máximo de figuras definido para su tipo de estructura correspondiente.

13) Definir el arreglo de apuntadores a funciones *(*ap_fig[NEST])(void *);* los miembros de este arreglo son apuntadores a funciones con un apuntador a void como único argumento, este arreglo tiene tantos apuntadores como tipos de estructuras existen en el esquema general.

14) Definir un arreglo de apuntadores a elementos tipo void *(*ap_est[NEST])* cuyos elementos son los apuntadores a la dirección de inicio de cada uno de los arreglos de estructuras que se definieron con anterioridad.

15) Definir las funciones que son de gran ayuda para la implantación del manejador, a continuación se describen:

FUNCIÓN LEE_ENT: Esta función lee N enteros de un archivo de datos y los guarda en localidades de memoria a partir de una dirección especifica, el archivo de datos se maneja de tres distintas maneras: modo>0 para abrir y leer, modo<0 para cerrar y leer, modo=0 sólo para leer.

La función lee_ent se define:

*void lee_ent(char *nombre, int *dato, int num_dat, int modo);*

de esta se observa lo siguiente: no regresa ningún valor por ser tipo void; el apuntador a caracteres **nombre* es el nombre del archivo sobre el que se van a leer los enteros; **dato* es la dirección a partir de la cual se deben almacenar los enteros leídos; *num_dat* es el número de enteros a leer; modo indica la manera de trabajar en el archivo.

FUNCIÓN ESC_ENT: Esta función escribe enteros sobre un archivo. Son enteros que se encuentran localizados a partir de una dirección específica. Los modos en los que se trabaja con el archivo son los mismos que en la función anterior. Esta función está definida de la siguiente manera:

*void esc_ent(char *nombre, int *dato, int num_dat, int modo);*

FUNCIÓN LEE_CUERDA: Esta función sirve para leer N cuerdas de un archivo y guardarlas en memoria a partir de una dirección establecida, va leyendo caracter por caracter hasta que encuentra un regreso de línea que indica el fin de la cadena; se define de la siguiente forma:

*void lee_cuerda(char *nombre,char *dato, int num_dat, int ancho, int modo);*

en donde **nombre* es el nombre del archivo donde se va a trabajar, **dato* es la dirección a partir de la cual se van a almacenar las cuerdas, *num_dat* es el número de cuerdas que se van a leer, ancho es el número de caracteres que tiene cada cuerda y *modo* indica la manera en que se va a trabajar el archivo de datos: modo>0 lee y abre, modo<0 cierra y lee, modo=0 sólo lee.

FUNCIÓN ESC_CUERDA: Es similar a la función anterior, sólo que en esta las cuerdas se escribirán sobre un archivo.

Su definición es la siguiente:

*void esc_cuerda(char *nombre, char *dato, int num_dat, int ancho,int modo);*

15) Leer de un archivo de datos el número de enteros y de cuerdas de cada estructura existente en el esquema general y guardar estos datos en los arreglos *nent_fig* y *ncuer_fig* respectivamente.

16) Leer los nombres de los archivos donde se encuentran los parámetros para dibujar los objetos gráficos y guardarlos en el arreglo *nom_dat.*

17) Leer de un archivo de datos el número de figuras de cada tipo de estructura y guardarlos en el arreglo *num_fig.*

18) Inicializar los apuntadores a funciones del arreglo *ap_fig* con las funciones de dibujo de objetos gráficos del esquema general. El manejar este arreglo da la facilidad de poder hacer referencia a las funciones de dibujo por medio de un índice.

19) Inicializar los apuntadores del arreglo *ap_est* con las dirección de inicio de los arreglos de estructuras; estos apuntadores servirán para calcular las direcciones de las estructuras de los objetos del esquema general las cuales dependen del tipo de estructura del objeto y del número de figura que ocupa dentro de su arreglo correspondiente.

20) Inicializar el arreglo *tam_est* con el tamaño de cada una de las estructuras; el tamaño se obtiene por medio de la función de C *"sizeof"* que proporciona el número de bytes que ocupa una estructura de cualquier tipo.

21) Implantar un doble ciclo; el primero para hacer un recorrido para todos los tipos de estructuras existentes y el segundo para recorrer el número de objetos que hay por tipo de estructura. En este doble ciclo se leen de archivos los parámetros de los objetos y se guardan en los espacios de memoria reservados para este fin. También dentro de este ciclo se hace el cálculo de los campos LIGAH que indican la dirección de la estructura hijo para cada objeto.

22) Leer de un archivo el número de colecciones que existen en el esquema, el tipo de estructura y número de figura del objeto inicial de cada colección (objeto padre); inicializar los arreglos dir_ini y fig_ini con las direcciones de las estructuras de los objetos padre y con el tipo de estructura de los objetos padre de cada colección respectivamente.

23) Implementar un ciclo mediante el cual se calcule el campo LIGAP de las estructuras; empezar con la estructura inicial de cada colección la cual tiene LIGAP igual a NULL y posteriormente hacer un recorrido por toda la colección hasta llegar al objeto cuya estructura tiene como campo LIGAH igual a NULL.

CAPITULO 6

APLICACIÓN DEL MANEJADOR DE OBJETOS GRÁFICOS:EDITOR DE OBJETOS GRÁFICOS

EDITOR DE OBJETOS GRÁFICOS

Con base en el manejador descrito anteriormente se implantó un editor de los objetos gráficos para facilitar la definición y modificación de los mismos, en particular se propone utilizar el teclado de la PC para mover los objetos gráficos permitiendo así modificar fácilmente los parámetros asociados al objeto.

En el presente capítulo se describirán las características propias del editor junto con las dos librerías principales, funciones auxiliares que se requieren para su uso y se finaliza el capítulo con el algoritmo propuesto para su implantación.

Las estructuras de los objetos gráficos del editor al igual que las del manejador están constituidas por números enteros, cuerdas y apuntadores a enteros. Todas estas estructuras tienen sus primeros siete campos iguales (LIGAP, LIGAH, X, Y, BORRA, EST, NFIG) los dos primeros son los apuntadores a las estructuras de los objetos padre e hijo respectivamente; De la misma manera que en el manejador X e Y constituyen la coordenada de la pantalla en la que se deben dibujar los objetos; BORRA es el campo que indica si se enciende o apaga el objeto, EST indica cual es el tipo de estructura del

objeto hijo y NFIG es el número que el mismo ocupa dentro de su arreglo correspondiente; Adicionalmente estas estructuras cuentan con otros campos enteros que constituyen los parámetros necesarios para recuperar el objeto, además de un cierto número de cadenas de caractéres que sirven como identificadores del objeto.

Cada objeto gráfico tiene su función de recuperación respectiva programada con la ayuda de otras funciones primitivas propias de METAWINDOWS. Estas funciones de tiene como único argumento un apuntador a la estructura correspondiente en la que se tienen almacenados los parámetros necesarios para recuperar al objeto como por ejemplo: DIAM, BASE, LON, ANG etc.

Como se mencionó anteriormente uno de los fines del editor es el de modificar fácilmente los parámetros del cualquier objeto, en especial la posición. Para lograr ésto basta con hacer referencia a los campos de cada estructura y modificarlos, por ejemplo:

algo->DIAM se refiere al campo DIAM de la estructura tipo algo.

Cuando se desea cambiar la posición de un objeto se deben cambiar los campos X e Y de su estructura. En el programa editor se implantó una función encargada de variar la posición de un objeto según las flechas del teclado. Esta función tiene como único argumento un apuntador a la estructura del objeto que se desea modificar y cuando se presiona alguna flecha el valor de los campos X e Y se incrementa o decrementa.

Al igual que con el manejo de objetos aislados la modificación de la posición de una colección se realiza a través de una función encargada de modificar la posición de la colección de acuerdo con las flechas del teclado; Además la tarea principal de esta función es incrementar o decrementar los campos X e Y de cada estructura de la colección según la tecla oprimida. Esta función tiene dos argumentos, un apuntador a la

dirección de la estructura del objeto padre de la colección y un entero que indica su tipo de estructura.

Para la modificación de los parámetros de los objetos se desarrolló un medio de comunicación interactivo con el usuario por medio del cual se modifican los archivos de datos utilizando el teclado.

Tomando en cuenta las ventajas del lenguaje C se emplean de igual forma que en el manejador, las estructuras por medio de apuntadores. Por ello fue necesario definir dentro del editor una estructura auxiliar que posee los campos comunes a todas las demás estructuras. La estructura propuesta se presenta a continuación:

```
typedef struct_auxiliar
{int *LIGAP;
 int *LIGAH;
 short X;
 short Y;
 short BORRA;
 short EST;
 short NFIG;
 short EDO;
 short DIAM;
 short ANG;
 char NOM[LCUE];
 char NUM[LCUE];
 }auxl;
```

Como resultado de la estructura y forma de manejo del editor, éste se puede utilizar para la elaboración de esquemas de sistemas de varios tipos, en donde es necesario definir un código que varia según el tipo de sistema considerado, este código se realizó tratando de implantar el editor de la forma más general posible.

Se propusieron dos librerías cuya codificación dependen del tipo de sistema del que se trate.

A continuación se describirá el contenido de cada una de estas librerías con el fin de aclarar la información que ellas contienen.

1) LIB1.C

En esta librería se encuentran las definiciones de estructuras, constantes, arreglos y funciones.

a) Definición de los tipos de estructuras del esquema: Estos tipos de estructuras representan los diferentes objetos gráficos que manejará el editor. Un ejemplo de definición de tipo de estructura para un caso general es el siguiente:

```
typedef struct _algo
{short X;
 short Y;
 short BORRA;
 short PAR1;
 short PAR2;
       .
       .
       .
 short PARN;
 char NOM[8];
 char NUM[8];
 }algo;
```

b) Definición de las funciones de dibujo: Esta parte de la librería la constituye una lista de declaraciones de funciones de dibujo de los objetos gráficos. Como ejemplo de la declaración de una función de dibujo se tiene:

```
# SEQ void FrameAlgo(far * algo);
```

En donde el nombre de la función de dibujo es *FrameAlgo* y su único parámetro es un apuntador a la estructura tipo *algo*.

c) Definición del número máximo de objetos diferentes para cada tipo de estructura, número de tipos de estructuras y número de colecciones en el esquema: Estos valores se definen dentro de la librería como constantes que delimitan el número de figuras para cada tipo de estructura; Además se tienen dos constantes, NEST y NCOLMAX las cuales definen el número máximo de tipos de estructuras que manejará el editor y el número de colecciones máximas presentes en el esquema respectivamente. A continuación se muestra como ejemplo la definición del número máximo de figuras de la estructura tipo algo:

#define NMAXALGO 30

d) Definición de arreglos de estructuras: En esta parte de la librería se definen los distintos arreglos de estructuras que almacenan los parámetros de los objetos gráficos del editor. Por tanto habrá tantos arreglos como tipos de estructuras existan en el esquema y cada arreglo tendrá tantos elementos como lo indique en la definición. Como ejemplo de la definición de un arreglo de estructura tipo *algo* tenemos:

algo arr_algo[NMAXALGO];

e) Funciones de recuperación: Además de la definición de los puntos anteriores se incluyó en esta librería el código de las funciones de recuperación. Todas estas funciones manejan los campos de cada estructura y las funciones primitivas de dibujo incluidas en el paquete de Metawindows para construir la representación del objeto gráfico a recuperar. Si se desea implantar el editor para un sistema determinado es necesario codificar estas funciones a criterio del programador con objeto de lograr la representación de un objeto.

2) LIB2.C

Esta librería contiene la inicialización de cada uno de los arreglos que se utilizan en la implantación del editor -ap_fig, ap_est, tam_est-.

a) Inicialización de los apuntadores a funciones: Como se ha escrito ya, existe un arreglo de apuntadores a funciones cuyos elementos sirven para hacer referencia a las funciones de recuperación. Este arreglo de apuntadores -ap_fig- se inicializa con las funciones de recuperación que tiene el editor en cuestión (funciones cuyo código ha sido definido previamente en LIB1.C); Este arreglo tendrá tantos elementos como tipos de estructuras (NEST) maneje el editor y la inicialización de cada elemento se hará de la siguiente manera:

ap_fig[n]=FrameAlgo;

donde *n* va de cero hasta NEST menos uno.

b) Inicialización de apuntadores a arreglos de estructuras: Dentro de esta parte de la librería se inicializa el arreglo de apuntadores a arreglos de estructuras. En la librería LIB1.C se encuentra la definición de arreglos de estructuras que contendrán los parámetros de los objetos gráficos del esquema general; Esta inicialización se lleva a cabo de la siguiente manera:

ap_est[n]=arr_algo;

donde *n* va desde cero hasta NEST (tipos de estructuras) menos uno y *arr_algo* es el arreglo de estructuras tipo *algo.*

c) Inicialización del arreglo de tamaños de estructuras: Este arreglo contiene el tamaño en bytes de cada estructura que maneja el editor, y se inicializa con la función de C *sizeof* siguiente manera:

tam_est[n]=sizeof(algo);

donde *n* va de cero hasta NEST menos uno y *algo* es el tipo de estructura del que se desea conocer su tamaño en bytes.

Por último se describirán las funciones auxiliares que se desarrollaron para la implantación del editor de objetos gráficos.

FUNCIÓN RESALTA: Esta función se utiliza cuando se modifican datos desplegados en pantalla. Así se resalta por inversión de la pantalla el rectángulo donde se va ha encontrar el texto a modificar y deja la pluma color negro preparada para pintar sobre el fondo invertido. Su definición es la siguientete:

void resalta(int xi ni, int yini, int deltay, int ancho, int pos_act,int color);

Esta definición indica que la función resalta no regresa ningún valor por ser tipo *void*, sus argumentos *xi ni* y *yini* forman la coordenada de una de las esquinas del rectángulo invertido; *deltay* indica el ancho del rectángulo; *ancho* es el largo del rectángulo; *pos_act* es el número de linea que se esta modificando dentro de N número de líneas que pueden ser modificadas sobre una misma pantalla; *color* es el color original del fondo del rectángulo invertido.

FUNCIÓN REGRESA: Esta función regresa al color original un rectángulo invertido y deja la pluma preparada para escribir sobre un fondo negro. La definición de esta función es la siguiente:

void regresa(int xi ni, int yini, int deltay, int ancho, int pos_act, int color);

Los argumentos son iguales a los de la función resalta.

FUNCIÓN LEECAR: Esta función lee un caracter ascii del teclado mediante la interrupción 24 de sistema operativo. La definición de esta función es la siguiente:

*void leecar(int *ah, int *al);*

No regresa ningún valor, cuenta con dos argumentos apuntadores a enteros, el primero **ah* representa la parte alta de el código ascii del caracter leído y el segundo **al*, representa la parte baja del caracter.

FUNCIÓN BORRA: Esta función apaga caracteres sobre la pantalla apartir de una posición dada respetando el estado de la pluma, se define de la siguiente manera:

void borra(int x, int y, int campo);

No regresa ningún valor por ser de tipo *void*; los enteros *x* e *y* representan la posición a partir de la cual se apagan (borran) los caractéres; el argumento campo es el número de caracteres a borrar.

FUNCIÓN SI: Esta función obtiene una respuesta si o no del operador, además de hacer eco de la tecla oprimida. Su definición es la siguiente:

int si(void);

Regresa un valor entero, este valor es cero si la respuesta es negativa y uno en caso contrario; Por tener un argumento tipo *void* para llamar esta función no es necesario pasar ningún argumento.

FUNCIÓN DIBUJA_COLECCION: Esta subrutina dibuja una colección de objetos gráficos unidos en forma de una lista doblemente ligada, parte del objeto inicial llamado objeto padre y dibuja (enciende) todos los objetos que le suceden hasta encontrar a aquel cuya liga es nula.

Su definición sigue el siguiente formato:

*void dibuja_colección(int *e_ini, int f_ini);*

Por ser de tipo void no regresa ningún valor, su argumento **e_ini* es la dirección de la estructura del objeto padre de la colección y *f_ini* es el tipo de esta estructura.

FUNCIÓN P_COLECCION: Esta función determina: a) La dirección de la estructura inicial (estructura del objeto padre) de la colección a la que pertenece la figura. b) El tipo de estructura del objeto padre *(*fig_p)*; se define de la siguiente manera:

*void p_colección(int *apent, int **ap_p, int *fig_p);*

y no regresa ningún valor; **apent* es la dirección de la figura que pertenece a la colección de la que se desean conocer los datos, ***ap_p* es el argumento que contendrá la dirección obtenida de la estructura del objeto padre y **fig_p* es su tipo de estructura.

FUNCIÓN INC_X: Esta función incrementa el valor que tiene el campo de posición X de todas las estructuras de una colección que le suceden a la estructura cuya dirección se pasa como argumento de esta función. Su definición esta dada por:

*void inc_x(auxi *auxiliar, int incx);*

en donde el argumento **auxiliar* indica la dirección de la estructura a partir de la cual se hacen las modificaciones, es conveniente hacer notar que aunque este apuntador se define del tipo *auxi* es posible pasar cualquier otro tipo de estructura y hacer referencia al campo X por medio de *auxiliar->X*, debido a que los primeros campos son los mismos en todas las estructuras .

El argumento incx es el valor que se tiene que sumar al campo X.

FUNCIÓN INC_Y: Esta función incrementa o decrementa el valor de *incy* al campo Y, de la misma forma que se hace en la función anterior. Su definición se forma como:

*void inc_y(auxi *auxiliar, int incy);*

FUNCIÓN CAMBIA_BORRA: Esta función modifica el campo BORRA de las estructuras de una colección apartir de la estructura cuya dirección se pasa como argumento, la definición esta dada por:

*void (auxi *auxiliar, int borra);*

Y el argumento **auxiliar* es la dirección de la estructura a partir de la cual se empieza a modificar los campos BORRA; el argumento borra es el valor que se asigna a *auxiliar->BORRA* de cada estructura.

FUNCIÓN CUERDA: Esta función obtiene una cadena de caracteres del teclado y hace eco en la pantalla gráfica, una cuerda termina cuando se teclea un regreso de línea. La definición de esta función es:

void cuerda(int x, int y, int campo);

en donde *x* e *y* definen la coordenada de la pantalla gráfica donde se empezará a obtener la cadena de caracteres; el argumento *campo* indica la longitud de la cadena. La cuerda que se obtiene por medio de esta función es almacenada en una variable global definida como un arreglo de caracteres.

FUNCIÓN NUM_ENT: La función num_ent obtiene un número entero por medio de la concatenación de caractéres válidos 0 al 9, +,- , en modo gráfico de la pantalla. La función es la siguiente:

int num_ent(int x, int y, int campo);

Y regresa el entero concatenado; los argumento *x* e *y* constituyen la coordenada en la pantalla gráfica donde se empezará a capturar el número; el argumento *campo* indica el máximo número de caracteres válidos.

FUNCIÓN MODIFICA_FIGURA: Esta función sirve para controlar el movimiento de una figura en la pantalla, permite modificar la posición de un elemento seleccionado por medio de las flechas del teclado, además de modificar los campos X e Y de la estructura del objeto con el que se está trabajando; Esta dotada para variar el incremento en X y en Y por medio de las teclas + y - del teclado. La función se define :

*void mofica_figura(auxi *auxiliar);*

Y no regresa ningún valor, su argumento indica la dirección de la estructura del objeto gráfico a modificar y es posible modificar figuras de distinto tipo que *auxi* debido a que los campos X e Y están presentes al principio de todas las estructuras.

FUNCIÓN MODIFICA_ENTERO: Esta función permite modificar N números enteros que se encuentran desplegados en pantalla en renglones consecutivos, modifica el valor de los mismos en memoria y controla el resaltado de los rectángulos en los que se encuentran dichos números. La función se define:

*void modifica_ent(int *apunt, int num_dat, int xi ni, int yini, int deltay, int ancho, int pos_num, int campo);*

en donde **apunt* es el apuntador a la dirección del primer entero a modificar; *num_dat* es el número de enteros que se van a modificar; *xi ni* e *yini* forman la coordenada en pantalla donde va a empezar el rectángulo de resaltado para modificar los enteros; *deltay* es el ancho del rectángulo de resaltado; *ancho* es el largo del rectángulo; *pos_num* es el valor de la coordenada X conteniendo números a modificar; y *campo* es el número de caracteres que tienen dichos números.

FUNCIÓN MODIFICA_CUERDA: Esta función realiza la modificación de N cuerdas tanto en la pantalla como en memoria. La función se define de la siguiente manera:

*void modifica_cuerda(char *apunt, int num_dat, int xi ni, int yini, int deltay, int ancho, int pos_num, int campo);*

Donde los argumentos son los mismos que en la función anterior, a excepción de **apunt* que es la dirección a partir de la cual se encuentran guardadas las *num_dat* cadenas a modificar.

FUNCIÓN MODIFICA_COL: Esta función realiza una tarea semejante a la de modifica_fig sólo que trabaja con colecciones de objetos gráficos, permite mover las colecciones por medio de las flechas del teclado; Utiliza las teclas de + y - para modificar el número de pixeles que se incrementarán o decrementarán a los campos X e Y de la estructura. Su definición es:

*void modifica_col(auxi *auxiliar, int fig_p);*

Y no regresa ningún valor, su argumento **auxiliar* indica la dirección de la estructura a partir de la cual se va a modificar la colección; en algunos casos esta estructura es la del objeto padre y en otros es la de un objeto cualquiera de la colección a partir del cual se desean hacer cambios la colección. La definición de la función especifica una estructura tipo auxiliar, sin embargo, debido a que los cambios para modificar un objeto se efectúan sobre los primeros campos comunes a todas las estructuras es posible hacer los cambios en cualquier tipo de estructura sin que ésta sea necesariamente tipo *auxi*.

FUNCIÓN DESPLIEGA_LET: Esta función lee N cuerdas de un archivo y las despliega en pantalla en renglones consecutivos. La función se define:

*int despliega_let(char *nombre, int num_ren, int xi ni, int yini, int deltay);*

en donde **nombre* es el nombre del archivo de datos del que se leen las cuerdas; *num_ren* es el número de cuerdas que se leen y despliegan; *xi ni e yini* constituyen la coordenada en la que se va empezar a escribir la primera cuerda; *deltay* es el incremento de *yini* para seguir escribiendo las cuerdas posteriores.

FUNCIÓN DESPLIEGA_CUERDA: Esta función es semejante a la anterior, sólo que no leerá las cuerdas de un archivo sino de memoria a partir de una dirección especificada.

La definición de la función es como sigue:

*int despliega_cuerda(char *dato, int num_dat, int xi ni, yini, int deltay, int ancho);*

en donde **dato* es la dirección a partir de la cual se encuentran almacenadas las cuerdas a desplegar; *num_dat* es el número de cuerdas a desplegar; *xi ni e yini* constituyen la coordenada para desplegar la primera cuerda; *deltay* es el incremento que se da a yini para desplegar las siguientes cuerdas; *ancho* indica el número de caracteres de cada cuerda.

FUNCIÓN DESPLIEGA_ENT: Es muy parecida a la función anterior y sólo trabaja con números enteros. La definición es:

*int despliega_ent(int *dato, int num_dat, int xi ni, int yini, int deltay);*

en donde **dato* es la dirección a partir de la cual se encuentran los números a desplegar y los demás argumentos son los mismos que en la función anterior.

El algoritmo del editor de objetos gráficos es básicamente el mismo del manejador de objetos gráficos y difiere en algunas definiciones y asignaciones que a continuación se mencionan:

1) Definir una estructura tipo auxiliar, mostrada anteriormente, que sirva para el manejo de apuntadores.

2) Diseñar las dos librerías -LIB1.C y LIB2.C-. lib1.c debe de ir colocada en la sección de definiciones del sistema y lib2.c dentro del cuerpo del programa principal.

3) Leer los nombres de los archivos de pantallas de despliegue y guardarlos en un arreglo de cadenas de caracteres previamente definido (nom_des.dat).

4) Inicializar el modo gráfico de la pantalla y se establecer los límites en coordenadas virtuales que se van a manejar para la edición del esquema.

5)Por medio de la implantación de un ciclo se debe preguntar por el tipo de estructura y el número de figura del objeto con el que se desee trabajar; el editor con ayuda de las pantallas de despliegue pregunta al usuario si desea hacer alguna modificación de los campos del objeto gráfico. En caso afirmativo se podrán realizar por medio del teclado, y se han terminado las modificaciones el editor pregunta por los límites en X e Y que definirán los rangos de la pantalla, estos valores establecen el área de trabajo con el manejo de coordenadas virtuales; con estos márgenes establecidos se recuperan todos los objetos del esquema mediante un ciclo dentro del cual se calcula el apuntador a las estructuras de cada objeto y se manda llamar a la función de recuperación indicada por medio del arreglo de apuntadores a funciones; Después comienza el ciclo de modificación de la posición de los objetos gráficos por medio de las flechas del teclado. En este momento se tienen 3 opciones de modificación: modificar sólo el objeto elegido, modificar toda la colección a la que pertenece dicho objeto; modificar dicho objeto y todos los que están ligados a el por medio de la LIGAH (objetos hijo). Es conveniente mencionar que todos los cambios hechos en pantalla también se realizan en memoria por lo que al terminar este ciclo los parámetros de las estructuras ya han sido actualizadas.

6) Una vez que se tienen las modificaciones de los objetos hechas en memoria se implantó otro ciclo para guardar la información en los archivos de datos.

A continuación se presenta un ejemplo del editor para el caso de sistemas hidráulicos:

EDITOR PARA EL SIMULADOR DEL SISTEMA CUTZAMALA.

Como aplicación del editor de objetos gráficos se realizó su implantación para el sistema hidráulico de Cutzamala.

Por medio de este editor se elaboró un esquema que incluye los elementos básicos del sistema, plantas de bombeo, canales, presas, tuberías, etcétera.

A continuación se mostrará como se codificaron las librerías LIB1.C Y LIB2.C.

1) LIB1.C

a) El Editor para el esquema del sistema Cutzamala maneja básicamente trece objetos gráficos: bombas tipo 1 y 2, filtros, tuberías (canales, túneles, tuberías), codos, presas, tanques, vertedores, plantas, válvulas tipo 1, 2 y 3, y letreros.

En el apéndice A se puede observar la definición de las estructuras para éstos objetos gráficos; estas estructuras son las siguientes: bom (tipo bomba), fil (tipo filtro), tu (tipo tubería), codo (tipo codo), pre (tipo presa), tan (tipo tanque), vert (tipo vertedor), plan (tipo planta), val (tipo válvula) y let (tipo letrero).

b) En la parte de definición de las funciones de recuperación sólo se listan dichas funciones con sus parámetros correspondientes.

c) Para las trece estructuras que existen en el editor del sistema Cutzamala se definió un número máximo de objetos para cada uno de ellos.

El esquema general del Sistema Cutzamala cuenta con 36 bombas tipo 1, 6 filtros, 178 tuberías, 83 codos, 6 presas, 15 tanques, 2 vertedores, 7 plantas, 36 válvulas tipo 1, y 4 válvulas tipo 2.

La definición de estas constantes se muestra en el apéndice A.

d) Cada una de las estructuras definidas cuenta con un arreglo que tiene tantos elementos como la constante de máximo de objetos para cada una de ellas lo especifica.

La lista de arreglos se anexa en el apéndice A.

e) Las funciones de recuperación que se programaron para el editor del sistema Cutzamala son las siguientes:

FrameB1(bom *NOMBRE):

FrameB2(bom *NOMBRE):

FrameFil(fil *NOMBRE):

FrameTu(tu *NOMBRE):

FrameCodo(codo *NOMBRE):

FramePre(pre *NOMBRE):

FrameTan(tan *NOMBRE):

FrameVert(vert *NOMBRE):

FramePlan(plan *NOMBRE):

FrameVl1(val *NOMBRE):

FrameVl2(val *NOMBRE):

FrameVl3(val *NOMBRE):

FrameLet(let *NOMBRE).

En el apéndice A se pueden observar estas rutinas de recuperación.

La codificación de estas funciones es la parte más laboriosa de la implantación pues requiere que el programador siga ciertos criterios definidos para la construcción de los objetos.

2) LIB2.C

Esta librería se muestra en el apéndice B.

a) El arreglo de apuntadores a funciones consta de trece elementos por ser trece los objetos que maneja el editor del sistema Cutzamala y se inicializa con las funciones de recuperación cuyo código se definió en la librería LIB1.C.

b) El arreglo de apuntadores a estructuras se inicializa igualando cada uno de sus elementos con el arreglo de estructura que le corresponde.

c) El arreglo de tamaños de estructuras se inicializa con la medida en bytes de cada estructura, obtenida por medio de la función *sizeof.*

A continuación se mencionarán los archivos de datos que se utilizan en el editor del sistema Cutzamala:

num_par.dat: Contiene el número de enteros y de cuerdas para cada una de las trece estructuras.

nom_arch.dat: Contiene los nombres de los archivos donde se encuentran los parámetros de cada objeto gráfico y los nombres de los archivos con las pantallas de despliegue. Los archivos que contiene los parámetros son los siguientes:

b1_dat.dat: parámetros de las bombas tipo 1, b2_dat.dat: parámetros de las bombas tipo 2, fil_dat.dat: parámetros de los filtros. tu_dat.dat: parámetros de tuberías, túneles y canales, codo_dat.dat: parámetros de codos, Pre_dat.dat: parámetros de presas, tan_dat.dat: parámetros de tanques. vert_dat.dat: parámetros de vertedores, plan_dat.dat: parámetros de plantas, val1_dat.dat: parámetros de válvulas tipo 1, val2_dat.dat: parámetros de válvulas tipo 2. val3_dat.dat: parámetros de válvulas tipo 3.

Los archivos de las pantallas de despliegue son los siguientes: b1_des.dat, fil_des.dat, tu_des.dat, codo_des.dat, pre_des.dat, tan_Des.dat, vert_des.dat, plan_des.dat, val_des.dat.

num_fig.dat: Contiene el número de figuras para cada tipo de estructura.

coleccion_dat.dat: Contiene el número de colecciones que existen en el esquema, el tipo de estructura del objeto padre y su número de figura para cada colección

figuras.dat: Contiene la pantalla de despliegue para preguntar por el objeto que se desea seleccionar.

edit_des.dat: Contiene la pantalla de despliegue para preguntar por los rangos en X e Y que delimitan el área de trabajo.

pregun1.dat: Contiene la pantalla de despliegue para preguntar por el tipo de tarea que se desea realizar con el objeto elegido: modificar únicamente el objeto, modificar toda su colección, modificar sólo los objetos ligados a el.

El ejemplo de la construcción de algunos de los archivos anteriores se pueden observar en el apéndice C.

El esquema del sistema Cutzamala cuenta con varias vistas fundamentales. Algunas de estas se presenta en el apéndice D.

CONCLUSIONES

Con el desarrollo del manipulador se obtuvo una técnica eficiente para el manejo de objetos gráficos; La principal ventaja del manejador es que la forma en que cada objeto gráfico se representa en memoria, el manejo de las funciones de dibujo y de las estructuras de los objetos por medio de apuntadores simplifica y generaliza su concepción.

Entre sus principales aplicaciones esta la edición de esquemas y el despliegue animado.

Otra característica del manejador de objetos gráficos es la representación en memoria para cada objeto gráfico y el poder modificar su parámetros.

Esto permite manejar un mismo objeto gráfico con diferentes características y afectar mímicos y variables en el tiempo real.

El manejador de objetos gráficos puede trabajar con varios tipos de sistemas, sin embargo para cada sistema es necesario desarrollar dos librerías LIB1.C y LIB2.C.

Como trabajo posterior se sugiere dotar al manejador de librerías para el manejo de objetos gráficos de los sistemas, más comunes en ingeniería como en el caso de sistemas hidráulicos, mecánicos, eléctricos etc.

REFERENCIAS

1.- Luis Alvarez Icaza, Griselda Gutiérrez Izquierdo, Irma Laura Camacho. Simulador de la Operación del Sistema Cutzamala. Informa del Instituto de Ingeniería. UNAM. Elaborado para la Comisión de Aguas del Valle de México. SARH. México D.F. Junio 1992.165 pp.

2.- Metagraphics Software Co. Metawindow versión 3: Reference Manual. Metagraphics Software Co. USA:1989. 259 pp.

APÉNDICE A

1.- Listado del Editor para el Sistema Cutzamala.

2.- Listado de función auxiliar funcione.C.

3.- Listado de función auxiliar Rect.C

1.-LISTADO DEL EDITOR DEL SISTEMA CUTZAMALA

```
/*utiliza campos de ligas a padres y a hijos para el manejo de las*/
/*** Maneja Colecciones de figuras incluyendo la figura de letrero  /*** colecciones
                **/
#include <dos.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <Grconst.h>
#include <Grports.h>
#include <Grextrn.h>
#include <math.h>
metaPort virtual;
penState *estado;        /* apuntador para guardar  y recuperar
                         el edo. de la pluma antes y
                  despues del dibujo de cada
        figura*/
rect scrnR,                /* rectangulo por omisión, coordenadas máximas */
    marca,                 /* rectangulo para resaltado */
    vista;             /* rectangulo para dibujo */
#include "lib1.c"
/**definición de estructura auxiliar para manejo de apuntadores, con los mismos
campos en común que todas las demás estructuras de objetos gráficos*/
typedef struct _auxiliar
 { int * LIGAP;
   int * LIGAH;
   short X;
   short Y;
   short BORRA;
   short EST;
   short NFIG;
   short EDO;
   short DIAM;
   short ANG;
   char  NOM[LCUE];
   char  NUM[LCUE];
   }auxi;
#define NULO     0              /* valor numérico de algunos ASCII */
#define BELL     7
#define REGRE    8
#define LF       10
```

```
#define CR        13
#define ESC       27
#define LINULA    38
#define MAS       43
#define MENOS     45
#define PUNTO     46
#define ARRI      72
#define IZQ       75
#define DER       77
#define ENEMAY    78
#define ABA       80
#define ESEMAY    83
#define ENEMIN    110
#define ESEMIN    115
#define MAXCOL    79
#define LCUER     11
#define LNOM      16
#define XGLMIN    0
#define XGLMAX    5000
#define YGLMIN    0
#define YGLMAX    3000
int  ascii_al, ascii_ah,    /* auxiliares para lectura de teclado */
    c,i,j,k,l,m,n,o,        /* auxiliares */
    *apent,*apent1,             /* apuntadores para los datos a leer */
    incx=1,                 /* incrementos de x e y para mov. de figs. */
    incy=1,
    *ap_pa,fig_p,               /*apuntador a la estructura padre y tipo de estructura*/
    nent_fig[NEST],             /* numero de parámetros enteros en cada figura */
    ncuer_fig[NEST],            /* numero de cuerdas en cada figura */
    n_fig[NEST],        /*numero de figuras distintas en cada estructura */
    fig_ini[NCOLMAX],       /*arreglo de figuras que inician cada colección*/
    tam_est[NEST+1];            /*arreglo de tamano de las estructuras */
int  *ap_auxest,                  /*apuntador auxiliar para calculo de
                            apuntador a nodo hijo*/
    *dir_ini[NCOLMAX],      /*arreglo de apuntadores a figuras de inicio
                de cada*/
    **liga;         /* coleccion */
int  diferencia;            /*diferencia para calcular el numero de
        bytes que ocupa*/
                                /* el apuntador liga padre e hijo*/
int  resp;                  /* varible auxiliar*/
auxi  *ap_auxant;
int  *ap_auxpos;        /* apuntador aux que senala a la estructura
                    hijo*/
auxi *est_aux;          /* apuntador auxiliar a estructura*/
```

```
char *apcuer,                    /* apuntador a cuerdas */
   linea[40],            /* linea para modificaciones */
   nom_dat[NEST][LNOM], /*nombres de archivos de dato de cada
                          estructura*/
   nom_des[NEST][LNOM];
void (*ap_fig[NEST])(void *), /*apuntador a las funciones*/
   *ap_est[NEST+1],       /*apuntador a las estructuras */
   *apest;
FILE *fp;                /* apuntador a archivos */
/*********************************************************************
 Resalta por inversion el rectangulo donde se puede hacer la
 modificacion.  Deja la pluma preparada para pintar negro sobre
 fondo coloreado.                                          */
void resalta(int xini, int yini, int deltay, int ancho, int pos_act, int color)
{
int  x0,x1,y0,y1;                    /*auxiliares*/
x0=xini-deltay/4;
x1=xini+ancho+deltay/4;
y0=yini+deltay*(pos_act-1)+deltay/4;
y1=yini+deltay*(pos_act)+deltay/4;
SetRect(&marca,x0,y0,x1,y1);   /* rectangulo para marcado */
InvertRect(&marca);
PenColor(0); BackColor(color);
}
/*********************************************************************
 Invierte el resaltado del cuadrado donde se pudo hacer la
 modificacion.  Deja la pluma preparada para pintar coloreado sobre
 fondo negro.                                              */
void regresa(int xini, int yini, int deltay, int ancho, int pos_act, int color)
{
int  x0,x1,y0,y1;                    /*auxiliares*/
x0=xini-deltay/4;
x1=xini+ancho+deltay/4;
y0=yini+deltay*(pos_act-1)+deltay/4;
y1=yini+deltay*(pos_act)+deltay/4;
SetRect(&marca,x0,y0,x1,y1);  /* rectangulo para marcado */
InvertRect(&marca);
PenColor(color); BackColor(0);
}
/*********************************************************
Objetivo: leer un caracter ascii del teclado mediante
 la interrupcion 24 del sistema operativo      */
void  leecar(int *ah,int *al)     /* utiliza declaracion de dos.h */
{ union REGS inregs,outregs;
```

```
 inregs.x.ax = 0;              /* limpia acumulador a antes de
                            entrar */
 int86(0x16,&inregs,&outregs);        /* llama a la interrupción */
 *ah = outregs.x.ax / 256;      /* obtiene la respuesta */
 *al = outregs.x.ax % 256;
}
/*************************************************************
 Objetivo: borrar "campo" espacios a partir de x, y
 Respeta la situación de la pluma */
void borra(int x, int y, int campo)
{
int  i;                                  /*auxiliar*/
for(i=0;i<campo;i++) linea[i]=' '; /* borra espacio de despliegue */
linea[campo]='\0';
MoveTo(x,y);
DrawString(linea);
linea[0]='\0';                    /* borra cadena */
MoveTo(x,y);
}
/*************************************************************
Objetivo: obtener una respuesta "s" o "n" del operador
Nota: la pluma debe llegar posicionada */
int si(void)
{
int  correcto=0;
do
 {
 leecar(&ascii_ah,&ascii_al);       /* lee caracter */
 switch(ascii_al)
  {
   case ENEMAY:                     /* acepta N, S, n, s */
   case ESEMAY:
   case ENEMIN:
   case ESEMIN:
    correcto=1;
    DrawChar((char)ascii_al);
    break;
   default:
    putchar(BELL);          /* en todos los demas casos no acepta el car. */
    break;
  }; /*switch*/
 } while (!correcto);
return ascii_al;                  /* regresa valor */
} /*si*/
/*dibuja una coleccion de figuras empezando desde la figura padre */
```

```
void dibuja_coleccion(int *e_ini,int f_ini)
{
 ap_auxant=(auxi *)e_ini;
 ap_fig[f_ini]((int *)ap_auxant);
 ap_auxpos=(int *)ap_auxant->LIGAH;
 /*dibuja cada coleccion completa hasta que termine la liga*/
 while (ap_auxpos!=NULL)
   {
   /* dibuja la figura de la coleccion que corresponde*/
   ap_fig[ap_auxant->EST]((int *)ap_auxpos);
   ap_auxant=(auxi *)ap_auxpos;
   ap_auxpos=(int *)ap_auxant->LIGAH;
   }
}/**dibuja_coleccion*/
/*encuentra la direccion de inicio y el tipo de figura de inicio del padre de la coleccion a
la que pertenece la figura que se pasa como parametro*/
void p_coleccion(int *apent,int **ap_p,int *fig_p)
{
 int cont,aux;
 est_aux=(auxi *)apent;
 while (est_aux!=NULL)
   {
   *ap_p=(int *)est_aux;
   est_aux=(auxi *)est_aux->LIGAP;
   }
 aux=0;
 cont=1;
 while (cont)
 {
   if(*ap_p==dir_ini[aux])
     cont=0;
   else
     aux++;
 }
*fig_p=fig_ini[aux];
}/**for (o=0..*/
/**incrementa en X una coleccion completa*/
void inc_x(auxi *auxiliar,int incx)
{
 ap_auxant=auxiliar;
 ap_auxant->X+=incx;
 est_aux=(auxi *)ap_auxant->LIGAH;
 while (est_aux!=NULL)
   {
   est_aux->X+=incx;
```

```
  ap_auxant=est_aux;
  est_aux=(auxi *)ap_auxant->LIGAH;
  }
}
/**incrementa en Y una coleccion completa*/
void inc_y(auxi *auxiliar,int incy)
{
 ap_auxant=auxiliar;
 ap_auxant->Y+=incy;
 est_aux=(auxi *)ap_auxant->LIGAH;
 while (est_aux!=NULL)
  {
  est_aux->Y+=incy;
  ap_auxant=est_aux;
  est_aux=(auxi *)ap_auxant->LIGAH;
  }
}
/*cambia el campo borra segun el parametro "borra" en una
  coleccion completa*/
void cambia_borra(auxi *auxiliar,int borra)
{
 ap_auxant=auxiliar;
 ap_auxant->BORRA=borra;
 est_aux=(auxi *)ap_auxant->LIGAH;
 while (est_aux!=NULL)
  {
  est_aux->BORRA=borra;
  ap_auxant=est_aux;
  est_aux=(auxi *)ap_auxant->LIGAH;
  }
}
/*****************************************************************
Objetivo: obtener una cuerda de caracteres. El cursor se coloca
en x,y; se preparan campo caracteres para la lectura        */
void cuerda(int x, int y, int campo)
{
int   bueno=1, ncar=0, signo=1, punto=0, ent= 0;
char  c;
borra(x, y, campo);  /* borra espacio para escritura */
while (bueno)              /*sigue mientras no haya regreso de carro */
 {
  leecar(&ascii_ah,&ascii_al);    / * lee caracter */
  c = (char)ascii_al;                /* lo pasa a char */
  if (!((ascii_al == CR) ||(ascii_al == REGRE) || (ncar>=campo)))
```

```
    {strncat(linea,&c,1);      /*lo guarda en linea para posible
                                 borrado */
    DrawChar((char)ascii_al); /* eco */
    ncar++;}
   switch(ascii_al)
    {
    case REGRE:                  /* Regreso, backspace */
     if(ncar)
      { linea[ncar-1]= ' ';     /* despliega espacio donde se
                                   borro */
        MoveTo(x,y);           DrawString(linea);
        linea[ncar-1]='\0';     /*ajusta longitud de linea */
        MoveTo(x,y);
        DrawString(linea);       /* reposiciona pluma */
        ncar--;                    /* ajusta numero de
                                   caracteres */
      } /*if (ncar)*/
     break;
    case CR:                     /* Regreso de carro, car return */
     bueno = 0;                  /* levanta bandera para terminacion */
     break;
    default:
     break;
   }; /*switch*/
 };/*while (bueno)*/
 } /*cuerda*/
 /*************************************************************
 Objetivo: obtener un numero entero a partir de la concatenacion
  de caracteres validos (0..9,+,-)                 */
 int num_ent(int x, int y, int campo)
 {
 int  bueno=1, ncar=0, signo=1, punto=0, ent= 0;
 char  c;
 borra(x, y, campo); /* borra espacio para escritura */
 while (bueno)          /* sigue mientras no haya regreso de carro */
  {
  leecar(&ascii_ah,&ascii_al);   /* lee caracter */
  c = (char)ascii_al;            /* lo pasa a char */
  if (ascii_al != 0 && isdigit(ascii_al)) /*si es numero lo
                                          procesa */
  {  ent = ent*10 + ascii_al - '0';        /*acumula numero */
     strncat(linea,&c,1);                   /* lo guarda en
                                         linea para posible
                                    borrado */
     DrawChar((char)ascii_al);              /* eco */
```

```
  ncar++;
}
else                                    /* si no es numero lo
                                        interpreta */
 switch(ascii_al)  {
  case REGRE:                           /* Regreso, backspace */
   if(ncar)
    { if(isdigit(linea[ncar-1]))
       ent = (ent - linea[ncar-1] + '0')/10;  /* ajusta
                                                 numero */
      linea[ncar-1]= ' ';
                                        /* despliega espacio
                                        donde se borro*/
      MoveTo(x,y);
      DrawString(linea);
      linea[ncar-1]='\0';     /*ajusta longitud de linea */
      MoveTo(x,y);
      DrawString(linea);  /* reposiciona pluma */
      ncar--;                 /* ajusta numero de caracteres */
    } /*if (ncar)*/
   break;
  case CR:                          /* Regreso de carro,car return */
   bueno = 0;                           /* levanta bandera para terminacion */
   break;
  case MENOS:                           /* signo - */
   if(!ncar)                        /* procesa solo si no hay carac. previos */
    { signo = -1;                       /* cambia signo */
      strncat(linea,&c,1);          /* guarda, desp. y ajusta cont. de car. */
      DrawChar((char)ascii_al);
      ncar++;
    } /*if (!ncar)*/
   else putchar(BELL);      /*campana cuando hay numeros previos */
   break;
  case MAS:                             /*signo + */
   if(!ncar)      /*procesa solo si no hay carac. previos */
    { strncat(linea,&c,1);              /* guarda, desp. y ajusta
                                    cont. de car. */
      DrawChar((char)ascii_al);
      ncar++;
    } /*if (!ncar)*/
   else putchar(BELL);      /*campana cuando hay numeros previos */
   break;
  default:
   putchar(BELL);           /*en todos los demas casos no acepta
                          el car. */
```

```
    break;
   }; /*switch*/
};/*while (bueno)*/
return ent*signo;       /* regresa valor */
} /*num_ent*/


/*************************************************
 Lee num_dat enteros del archivo apuntado por
 *nombre, los regresa apuntados por *dato
 El archivo se maneja segun el parametro modo
 modo>0 => apertura y lectura; modo<0 => cierre y lectura
 modo=0 => unicamente lectura */
int lee_ent(char *nombre,int *dato, int num_dat, int modo)
{
int i,j;                                    /* auxiliar */
if (modo > 0 ) fp = fopen(nombre,"r");  /* abre archivo con datos */
for (i=0; i<num_dat; i++)
 {  fscanf(fp,"%d",dato++); j=getc(fp);}/* lee de archivo con
                                           formato */
if (modo < 0) i = fclose(fp);               /* cierra archivo */
return(0); }
/*************************************************
 Escribe num_dat enteros del archivo apuntado por
 *nombre, los regresa apuntados por *dato
 El archivo se maneja segun el parametro modo
 modo>0 => apertura y escritura; modo<0 => cierre y escritura
 modo=0 => unicamente escritura */
int esc_ent(char *nombre,int *dato,int num_dat,int modo)
{
int i;                                      /* auxiliar */
if (modo > 0 ) fp = fopen(nombre,"w");  /* abre archivo con datos */
for (i=0; i<num_dat; i++)
 { if (i!=(num_dat-1))
    {fprintf(fp,"%d ",*dato); /*escribe en archivo con espacio */
     dato++;}
  else
    {fprintf(fp,"%d\n",*dato);/* escribe en archivo con nueva
                                 linea */
     dato++;}
 }
if (modo < 0) i = fclose(fp);   /* cierra archivo */
return(0); }
/*************************************************
 Lee num_dat cuerdas del archivo apuntado por
```

```
*nombre, las regresa apuntadas por *dato.
Cada cuerda tiene ancho caracteres.
El archivo se maneja segun el parametro modo
modo>0 => apertura y lectura; modo<0 => cierre y lectura
modo=0 => unicamente lectura. */
int lee_cuerda(char *nombre,char *dato, int num_dat, int ancho, int modo)
{
int  i,j;                                   /*auxiliar */
int c;
char *a;
if (modo>0) fp = fopen(nombre,"r");   /* abre archivo con datos */
for (i=0; i<num_dat; i++)
 {
  a=dato;
  for (j=0; j<ancho-2; j++)
   {
    c=getc(fp);
    if (c==LF)break;
    *dato=(char)c; dato++;
   }
  *dato=NULO;
  dato=a+ancho;
 }
if (modo<0) i = fclose(fp);                    /* cierra archivo */
return(0); }
/*************************************************
Escribe num_dat cuerdas del archivo apuntado por
*nombre, las regresa apuntadas por *dato.
Cada cuerda tiene ancho caracteres.
El archivo se maneja segun el parametro modo
modo>0 => apertura y escritura; modo<0 => cierre y escritura
modo=0 => unicamente escritura. */
int esc_cuerda(char *nombre,char *dato, int num_dat, int ancho, int modo)
{
int  i;                                         /* auxiliar */
if (modo>0) fp = fopen(nombre,"w"); /* abre archivo con datos */
for (i=0; i<num_dat; i++)
 { fprintf(fp,"%s\n",dato);
   dato=dato+ancho;}                            /* escribe en archivo con
                                                formato */
if (modo<0) i = fclose(fp);                     /* cierra archivo */
return(0); }
/*********************************************************************
Objetivo: controlar el movimiento en el despliegue de una figura */
void modifica_fig(auxi *auxiliar)
```

```
{
do
 {
 leecar(&ascii_ah,&ascii_al);                    /* lee caracter */
 if(!ascii_al)
  {
   switch(ascii_ah)
    {
     case ARRI:
      auxiliar->BORRA=1;
      ap_fig[j]((void *)auxiliar);
      auxiliar->Y-=incy;
      auxiliar->BORRA=0;
      ap_fig[j]((void *)auxiliar);
      break;
     case ABA:
      auxiliar->BORRA=1;
      ap_fig[j]((void *)auxiliar);
      auxiliar->Y+=incy;
      auxiliar->BORRA=0;
      ap_fig[j]((void *)auxiliar);
      break;
     case IZQ:
      auxiliar->BORRA=1;
      ap_fig[j]((void *)auxiliar);
      auxiliar->X-=incx;
      auxiliar->BORRA=0;
      ap_fig[j]((void *)auxiliar);
      break;
     case DER:
      auxiliar->BORRA=1;
      ap_fig[j]((void *)auxiliar);
      auxiliar->X+=incx;
      auxiliar->BORRA=0;
      ap_fig[j]((void *)auxiliar);
      break;
     default:
      putchar(BELL);
      break;
    } /*switch y ascii_al = 0*/
  }
 else
  {
   switch(ascii_al)
    {
```

```
    case MAS:
     if (incx < 100) incx=incx*10;
     if (incy < 100) incy=incy*10;
     break;
    case MENOS:
     if (incx > 1) incx=incx/10;
     if (incy > 1) incy=incy/10;
     break;
    case ESC:
     break;
    default:
     putchar(BELL);
     break;
   } /*switch ascii_al */
  } /* else !ascii_al */
 }
while (!(ascii_ah == 1 && ascii_al == ESC));
}
/*****************************************************************
 Objetivo: controlar el resaltado y modificacion de numeros */
void modifica_ent(int *apun,int num_dat, int xini, int yini, int deltay,int ancho, int
pos_num, int campo)
{
int  color=15,i=0;                              /* auxiliares */
resalta(xini, yini, deltay, ancho, i, color);   /* resalta primera
                                    posicion */
do
 {
 leecar(&ascii_ah,&ascii_al);                   /* lee caracter */
 if(ascii_al == 13)                             /*si <CR> borra espacio y lee num */
   *apun=num_ent(pos_num,yini+deltay*i,campo);
 else
   {
   if(!ascii_al)
    {
    switch(ascii_ah)
      {
      case ARRI:
       {regresa(xini, yini, deltay, ancho, i, color);
        if (i!=0) {i=i-1; apun -= 1;}
        else {i=num_dat-1; apun+= num_dat-1;};
        resalta(xini, yini, deltay, ancho, i, color);};
       break;
      case ABA:
       {regresa(xini, yini, deltay, ancho, i, color);
```

```
        if(i==(num_dat-1)){i=0; apun -= num_dat-1;}
        else {i=i+1; apun += 1;};
        resalta(xini, yini, deltay, ancho, i, color);};
        break;
       default:
        putchar(BELL);
        break;
      } /*switch y ascii_al = 0*/
    }
   else if(ascii_al != ESC) putchar(BELL);
  }
 }
while (!(ascii_ah == 1 && ascii_al == ESC));

regresa(xini, yini, deltay, ancho, i, color); /* regresa ultima
                                                 posicion */
}
/*****************************************************************
 Objetivo: controlar el resaltado y modificacion de cuerdas */
void modifica_cuerda(char *apun,int num_dat, int xini, int yini, int deltay,int ancho, int
pos_num, int campo)
{
int  color=15,i=0;                          /* auxiliares */
resalta(xini, yini, deltay, ancho, i, color);     /* resalta primera
                                posicion */
do
 {
 leecar(&ascii_ah,&ascii_al);                 /* lee caracter */
 if(ascii_al == 13)                           /*si <CR> borra espacio y lee cuerda */
   {cuerda(pos_num,yini+deltay*i,campo);
   strcpy(apun,linea); }
 else
   {
   if(!ascii_al)    /*dice que los caracteres son flechas*/
    {
    switch(ascii_ah)
      {
      case ARRI:
       {regresa(xini, yini, deltay, ancho, i, color);
       if (i!=0) {i=i-1; apun -= campo;}
       else {i=num_dat-1; apun+= campo*(num_dat-1);};
       resalta(xini, yini, deltay, ancho, i, color);};
       break;
      case ABA:
       {regresa(xini, yini, deltay, ancho, i, color);
```

```
        if(i==(num_dat-1)){i=0; apun -= campo*(num_dat-1);}
        else {i=i+1; apun += campo;};
        resalta(xini, yini, deltay, ancho, i, color);};
        break;
       default:
        putchar(BELL);
        break;
      } /*switch y ascii_al = 0*/
    }
   else if(ascii_al != ESC) putchar(BELL);
  }
 }
while (!(ascii_ah == 1 && ascii_al == ESC));
regresa(xini, yini, deltay, ancho, i, color);          /* regresa ultima posicion */
}
/***modifica una coleccion completa de elementos.**/
void modifica_col(auxi *auxiliar,int fig_p)
{
do
 {
 leecar(&ascii_ah,&ascii_al);  /* lee caracter */
 if(!ascii_al)
  {
  switch(ascii_ah)
   {
    case ARRI:
     cambia_borra(auxiliar,1);
     dibuja_coleccion((int *)auxiliar,fig_p);
     inc_y(auxiliar,-incy);
     cambia_borra(auxiliar,0);
     dibuja_coleccion((int *)auxiliar,fig_p);
     break;
    case ABA:
     cambia_borra(auxiliar,1);
     dibuja_coleccion((int *)auxiliar,fig_p);
     inc_y(auxiliar,incy);
     cambia_borra(auxiliar,0);
     dibuja_coleccion((int *)auxiliar,fig_p);
     break;
    case IZQ:
     cambia_borra(auxiliar,1);
     dibuja_coleccion((int *)auxiliar,fig_p);
     inc_x(auxiliar,-incx);
     cambia_borra(auxiliar,0);
     dibuja_coleccion((int *)auxiliar,fig_p);
```

```
      break;
    case DER:
      cambia_borra(auxiliar,1);
      dibuja_coleccion((int *)auxiliar,fig_p);
      inc_x(auxiliar,incx);
      cambia_borra(auxiliar,0);
      dibuja_coleccion((int *)auxiliar,fig_p);
      break;
    default:
      putchar(BELL);
      break;
    } /*switch y ascii_al = 0*/
  }
  else
  {
   switch(ascii_al)
    {
    case MAS:
      if (incx < 100) incx=incx*10;
      if (incy < 100) incy=incy*10;
      break;
    case MENOS:
      if (incx > 1) incx=incx/10;
      if (incy > 1) incy=incy/10;
      break;
    case ESC:
      break;
    default:
      putchar(BELL);
      break;
    } /*switch ascii_al */
  } /* else !ascii_al */
 }
while (!(ascii_ah == 1 && ascii_al == ESC));
}
/*************************************************************************/
/* Lee num_ren renglones del archivo apuntado por nombre. Escribe
   el primer renglon en (xini, yini) y suma deltay a yini para cada
   uno de los que sigue. */
int despliega_let(char *nombre, int num_ren, int xini,
                  int yini, int deltay)
{
FILE  *fp;
int  i;                    /* auxiliar */
char linea[MAXCOL+2],              /* linea para despliegue */
```

```
   * lin;                          /* apuntador a a la linea
                                corriente de pagina */
i=SystemFont(16);                       /* selecciona el font para escritura */
fp = fopen(nombre,"r");                 /* abre archivo con datos */
for (i=0; i<num_ren; i++)       /* lo escribe renglon a renglon */
 { lin = fgets(linea, MAXCOL, fp);
   linea[strlen(linea)-1]='\0';
   MoveTo(xini,yini);
   DrawString(linea);
   yini= yini + deltay; }
i = fclose(fp);                         /* cierra archivo */
return(0); }                            /* regreso provisional */
/************************************************/
int despliega_cuerda(char *dato, int num_dat,int xini, int yini, int deltay, int ancho)
{
char linea[MAXCOL+2];                   /* linea para despliegue */
int  i;                         /* auxiliar */
i=SystemFont(16);               /* selecciona el font para
                        escritura */
for (i=0; i<num_dat; i++)       /*lo escribe renglon a renglon */
 {  sprintf(linea,"%s",dato);    /* copia numero a cadena */
   MoveTo(xini,yini);           /* posiciona cursor */
   DrawString(linea);           /* escribe */
   dato = dato + ancho;         /* apunta al siguiente dato */
   yini= yini + deltay; }       /* nueva posicion del cursor */
return(0); }                            /* regreso provisional */
/************************************************/
int despliega_ent(int *dato, int num_dat, int xini, int yini, int deltay)
{
char linea[MAXCOL+2];                           /* linea para despliegue */
int  i;                                 /* auxiliar */
i=SystemFont(16);                       /* selecciona el font para
                                escritura */
for (i=0; i<num_dat; i++)                       /* lo escribe renglon a
                                renglon */
 {  sprintf(linea,"%d",*dato);                  /* copia numero a cadena */
   MoveTo(xini,yini);                   /* posiciona cursor */
   DrawString(linea);                   /* escribe */
   dato ++;                             /*apunta al siguiente dato */
   yini= yini + deltay; }               /* nueva posicion del cursor */
return(0); }                            /* regreso provisional */
/*********************************************
/************************************************/
void main (void)
{
```

```
#include "lib2.c"  /**libreria de definicion de apuntadores**/

i=QueryGrafix();                    /* inicia metawindows */
i=InitGrafix(-EGA640x480);              /* monitor VGA */
ScreenRect(&scrnR);                 /* rectangulo a toda la
                                    pantalla */
SetDisplay(GrafPg0);                /* pagina cero de
                                    despliege*/
EraseRect(&scrnR);                      /* borra pantalla */
InitPort(&virtual);
SetPort(&virtual);
PortOrigin(1);
SetOrigin(0,0);
SetRect(&scrnR,XGLMIN,YGLMIN,XGLMAX,YGLMAX);
VirtualRect(&scrnR);
PenColor(15);BackColor(0);              /* inicia colores */
i=SystemFont(16);                   /* selecciona el font para
                                escritura */
DupRect(&scrnR,&vista);     /* Copia las coordenas
                                globales a las de       dibujo */
/* Lee el numero de parametros enteros y de cuerdas para cada
   tipo de estructura */
apent = nent_fig;       /* apunta al arreglo de num. de enteros */
i = lee_ent("num6_par.dat",apent,NEST,1);  /* abre y lee archivo */
apent = ncuer_fig;      /* apunta al arreglo de num. de cuerdas */
i = lee_ent("num6_par.dat",apent,NEST,-1); /* lee y cierra archivo */
/*Lee el nombre de los archivos con los datos para cada tipo de estructura*/
apcuer = &nom_dat[0][0];    /*apunta a los nombres de los archivos
                        de datos */
i = lee_cuerda("nom5_arch.dat",apcuer,NEST,LNOM,1);/ * abre y lee
                        archivo */
apcuer = &nom_des[0][0];    /*apunta a nombres de archivos
                        de despliegues */
i = lee_cuerda("nom5_arch.dat",apcuer,NEST,LNOM,-1);  /*lee y cierra
                        archivo*/
/* Lee el numero de objetos para cada tipo de estructura */
apent = n_fig;                          /* apunta al arreglo de num. de figuras */
i = lee_ent("num5_fig.dat",apent,NEST,1);  /* abre y lee archivo */
i = lee_ent("num5_fig.dat",apent,0,-1);    /* cierra archivo */
/* Lee los datos del numero de figuras de cada tipo de estructura */
for (j=0; j<NEST; j++)   /* controla el numero de estructura */
  {
  i = lee_ent(nom_dat[j],apent,0,1);  /* abre archivo sin leer datos */
  for (k=0; k<n_fig[j]; k++)            /* controla el numero de figura */
   {
```

```
/*se calcula el numero de bytes que ocupa el campo liga en cada estructura*/
    diferencia=tam_est[j]-nent_fig[j]*2-ncuer_fig[j]*LCUER;
/*la direccion del primer entero es la direccion del inicio de la
  estructura mas el tamano de cada miembro por el numero de figura
  en curso.*/
/*mas el tamano del campo liga Todo expresado en palabras */
    apent = (int *)ap_est[j]+tam_est[j]*k/2+diferencia/2;
 /*la direccion de la primera cuerda es la direccion
 del inicio de la estructura mas el tamano de la estructura por
 el numero de figura en curso mas el tamano de los enteros de
 la estructura mas el tamano del campo liga.Todo en bytes */
    apcuer = (char*)ap_est[j]+tam_est[j]*k+nent_fig[j]*2+diferencia;
    i = lee_ent(nom_dat[j],apent,nent_fig[j],0);                    /*lee datos enteros
*/
    i = lee_cuerda(nom_dat[j],apcuer,ncuer_fig[j],LCUER,0);        /*lee cuerdas*/
 /*se calcula el campo liga para cada figura, es la direccion de la*/
 /*figura hijo*/
   apent=(int *)apent-diferencia/2;    /*es el inicio de cada
                                        figura*/
    est_aux=(auxi *)apent;
    est_aux->LIGAH=(int *)ap_est[est_aux->EST]+tam_est[est_aux->EST]*(est_aux-
>NFIG-1)/2;
   }
  i = lee_ent(nom_dat[j],apent,0,-1);/* cierra archivo sin leer
                                      datos */
  }
/*Inicia lectura del numero de colecciones de objetos y tipo de estructura y figuras que
inician cada coleccion*/
i = lee_ent("coleccion.dat",apent,0,1);          /* abre archivo sin leer
                                                 datos */
apent=&l;
i = lee_ent("coleccion.dat",apent,1,0);
for (o=0;o<l;o++)      /* se hace para l numero de colecciones*/
  {
 /** se leen los datos de estructura y numero de figura*/
  apent=&j;
 /* lee tipo de estructura inicial*/
  i = lee_ent("coleccion.dat",apent,1,0);
       apent=&k;
  i = lee_ent("coleccion.dat",apent,1,0);       /*lee numero de figura inicial*/
  /**se calcula la direccion de inicio de cada coleccion*/
  dir_ini[o]=(int *)ap_est[j]+tam_est[j]*(k-1)/2;
  fig_ini[o]=j;
  } /**for (o=0...*/
```

```
i = lee_ent("coleccion.dat",apent,0,-1);        /*cierra archivo sin leer
          datos */
/*a continuacion se calculan las direcciones de las figuras padres
para cada figura de las colecciones que existen**/
for (o=0;o<l;o++)
{
 ap_auxant=(auxi *)dir_ini[o];
 ap_auxant->LIGAP=NULL;
 est_aux=(auxi *)ap_auxant->LIGAH;
while (est_aux!=NULL)
  {
  est_aux->LIGAP=(int *)ap_auxant;
  ap_auxant=est_aux;
  est_aux=(auxi *)ap_auxant->LIGAH;
  }
}/**for (o=0..*/
/*Inicia el ciclo de control de seleccion de estructura y numero de figura, modificacion
de datos y despliegue con movimiento auxiliado por cursores ya sea de una sola figura
o por colecciones*/
j=-1;                    /* inicia indicador para leer num. de est. */
/*repita hasta que se seleccione j=NEST que fuerza salida con break*/
while (1)
 {                       /*while (j<0..)*/
  EraseRect(&scrnR); /* borra pantalla */
  /*despliega las posibles figuras a dibujar */
  i = despliega_let("figura6.dat",18,400,300,120);
  apent = n_fig;          /* apunta al arreglo de numero de figuras */
  /* despliega los num. de fig. validos */
  i = despliega_ent(apent,NEST,3000,300,120);
  j=num_ent(2600,2100,5);    /* obtiene el num. de est. */
  if (j>=0 && j<=NEST-1)  /* verifica si el num. de est. es valido */
   {                          /*if(j>=0..)*/
   /*verifica si existen mas de cero figuras del tipo de estructura*/
    if(n_fig[j] > 0)
     {          /*if(n_fig..)*/
      k=-1;                  /* inicia indicador para num. de fig. */
      while (k<0 || k>n_fig[j])
       {                     /*while(k<0..)*/
        k=num_ent(2600,2340,5);       /* obtiene el num. de fig. */
        if(k> 0 && k<=n_fig[j])  /* verifica num.de fig. valido */
         {                   /*if(k>0..)*/
          /*incia ciclo para preguntar por tarea*/
          resp=0;
          EraseRect(&scrnR);    /* borra pantalla */
          /* pregunta si se va a modificarla figura,la coleccion
```

```
            a la que pertenece y sus hijos*/
      l = despliega_let("pregun1.dat",4,500,1500,120);
      while(resp<=0 || resp>3)
      {
      resp=num_ent(2100,1500,5);  /* obtiene la respuesta */
      if(resp<=0 || resp>3) putchar(BELL);
      }                                          /*while(resp...*/
      apent = (int *)ap_est[j]+tam_est[j]*(k-1)/2;
      /**busca la direccion del padre de la coleccion*/
      /* a la que pertenece la figura y la regresa en ap_p*/
      p_coleccion(apent,&ap_pa,&fig_p);
      EraseRect(&scrnR);     /* borra pantalla */
      l = despliega_let(nom_des[j],22,400,300,120);
      apent = (int *)ap_est[j]+tam_est[j]*(k-1)/2+diferencia/2;
      l = despliega_ent(apent,nent_fig[j],2500,300,120);
      apcuer =
       (char*)ap_est[j]+tam_est[j]*(k)+nent_fig[j]*2+diferencia;
l=despliega_cuerda(apcuer,ncuer_fig[j],2500,300+120*nent_fig[j],120,LCUER);
      MoveTo(3100,2700);
      c=si(); c=tolower(c);    /* acepta respuesta y pasa a minusculas*/
      if (c=='s') modifica_ent(apent,nent_fig[j],400,300,120,2800,2500,5);
      MoveTo(3100,2820);
      c=si(); c=tolower(c);    /* acepta respuesta y pasa a
                        minusculas*/
      if (c=='s')
       modifica_cuerda(apcuer,ncuer_fig[j],400,300+120*nent_fig[j],120
,2900,2500,L_CUE);
      EraseRect(&scrnR);     /* borra pantalla */
      /* despliega las coordenadas de dibujo */
      l = despliega_let("edit_des.dat",10,400,300,120);
      apent1 = (int *)&vista;/*apunta al rect. correspondiente */
      /*despliega las coordenadas de dibujo*/
      l = despliega_ent(apent1,4,2000,300,120);
      MoveTo(3100,900);
      c=si(); c=tolower(c);    /* acepta respuesta y pasa a
                        minusculas*/
                                    /* modif. rec. de vista */
      if (c=='s') modifica_ent(apent1,4,400,300,120,2300,2000,5);
      EraseRect(&scrnR);     /* borra pantalla */
      VirtualRect(&vista);    /* cambia coordenadas a las de
                        dibujo */
      /* Dibuja todas las estructuras */
      for (m=0; m<NEST; m++)/*controla el numero de estructura */
       {                /*for(m..)*/
```

```
        for (n=0; n<n_fig[m]; n++) /* controla el numero de
                                      figura */
{                                   /*for(n..)*/
/*la direccion del primer entero es la direccion del inicio
 de la estructura mas el tamano de cada miembro por el numero
 de figura en cuerso. Todo expresado en palabras */
          apent1 = (int *)ap_est[m]+tam_est[m]*n/2;
          ap_fig[m](apent1);
        }                                   /*for(n..)*/
      }                                     /*for(m..)*/
      apent=(int *)apent-diferencia/2;
      switch(resp)
      {
      case 1:
          modifica_fig((auxi *)apent);
          break;
      case 2:
          modifica_col((auxi *)ap_pa,fig_p);
          break;
      case 3:
          modifica_col((auxi *)apent,j);
          break;
      default:
          break;
      }/*switch...*/
      /* if(resp==1..*/
      VirtualRect(&scrnR);    /* regresa a coordenadas
                              globales */
    } /*if(k>0..)*/
    else putchar(BELL);
   } /*while(k<0..)*/
  } /*if(n_fig..)*/
 } /*if(j>=0..)*/
 else
 /*fuerza la terminacion de iteracion sobre num. de est. */
  if (j==NEST) break;
  else putchar(BELL);                   /*no se selecciono num. valido de est,va por otro
*/
 } /*while (j<0..)*/
/*Escribe los datos del numero de figuras por tipo de estructura */
for (j=0; j<NEST; j++)          /* controla el numero de estructura */
  {
  i = esc_ent(nom_dat[j],apent,0,1);    /* abre archivo sin escribir datos */
  for (k=0; k<n_fig[j]; k++)            /* controla el numero de figura */
  {
```

```
/*la direccion del primer entero es la direccion del
inicio de la estructura mas el tamano de cada miembro por el numero de figura en
cuerso. Todo expresado en palabras */
    diferencia=tam_est[j]-nent_fig[j]*2-ncuer_fig[j]*LCUER;
    apent = (int *)ap_est[j]+tam_est[j]*k/2+diferencia/2;
/*la direccion de la primera cuerda es la direccion del inicio
 de la estructura mas el tamano de la estructura por el numero
 de figura en curso mas el tamano de los enteros de la estructura.
 Todo expresado en bytes */
    apcuer = (char *)ap_est[j]+tam_est[j]*k+nent_fig[j]*2+diferencia;
     /* escribe datos enteros */
    i = esc_ent(nom_dat[j],apent,nent_fig[j],0);
    /* escribe cuerdas */
    i = esc_cuerda(nom_dat[j],apcuer,ncuer_fig[j],LCUER,0);
    }
 /* cierra archivo sin escribir datos */
  i = esc_ent(nom_dat[j],apent,0,-1);
  }
VirtualRect(&scrnR);                    /* regresa a coordenadas globales */
SetDisplay(TextPg0);          /* regresa despliegue a condiciones normales */
i=QueryError();               /* obtiene e imprime codigo de error */
printf("QueryError=%d/%d\n", i >>7,i & 127);
getchar();
}
```

2. -LISTADO DE FUNCIÓN AUXILIAR FUNCIONE.C

```
#include "math.h"
void tranf(punto,tipo,ang,delta,x,y)
point * punto, * delta;
short  tipo,ang,x,y;
{
 double angy;
 punto->X=0;
 punto->Y=0;
 x = x - delta->X;
 y = y - delta->Y;
 switch (tipo) {
  case 1:
  /* MATRIZ DE ROTACION CON UN ANGULO TETA(GRADOS) */
   angy = (3.14159 * ang)/180.0;
    punto->X = (short) (x*cos(angy) + y*sin(angy) + 0.5);
    punto->Y = (short) (-x*sin(angy) + y*cos(angy)+ 0.5);
    break;
  case 2:
  /* MATRIZ DE TRASLACION */
    punto->X = x+delta->X;
```

```
    punto->Y = y+delta->Y;
    break;
  case 3:
   /*  MATRIZ DE ESCALAMIENTO AL DOBLE  */
    punto->X = x * delta->X;
    punto->Y = y * delta->Y;
    break;
 };   /***** end del switch  *******/
 punto->X = punto->X + delta->X;
 punto->Y = punto->Y + delta->Y;
}
```

3.-LISTADO DE FUNCIÓN AUXILIAR RECT.C

```
/*dibuja un los contornos redondeados de un rectangulo*/
void FrameRectRound(rect *R,int diamx, int diamy)
{
rect aux;  /*variables auxiliares*/
int x1,y1,x2,y2;
x1=R->Xmin+diamx/2;
y1=R->Ymin+diamy/2;
x2=R->Xmax-diamx/2;
y2=R->Ymax-diamy/2;
/*a continuacion dibuja las lineas rectas de el rectangulo*/
MoveTo(x1,R->Ymin);
LineTo(x2,R->Ymin);
MoveTo(R->Xmax,y1);
LineTo(R->Xmax,y2);
MoveTo(x2,R->Ymax);
LineTo(x1,R->Ymax);
MoveTo(R->Xmin,y2);
LineTo(R->Xmin,y1);
/*se define el rectangulo que sirve de auxiliar para dibujar el arco superior izquierdo */
SetRect(&aux,R->Xmin,R->Ymin,R->Xmin+diamx,R->Ymin+diamy);
/*a continuacion se dibujan el arco superior izquierdo*/
FrameArc(&aux,900,900);
/*se define el rectangulo que sirve de auxiliar para dibujar el arco superior derecho */
SetRect(&aux,R->Xmax-diamx,R->Ymin,R->Xmax,R->Ymin+diamy);
FrameArc(&aux,0,900);
/*se define el rectangulo que sirve de auxiliar para dibujar el arco inferior izquierdo*/
SetRect(&aux,R->Xmin,R->Ymax-diamy,R->Xmin+diamx,R->Ymax);
FrameArc(&aux,1800,900);
/*se define el rectangulo que sirve de auxiliar para dibujar el arco inferior izquierdo */
SetRect(&aux,R->Xmax-diamx,R->Ymax-diamy,R->Xmax,R->Ymax);
/*a continuacion se dibujan el arco superior izquierdo*/
FrameArc(&aux,2700,900);
}/*end frameRectRound*/
```

APÉNDICE B

1.- Listado de librería LIB1.C

2.- Listado de la librería LIB2.C

1.-LISTADO DE LIBRERÍA LIB1.C

```
/*  definicion de estructuras  de objetos gráficos */
#define MicrosoftC  1   /* 5.0, 5.1   */
/* Language Fixups */
#if     MicrosoftC
#define  FAR_malloc _fmalloc    /* Must use 'far' pointers */
#define  FAR_free   _ffree      /*or MetaWINDOW bitmaps. */
#endif   /*MicrosoftC*/
#define LCUE   11
typedef struct _val
 {  int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short EDO;
    short DIAM;
    short ANG;
    char  NOM[LCUE];
    char  NUM[LCUE];
    }val;
typedef struct _T1
  { short long1;
    short long2;
    short X,Y;
  }T1;
typedef struct _bom
  { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short EDO;
    short DIAM;
```

```
    short ANG;
    char  NOM[LCUE];
    char  NUM[LCUE];
   }bom;
typedef struct _tub
   { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short EDO;
    short DIAM;
    short LON;
    short ANGT;
    short TIPO;
    char  NOM[LCUE];
    char  NUM[LCUE];
   }tub;
typedef struct _vert
   { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short EDO;
    short BASE;
    short ALT;
    short TIPO;
    short TECHO;
    char  NOM[LCUE];
    char  NUM[LCUE];
   }vert;

typedef struct _pres
   { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
```

```
    short EDO;
    short ALT;
    short TIPO;
    char  NOM[LCUE];
    char  NUM[LCUE];
    }pres;
  typedef struct _plan
   { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short BASE;
    short ALT;
    char  NOM[LCUE];
    char  NUM[LCUE];
    }plan;
  typedef struct _filt
   { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short EDO;
    short BASE;
    short ALT;
    short TIPO;
    char  NOM[LCUE];
    char  NUM[LCUE];
    }filt;
  typedef struct _tanq
   { int  * LIGAP;
    int  * LIGAH;
    short X;
    short Y;
    short BORRA;
    short EST;
    short NFIG;
    short EDO;
    short BASE;
    short ALT;
```

```
  short TIPO;
  short N_CON;
  short POS_CON[3];/*este arreglo esta formado por la posicion y
                          /* el lado en que se quiere la conexion
                      izquierdo(1) derecho(2),abajo(3) */
  short DIAM_CON[3];
  char  NOM[LCUE];
  char  NUM[LCUE];
 }tanq;
typedef struct _codo
 { int  * LIGAP;
  int  * LIGAH;
  point V1;
  short BORRA;
  short EST;
  short NFIG;
  short EDO;
  point V2;
  point V3;
  short TIPO;   /**1 para rectangulito y 2 para trapecio **/
 }cod;

typedef struct _letrero
 {int  * LIGAP;
 int  * LIGAH;
 short X,
     Y,
     BORRA,
     EST,
     NFIG,
     IDENT,          /*valor entre 0.011 y 1.00*/
     TAM;
 char  CADENA[LCUE];
 char  NUM[LCUE];
 }letrero;

/** definicion de funciones de recuperacion **/

#if    m_MSDOS
#define SEQ far
#endif/* m_MSDOS */
void  SEQ  SetLet(letrero far, short,short,short,short,short,char *);
void  SEQ  SetT  (T1 far *, short,short,short,short);
void  SEQ  SetBom(bom far *, short,short,short,short,short,short,
                        char*,char *);
```

```
void SEQ SetVal(val far *, short,short,short,short,short,short,
char *,char *);
void SEQ SetTanq(tan far *, short,short,short,short,short,
short,short,short,short, short, char *,char * );
void SEQ SetTub(tub far *, short,short,short,short,short,short,
short,short,char*,char *);
void SEQ SetCodo(cod far *, point,short,short,point,point,short);
void SEQ SetVert(vert far *, short,short,short,short,short,
        short,short,short,char*,char *);
void SEQ SetPresa(pres far *, short,short, short,short,
short,short,char *,char *);
void SEQ SetPlan(plan far *, short,short,short,short,short,
                char *,char *);
void SEQ SetFil(filt far *, short,short,short,short,short,short,
    short, char *,char *);
void SEQ tranf(point *, short, short, point *, short,short);
void SEQ Letrero(letrero far *);
void SEQ FrameT(T1 far *);
void SEQ FrameBom1(bom far *);
void SEQ FrameBom2(bom far *);
void SEQ FrameRectRound(rect far *,int,int);
void SEQ FrameVl1    (val far *);
void SEQ FrameVl2    (val far *);
void SEQ FrameVl3    (val far *);
void SEQ FrameTan    (tanq far *);
void SEQ FrameTu     (tub far *);
void SEQ FrameCodo   (cod far *);
void SEQ FrameVert   (vert far *);
void SEQ FramePresa  (pres far *);
void SEQ FramePlan   (plan far *);
void SEQ FrameFil    (filt far *);
/*definicion de numero maximo de objetos para cada tipo de estructura*/
#define NVALMAX  50          /*número maximo de figuras de cada
                            tipo a dibujar*/
#define NBOMMAX  50
#define NTUBMAX  200
#define NVERTMAX 10
#define NPRESMAX 10
#define NPLANMAX 10
#define NFILTMAX 20
#define NTANQMAX 20
#define NCODOMAX 100
#define NLETMAX  20
#define NEST     13          /*numero maximo de estructuras en el
                        esquema*/
```

```
#define NCOLMAX  20                /*numero maximo de colecciones en el esquema*/
/***DEFINICION DE ARREGLOS DE ESTRUCTURAS**/
bom  bomba1[NBOMMAX],      /* estructura tipo bomba  */
     bomba2[NBOMMAX];
tanq  tanque[NTANQMAX];            /* estructura tipo tanque */
tub  tuberia[NTUBMAX];             /* estructura tipo tuberia  */
tanq  vertedor[NVERTMAX];          /* estructura tipo vertedor */
pres  presa[NPRESMAX];             /* estructura tipo presa  */
plan  planta[NPLANMAX];            /* estructura tipo planta */
filt  filtro[NFILTMAX];        /* estructura tipo filt  */
cod  codo[NCODOMAX];               /* estructura tipo codo */
val  valvula1[NVALMAX],            /* estructura tipo valvula*/
     valvula2[NVALMAX],
     valvula3[NVALMAX];
letrero letreros[NLETMAX]; /* estructura tipo letrero*/

/***DEFINICION DE FUNCIONES DE RECUPERACION PARA OBJETOS
GRAFICOS*/
#include "funcione.c"
#include "math.h"
#include "rect.c"
void FrameBom1(NOMBRE)
bom * NOMBRE;
{
  rect uno,scrnR;
  short i=0;
  point origen,final,cuadrote[2],puntos[4];
  polyHead  cabeza[1];
  GetPenState(estado); *obtien el edo. en que se encuentra la pluma*/
  cabeza[0].polyBgn = 0;
  cabeza[0].polyEnd = 3;
  cabeza[0].polyRect = scrnR;
  puntos[0].X = NOMBRE->X - (5/2 * NOMBRE->DIAM);
  puntos[0].Y = NOMBRE->Y;
  puntos[1].X = NOMBRE->X;
  puntos[1].Y = NOMBRE->Y;
  puntos[2].X = NOMBRE->X;
  puntos[2].Y = NOMBRE->Y + NOMBRE->DIAM;
  puntos[3].X = NOMBRE->X - (5/2 * NOMBRE->DIAM);
  puntos[3].Y = NOMBRE->Y + NOMBRE->DIAM;
  /*  SE CALCULAN LAS COORDENADAS DEL RECTANGULO DEL ARCO  */
  cuadrote[0].X= NOMBRE->X-(7/2 * NOMBRE->DIAM);
  cuadrote[0].Y= NOMBRE->Y;
  cuadrote[1].X= NOMBRE->X- NOMBRE->DIAM;
  cuadrote[1].Y= NOMBRE->Y + NOMBRE->DIAM + (2 * NOMBRE->DIAM);
```

```
/*se observa si se va a borrar o no , si no escoge un color segun el edo*/
 NOMBRE->EDO=NOMBRE->EDO%3;
 if (!NOMBRE->BORRA)
 {
 switch (NOMBRE->EDO)  {
   case 0 : BackColor(4);  /*  PRENDIDA  */
          break;
   case 1 : BackColor(1);  /*  APAGADA  */
          break;
   case 2 : BackColor(14);  /*  DESCOMPUESTA  */
          break;
   default :break;  }
 PenColor(15);
 }
 else
   {
    BackColor(0);
    PenColor(0);
   }
 origen.X = puntos[3].X; /* puntos[i].X - (puntos[0].X -
                          puntos[1].X/2); */
 origen.Y = puntos[3].Y; /* puntos[i].Y - (puntos[3].Y -
                          puntos[0].Y/2); */
SetRect(&uno,cuadrote[0].X,cuadrote[0].Y,cuadrote[1].X,cuadrote[1].Y)
 FillOval(&uno,0);
 for (i=0; i<4; i++)       {
  tranf(&final,1,NOMBRE->ANG,&origen,puntos[i].X,puntos[i].Y);
  puntos[i].X = final.X;
  puntos[i].Y = final.Y;   }
 FillPoly(1,cabeza,puntos,0);
 BackColor(0);
 SetPenState(estado);/* recupera el edo. en que se encontraba la
          pluma*/
} /* END DEL PROGRAMA */
void  FrameBom2(NOMBRE)
bom * NOMBRE;
{
 rect uno,scrnR;
 short i=0,x3,y3,x4,y4;
 point origen,final,cuadrote[2],puntos[4];
 polyHead  cabeza[1];
 GetPenState(estado);/*obtien el edo. en que se encuentra la pluma*/
 cabeza[0].polyBgn = 0;
 cabeza[0].polyEnd = 3;
 cabeza[0].polyRect = scrnR;
```

```
puntos[0].X = NOMBRE->X - (5/2 * NOMBRE->DIAM);
puntos[0].Y = NOMBRE->Y;
puntos[1].X = NOMBRE->X;
puntos[1].Y = NOMBRE->Y;
puntos[2].X = NOMBRE->X;
puntos[2].Y = NOMBRE->Y + NOMBRE->DIAM;
puntos[3].X = NOMBRE->X - (5/2 * NOMBRE->DIAM);
puntos[3].Y = NOMBRE->Y + NOMBRE->DIAM;
/*  SE CALCULAN LAS COORDENADAS DEL RECTANGULO DEL ARCO  */
cuadrote[0].X= NOMBRE->X-(7/2 * NOMBRE->DIAM);
cuadrote[0].Y= NOMBRE->Y;
cuadrote[1].X= NOMBRE->X- NOMBRE->DIAM;
cuadrote[1].Y= NOMBRE->Y + NOMBRE->DIAM + (2 * NOMBRE->DIAM);
/*  CIRCULO CONCENTRICO  */
x3 = cuadrote[0].X + NOMBRE->DIAM/2;
y3 = cuadrote[0].Y + 3*NOMBRE->DIAM/4;
x4 = cuadrote[1].X - NOMBRE->DIAM/2;
y4 = cuadrote[1].Y - 3*NOMBRE->DIAM/4;
/*escoge un backcolor si se va a borrar y otro backcolor diferente si no*/
NOMBRE->EDO=NOMBRE->EDO%3;
if (!NOMBRE->BORRA)
{
switch (NOMBRE->EDO) {
  case 0 : BackColor(4);  /* PRENDIDA */
           break;
  case 1 : BackColor(1);  /* APAGADA  */
           break;
  case 2 : BackColor(14); /* DESCOMPUESTA */
           break;
  default :break;    }
 PenColor(15);
 }
 else
    { BackColor(0);PenColor(0);}
origen.X = puntos[3].X;
origen.Y = puntos[3].Y;
for (i=0; i<4; i++)      {
   tranf(&final,1,NOMBRE->ANG,&origen,puntos[i].X,puntos[i].Y);
   puntos[i].X = final.X;
   puntos[i].Y = final.Y;   }
 FillPoly(1,cabeza,puntos,0);
SetRect(&uno,cuadrote[0].X,cuadrote[0].Y,cuadrote[1].X,cuadrote[1].Y)
 FillOval(&uno,0);
 SetRect(&uno,x3,y3,x4,y4);
 FillOval(&uno,16);
```

```
  BackColor(0);
  SetPenState(estado); /*recupera el edo. en que se encontraba la
                pluma*/
} /* END DEL PROGRAMA */
void  FrameFil(NOMBRE)
filt * NOMBRE;
{
short i,x,y,bas,alt,x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,x7,y7,x8,y8,z;
short  disty,finy1,dis,color;
short margeny;
rect uno,scrnR;
polyHead  cabeza[1];
point puntos[8], puntitos[8];
GetPenState(estado);/*obtien el edo. en que se encuentra la pluma*/
 if (NOMBRE->BORRA) {color=0;PenColor(0);}else {color=3; PenColor(15);}
  x = NOMBRE->X;
  y = NOMBRE->Y;
  bas = NOMBRE->BASE;
  alt = NOMBRE->ALT;
  dis =0;
  if (NOMBRE->TIPO == 1) {
    x1 = NOMBRE->X + bas/3;
    y1 = y - alt/5;
    x2 = NOMBRE->X + (4*bas)/3;
    y2 = y - alt/5;
    x3 = NOMBRE->X + bas/3;
    y3 = y + (2*alt)/5;
    x4 = NOMBRE->X + (4*bas)/3;
    y4 = y + (2*alt)/5;
    x5 = NOMBRE->X + (2*bas)/3;
    y5 = y + (3*alt)/5;
    x6 = NOMBRE->X + bas;
    y6 = y + (3*alt)/5;
    x7 = NOMBRE->X + (2*bas)/3;
    y7 = y + (4*alt)/5;
    x8 = NOMBRE->X + bas;
    y8 = y + (4*alt)/5;
    dis = NOMBRE->X;}
   else
    if (NOMBRE->TIPO == 2 ) {
      x1 = NOMBRE->X - (4*bas)/3;
      y1 = NOMBRE->Y - alt/5;
      x2 = NOMBRE->X - bas/3;
      y2 = NOMBRE->Y - alt/5;
      x3 = NOMBRE->X - (4*bas)/3;
```

```
        y3 = NOMBRE->Y +(2*alt)/5;
        x4 = NOMBRE->X - bas/3;
        y4 = NOMBRE->Y +(2*alt)/5;
        x5 = NOMBRE->X -bas;
        y5 = NOMBRE->Y +(3*alt)/5;
        x6 = NOMBRE->X -(2*bas)/3;
        y6 = NOMBRE->Y +(3*alt)/5;
        x7 = NOMBRE->X - bas;
        y7 = NOMBRE->Y + (4*alt)/5;
        x8 = NOMBRE->X - (2*bas)/3;
        y8 = NOMBRE->Y + (4*alt)/5;
        dis = x1-bas/3; }
      else
        printf("error en el tipo de vertedor\n");
/*  coloca los puntos del poli  */
    cabeza[0].polyBgn = 0;
    cabeza[0].polyEnd = 7;
    cabeza[0].polyRect = scrnR;
    z=0;
    puntos[z].X = x8;
    puntos[z].Y = y8;
    z++;
    puntos[z].X = x6;
    puntos[z].Y = y6;
    z++;
    puntos[z].X = x4;
    puntos[z].Y = y4;
    z++;
    puntos[z].X = x2;
    puntos[z].Y = y2;
    z++;
    puntos[z].X = x1;
    puntos[z].Y = y1;
    z++;
    puntos[z].X = x3;
    puntos[z].Y = y3;
    z++;
    puntos[z].X = x5;
    puntos[z].Y = y5;
    z++;
    puntos[z].X = x7;
    puntos[z].Y = y7;
    disty = y8-y1;
    margeny = (short) (disty * (NOMBRE->EDO/100.0));
    finy1 = y8 - margeny;
```

```
  for (i = 0; i <=7; i++) {puntitos[i] = puntos[i];};
  if (NOMBRE->TIPO == 1 || NOMBRE->TIPO ==2) {
    MoveTo(x,y);
    LineTo(x1,y);
    for (i = 0; i <=7; i++) {puntos[i] = puntitos[i];};
    BackColor(color);
    ErasePoly(1,cabeza,puntos);
    for (i = 0; i <=7; i++) {puntos[i] = puntitos[i];};
    BackColor(16);
    SetRect(&uno,x1,y1,x2,finy1);
    FillRect(&uno,0);
    for (i = 0; i <=7; i++) {puntos[i] = puntitos[i];};
    FramePoly(1,cabeza,puntos);
    SetPenState(estado);/*recupera el edo. en que se encontraba la
                pluma*/
               }
}
void  FrameTu(NOMBRE)
tub * NOMBRE;
{
 short z,j,i=0,margeny,dm1,X1,X2,Y1,Y2,color;
 point puntos[4],sust[4],origen,final;
 polyHead  cabeza[1];
 rect scrnR;
GetPenState(estado);  /*obtiene el edo. en que se encuentra la pluma*/
if (NOMBRE->BORRA) {color=0;PenColor(0);} else {color=3;PenColor(15);}
/*** PARA TIPO 1 = TUBERIA
     PARA TIPO 2 = TUNEL
     PARA TIPO 0 = CANAL    ***/
 NOMBRE->TIPO=NOMBRE->TIPO % 3;
 cabeza[0].polyBgn = 0;
 cabeza[0].polyEnd = 3;
 cabeza[0].polyRect = scrnR;
 puntos[0].X = NOMBRE->X;
 puntos[0].Y = NOMBRE->Y;
 puntos[1].X = NOMBRE->X + NOMBRE->LON;
 puntos[1].Y = NOMBRE->Y;
 puntos[2].X = NOMBRE->X + NOMBRE->LON;
 puntos[2].Y = NOMBRE->Y + NOMBRE->DIAM;
 puntos[3].X = NOMBRE->X;
 puntos[3].Y = NOMBRE->Y + NOMBRE->DIAM;
 X1=puntos[0].X;
 X2=puntos[2].X;
 Y1=puntos[0].Y;
 Y2=puntos[2].Y;
```

```
dml = NOMBRE->DIAM/3;
for (j=0; j<=3; j++)
   sust[j] = puntos[j];
origen.X = NOMBRE->X;
origen.Y = NOMBRE->Y;
/* se calcula y se dibuja el nivel de agua */
margeny = (short) ((puntos[3].Y - puntos[0].Y) * (NOMBRE->EDO/100.0));
if ((NOMBRE->ANGT >=0 && NOMBRE->ANGT <= 180) || (NOMBRE->ANGT
>= 270 && NOMBRE->ANGT <= 360))
    puntos[0].Y = puntos[1].Y = (puntos[3].Y - margeny);
if (NOMBRE->ANGT >180 && NOMBRE->ANGT <270)
  puntos[2].Y = puntos[3].Y = (puntos[1].Y + margeny);
for (j=0; j<=3; j++) {
    tranf(&final,1,NOMBRE->ANGT,&origen,puntos[j].X,puntos[j].Y);
    puntos[j].X = final.X;
    puntos[j].Y = final.Y; };
BackColor(color);
FillPoly(1,cabeza,puntos,0);    /*** NIVEL DE AGUA ***/
BackColor(16);
cabeza[0].polyBgn = 0;
cabeza[0].polyEnd = 1;
cabeza[0].polyRect = scrnR;
/*** SE DIBUJAN LAS LINEAS EXTERIORES ***/
for (i=0;i<=1;i++) {
   tranf(&final,1,NOMBRE->ANGT,&origen,sust[i].X,sust[i].Y);
   puntos[i].X = final.X;
   puntos[i].Y = final.Y; };
FramePoly(1,cabeza,puntos);
cabeza[0].polyBgn = 2;
cabeza[0].polyEnd = 3;
for (i=2;i<=3;i++) {
   tranf(&final,1,NOMBRE->ANGT,&origen,sust[i].X,sust[i].Y);
   puntos[i].X = final.X;
   puntos[i].Y = final.Y; };
FramePoly(1,cabeza,puntos);
 if (NOMBRE->TIPO == 2) {  /*  TUNEL  */
    PenDash(4);
    z = NOMBRE->Y + NOMBRE->DIAM/2;
    tranf(&final,1,NOMBRE->ANGT,&origen,NOMBRE->X,z);
    MoveTo(final.X,final.Y);
    tranf(&final,1,NOMBRE->ANGT,&origen,NOMBRE->X+NOMBRE->LON,z);
    LineTo(final.X,final.Y);
    PenDash(0);   };
 if (NOMBRE->TIPO == 0) {  /* CANAL */
    puntos[0].X = NOMBRE->X;
```

```
      puntos[0].Y = NOMBRE->Y + dm1;
      puntos[1].X = NOMBRE->X + NOMBRE->LON;
      puntos[1].Y = NOMBRE->Y + dm1;
      for (i=0;i<=1;i++) {
         tranf(&final,1,NOMBRE->ANGT,&origen,puntos[i].X,
          puntos[i].Y);
         puntos[i].X = final.X;
         puntos[i].Y = final.Y;  };
      FramePoly(1,cabeza,puntos);
      puntos[0].X = NOMBRE->X;
      puntos[0].Y = NOMBRE->Y + (2*dm1);
      puntos[1].X = NOMBRE->X + NOMBRE->LON;
      puntos[1].Y = NOMBRE->Y + (2*dm1);
      for (i=0;i<=1;i++) {
         tranf(&final,1,NOMBRE->NGT,&origen,puntos[i].X,
          puntos[i].Y);
         puntos[i].X = final.X;
         puntos[i].Y = final.Y;  };
      FramePoly(1,cabeza,puntos);
     }
  BackColor(0);
  SetPenState(estado); /* recupera el edo. en que se encontraba la pluma*/
 }
void  FrameCodo(NOMBRE)
cod * NOMBRE;
{
  point punto[4];
  polyHead  cabeza[1],head[1];
  rect  scmR;
  short z,distx,disty,margenx,margeny,color;
  GetPenState(estado);/*obtiene el edo. en que se encuentra la
                pluma*/
  if (NOMBRE->BORRA) {color=0;PenColor(0);} else {color=3;PenColor(15);}
    if (NOMBRE->TIPO == 1) {
        distx = NOMBRE->V2.X;
        disty = NOMBRE->V2.Y;
        margenx=(short) (distx * (NOMBRE->EDO/100.0));
        margeny=(short) (disty * (NOMBRE->EDO/100.0));
        punto[1].X= (short) (NOMBRE->V1.X+NOMBRE->V2.X-margenx);
        punto[1].Y= (short)(NOMBRE->V1.Y+NOMBRE->V2.Y-margeny);
        distx=NOMBRE->V3.X - NOMBRE->V2.X;
        disty=NOMBRE->V3.Y - NOMBRE->V2.Y;
        margenx=(short) (distx * (NOMBRE->EDO/100.0));
        margeny=(short) (disty * (NOMBRE->EDO/100.0));
        punto[2].X=NOMBRE->V1.X+NOMBRE->V2.X+margenx;
```

```
    punto[2].Y=NOMBRE->V1.Y+NOMBRE->V2.Y+margeny;
    cabeza[0].polyBgn  = 1;
    cabeza[0].polyEnd  = 3;
    cabeza[0].polyRect = scrnR;
    z=1;
    punto[z].X = punto[1].X;
    punto[z].Y = punto[1].Y;
    z=2;
    punto[z].X = punto[2].X;
    punto[z].Y = punto[2].Y;
    z=3;
    punto[z].X = NOMBRE->V1.X+NOMBRE->V2.X;
    punto[z].Y = NOMBRE->V1.Y+NOMBRE->V2.Y;
    BackColor(color);
    FillPoly(1,cabeza,punto,0);
  }
 if (NOMBRE->TIPO == 2) {
    distx = NOMBRE->V2.X;
    disty = NOMBRE->V2.Y;
    margenx=(short) (distx * (NOMBRE->EDO/100.0));
    margeny=(short) (disty * (NOMBRE->EDO/100.0));
    punto[1].X= (short) (NOMBRE->V1.X+margenx);
    punto[1].Y= (short)(NOMBRE->V1.Y+margeny);
    distx=NOMBRE->V3.X - NOMBRE->V2.X;
    disty=NOMBRE->V3.Y - NOMBRE->V2.Y;
    margenx=(short) (distx * (NOMBRE->EDO/100.0));
    margeny=(short) (disty * (NOMBRE->EDO/100.0));
    punto[2].X=NOMBRE->V1.X+NOMBRE->V3.X-margenx;
    punto[2].Y=NOMBRE->V1.Y+NOMBRE->V3.Y-margeny;
    head[0].polyBgn  = 1;
    head[0].polyEnd  = 4;
    head[0].polyRect = scrnR;
    z=1;
    punto[z].X = punto[1].X;
    punto[z].Y = punto[1].Y;
    z=2;
    punto[z].X = punto[2].X;
    punto[z].Y = punto[2].Y;
    z=3;
    punto[z].X = NOMBRE->V1.X+NOMBRE->V3.X;
    punto[z].Y = NOMBRE->V1.Y+NOMBRE->V3.Y;
    z=4;
    punto[z].X = NOMBRE->V1.X;
    punto[z].Y = NOMBRE->V1.Y;
    BackColor(color);
```

```
      FillPoly(1,head,punto,0);
    }
    MoveTo(NOMBRE->V1.X ,NOMBRE->V1.Y);
  LineTo(NOMBRE->V1.X+NOMBRE->V3.X,NOMBRE->V1.Y+NOMBRE->V3.Y);
    BackColor(16);
    SetPenState(estado); /*recupera el edo. en que se encontraba la
                pluma*/
  }
void  FramePresa(NOMBRE)
pres * NOMBRE;
{
  short i=0,x,y,x1,x2,y1,bas,x3,x4,z;
  point puntos1[3],puntos2[4];
  polyHead  cabeza[1];
  rect  scrnR;
  short a,a1,b,b1,distx,disty,margenx,margeny,color1,color2;
  GetPenState(estado);
  if(NOMBRE->BORRA) {color1=0;PenColor(0);color2=0;}
  else {color1=3;color2=15;}
  y = NOMBRE->Y;
  bas =(short) (0.5 * NOMBRE->ALT);
  y1 = y - NOMBRE->ALT;
  switch (NOMBRE->TIPO){     /* presa a la derecha  */
    case 1: {
          x1=(short) (NOMBRE->X-(bas*.5));
          x2=NOMBRE->X-(1*bas/4);
          x3=NOMBRE->X;
          x4=(short) (NOMBRE->X-(2.5*bas));
          break;      }
    case 2:  {
          x1=(short) (NOMBRE->X+(bas*.5));
          x2=NOMBRE->X+(1*bas/4);
          x3=NOMBRE->X;
          x4=(short) (NOMBRE->X+(2.5*bas));
          break;      }
    default: printf("error en el tipo de presa\n");    }
  switch (NOMBRE->TIPO){        /* vertedor a la derecha  */
    case 1: {
          distx=x1-x4;
          disty=y-y1;
          margenx=(short) (distx * (NOMBRE->EDO/100.0));
          margeny=(short) (disty * (NOMBRE->EDO/100.0));
          a= (short) (x1-((x1-x4) * (NOMBRE->EDO/100.0)));
          b=y-margeny;
          distx=x2-x1;
```

```
            margenx=(short) (distx * (NOMBRE->EDO/100.0));
            a1=x1+margenx;
            b1=y-margeny;
            break;  }
             case 2: {
            distx=x4-x1;
            disty=y-y1;
            margenx=(short) (distx * (NOMBRE->EDO/100.0));
            margeny=(short) (disty * (NOMBRE->EDO/100.0));
            a=x1+margenx;
            b=y-margeny;
            distx=x1-x2;
            margenx=(short) (distx * (NOMBRE->EDO/100.0));
            a1=x1-margenx;
            b1=y-margeny;
            break;  }          }
if ((NOMBRE->TIPO == 1) || (NOMBRE->TIPO == 2)) {
  cabeza[0].polyBgn = 0;
  cabeza[0].polyEnd = 2;
  cabeza[0].polyRect = scrnR;
  z=0;
  puntos1[z].X = a;
  puntos1[z].Y = b;
  z=1;
  puntos1[z].X = a1;
  puntos1[z].Y = b1;
  z=2;
  puntos1[z].X = x1;
  puntos1[z].Y = y;
  BackColor(color1);
  FillPoly(1,cabeza,puntos1,0);
  cabeza[0].polyBgn = 0;
  cabeza[0].polyEnd = 3;
  cabeza[0].polyRect = scrnR;
  z=0;
  puntos2[z].X = x1;
  puntos2[z].Y = y;
  z=1;
  puntos2[z].X = x2;
  puntos2[z].Y = y1;
  z=2;
  puntos2[z].X = x3;
  puntos2[z].Y = y1;
  z=3;
  puntos2[z].X = NOMBRE->X;
```

```
    puntos2[z].Y = y;
    BackColor(color2);
    PaintPoly(1,cabeza,puntos2);
    MoveTo(x4,y1);
    LineTo(x1,y);
   if (NOMBRE->TIPO == 1)
      x = x4;
   if (NOMBRE->TIPO == 2)
      x = NOMBRE->X;
   BackColor(0);
   SetPenState(estado);
  }; /*   del while del tipo   */
}
void FrameTan (NOMBRE)
tanq * NOMBRE;
{
 rect uno;
 static int x1,y1,x2,y2,j,t1x,t1y,t2x,t2y,t3x,t3y,t4x,t4y,t5x,t5y,t6x,t6y,
      dist,margen,color1,color2;
 GetPenState(estado); /*obtien el edo. en que se encuentra la pluma*/
 if (NOMBRE->BORRA) {color1=0;color2=0;} else {color1=3;color2=15;};
 SetDisplay(GrafPg0);
 x1 = NOMBRE->X;
 y1 = NOMBRE->Y - NOMBRE->ALT;
 x2 = NOMBRE->X + NOMBRE->BASE;
 y2 = NOMBRE->Y ;
 t1x=0;  t1y=0;
 t2x=0;  t2y=0;
 t3x=0;  t3y=0;
 t4x=0;  t4y=0;
 t5x=0;  t5y=0;
 t6x=0;  t6y=0;
 for (j=0; j< NOMBRE->N_CON; j++){
 if (NOMBRE->POS_CON[j] == 1) {
    t1x= x1;
    t1y= y2;
    t2x= x1;
    t2y= y2 - NOMBRE->DIAM_CON[j];
    BackColor(color1);
    SetRect(&uno,t2x-NOMBRE->DIAM_CON[j],t2y,t1x,t1y);
    FillRect(&uno,0); /*PaintRect(&uno);*/
    BackColor(color2);PenColor(color2);
    MoveTo(t1x,t1y);
    LineTo(t1x-NOMBRE->DIAM_CON[j],t1y);
    MoveTo(t2x,t2y);
```

```
    LineTo(t2x-NOMBRE->DIAM_CON[j],t2y);
                }
  if (NOMBRE->POS_CON[j] == 2) {
    t3x= x2;
    t3y= y2;
    t4x= x2;
    t4y= y2 - NOMBRE->DIAM_CON[j];
    BackColor(color1);
    SetRect(&uno,t4x,t4y,t3x+NOMBRE->DIAM_CON[j],t3y);
    FillRect(&uno,0);
    BackColor(color2);PenColor(color2);
    MoveTo(t3x,t3y);
    LineTo(t3x+NOMBRE->DIAM_CON[j],t3y);
    MoveTo(t4x,t4y);
    LineTo(t4x+NOMBRE->DIAM_CON[j],t4y);
                }
  if (NOMBRE->POS_CON[j] == 3) {
    t5x= x1 + (x2 - x1)/2-(NOMBRE->DIAM_CON[j]/2);
    t5y= NOMBRE->Y;
    t6x= t5x + NOMBRE->DIAM_CON[j];
    t6y= NOMBRE->Y;
    BackColor(color1);
    SetRect(&uno,t5x,t5y,t6x,t6y+NOMBRE->DIAM_CON[j]);
    FillRect(&uno,0);
    BackColor(color2);PenColor(color2);
    MoveTo(t5x,t5y);
    LineTo(t5x,t5y+NOMBRE->DIAM_CON[j]);
    MoveTo(t6x,t6y);
    LineTo(t6x,t6y+NOMBRE->DIAM_CON[j]);
                }
                  }/* END DEL FOR*/
  dist = y2 - y1;
  margen = (short) (dist * (NOMBRE->EDO/100.0));
  dist = NOMBRE->Y - margen;
  SetRect(&uno,x1,dist,x2,y2);
  BackColor(color1);
  FillRect(&uno,0);
  BackColor(16);
  SetRect(&uno,x1,y1,x2,y2);
  FrameRect(&uno);
  if (NOMBRE->TIPO == 2 ) {
     PenColor(0);
     MoveTo(x1,y1);
     LineTo(x2,y1);     };
  PenColor(color1);
```

```
  MoveTo(t1x,t1y);
  LineTo(t2x,t2y);
  MoveTo(t3x,t3y);
  LineTo(t4x,t4y);
  MoveTo(t6x,t6y);
  LineTo(t5x,t5y);
  PenColor(color2);
  BackColor(0);
  BackColor(0);
 SetPenState(estado);/*recupera el edo. en que se encontraba la
                pluma*/
}
void  FrameVert(NOMBRE)
vert * NOMBRE;
{
 short i=0,x,y,x1,y1,x2,y2,x3,y3,x4,y4,dist,b,margen,pos,color;
 rect uno;
  GetPenState(estado); /*obtien el edo. en que se encuentra la pluma*/
  if (NOMBRE->BORRA) {color=0;PenColor(0);} else {color=3;PenColor(15);}
  x = NOMBRE->X;
  y = NOMBRE->Y;
  x1 = NOMBRE->X;
  y1 = NOMBRE->Y-NOMBRE->ALT;
  y2 = NOMBRE->Y;
  y3 = y2 - (2*NOMBRE->ALT)/3;
  y4 = y3;
  if (NOMBRE->TIPO == 1) {     /* vertedor a la derecha  */
    x2 = NOMBRE->X + NOMBRE->BASE;
    pos=NOMBRE->X;
    x3 = x2;
    x4 = x3 + (NOMBRE->BASE/4);
    }
   else
    if (NOMBRE->TIPO == 2) {
      x2 = NOMBRE->X- NOMBRE->BASE;
      pos=x2;
      x3 = x2;
      x4 = x3 - (NOMBRE->BASE/4);
      }
  if ((NOMBRE->TIPO ==1) || (NOMBRE->TIPO == 2)) {
    dist = y2 - y3;
    margen = (short) (dist * (NOMBRE->EDO/100.0));
    b = NOMBRE->Y - margen;
    SetRect(&uno,x1,b,x2,y2);
    BackColor(color);
```

```
    FillRect(&uno,0);
    BackColor(16);
    MoveTo(x1,y1);
    LineTo(x,y);
    LineTo(x2,y2);
    LineTo(x3,y3);
    LineTo(x4,y4);
    if (NOMBRE->TECHO == 1) {
    MoveTo(x1,y1);
    LineTo(x4,y1);          };
    if (NOMBRE->TIPO == 2)     /* vertedor a la derecha  */
      x = NOMBRE->X - (5*NOMBRE->BASE)/4;
     else
      x = NOMBRE->X + 2;
   BackColor(0);
   SetPenState(estado);              /* recupera el edo. en que se encontraba la pluma*/
  } /* end del if  */
}
void  FramePlan(NOMBRE)
plan * NOMBRE;
{
 short x,y,x1,y1,x2,y2,band=0,diamx=0,diamy=0;
 rect uno;
GetPenState(estado);                /*obtiene el edo. en que se encuentra la pluma*/
if (NOMBRE->BORRA) PenColor(0);
   x = NOMBRE->X;
   y = NOMBRE->Y;
 x1 = x;
 y1 = y - NOMBRE->ALT/4;
 x2 = x + NOMBRE->BASE;
 y2 = y + (3 * NOMBRE->ALT)/4;
 BackColor(16);
 SetRect(&uno,x1,y1,x2,y2);
 EraseRoundRect(&uno,NOMBRE->BASE,NOMBRE->ALT/2);
 SetRect(&uno,x1,y1,x2,y2);
 FrameRectRound(&uno,NOMBRE->BASE,NOMBRE->ALT/2);
 diamx = x2 - x1;
 diamy = y2 - y1;
if (NOMBRE->BORRA) PenColor(0);
BackColor(0);
SetPenState(estado);  /*recupera el edo. en que se encontraba la
        pluma*/
}
void  FrameVl1(NOMBRE)
val * NOMBRE;
```

```
{
 T1 t;
 rect uno;
 short x,x3,x4,y,y3,y4;
 short X1,Y1,X2,Y2;
 short lon1,lon2,x8,y8,a,b,a1,b1,inter;
 short distx,disty,margenx,margeny,color;
 float mientras;
 GetPenState(estado);
 if (NOMBRE->BORRA) {color=0;PenColor(0);} else {color=3;}
 inter = NOMBRE->EDO;
 x = NOMBRE->X;
 y = NOMBRE->Y;
 x3=x;
 y3=y+NOMBRE->DIAM;
 x4=x+NOMBRE->DIAM;
 y4=y;
 /*  se calculan las coordenadas del rectangulo  */
X1=x-NOMBRE->DIAM/5;
Y1=y-NOMBRE->DIAM/2;
X2=x4+NOMBRE->DIAM/5;
Y2=y4+(3*NOMBRE->DIAM)/2;
distx=X2-X1;
disty=Y2-Y1;
mientras = (float) (inter/100.0);
margenx=(short) (distx * (1- mientras)/2.0);
margeny=(short) (disty * (1- mientras)/2.0);
a=X1+margenx;
b=Y1+margeny;
a1=X2-margenx;
b1=Y2-margeny;
/*se calculan las coordenadas y medidas para la T de la valvula*/
x8=x+NOMBRE->DIAM/2;
y8=Y1;
lon1=NOMBRE->DIAM/2;
lon2=NOMBRE->DIAM/4;
if (inter>0 && inter <101 ) {
  SetRect(&uno,X1,Y1,X2,Y2);
  BackColor(16);
  FillOval(&uno,0);
  FrameOval(&uno);
  BackColor(color);
  SetRect(&uno,a,b,a1,b1);
  FillOval(&uno,0);
  MoveTo(X1+(x3-X1)/2,y);
```

```
 LineTo(X2-(X2-x4)/2,y4+NOMBRE->DIAM);
 MoveTo(X1+(x3-X1)/2,y3);
 LineTo(X2-(X2-x4)/2,y4);
 SetT(&t,lon1,lon2,x8,y8);
 FrameT(&t);
  BackColor(0);
  SetPenState(estado);
}
}
void  FrameVl2(NOMBRE)
val * NOMBRE;
{
 T1 t;
 rect uno;
 short x,y,aux;
 short X1,Y1,X4,Y4;
 short lon1,lon2,x8,y8,i=0;
 GetPenState(estado);
 x = NOMBRE->X;
 y = NOMBRE->Y;
 /*se calculan las coordenadas del rectangulo*/
X1=x;
Y1=y;
X4=x+(NOMBRE->DIAM);
Y4=y+2*NOMBRE->DIAM;
/*se calculan las coordenadas y medidas para la T de la valvula*/
x8=X1+NOMBRE->DIAM/2;
y8=Y1;
lon1=NOMBRE->DIAM/2;
lon2=NOMBRE->DIAM/4;
/*se observa si se va a borrar o no,si no escoge un color segun el edo*/
aux=(NOMBRE->EDO-1)%3;
if (!NOMBRE->BORRA)
  {
switch (aux) {
 case 0 : BackColor(4);  /*  PRENDIDA  */
        break;
 case 1 : BackColor(1);  /*  APAGADA   */
        break;
 case 2 : BackColor(14); /*  DESCOMPUESTA  */
        break;
 default :break; }
 }
 else  {BackColor(0);PenColor(0);}
  SetRect(&uno,X1,Y1,X4,Y4);
```

```
  FillRect(&uno,0);
  MoveTo(X1,Y1);
  LineTo(X4,Y4);
  MoveTo(X1,Y4);
  LineTo(X4,Y1);
  SetT(&t,lon1,lon2,x8,y8);
  FrameT(&t);
 BackColor(0);
 SetPenState(estado);
return;
}
void FrameVl3(NOMBRE)
val * NOMBRE;
{
 rect uno;
 short x,y,aux;
 short X1,Y1,X4,Y4,i=0,X2,Y2,X3,Y3;
  GetPenState(estado);
 x = NOMBRE->X;
 y = NOMBRE->Y;
 /*  se calculan las coordenadas del rectangulo  */
 X1=x;
 Y1=y;
 X4=x+NOMBRE->DIAM;
 Y4=y+2*NOMBRE->DIAM;
aux=(NOMBRE->EDO-1)%3;
if (!NOMBRE->BORRA)
 {

 switch (aux) {
   case 0 : BackColor(4);  /*  PRENDIDA  */
        break;
   case 1 : BackColor(1);  /*  APAGADA  */
        break;
   case 2 : BackColor(14); /*  DESCOMPUESTA  */
        break;
   default :break; }
  }
 else  {BackColor(0);PenColor(0);}
   SetRect(&uno,X1,Y1,X4,Y4);
   FillRect(&uno,0);
   MoveTo(X1,Y1);
   LineTo(X4,Y4);
   /*   rectangulo pequeno sombreado            */
   X3=X1;
```

```
    Y3=Y4-NOMBRE->DIAM/5;
    X2=X1+NOMBRE->DIAM/5;
    Y2=Y4;
    SetRect(&uno,X3,Y3,X2,Y2);
    FrameRect(&uno);
    PaintRect(&uno);
     BackColor(0);
     SetPenState(estado);
}
void FrameT(nom)
 T1 * nom;
{  short x1,y1,x2,y2,x3,y3,x4,y4;
x1=nom->X;
y1=nom->Y;
x2=x1;
y2=y1-nom->long1;
x3=x2-nom->long2;
y3=y2;
x4=x2+nom->long2;
y4=y2;
MoveTo(x1,y1);
LineTo(x2,y2);
MoveTo(x3,y3);
LineTo(x4,y4);
}
void SetT(nom1,lo1,lo2,px,py)
  T1 * nom1;
  short lo1,lo2,px,py;
{
  nom1->long1=lo1;
  nom1->long2=lo2;
  nom1->X=px;
  nom1->Y=py;
  }
/*****DIBUJA UN LETRERO CON EL TAMANO SEGUN LA PANTALLA  QUE
SE ESTE USANDO**/
void Letrero(letrero *NOMBRE)
{
 short i=0,j,font;
 int    loadErr;
 float p1,p3,p2,tf[6];
 float tamdmax,tamdib,exp;
GetPenState(estado);
tamdmax=5830.00f; /***tamano de digonal de pantalla maxima*/
/*calculo de diagonal de pantalla con coordenas virtuales*/
```

```
exp = (float)pow((double)(vista.Xmax-vista.Xmin),2.00);
tamdib = exp;
exp = (float)pow((double)(vista.Ymax-vista.Ymin),2.00);
tamdib = tamdib+exp;
tamdib=(float)sqrt((double)tamdib);
tf[0]=0.011f;
tf[1]=0.014f;
tf[2]=0.017f;
tf[3]=0.018f;
tf[4]=0.021f;
tf[5]=0.027f;
tf[6]=1.000f;
p2=(float)NOMBRE->TAM*0.001;/***segun el tamano de letrero escoge una
                              proporcion a la pantalla*/
p1=(tamdmax/tamdib);
p3=p2*p1;
font=-1;
for (j=0; j<=5 ;j++)
{
   if (tf[j]<=p3 && tf[j+1]>=p3  )
   {
     font =j;
     j=5;
   }
}
if (font>=0)
{
 switch(font)
 {
    case 0:
          font=0;
          break;
    case 1:
          font=8;
          break;
    case 2:
          font=16;
          break;
    case 3:
          font=24;
          break;
    case 4:
          font=72;
          break;
    case 5:
```

```
            font=48;
            break;
  }
  if (NOMBRE->BORRA==1)
    {BackColor(0);PenColor(0);}
  loadErr=SystemFont(font);
  MoveTo(NOMBRE->X,NOMBRE->Y);
  i=0;
  do
  {
      DrawChar(NOMBRE->CADENA[i]);
      i++;
  } while ( (i <= 8) && (NOMBRE->CADENA[i] != '\0'));
 }/*if (font>=0..*/
 PenColor(15);
 SetPenState(estado);
 }/*letrero*/
```

2.-LISTADO DE LA LIBRERÍA LIB2.C

```
/**DEFINICION DE APUNTADORES A FUNCIONES**/
ap_fig[0]=FramePlan;
ap_fig[1]=FrameBom1;
ap_fig[2]=FrameBom2;
ap_fig[3]=FrameFil;
ap_fig[4]=FrameTu;
ap_fig[5]=FrameCodo;
ap_fig[6]=FramePresa;
ap_fig[7]=FrameTan;
ap_fig[8]=FrameVert;
ap_fig[9]=FrameVl1;
ap_fig[10]=FrameVl2;
ap_fig[11]=FrameVl3;
ap_fig[12]=Letrero;
/***DEFINICION DE APUNTADORES A ESTRUCTURAS***/
/* Inicia apuntadores a estructuras para uso indexado posterior */
/* ap_est = arreglo de apuntadores a estructuras*/
ap_est[0]=planta;
ap_est[1]=bomba1;
ap_est[2]=bomba2;
ap_est[3]=filtro;
ap_est[4]=tuberia;
ap_est[5]=codo;
ap_est[6]=presa;
ap_est[7]=tanque;
ap_est[8]=vertedor;
```

```
ap_est[9]=valvula1;
ap_est[10]=valvula2;
ap_est[11]=valvula3;
ap_est[12]=letreros;
ap_est[13]=NULL;
/**INICIALIZACION DE ARREGLO DE TAMANO DE ESTRUCTURAS**/
/* Encuentra el tamano de cada estructura */
tam_est[0]=sizeof(plan);
tam_est[1]=sizeof(bom);
tam_est[2]=sizeof(bom);
tam_est[3]=sizeof(filt);
tam_est[4]=sizeof(tub);
tam_est[5]=sizeof(cod);
tam_est[6]=sizeof(pres);
tam_est[7]=sizeof(tanq);
tam_est[8]=sizeof(vert);
tam_est[9]=sizeof(val);
tam_est[10]=sizeof(val);
tam_est[11]=sizeof(val);
tam_est[12]=sizeof(letrero);
tam_est[13]=0;
```

APÉNDICE C

1.-Archivo de datos de un objeto gráfico.

1.-ARCHIVO DE DATOS DE UN OBJETO GRÁFICO

3121 482 0 12 1 50 120 2

175 2800 0 9 2 50 75 1
Colorines

940 2825 0 2 4 0 240 2
V. Bravo

727 2375 0 6 2 50 75 1
Tilostoc

2563 1443 0 5 1 50 65 2
D. Guerra

1876 980 0 12 0 50 50 50 1
Chilesdo
V. Vict.

Archivo de parametros para dibujar presas.

Los datos se presentan en el siguiente orden:

X, Y, BORRA, EST, NFIG, EDO, ALT, TIPO,
IDENTIFICADOR 1,
IDENTIFICADOR 2.

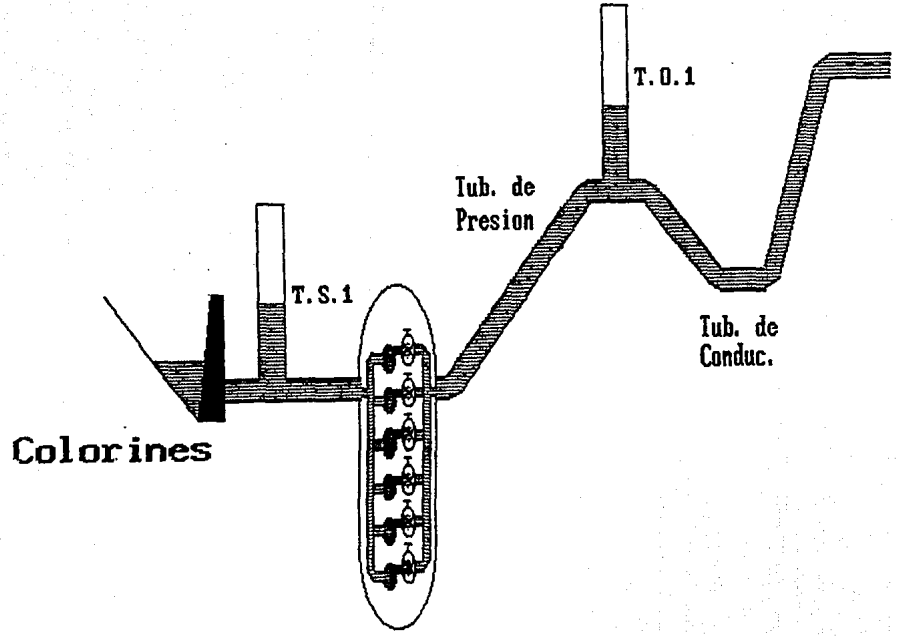
APÉNDICE D

1.-Esquema general del Sistema Cutzamala.

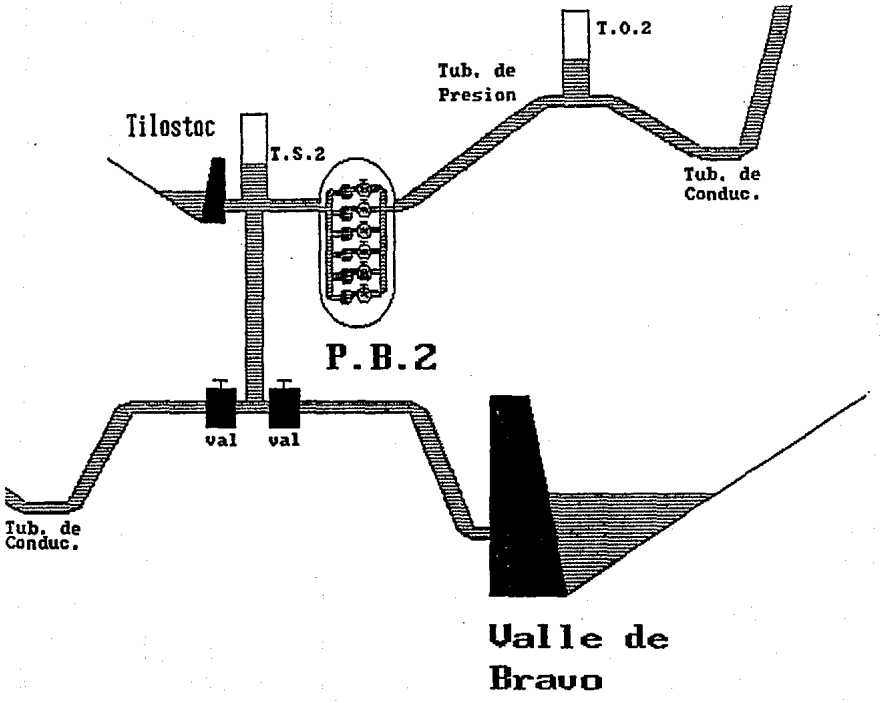
2.-Vistas del Sistema Cutzamala.

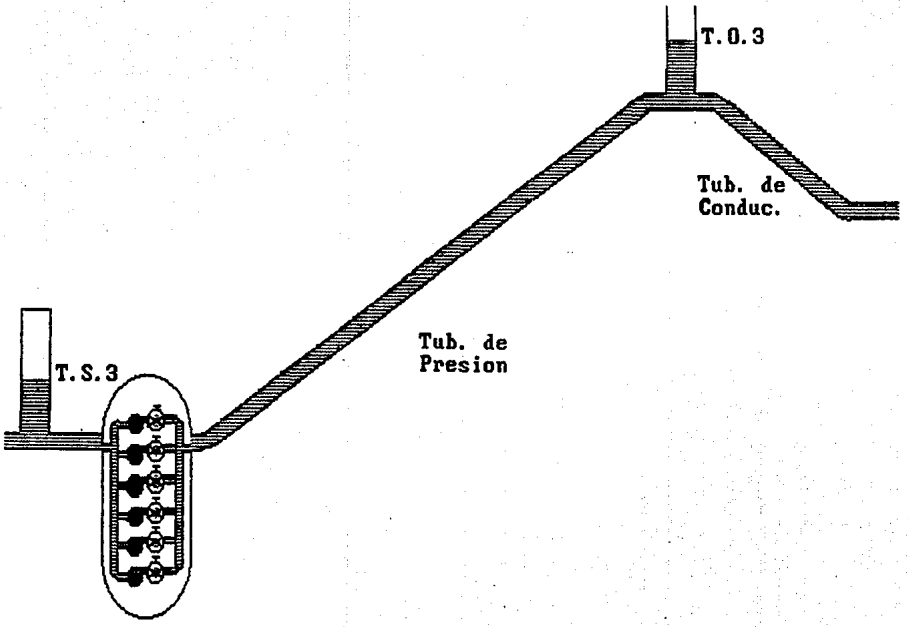
1.-ESQUEMA GENERAL DEL SISTEMA CUTZAMALA

2.-VISTAS DEL SISTEMA CUTZAMALA

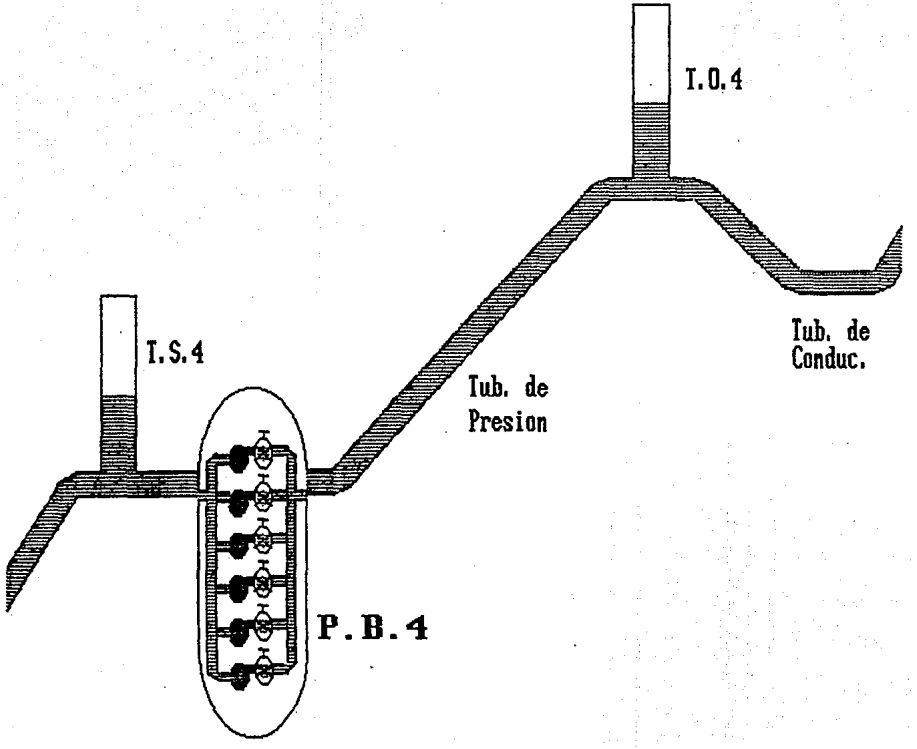


P.B.1


T.O.2
Tub. de
Presion
Tilostoc
T.S.2
Tub. de
Conduc.
P.B.2
val
val
Tub. de
Conduc.
Valle de
Bravo

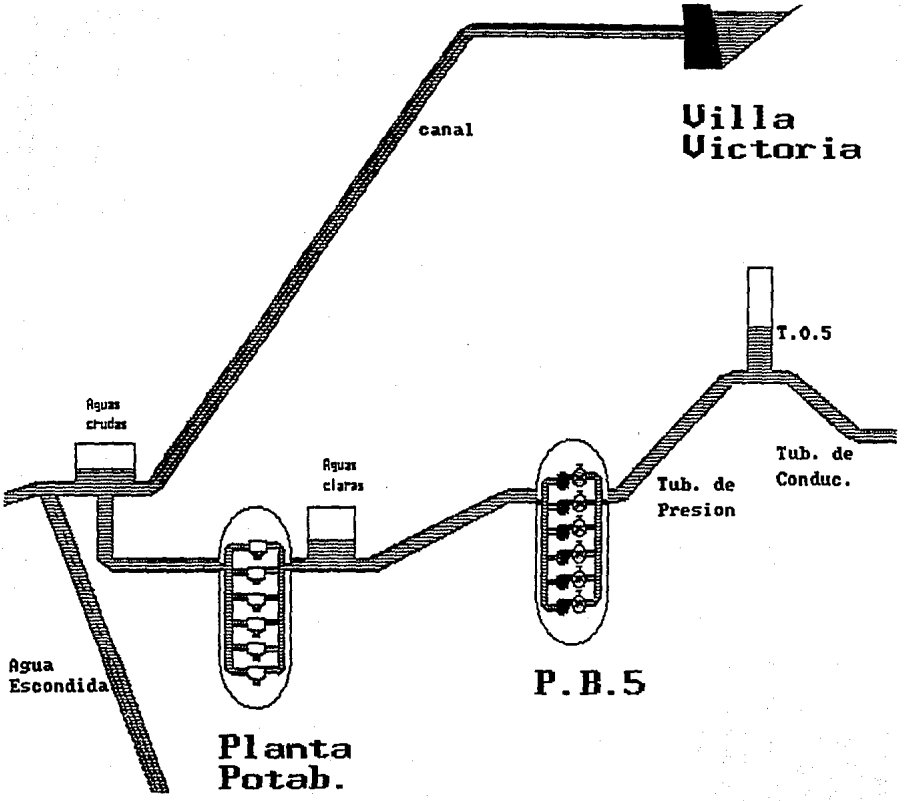


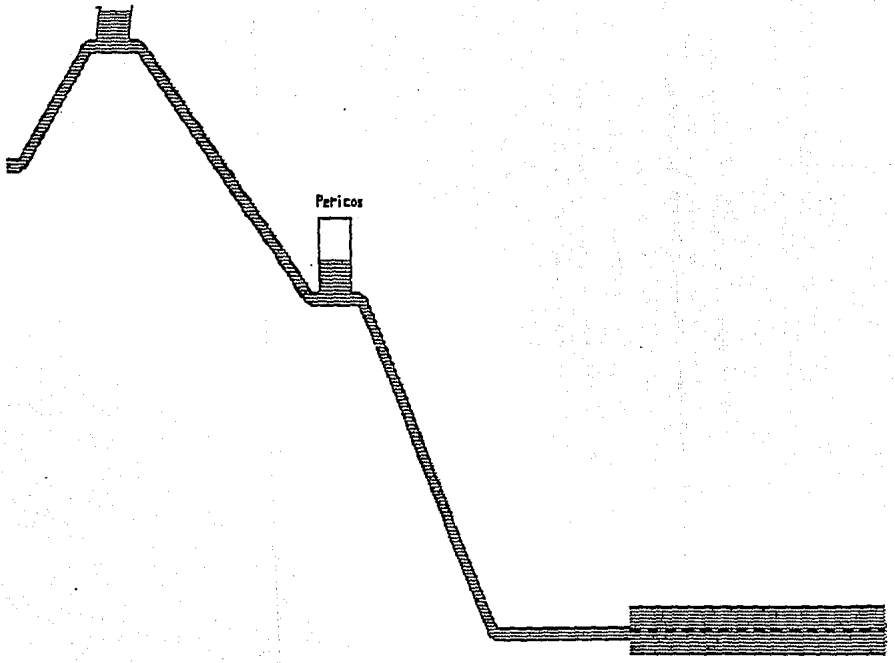
P.B.3


T.O.4
T.S.4
Tub. de
Presion
Tub. de
Conduc.
P.B.4

Aguas
claras
Agua
Escondida
Planta
Potab.
Donato
Guerra
canal

P.B.6


Villa
Victoria
canal
T.0.5
Aguas
crudas
Aguas
claras
Tub. de
Conduc.
Tub. de
Presion
Agua
Escondida
P.B.5
Planta
Potab.



Tunel
San Jose