UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

INTERFAZ GRAFICA DE USUARIO PARA SU APLICACION A SISTEMAS COMPUTARIZADOS

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A N

VICENTE RENE HERNANDEZ OLVERA

HUGO ABAD MERLIN PEREZ

DIRECTOR: FIS RAYMUNDO HUGO RANGEL GUTIERREZ

MEXICO, D. F. 1996

Agradecimientos:

A mis padres Ma. de Jesús y Vicente Héctor, que me dieron la oportunidad de continuar mis estudios.

A mis hermanos Raúl, Esperanza, Guillermo y Omar; por su apoyo y comprensión.

A todos mis amigos que me dieron ánimos para terminar esta tesis.

Vicente René Hernández Olvera

A mis padres Juan Merlín Marín y María Pérez Ramírez por haberme dado lo mas importante de este mundo la vida, cariño, buenos consejos y el apoyo necesario para lograr esta meta.

A mis hermanos por la unión que siempre hemos mantenido.

A mis abuelitos por ser parte de mi vida.

Hugo Abad Merlín Pérez

Al Fís. Raymundo Hugo Rangel, por su
colaboración y apoyo en la realización
de este trabajo

Vicente René Hernández Olvera

Hugo Abad Merlín Pérez

Indice

CAPITULO 2
PLANEACION DEL PROYECTO

CAPITULO 3
MODELOS DE OBJETOS PARA LA IGU

CAPITULO 4
ANALISIS Y DISEÑO DE LA IGU

CAPITULO 5
IMPLEMENTACION EN LENGUAJE C++

CAPITULO 6
ANALISIS Y DISEÑO DE LA APLICACION

CAPITULO 7
IMPLEMENTACION DE LA APLICACION

INTRODUCCION

ANTECEDENTES

INTERFACES GRAFICAS

"Una imagen, dice más que mil palabras". Las interfaces gráficas no son nuevas, han estado presentes a través de toda la historia de la humanidad y corresponden a los más valiosos legados que nos han dejado las civilizaciones antiguas. Ejemplos de ellas podemos encontrar: en las paredes de las cavernas en España y Francia, en las pirámides de Egipto y en los muros de muchos otros países.

Dada la importancia histórica que han tenido las interfaces gráficas, éstas no pueden faltar como uno de los componentes más importantes de cualquier sistema de cómputo, y el éxito de éste, dependerá en gran medida del adecuado diseño de la interfaz gráfica.

Los sistemas de cómputo, frecuentemente carecen de buenas interfaces por una variedad de razones, incluyendo la falta de una buena metodología de diseño de interfaces de usuario y de buenas herramientas para implementar la interfaz. Una de las frecuentes tragedias de los desarrolladores de programas de aplicación, es la cantidad de tiempo y esfuerzo que invierten en diseñarlos e implementarlos, y que éstos resulten en una pobre interfaz de usuario.

Las interfaces de usuario, son casi siempre incluidas en revisiones de productos de *software*, simplemente, por que si la aplicación no es fácil de usar, algunos usuarios se ven frustrados y abandonan la aplicación, o bien, ésta no resultaría lo productiva que ellos quisieran. Los programas de aplicación son juzgados por muchos factores, que incluyen funcionalidad, velocidad y facilidad de uso. La facilidad de uso implica que los usuarios puedan ser capaces de aprender a usar la aplicación de una forma sencilla.

La interfaz gráfica convierte a la computadora en una herramienta más poderosa, en algo más familiar de usar. La mayoría de usuarios de sistemas operativos como DOS, UNIX, VMS; se ven obligados a estar tecleando comandos, sin embargo, no todos entienden bien la función que éstos realizan, es aquí donde sistemas operativos provistos con una interfaz gráfica, como: *Apple-Macintosh* y *Windows*, a través de representaciones gráficas nos dan una mayor información de la operación que se esta realizando, además de ahorrarle tiempo al usuario en proporcionar comandos. Por esta razón, no resulta extraño que sistemas operativos como UNIX y VMS hayan creado también sus propias interfaces gráficas.

JUSTIFICACION

Debido a la importancia actual de las interfaces gráficas de usuario en los sistemas de cómputo, se desarrollaron un conjunto de herramientas para crear interfaces gráficas que permitan el uso del ratón, de ventanas, menús, cuadros de diálogo, barras de despliegue, botones; de manera que estas utilerías puedan ser incorporadas por los programadores en diversas aplicaciones que así las requieran.

Se hizo la consideración de que estas herramientas pudieran utilizarse en equipos de cómputo de bajo costo y que no demandaran demasiados recursos como: memoria, espacio en dispositivos de almacenamiento, tipo de procesador; que es el caso de interfaces gráficas complejas como *Windows*; que es altamente demandante en cuanto a recursos de la computadora.

Apoyándonos en lo anterior, se decidió diseñar las herramientas para trabajar en equipos, que cuenten con un mínimo de memoria de 640 KB , con 1 MB de espacio en disco duro o una unidad de disco flexible, y con uno de los siguientes adaptadores gráficos: EGA, VGA o Hércules. En los tres adaptadores gráficos se trabajará en forma monocromática para que se tenga la misma calidad de presentación, se tomó en consideración también, que dichas tarjetas de gráficos cuentan con una resolución y cociente de aspecto adecuados para las herramientas.

Para facilitar el uso de la interfaz gráfica, se utiliza un dispositivo de posicionamiento en pantalla, el popular *mouse* o ratón; tomando en cuenta su bajo costo, así como sus características para el rápido manejo de los objetos gráficos.

La mayor parte de la programación de las herramientas para la interfaz gráfica, se realizó en un lenguaje orientado a objetos (C++), para aprovechar las ventajas que proporciona este tipo de lenguajes, y aplicándolo a objetos gráficos. Sin embargo, debido a algunas limitaciones de las librerías gráficas con las que cuenta la interfaz gráfica de *Borland* (BGI *Borland Graphic Interface)*, se usaron algunas funciones de una librería de gráfica en lenguaje ensamblador para hacer más rápida la interfaz; como es el caso del manejo de ventanas, de la escritura al área de memoria de los adaptadores de vídeo, y la escritura de cadenas de caracteres en modo gráfico. Otra parte de la programación fue desarrollada en lenguaje C convencional.

Como ejemplo del uso que se le puede dar a las herramientas desarrolladas, se realizó una aplicación sencilla que permite editar archivos de imágenes (monocromáticas.) Se eligió el formato PCX, ya que éste, es de los formatos más compactos en cuanto a código y más fáciles de manejar. Dicha aplicación permitirá ver los alcances que pueden proporcionar las herramientas creadas y el uso que pueda dárseles en otras aplicaciones.

CAPITULO 1. PRINCIPIOS Y METODOLOGIA DE LAS INTERFACES DE USUARIO

1.1. DISEÑO DE LA INTERFAZ DE USUARIO

Cuando se diseña una interfaz gráfica, se necesita considerar no sólo las operaciones de graficación que se efectuarán, sino también la forma en que estas operaciones se pondrán a disposición del usuario. Esta interfaz debe de diseñarse de manera que proporcione un medio adecuado y efectivo para que el usuario accese funciones de gráficos básicos como: el despliegue de objetos, establecimiento de atributos, realización de transformaciones, etc.

Cualquiera que sea el tipo de aplicación al que se destine la interfaz, necesitamos decidir que diálogo interactivo sirve mejor al usuario, el tipo de rutina de manipulación que se utilizará y los dispositivos de salida que resultan adecuados para el tipo de aplicación implicada. Una interfaz pobremente diseñada aumenta las posibilidades de que el usuario cometa errores, y puede incrementar significativamente el tiempo que tarda el usuario en realizar una actividad.

1.2. PROBLEMAS A LOS QUE SE ENFRENTAN LOS DISEÑADORES E IMPLEMENTADORES DE INTERFACES DE USUARIO

Se pueden identificar varios problemas en el diseño de las interfaces de usuario cuando se emplean en sistemas de aplicación y se deben principalmente a que:

- Los desarrolladores de la aplicación, no entienden el grado de conocimientos y familiaridad del usuario con la aplicación, y desarrollan interfaces que no son apropiadas para el usuario en cuestión.

- Los programadores invierten años aprendiendo como escribir programas, y no en saber como las personas usan las computadoras para hacer su trabajo.

- Los desarrolladores de aplicaciones no toman el tiempo necesario para diseñar, modelar y probar las interfaces de usuario.

- Todas las interfaces de usuario deben ser modeladas, probadas y evaluadas antes de ser introducidas a los usuarios.

1.3. FASES EN EL CICLO DE UNA INTERFAZ DE USUARIO

El ciclo de vida de la interfaz de usuario consiste de cuatro fases, que comprenden la determinación del diseño y planeación de implementación, formulación y análisis de requerimientos, diseño de la interfaz de usuario y la implementación de la interfaz. Las últimas tres fases se repiten hasta que la interfaz de usuario resultante es terminada.

1.3.1. Fase 1. Desarrollo del diseño de la interfaz de usuario y planeación de la implementación

Se define el alcance y metas de la interfaz de usuario. Se determina que aplicaciones podrá dar soporte la interfaz de usuario y los requerimientos de compatibilidad con las interfaces de usuario existentes.

Se determinan los recursos disponibles para diseñar e implementar la interfaz de usuario, que tipos de diseños e implementaciones serán necesarios durante el proyecto, hacia que usuarios será enfocada la interfaz y que herramientas son necesarias durante el diseño y desarrollo.

Se analizan los costos de implementación y mantenimiento de la interfaz gráfica de usuario propuesta, así como sus beneficios potenciales.

1.3.2. Fase 2. Formulación y análisis de requerimientos

Se identifican los usuarios potenciales de la aplicación y los problemas que se tendrán que resolver. Se categorizan los usuarios potenciales en clases, basados en su comprensión de la aplicación y su familiarización con las computadoras.

1.3.3. Fase 3. Diseño de la interfaz de usuario

Se diseñan los objetos gráficos con los cuales el usuario interactuará. Ejemplos de objetos de interacción incluyen: menús, cuadros de diálogo, cuadros de comandos, iconos, ventanas y otros tipos de objetos que presentan información. Los objetos de interacción son los mecanismos por los cuales, los usuarios manipulan objetos conceptuales. La mayoría de objetos de interacción, presentan información para desplegarla visualmente al usuario y también es presentada a veces en mensajes de audio.

El diseñador de diálogos especifica los comandos que describirán el intercambio de información entre el usuario, vía objetos de interacción y las aplicaciones fundamentales.

1.3.4. Fase 4. Implementación de la interfaz de usuario

Los diseñadores de las interfaces usan los prototipos y herramientas de implementación disponibles para desarrollar la interfaz de usuario.

Hay muchas herramientas en el mercado que pueden ser usadas para construir interfaces de usuario. Estas herramientas incluyen procesadores de comandos, sistemas formateadores de menús, sistemas de ventanas, conjunto de herramientas de interfaces de usuario y sistemas de manejo de interfaces.

1.4. TIPOS DE INTERFACES DE USUARIO

1.4.1. De comandos de diálogo

Este tipo de interfaz requiere que los usuarios sean capaces de formular comandos y especificar sus parámetros usando una sintaxis formal.

El usuario proporciona comandos en un cuadro de diálogo, tecleándolos. Este estilo de diálogo es empleado por usuarios experimentados que usan frecuentemente la aplicación.

1.4.2. De diálogo (de preguntas y respuestas)

El mecanismo de ejecución de la interfaz indica al usuario que proporcione comandos y parámetros, por medio de preguntas a las cuales el usuario responderá. Para usuarios expertos este estilo de diálogo resulta tedioso y repetitivo. Este estilo de diálogo es el más apropiado para usuarios con experiencia en los comandos y opciones usadas.

1.4.3. De menús y formatos de captura

El mecanismo de ejecución indica al usuario con menús y formatos de captura, en lugar de preguntas. El usuario responde seleccionando opciones de menús y proporcionando simples respuestas en los cuadros de captura. Este estilo de diálogo es apropiado para usuarios inexpertos.

1.5. COMPONENTES DE LA INTERFAZ DE USUARIO

1.5.1. Modelo de usuario

El diseño de la interfaz de usuario comienza con el modelo de usuario. El modelo determina la estructura conceptual que se le presentará al usuario. El modelo describe para lo que está diseñado el sistema y de que operaciones de graficación se dispone.

Una vez que se ha establecido el modelo de usuario, las otras componentes de la interfaz pueden desarrollarse. La etapa final en el diseño del modelo consiste en elaborar el manual del usuario, el cual explica el sistema y ofrece ayuda en su uso.

Básicamente, el modelo del usuario define el sistema de gráficas en términos de objetos y las operaciones que pueden efectuarse sobre éstos. Un programa de diseño de circuitos podría utilizar elementos eléctricos o lógicos para los objetos, con operaciones de posicionamiento disponibles para agregar o suprimir elementos dentro del diseño total del circuito.

Objetos como elementos de circuitos se denominan objetos de aplicación, además de estos objetos, el modelo del usuario podría contener otros objetos que se utilizan para controlar operaciones de graficación, estos se llaman objetos de control.

Ejemplos de objetos de control, son los cursores para seleccionar posiciones en la pantalla, símbolos de selección del menú y retículas de posicionamiento. Las operaciones de que se disponen en una interfaz gráfica de usuario permiten el uso de objetos de manipulación y de control.

Sólo deben de emplearse objetos bien conocidos en el modelo de usuario. El modelo no debe contener referencias de conceptos que puedan ser desconocidas para el usuario. Toda la información contenida en el modelo del usuario debe de presentarse en el lenguaje de aplicación.

En conjunto, el modelo del usuario debe de ser lo mas simple y consistente que sea posible. Un modelo complicado le es difícil de entender y le resulta complicado trabajar con él, en forma eficiente. El número de objetos y operaciones de graficación del modelo deben de minimizarse a sólo aquellas que se necesiten en la aplicación. Esto hace que al usuario le sea fácil aprender el sistema. Por otro lado, si la interfaz se simplifica demasiado, puede ser fácil de aprender pero difícil de aplicar. El diseñador del modelo de usuario debe de buscar también consistencia. Los objetos y las operaciones no deben definirse en diferentes formas cuando se utilicen en distintos contextos. Por ejemplo, un solo símbolo no debe servir como objeto de aplicación y objeto de control, dependiendo del modo de interacción. Esto hace difícil que un usuario lleve el registro del significado de los símbolos. Es mucho más fácil para un usuario de la interfaz, si los objetos y las operaciones se definen y utilizan de una manera consistente.

El punto inicial para elaborar un modelo de usuario a menudo consiste en realizar un análisis de la tarea. Las conclusiones del análisis de tareas puede formar la base para decidir que tipos de operaciones y objetos se necesitan, así, como la forma en que la interfaz gráfica debe de presentarse al usuario.

1.5.2. Lenguaje de comando

El lenguaje interactivo elegido debe de ser lo más natural posible para que el usuario lo aprenda, con todas las operaciones especificadas en términos relativos al área de aplicación. Los comandos deben de diseñarse de modo que el usuario no tenga que aprender nuevos conceptos, ni un nuevo lenguaje.

Minimización de la operación: Cada operación del lenguaje de comando debe de estructurarse de manera que un usuario pueda entenderla con facilidad y recuerde el objetivo de la misma. Deben evitarse los formatos de comandos confusos, complicados, inconsistentes y abreviados; ya que sólo confunden al usuario y reducen la efectividad de la interfaz.

El lenguaje de comando debe de estructurarse, de modo que, no se pida al usuario distraer su atención constantemente de un dispositivo de entrada a otro. Para un usuario menos experimentado, un lenguaje de comando con pocas operaciones fácilmente asimilables, es por lo general, más efectivo que un conjunto de operaciones grande y general. Un conjunto de comandos simplificado es fácil de aprender y de recordar, y el usuario puede concentrarse en la aplicación en vez de en el lenguaje.

No obstante, un usuario experimentado podría hallar algunas aplicaciones difíciles de manejar con un conjunto pequeño de comandos diseñado para el usuario principiante.

Para dar entrada a varios usuarios, los lenguajes de comando deben de diseñarse en varios niveles. Para los usuarios sin experiencia pueden utilizar el nivel más bajo, el cual contiene el conjunto mínimo de comandos y los expertos pueden usar los conjuntos mayores de comandos en los niveles superiores.

1.5.3. Facilidades de ayuda para el usuario

Es muy importante que se incluyan facilidades de ayuda en el lenguaje de comando. Diferentes niveles de ayuda permiten a los usuarios principiantes obtener instrucciones detalladas, mientras que los más experimentados pueden obtener solicitudes de entrada breves que no interrumpan su concentración.

Las facilidades de ayuda pueden incluir una sesión tutorial que ofrezca instrucción acerca de como utilizar el sistema. Un usuario principiante puede revisar la interfaz, hacer un repaso general de las funciones y de la forma en que opera el conjunto de comandos básicos.

Si se dispone de diferentes niveles de ayuda, un usuario principiante puede seleccionar el nivel de ayuda más elemental para recibir solicitudes de entrada y explicaciones detalladas en cada etapa durante la aplicación de la interfaz.

1.5.4. Respaldo de información y manejo de errores

Durante cualquier secuencia de operaciones debe de disponerse de algún mecanismo sencillo de respaldo o cancelación. Con frecuencia, una operación puede cancelarse antes de que se complete la ejecución, con el sistema restituido en el estado en que se encontraba antes de que se iniciará la operación. Con la capacidad del respaldo en algún punto, un usuario puede explorar con confianza las capacidades del sistema, sabiendo que pueden eliminarse los efectos de cualquier error.

Algunas veces un sistema puede respaldarse a través de varias operaciones, permitiendo al usuario volver a colocar el sistema en algún punto especificado. Si no hay una facilidad de respaldo, pueden utilizarse otros métodos para ayudar a los usuarios a superar los efectos de los errores. Podría pedirse a un usuario que verifique algunos comandos antes de ejecutar las instrucciones. Los diagnósticos efectivos y los mensajes de error deben incorporarse en el lenguaje de comando para permitir a los usuarios evitar errores y entender lo que estuvo mal cuando se haya cometido un error. El mensaje de error debe ofrecer una explicación clara de lo que anda mal y de lo que se necesita hacer para corregir esa situación. Normalmente, el sistema de recuperación despreciará una entrada incorrecta e informará al usuario del error, en una forma que ayude a éste a determinar la entrada adecuada en ese punto.

1.5.5. Tiempo de respuesta

Es el tiempo que tarda el sistema en responder a la entrada de un usuario dependiendo de la complejidad de la tarea solicitada. Para muchas solicitudes de entrada de rutinas, el sistema puede responder en forma inmediata. Cuando un usuario proporciona una solicitud de procesamiento complicada, puede esperarse alguna demora y este retraso podría utilizarse para planificar la siguiente fase de la aplicación.

Sin importar la complejidad de la solicitud de entrada, los usuarios pueden esperar que los sistemas proporcionen algún tipo de respuesta inmediata, de lo contrario, no pueden estar seguros de que la entrada fue recibida y que el sistema realiza el procesamiento. Debe de diseñarse una interfaz gráfica de usuario, para dar respuesta instantánea a la entrada del usuario. Si el tiempo de procesamiento va a ser largo, la respuesta inmediata simplemente permite al usuario saber que la entrada se ha recibido. Para el usuario principiante, esta respuesta podría ser un mensaje que afirme que la entrada esta siendo procesada. Con usuarios experimentados, un cursor centelleante, o un cambio de color o intensidad pueden servir para el mismo fin.

1.5.6. Tipos de lenguaje de comandos

Existen varios posibles tipos de lenguaje de comandos y la elección del formato de comandos de entrada depende de varios factores. Entre estos factores se incluyen los objetivos del paquete, el tipo de dispositivo de entrada que se utiliza y el tipo de usuario.

El lenguaje de comandos puede conformarse de modo que la secuencia de acciones de entrada sea dirigida por la interfaz gráfica o bien por el usuario.

Cuando la interfaz dirige la entrada, se indica al usuario que tipo de acción se espera en cada etapa. Este es un método particularmente efectivo para los usuarios principiantes, donde se utilizan solicitudes de entrada y menús para explicar lo que se pide, y como dar la entrada. Los usuarios en algunos casos, pueden verse restringidos a un número limitado de respuestas (simplemente sí, no, o bien, un valor numérico.)

En otros sistemas, puede dirigirse al usuario para que seleccione una opción, de una lista de posibles opciones. Esta selección podrá hacerse a partir de un menú, utilizando algún tipo de dispositivo de entrada. Una vez que se haya escogido la opción, como la selección de un objeto, el usuario puede proceder a dar entrada de los parámetros adecuados.

Podrían utilizarse algunos menús y solicitudes de entrada para recordar al usuario las opciones de que dispone con cualquier operación seleccionada; en general, el usuario tiene la libertad de explorar las capacidades del sistema, sin seguir una secuencia de acciones fija.

Cuando sólo se establece un diálogo, no se pone a disposición del usuario ningún conjunto de comandos. La entrada se logra seleccionando opciones de un menú y dando respuestas simples a las solicitudes de entrada. Este método de obtención de la entrada es adecuado para principiantes, pero es ineficaz para aquellos usuarios que tienen más experiencia, los cuales pueden utilizar mejor las capacidades de la interfaz con un conjunto de comandos de entrada. Los comandos pueden diseñarse para que puedan ser proporcionados con las teclas de función, o desde un teclado estándar. Para minimizar el tiempo de aprendizaje del usuario, la sintaxis de los comandos de entrada debe de ser simple y directa.

1.5.7. Diseño del menú

Cuando se emplean menús en un programa, el usuario es liberado de la carga de recordar opciones de entrada. Esto no sólo reduce la cantidad de memorización que requiere el usuario para listar la gama de opciones disponibles; sino que también impide que el usuario seleccione opciones que no sean válidas en ese punto. Además, los menús pueden alterarse fácilmente para dar cabida a diferentes aplicaciones, mientras que las teclas de funciones deben de reprogramarse y volverse a rotular si se alteran.

Los menús pueden utilizarse como el mecanismo de entrada de operaciones y de parámetros. La selección iterativa del menú puede realizarse con muchos tipos de dispositivos de entrada; pueden utilizarse teclados, el ratón y otros dispositivos para hacer selecciones posicionando el cursor u otro símbolo en una opción de menú.

También puede utilizarse un teclado para dar el nombre de identificación o el número de un elemento de menú. En general, los menús con menos opciones son más efectivos, ya que reducen la cantidad de tiempo de búsqueda que se necesita para encontrar una opción determinada y ocupan menos espacio en la pantalla. Normalmente, los menús se colocan en un lado de la pantalla, de modo que no interfieran con la imagen desplegada. Cuando se va a presentar un menú extenso con largas descripciones de cada opción, quizá tenga que ocupar toda la pantalla, de tal manera, que la imagen y el menú se desplieguen en forma alternada. Esto puede tener un efecto muy negativo en la sucesión de ideas del usuario, ya que la continuidad visual con la imagen se pierde cada vez que se hace una selección del menú.

Los menús que ocupan toda la pantalla deben evitarse acortando las descripciones de las opciones de menú y dividiendo las selecciones en dos o más submenús. Los elementos listados en un menú pueden presentarse como cadenas de caracteres, o bien, como iconos gráficos; las ventajas de los iconos son que, por lo general, ocupan menos espacio y se reorganizan más rápidamente que las descripciones de texto correspondientes. Un usuario sin experiencia puede encontrar los iconos más difíciles de usar en un principio, pero una vez que aprenda el conjunto de iconos, las entradas pueden hacerse más rápido y con menos errores.

1.5.8. Retroalimentación

Una parte importante de cualquier sistema de gráficas es la cantidad de retroalimentación suministrada a un usuario. El sistema necesita realizar un diálogo interactivo e informar al usuario lo que está haciendo el sistema en cada etapa. Esto es particularmente importante cuando el tiempo de respuesta es alto. Sin retroalimentación, un usuario puede empezar a preguntarse que esta haciendo el sistema y si los datos tienen que volverse a proporcionar. Conforme cada entrada del usuario es recibida, debe aparecer una respuesta inmediatamente en la pantalla.

El mensaje debe de ser breve e indicar con claridad el tipo de procesamiento que esta teniendo lugar. Esto no sólo informa al usuario que la entrada se ha recibido, sino que también le indica que el sistema esta trabajando; de manera que, pueda corregirse cualquier error de entrada. Si el procesamiento no puede efectuarse de forma inmediata, deberán de desplegarse algunos mensajes para mantener informado al usuario del avance del sistema; como bien puede ser algún mensaje que este centelleando, o un icono de espera que sirva para indicar al usuario que el sistema sigue trabajando. El sistema también podría permitir al usuario dar otros comandos o datos mientras se procesa una instrucción.

Los mensajes de retroalimentación deben de darse con la claridad suficiente para que tengan poca oportunidad de ser pasados por alto. Por otro lado, no deben de ser tan abrumadores que interrumpan la concentración del usuario. Cuando se despliegan mensajes en la pantalla, puede usarse una área de mensaje fija, de manera que sepa siempre donde buscar los mensajes.

En algunos casos, puede ser útil colocar mensajes de retroalimentación en la área de trabajo del usuario próxima al cursor, también pueden usarse diferentes colores para distinguir la retroalimentación de otros objetos desplegados. Otros métodos que podrían utilizarse para la retroalimentación de la selección del menú incluyen: el realce, el cambio de color y centelleo del elemento seleccionado.

El tipo de retroalimentación también dependerá del tipo de usuario, ya que un usuario principiante requiere de una retroalimentación más detallada, que no sólo indique con claridad lo que el sistema esta haciendo, sino también los datos que el usuario debe de proporcionar a continuación.

1.5.9. Formatos de salida en pantalla

La información que se presenta al usuario en una interfaz gráfica incluye: una combinación de imágenes, menús, mensajes de salida y otras formas de diálogo generadas por el sistema. Existen muchas posibilidades para disponer y presentar esta información de salida al usuario y al diseñador de un paquete de gráficas; se debe de considerar la mejor manera de diseñar los formatos de salida para lograr la mayor efectividad visual. Las consideraciones en el diseño de formatos de salida incluyen: estructuras de menús y mensajes, iconos y la área de trabajo. Las estructuras de menús y mensajes dependen de varios factores, además del nivel de experiencia del usuario y del tipo de aplicación, la estructura de menús y mensajes se verá influenciada por el área de trabajo de la pantalla.

La estructura de muchos símbolos que se utilizan en un paquete de gráficas depende del tipo de la aplicación para la cual se dirige la interfaz. Los iconos se escogen de modo que ofrezcan una imagen todavía más clara y simple del objeto u operación que se supone deben representar. Otros símbolos, como los cursores o apuntadores de menús, deben de diseñarse para que sean claramente diferentes de los otros iconos. En algunos casos, la interfaz podría permitir al usuario especificar la forma de algunos símbolos.

Hay tres componentes básicos del área de la pantalla, estos son: el área de trabajo del usuario, el área de menú y el área para solicitudes de entrada y mensajes de retroalimentación. Para hacer el área de trabajo lo más grande posible, las áreas de menú y mensajes deben de minimizarse. Si se desea una área de trabajo muy grande, las otras áreas podrían suprimirse cuando no se necesiten, de modo que el área de trabajo pueda ampliarse hasta llenar la pantalla. Una forma de permitir un empleo máximo de la pantalla, consiste en dar al usuario algún control sobre el tamaño de las áreas de menús y mensajes.

Puede ofrecerse mayor flexibilidad al organizar proyectos en la pantalla, permitiendo al usuario que se forme cualquier número de áreas de ventanas en sobreposición. En este esquema, el usuario especifica el área de la pantalla en la cual se desplegará un menú o imagen. A medida que se coloca cada nueva ventana en la pantalla, puede superponerse y oscurecer ventanas creadas con anterioridad.

1.5.10. Dispositivos de entrada de datos

Las interfaces gráficas utilizan varios tipos de datos de entrada, entre ellos se incluyen: valores de posiciones de coordenadas, cadenas de caracteres, valores que especifican opciones de un menú para la entrada de diferentes tipos de valores. Para proporcionarse estos datos pueden utilizarse diferentes tipos de dispositivos de entrada los cuales estarán en función del tipo de datos que quieren ser introducidos.

1.5.10.1. Clasificación de los dispositivos de entrada de datos

- **Dispositivos de localización:** este tipo de dispositivos permiten especificar una posición coordenada en la pantalla (x,y). Ejemplos de este tipo de dispositivos son: la palanqueta (*joystick*), cursores manuales, el lápiz óptico y el ratón.

- **Dispositivos de trazo:** sirven para especificar una serie de posiciones coordenadas en la pantalla, la entrada del dispositivo de trazo es equivalente a multiplicar varias veces las acciones realizadas por un dispositivo de localización, ejemplos de este tipo de dispositivos son: el lápiz óptico, tabletas ópticas y el ratón cuando tiene un movimiento continuo.

- **Dispositivos de cadena:** permiten proporcionar una entrada de cadena de caracteres, el dispositivo más común para realizar este propósito es el teclado.

- **Dispositivos evaluadores:** sirven para especificar valores escalares en sistemas de gráficas como: el ángulo de rotación, factores de escalamiento y de transformación. Un ejemplo de este tipo de dispositivos son los discos marcadores de control.

- **Dispositivos de elección:** son dispositivos para seleccionar opciones de un menú, en esta clasificación se encuentran: el lápiz óptico, los teclados de funciones y el ratón.

- **Dispositivos de recolección:** permiten seleccionar componentes de una imagen (fragmentos) y realizar transformaciones de figuras; el lápiz óptico se encuentra en ésta clasificación.

CAPITULO 2. PLANEACION DEL PROYECTO

2.1. VENTAJAS QUE OFRECE UNA INTERFAZ GRAFICA DE USUARIO

- Facilita la interacción del usuario en la utilización de programas de aplicación, ya que le proporcionan una serie de objetos gráficos que simplifican y hacen más amigable el uso del programa.

- A través del manejo de objetos gráficos , es más fácil que el usuario recuerde estos objetos, a que este aprendiendo largos y complicados comandos.

- La interfaz provee de una mejor presentación al programa de aplicación, con la incorporación de iconos, botones, cuadros de diálogo, menús.

- Con el manejo de dispositivos de posicionamiento en pantalla, se tiene un más rápido acceso a los datos y a las operaciones que realice el usuario.

2.2. NECESIDADES

- Proporcionar a los programadores herramientas que permitan la creación de interfaces gráficas, de una forma rápida y sencilla. Estas herramientas deberán reunir los elementos necesarios (objetos gráficos), para poder crear una interfaz gráfica de usuario completa.

- Las herramientas deberán de ser fáciles de usar en aplicaciones que así lo requieran y con un mínimo de esfuerzo por parte del programador para incorporarlas en sus programas.

- Diseñar herramientas para creación de interfaces que sean compatibles con el *hardware* existente, que no requieran de costosos equipos de cómputo, ni demasiados recursos del equipo. Además, deberán desarrollarse en un lenguaje de programación que reúna todas las características que permitan una rápida adecuación en los programas de aplicación y que permitan darle un fácil mantenimiento a los mismos.

- Los objetos que formen parte de las herramientas para construir interfaces gráficas deberán ser compatibles con las ya existentes, para que el usuario no pierda la familiaridad que tiene con otros tipos de interfaces gráficas.

- Deberá contar con un dispositivo de posicionamiento en pantalla, para que la interfaz sea más rápida de utilizar por medio de un ratón, palanqueta, lápiz óptico; o algún otro dispositivo de este tipo.

2.3. REQUERIMIENTOS PARA EL DISEÑO Y PLANEACION DE LA IMPLEMENTACION

2.3.1. Necesidades de *hardware*

2.3.1.1. De adaptadores de video

Para tener una mejor presentación y calidad de gráficos, la interfaz gráfica deberá trabajar en modo gráfico, para esto, el *hardware* de vídeo que se use, deberá de ser el que reúna las mejores características en cuanto a resolución, cociente de aspecto y ser un tipo de tarjeta común en el mercado. En la tabla que se muestra a continuación podemos ver las características de algunas de las tarjetas de vídeo más comunes en el mercado:

Tipo de tarjeta de video	Resolución (pixeles)
VGA (Video Graphics Array)	640x480
EGA (Enhanced Graphics Array)	640x350
CGA (Color Graphics Adapter)	640x200
Hércules	720x348

Las tarjetas de vídeo que cumplen mejor con los requisitos anteriormente mencionados, son: VGA (640 x 480 pixeles), EGA (640 x 350 pixeles) y Hércules (720 x 348 pixeles), por lo que se consideró que las herramientas serían realizadas para trabajar en estos tres tipos de tarjetas gráficas. Además, la interfaz debe de tener el mismo tipo de presentación para las tres clases de tarjetas, es decir, que las características de la interfaz no estén limitadas por el tipo de adaptador de vídeo. Por esta razón, la interfaz se manejará en forma monocromática. De esta manera, la exigencia de recursos de la computadora serán menores, ya que no se requiere de mucha memoria para almacenamiento de colores y su calidad es la misma para los tres tipos de adaptadores de vídeo.

2.3.1.2. De dispositivos de entrada de datos

La interfaz debe emplear un dispositivo de posicionamiento o localización en pantalla, para hacerla más fácil y rápida de usar. Entre los tipos de posicionamiento en pantalla más comunes encontramos: las palanquetas o *joysticks*, lápiz óptico, el ratón o *mouse*.

Por las características mencionadas en el capítulo anterior en relación a estos dispositivos, se considera que el ratón sea el dispositivo a ser utilizado en la interfaz, por ser un tipo de dispositivo que tiene mucha aplicación en las interfaces gráficas, por la facilidad para poder controlarlo (a través de su programa controlador o *driver* y por medio de interrupciones del sistema operativo), se consideró también su bajo costo y que la gran mayoría de usuarios cuenta con uno de estos dispositivos.

2.3.2. Requerimientos de *software*

2.3.2.1. Herramientas existentes para creación de gráficos

Se investigó sobre librerías gráficas existentes que pudieran ser empleadas en nuestro programa, como es el caso de la librería gráfica de Borland (BGI Borland Graphic Interface, GRAPHICS.H); se encontró una librería gráfica que contiene funciones en lenguaje ensamblador para el manejo de texto en modo gráfico, el manejo del adaptador de video y para limpiar la pantalla. Estas funciones tienen un mejor desempeño, comparado al de las funciones de la librería gráfica de Borland, que resultan muy lentas, sobre todo cuando el equipo en el que son utilizadas es una PC XT o una PC con procesador 80286.

2.3.2.2. De lenguaje de programación

Dado que se trata de un programa que manejará las características del adaptador de video en modo gráfico, y que además trabajará directamente con objetos gráficos; se tratará de aprovechar las características de lenguajes de programación que permitan hacer un manejo más óptimo del modo gráfico de la tarjeta de video.

Implementar la interfaz gráfica en un lenguaje orientado a objetos, tiene muchas ventajas y entre éstas, podemos mencionar algunas características de este tipo de programación, como el uso de: clases, objetos, variables de instancia, herencia, las facilidades de mantenimiento del código y además que estás características pueden ser aplicadas de forma más clara y sencilla a objetos de tipo gráfico. El lenguaje deberá también ser compatible con otros lenguajes; que reúna las características para un adecuado manejo del modo gráfico y de dispositivos de posicionamiento (el ratón en este caso); que tenga además disponibilidad y uso entre los programadores a los que estará enfocado el desarrollo de estas herramientas.

Tomando en cuenta los puntos anteriores se consideró hacer la programación en lenguaje C++, para aprovechar todas las características que ofrece un lenguaje de programación de este tipo. Por otro lado, como se mencionó anteriormente se utilizarán algunas rutinas escritas en lenguaje ensamblador y otras en lenguaje C. Esto para ciertas operaciones críticas en el diseño de interfaces gráficas como son: el manejo del adaptador gráfico, de ventanas y texto; así como de otras funciones que no requieran de la utilización de técnicas de programación orientada a objetos y que resultan más fáciles programarlas de la forma convencional en lenguaje C.

2.4. REQUERIMIENTOS PARA LA CREACION DE LAS HERRAMIENTAS PARA LA INTERFAZ GRAFICA

2.4.1. De objetos gráficos

Las herramientas ha desarrollarse deberán cubrir los objetos necesarios para tener una interfaz gráfica lo más completa posible. Todo el conjunto de herramientas formarán parte de una librería, la cual podrá ser utilizada por los programadores en sus diferentes aplicaciones.

Los objetos de interacción, que formarían parte de las herramientas y que consideramos como indispensables para una interfaz gráfica, son:

- ventanas
- menús
- cuadros de diálogo
- cuadros de captura o edición
- iconos
- barras de despliegue
- listas
- objetos de tipo texto (botones, mensajes)
- cursores

2.4.2. El tipo de usuario

Dado que la interfaz usará algunos objetos gráficos y un dispositivo localizador (no muy familiares para todos los usuarios); la interfaz será enfocada a usuarios que tengan ya una cierta experiencia en el manejo de un sistema de cómputo y que estén familiarizados con el uso del ratón. La forma de interacción con el usuario, sería a través de menús y formatos de captura; en lugar de sólo cuadros de preguntas y respuestas. El usuario respondería seleccionando opciones de menús, por medio del ratón y proporcionando información en los cuadros de captura. Este tipo de interfaz es el apropiado para los usuarios que cumplen con las características antes mencionadas.

2.5. ALCANCES

El uso que pueda dársele a las herramientas es muy amplio y depende sólo de la habilidad del programador para utilizarlas dentro de sus programas de aplicación.

Las herramientas podrán ser utilizadas para crear interfaces en programas de dibujo, de edición de texto; para elaborar interfaces de manejadores de bases de datos, administradores de archivos, herramientas de escritorio y en muchos otros programas de aplicación.

Las herramientas no estarán limitadas, siempre se tendrá la oportunidad de seguir desarrollando más código en base a éstas y pudiéndose crear objetos más sofisticados y por que no, llegar a formar una interfaz gráfica tan completa como *Microsoft-Windows*.

CAPITULO 3. MODELOS DE OBJETOS PARA LA IGU

3.1. OBJETOS GRAFICOS

3.1.1. Modos gráficos

Para la interfaz gráfica se utilizará el modo gráfico de los adaptadores de vídeo: EGA, VGA y Hércules. Se manejarán las resoluciones de: 640 x 350 pixeles para EGA, 640 x 480 para VGA y 720 x 438 para la tarjeta Hércules (en las tres tarjetas en forma monocromática.)

3.1.2. Escritorio

Es el fondo que cubre toda la pantalla, y sobre el cual son colocados los objetos gráficos como: ventanas, iconos, cuadros de diálogo. El escritorio puede tener un color de fondo o puede estar formado por imágenes, sus dimensiones dependerán de las características del adaptador gráfico.

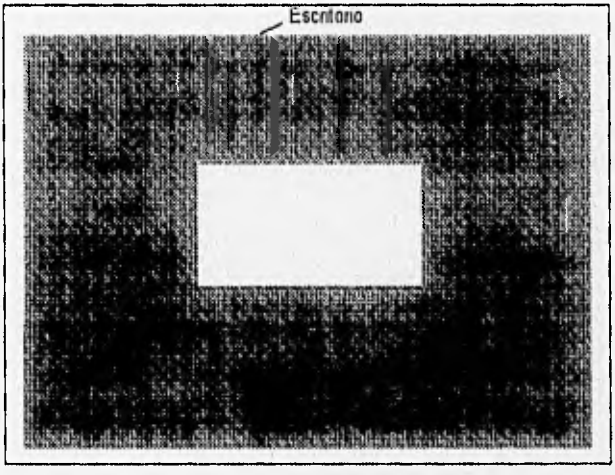


Fig. 3.1. Ejemplo de un escritorio.

3.1.3. Ratón

El ratón es un dispositivo de entrada , que consta de una pequeña caja de manipulación manual con una o varias ruedas en la base, que a medida que el ratón es empujado a través de una superficie plana, la(s) ruedecilla(s) registran la cantidad y dirección del movimiento, para convertirlo a un movimiento correspondiente del cursor en la pantalla. Los botones situados en la parte superior de este dispositivo, se emplean como interruptores para señalar la ejecución de alguna operación, como es el registro de la posición del cursor.

El ratón nos permite especificar una posición en la pantalla, tanto en modo de texto, como en modo gráfico. A través de un cursor (de forma: de flecha, de reloj de espera, I-centelleante, de cruz y otros), se puede hacer: la selección de comandos, presionar botones, crear gráficos, seleccionar texto. Para hacer uso del ratón se requiere de un programa que nos permita la interacción con el dispositivo físico (el archivo controlador del ratón.) El programa controlador del ratón realiza llamadas a las funciones de la interrupción 33h de DOS, por medio de los valores adecuados en los registros AX, BX, CX y DX, se pueden realizar operaciones con el ratón como: activarlo, desactivarlo, colocar un tipo de cursor, conocer el estado de sus botones, moverlo a la posición que se necesite; así como, conocer su posición actual dentro de la pantalla.

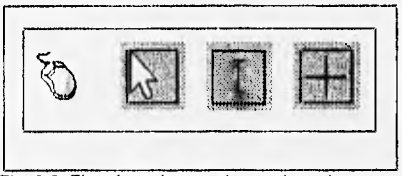


Fig. 3.2. El ratón y algunos de sus tipos de cursor.

3.1.4. Recuadros y ventanas

El recuadro tiene como función principal, el definir el área rectangular sobre la pantalla en base a dos coordenadas (x1,y1) y (x2,y2).

Los objetos de tipo ventana heredan las propiedades del recuadro, es decir, el definir una área rectangular en la pantalla; pero a diferencia del recuadro, la ventana es un objeto que ocupa espacio en memoria. Una ventana esta formada por una área rectangular (por lo general de color blanco) del tamaño de la ventana; así como, un cuadro negro alrededor de ésta (el marco o estructura de la ventana.) En los objetos ventana pueden colocarse: cadenas de texto, iconos, botones, y otros tipos de objetos gráficos.

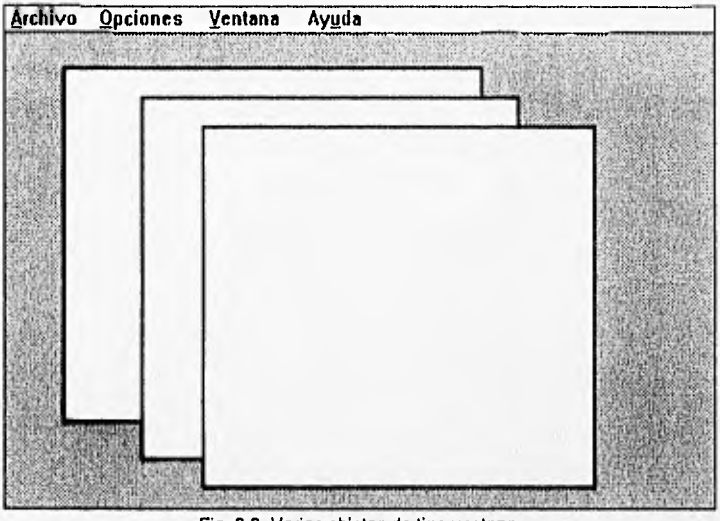


Fig. 3.3. Varios objetos de tipo ventana.

3.1.5. Menús

Un menú es la derivación de una ventana, como puede verse en la figura 3.4, que muestra la estructura de los menús. Los menús típicos están formados por una área rectangular colocada en la parte superior de la pantalla, que recibe el nombre de barra de menús (1), que contiene los títulos de los menús y cuya finalidad es definir las funciones de éstos, en grupos lógicos; si se selecciona uno de los títulos, un menú colgante (*pull-down*) aparecerá debajo de la barra de menús (2).

Los componentes de un menú colgante, son los índices del menú que se refieren a los nombres de las opciones que están dentro de un menú. Cada menú tiene un número fijo de líneas y columnas. El menú tiene una extensión horizontal y la altura se determina por el número de líneas de los índices del menú.

La distancia entre índices de un menú, en un menú horizontal se conoce como el espaciado de índices de menú. Los estados de un índice de menú pueden ser: activo, que aparecerá en letra normal (4); inactivo, que aparecerá en un color más claro, por lo general en color gris (5). Si el índice esta siendo seleccionado, éste aparecerá con los colores invertidos (3).

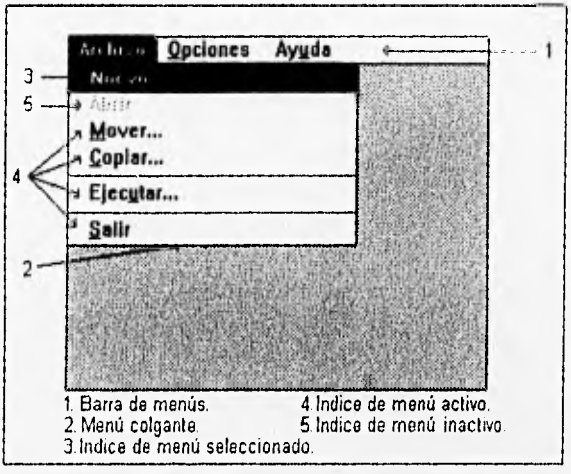


Fig. 3.4. Estructura de los menús.

3.1.6. Barras de despliegue

Son barras que aparecen en la parte inferior y/o en el extremo derecho de una ventana. Podemos tener barras de despliegue verticales y horizontales. Las barras de despliegue consisten de cinco elementos o partes, usando la barra de despliegue vertical como ejemplo base, hay dos flechas en los extremos de la barra; una hacia arriba y otra hacia abajo. El cuadro en el centro de la barra es el indicador de desplazamiento, el cual muestra la posición actual en la pantalla, en relación con el contenido total de la ventana; las zonas donde se mueve el indicador, son las zonas de desplazamiento inferior y superior. Ver la figura 3.5. que muestra ejemplos de las barras de despliegue.

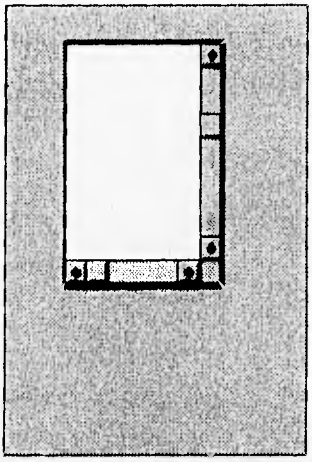


Fig. 3.5. Barras de despliegue: horizontal y vertical.

3.1.7. Mapas de bits e iconos

El mapa de bits, es una imagen almacenada en forma de un diseño creado por puntos o pixeles en pantalla. Podemos catalogar los mapas de bits en dos clases: mapas de bits de gran tamaño y mapas de bits pequeños o iconos.

Los mapas de bits de gran tamaño no tienen asociada ninguna actividad a realizar, el objetivo principal que tienen es sólo decorativo, ya que son imágenes demasiado grandes y que consumen mucho tiempo en ser desplegadas en pantalla. Como resultado de esto, no es conveniente usarlos para invocar alguna actividad.

Fig. 3.6. Un mapa de bits decorativo.

Los mapas de bits pequeños o iconos, son símbolos o figuras que pueden representar comandos complejos, en lugar de tener que teclear el comando o seleccionar uno de una lista de menús. Un icono puede ser seleccionado para invocar el comando, con la seguridad de que el icono ejecutará la actividad que representa. Las dimensiones de este tipo de mapas de bits se recomiendan, de acuerdo a los estándares de interfaces gráficas, que sean hasta de un máximo de 32 x 32 pixeles.

Fig. 3.7. Varios mapas de bits o iconos.

3.2. OBJETOS GRAFICOS DE TIPO TEXTO

Podemos definir seis objetos de tipo texto, éstos son: texto, campos de texto, campos de edición de texto, cuadros de selección, botones y listas. Todos estos objetos, tienen en común que están formados por simples cadenas de texto; sin embargo, cada uno de ellos tiene una función diferente, como se explicará a continuación.

3.2.1. Texto

El objeto de tipo texto más sencillo, es el objeto texto. Un objeto texto es una simple línea de texto que puede ser colocada dentro de una ventana. Este objeto tiene las características de no poder ser seleccionado, ni se puede alterar su contenido posteriormente en la ventana; éste puede ser únicamente leído y permanece todo el tiempo fijo.

El objeto de tipo texto, representa al tipo de control más simple en una interfaz gráfica. En la figura 3.8 se muestra un objeto de tipo texto.

3.2.2. Campos de texto

Los controles de tipo campos de texto, son parecidos a los controles de tipo texto, con la diferencia que el campo texto puede ser modificado después de haber sido agregado a una ventana, ya que este tipo de objetos manejan cadenas de longitud variable. Por ejemplo, el nombre de un archivo que es seleccionado en un cuadro de selección de archivos, requeriría de objetos de tipo campos de texto. Mientras que simples objetos de texto, que no cambien de apariencia durante la apertura de una ventana, pueden ser utilizados para el título de un menú o de un cuadro de diálogo.

Un campo de texto, define la longitud asociada con el texto y dibuja un recuadro alrededor de éste. Ver la figura 3.8, donde se muestra un objeto de tipo campo de texto.

3.2.3. Campo de edición de texto

El objeto campo de edición de texto, representa uno de los controles más complejos en la interfaz gráfica. Un objeto de campo de edición de texto, es similar al objeto campo de texto, excepto que en este tipo de objetos, puede ser editado su contenido si es seleccionado por el cursor del ratón. La característica más importante del objeto campo de edición es que éste puede ser seleccionado, si esto ocurre, un cursor vertical (en forma de I) aparecerá dentro de un recuadro, en donde podrá ser editado el texto. Ver figura 3.8.

3.2.4. Cuadros de selección

Este tipo de objetos tiene un pequeño círculo o cuadro, y texto asociado a éste, indicando el tipo de actividad que realizará si es seleccionado. El cuadro de selección tendrá dos estados: el de selección y no selección, si es seleccionado cambiará su apariencia, con algún tipo de relleno o con una marca (X). Generalmente, son colocados grupos de objetos de este tipo dentro de una ventana, y uno o varios de ellos pueden estar en estado de selección. Ver figura 3.8.

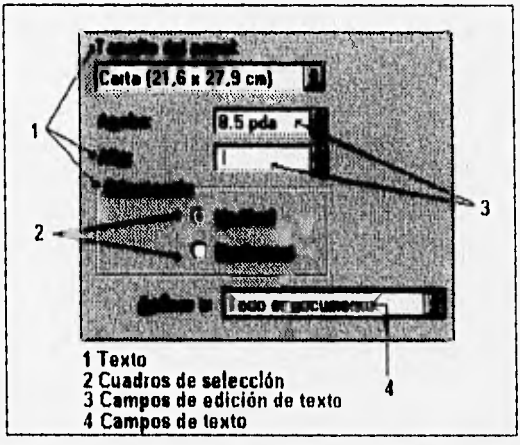


Fig. 3.8. Ejemplos de objetos texto: texto, campo de texto, campo de edición de texto y cuadros de selección.

3.2.5. Botones

Los botones son objetos utilizados por la interfaz gráfica como objetos de control. Un botón consiste de una área rectangular y tiene dos estados: en el primero, el botón puede estar activo o inactivo (que es el estado normal); el otro estado, es el estado de selección del botón, que aparecerá resaltado en otro color cuando es seleccionado por el ratón.

Los botones tienen un elemento texto que permite identificar la función que realizan, como ejemplo de ello tenemos que sirven para cerrar ventanas y cuadros de diálogo (éstos reciben el nombre de botones de comando), los objetos de tipo botón suelen tener asignada una cadena de texto o un icono en su interior, para describir la acción que realizan (por ejemplo: botones para *Cancelar, Aceptar, Sí, No.*)

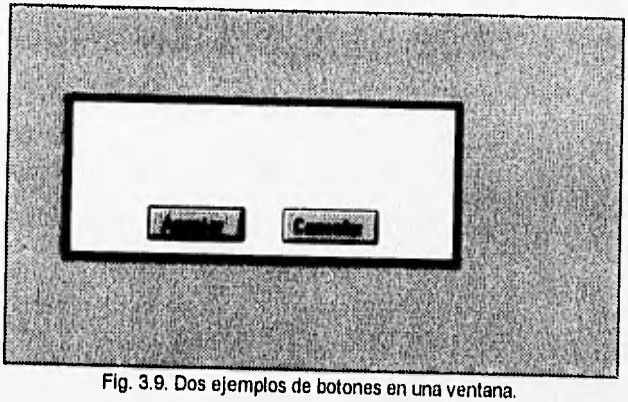


Fig. 3.9. Dos ejemplos de botones en una ventana.

3.2.6. Listas

Dentro de una ventana de aplicación, el objeto lista es un conjunto de cadenas disponibles en un recuadro (por ejemplo, una lista de todos los archivos contenidos en un directorio, ver fig.3.10.) Los objetos de tipo lista, aparecen siempre relacionados con las barras de despliegue, si el cuadro de la lista no puede contener toda la información existente, aparecerá una barra de desplazamiento para poderla mostrar y la información puede ser observada moviendo la barra.

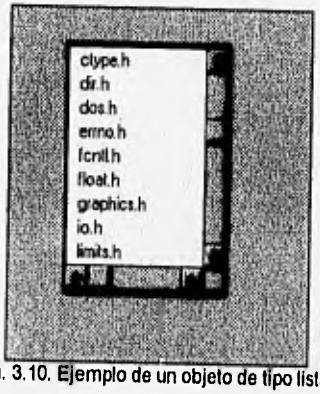


Fig. 3.10. Ejemplo de un objeto de tipo lista.

CAPITULO 4. ANALISIS Y DISEÑO DE LA IGU

4.1. DEFINICION DE LAS CLASES PARA LA IGU

Basándonos en las características de los objetos gráficos para interfaces gráficas que fueron mencionadas en el capítulo anterior, se realizó el siguiente análisis, considerando como clases base para otras clases, a las clases recuadro y texto.

Objeto:	Hereda propa:		Funciones:
	Recuadro	Texto	
Gráficos	X		- Constructor de la clase gráficos (para activar el modo gráfico.) - Destructor de la clase gráficos (para apagar el modo gráfico.)
Ratón	X		- Constructor de la clase ratón. - Función para detectar el controlador del ratón. - Función para activar el ratón. - Función para colocar el cursor del ratón en la pantalla. - Función para mover el ratón. - Función para detectar el estado de los botones del ratón. - Función para conocer la localización del ratón dentro de la pantalla. - Función para colocar el tipo de cursor. - Cerrar la ventana. -Destructor.
Ventana	X		- Constructor de la clase ventana. - Función para abrir la ventana. - Función para cerrar la ventana. - Destructor.

Objeto:	Hereda propa:		Funciones:
	Recuadro	Texto	
Menú	X		- Constructor de la clase. - Función para dibujar barra de menús. - Función para colocar títulos de menús. - Función para dibujar menús colgantes. - Destructor.
Barras de despliegue	X		- Constructor de la clase. - Función para dibujar barras de despliegue horizontales . - Función para dibujar barras de despliegue verticales. - Destructor.
Mapas de bits	X		- Constructor de la clase. - Función para dibujar el mapa de bits. - Destructor.
Objeto texto	X	X	- Constructor de la clase - Función para colocar el objeto texto - Destructor
Campos de texto	X	X	- Constructor de la clase - Función para colocar el campo de texto - Destructor
Campo de edición de texto	X	X	- Constructor de la clase - Función para dibujar el objeto campo de edición de texto - Destructor
Cuadros de selección	X	X	- Constructor de la clase - Función para dibujar el cuadro de selección - Destructor
Botones	X	X	- Constructor de la clase - Función para dibujar el objeto botón - Destructor
Listas	X	X	- Constructor de la clase listas - Función para dibujar el objeto lista - Destructor

En la figura 4.1, se muestra el diagrama de bloques de las clases para la interfaz gráfica:

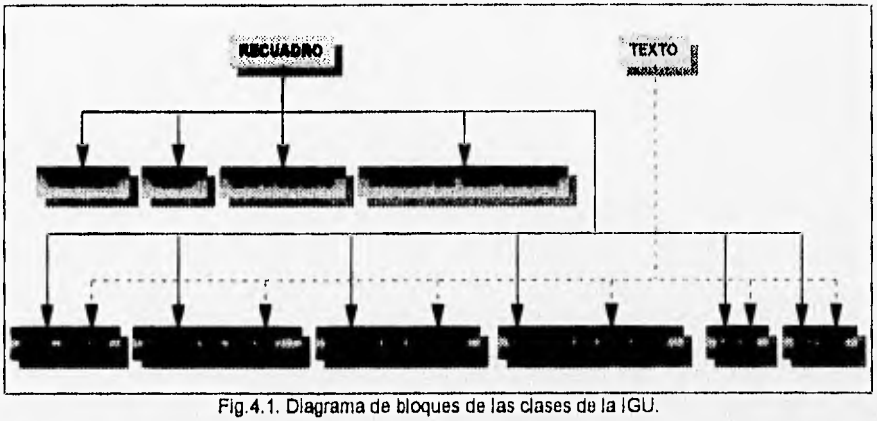


Fig.4.1. Diagrama de bloques de las clases de la IGU.

4.2. DEFINICION DE DATOS PARA LOS OBJETOS DE LA INTERFAZ GRAFICA

En la siguiente tabla se hace el análisis de los datos necesarios para definir los objetos gráficos:

Objeto	Datos
Adaptador gráfico	- Controlador gráfico. - Modo gráfico.
Ratón	- Registros AX, BX, CX, DX; para almacenar la información del ratón. - Coordenadas (x1,y1) para posición del ratón.
*Recuadro (Clase base)	- Coordenadas (x1,y1,x2,y2)
Ventana	- Coordenadas del recuadro (x1,y1,x2,y2) - Tamaño de la ventana (tamaño) - Estructura de datos para asociarle objetos a la ventana, una lista ligada (apuntador.)

Objeto	Datos
Menús	- Coordenadas para la barra de menús (x1,y1,x2,y2). - Estructura con los datos para los títulos de los menús. a) Título del menú (menú). b) Contador del número de títulos de menús (contador). - Estructura para almacenar los índices de menú que contienen: a) Nombre del índice (índice). b) Función asociada al índice (función). - Estructura para definir un menú completo incluyendo título e índices: a) Contador del número de índices de menú (contador). b) Datos de los títulos de los menús (título). c) Arreglo con los datos de los índices de menús (arreglo_índices).
Barras de despliegue	- Coordenadas para los recuadros que forman los elementos de las barras: a) Recuadro flecha1 (x1,y1,x2,y2). b) Recuadro flecha2 (x3,y3,x4,x4). c) Recuadro para el indicador de deslizamiento o posición (x5,y5,x6,y6). d) Recuadro hacia arriba (barra de despliegue vertical) o a la izquierda (barra de despliegue horizontal): (x7,y7,x8,y8). e) Recuadro hacia abajo (barra de despliegue vertical) o a la derecha (barra de despliegue horizontal): (x9,y9,x10,y10). - Coordenadas de inicio de la barra de despliegue (x11,y11). - Estructura de datos para indicar la ventana en que se encuentra asociado, requiere de una lista ligada (apuntador). - Estado de actividad para la barra de despliegue, si puede o no ser usada (actividad). - Posición del cursor, la barra indicadora (posición). - Longitud máxima y mínima de la barra (max, min).

Objeto	Datos
Mapas de bits	- Coordenadas (x1,y1,x2,y2). - Tamaño del mapa de bits (tamaño). - Apuntador para indicar la ventana en que se encuentra asociado requiere de una estructura de datos, una lista ligada (apuntador). - Estado de selección del mapa de bits si es seleccionado aparecerá con los colores invertidos (selección). - Estado de actividad para el mapa de bits, si puede o no ser usado (actividad). - Almacenamiento del mapa de bits (mapabits). - Función asociada al mapa de bits (función).
*Texto (Clase base)	- Coordenadas para colocar el texto (x1,y1). - Cadena de texto fija (texto).
Objeto texto	- Coordenadas para colocar el texto (x1,y1). - Cadena de texto fija (texto). - Estructura de datos para indicar la ventana en que se encuentra asociado, requiere de una lista ligada (apuntador). - Estado de actividad para el objeto texto, si puede o no ser usado (actividad).
Campo texto	- Coordenadas del recuadro (x1,y1,x2,y2). - Coordenadas para colocar el texto (x3,y3). - Cadena de texto (texto). - Longitud de la cadena de texto. - Estructura de datos para indicar la ventana en que se encuentra asociado el campo texto, requiere de una lista ligada (apuntador). - Estado de actividad para el objeto texto, si puede o no ser usado (actividad).

Objeto	Datos
Campo de edición de texto	- Coordenadas del recuadro (x1,y1,x2,y2) - Coordenadas para colocar el texto (x3,y3). - Cadena de texto (texto). - Longitud de la cadena de texto. - Posición del cursor para poder editar el campo de texto (cursor). - Función asociada que permita la edición del texto si se ha seleccionado este campo (función). - Estructura de datos para indicar la ventana en que se encuentra asociado, requiere de una lista ligada (apuntador). - Estado de actividad para el objeto texto, si puede o no ser usado (actividad).
Cuadros de selección	- Coordenadas del recuadro (x1,y1,x2,y2) - Coordenadas para colocar el texto (x3,y3). - Cadena de texto (texto). - Longitud de la cadena de texto (longitud). - Estado de selección del cuadro, si es seleccionado aparecerá con una (X) en el interior del recuadro situado a un lado del texto (selección). - Estructura de datos para indicar la ventana en que se encuentra asociado, requiere de una lista ligada (apuntador). - Estado de actividad para el objeto texto, si puede o no ser usado (actividad).
Botón	- Coordenadas del recuadro (x1,y1,x2,y2). - Contorno del recuadro (x3,y3,x4,y4). - Coordenadas para colocar el texto (x5,y5). - Cadena de texto (texto). - Longitud de la cadena de texto. - Estructura de datos para indicar la ventana en que se encuentra asociado, requiere de una lista ligada (apuntador). - Estado de actividad para el objeto texto, si puede o no ser usado (actividad).
Lista	- Coordenadas del recuadro que contiene la lista (x1,y1,x2,y2). - Dimensiones de la lista (largo y ancho). - Arreglo para almacenar cadenas de texto (arreglo_texto). - Contador del número de cadenas de texto (contador). - Coordenadas para colocar el texto (x3,y3). - Estructura de datos para indicar la ventana en que se encuentra asociado, requiere de una lista ligada (apuntador).

La fig. 4.2 muestra las clases: gráficos, texto, recuadro y ratón. La clase gráfico y ratón son clases simples que no heredarán sus propiedades a ninguna otra clase (de acuerdo al análisis anterior). Las clases texto y recuadro serán clases base que heredarán sus propiedades comunes a otras clases (herencia).

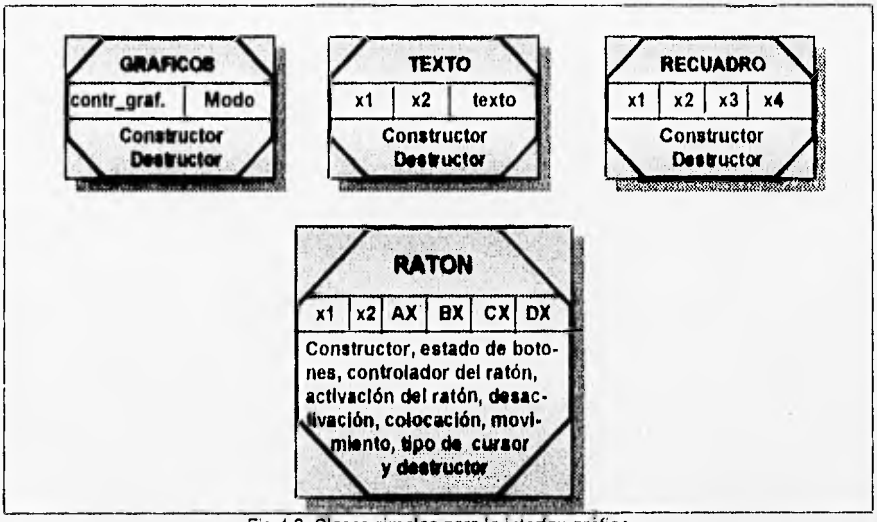
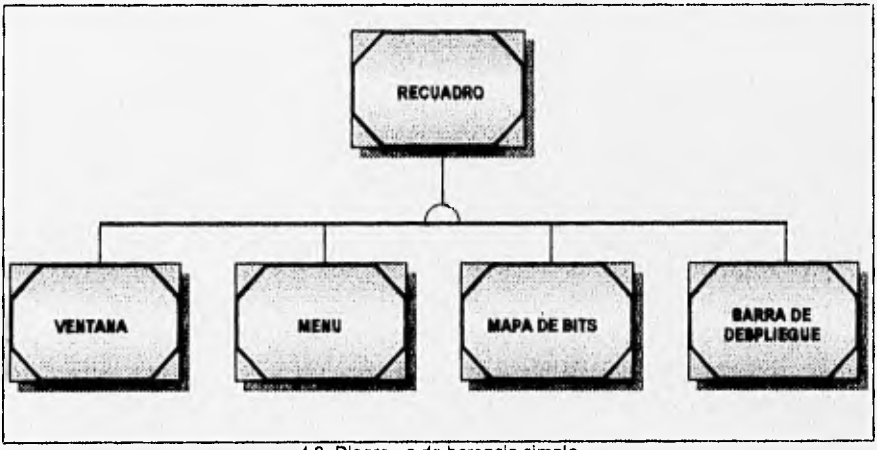
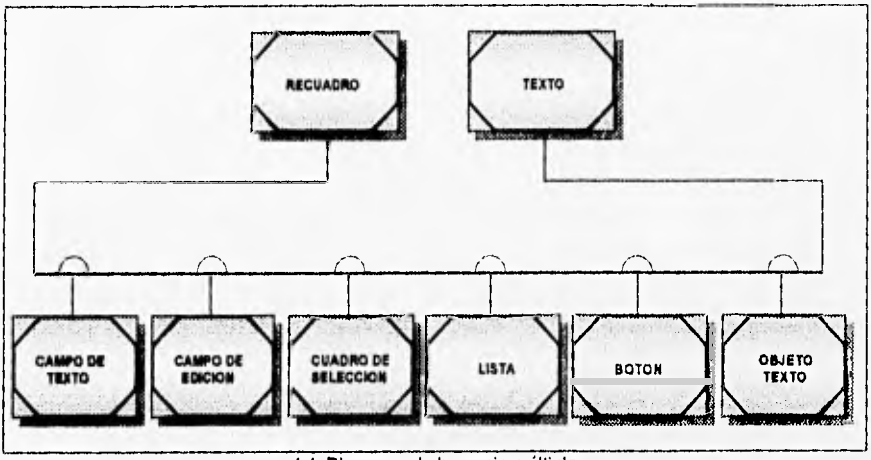


Fig.4.2. Clases simples para la interfaz gráfica

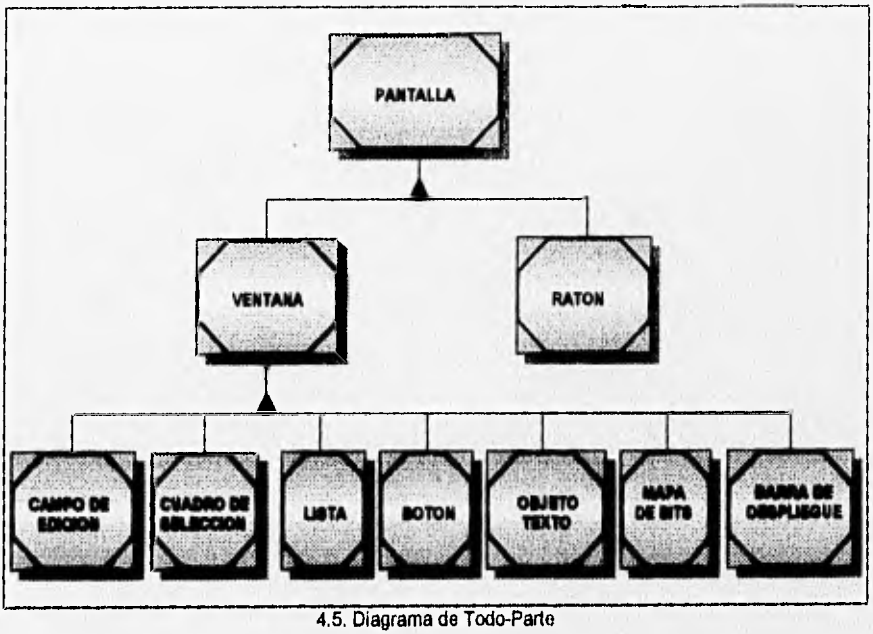
4.3. DIAGRAMAS DE HERENCIA SIMPLE, HERENCIA MULTIPLE Y DE TODO-PARTE



4.3. Diagrama de herencia simple



4.4. Diagrama de herencia múltiple



4.5. Diagrama de Todo-Parte

CAPITULO 5. IMPLEMENTACION EN LENGUAJE C++

5.1. DEFINICION DE LAS CLASES

5.1.1. Definición de la clase *Graficos*

La clase *Graficos* permite la creación de objetos que pueden inicializar y desactivar el modo gráfico de los adaptadores de vídeo EGA (640 x 350 pixeles), VGA (640 x 480 pixeles) y Hércules (720 x 348 pixeles); a continuación se muestra el código para definir esta clase:

```
class Graficos {
// Variables utilizadas por la clase Graficos
protected:
int d,m;  // almacenan el modo y controlador gráficos
public:
//  Funciones miembro de la clase
Graficos(void);
int IniciaGraficos(void);
~Graficos(void);
};
```

5.1.1.1. Funciones miembro de la clase *Graficos*

El constructor de la clase *Graficos*, detecta el modo y el controlador gráfico.

```
Graficos::Graficos(void) {
        detectgraph(&d,&m);
}
```

La función *IniciaGraficos*, inicializa el modo gráfico y coloca el cursor del ratón dentro de las coordenadas válidas para el adaptador de vídeo detectado.

```
int Graficos::IniciaGraficos(void)
{
        union REGS r;
        Raton R(r);
        if(d<0) return(0);
        Initgraph(&d,&m,"");
        if(graphresult() < 0) return(0);
        setcolor(getmaxcolor());
        Modografico=d;
        If(Modografico==VGA) Modografico=MCGA;
        DoTable(Modografico);
        R.Coloca_Raton(2,2,SCREENWIDE-8,SCREENDEEP-8,Modografico);
        return(1);
}
```

Destructor de la clase *Graficos*, apaga el modo gráfico

```
Graficos::~Graficos(void) {
        closegraph();
        if(Modografico==HERCMONO) Reg_Tarj_Herc();
        while(kbhit()) getch();
}
```

5.1.1.2. Funciones utilizadas por la clase *Graficos*

La función externa *DoTable(int x)*, es utilizada por la clase *Graficos* y forma parte de la librería de funciones en lenguaje ensamblador (FUNCS_ASM.OBJ). Esta función, dependiendo del valor que le entrega *initgraph*, crea una tabla de apuntadores y los indexa dependiendo de la línea de interés del área de vídeo. Esta función crea todos los posibles apuntadores para una tarjeta en particular; de esta manera, es más rápido el acceso a la información del área de vídeo.

La función *Reg_Tarj_Herc()* , restaura el modo texto de la tarjeta gráfica Hércules.

La función *R.Coloca_Raton*, coloca el cursor del ratón dentro de las coordenadas válidas de la tarjeta de vídeo.

Las constantes SCREENWIDE y SCREENDEEP; son constantes externas que se encuentran en el archivo FUNCS_ASM.OBJ (SCREENWIDE es el ancho de la pantalla en pixeles y SCREENDEEP es el largo de la pantalla en pixeles.)

5.1.2. Definición de la clase *Raton*

La clase *Raton* que se muestra a continuación, permite crear objetos que activen el control del ratón y su uso en la interfaz gráfica como dispositivo de posicionamiento en pantalla.

```
class Raton {
// Variables utilizadas por la clase Raton
protected:
        union REGS r;
        int mx,my,x,y,n; // Las coordenadas para colocar
 // el cursor del ratón
// Funciones miembro de la clase
public:
Raton (union REGS r1);
void Coloca_Raton(int mx1,int my1,int x1,int y1,int n1);
int Det_Contr_Raton(void);
void Activa_Raton(void);
void DesactivaRaton(void);
void MueveRaton(LOCALIZA *p1);
int BotonesRaton(void);
int Det_Fuera_Raton(LOCALIZA *p1);
void Posicion_Raton(LOCALIZA *p1);
void CursorEspera(void);
void CursorFlecha(void);
};
```

5.1.2.1. Funciones miembro de la clase *Raton*

El constructor de la clase *Raton*, inicializa la variable de tipo REGS usada por otras funciones miembro.

```
Raton::Raton(union REGS r1)
{
        r=r1;
}
```

Para hacer uso del ratón, se requiere de un programa que nos permita la interacción con el dispositivo físico. Cuando se carga el archivo controlador del ratón, este archivo maneja una interrupción de *software*. La comunicación con el controlador del ratón, se lleva a cabo por medio de una llamada a las funciones de la interrupción 33h de DOS y colocando los valores apropiados en los registros AX, BX, CX y DX.

La función miembro *Coloca_Raton*, coloca el ratón dentro de las coordenadas válidas del modo y controlador gráfico, por medio de las funciones 7 y 8 de la interrupción 33h de DOS. Los argumentos *mx*, *my:* son las coordenadas en la parte superior izquierda del rango del ratón; *x, y:* son las coordenadas en la parte inferior derecha (*mx, my, x, y*: son los límites en los cuales puede moverse el ratón en la pantalla.) El argumento *n* corresponde al modo gráfico detectado. A continuación se ilustra el código para la función *Coloca_Raton*.

```
void Raton::Coloca_Raton(int mx1,int my1,int x1,int y1,int n1)
{
        char *p;

        mx=mx1; my=my1;
        x=x1; y=y1; n=n1;
        // Verifica si esta trabajando con una tarjeta Hércules
        if(n==HERCMONO) {
                p=(char *)MK_FP(0x0040,0x0049);
                Tarj_Herc=*p;
                *p=6;
                r.x.ax=0;
                int86(0x33,&r,&r);
        }
        r.x.ax=0x0007;
        r.x.cx=mx;
        r.x.dx=x;
        int86(0x33,&r,&r);
        r.x.ax=0x0008;
        r.x.cx=my;
        r.x.dx=y;
        int86(0x33,&r,&r);
}
```

La función miembro *Det_Contr_Raton*, detecta e inicializa la comunicación con el controlador del ratón, utilizando la función 0, en el registro AX, de la interrupción 33h de DOS.

```
inl Raton::Det_Contr_Raton(void)
{
        r.x.ax=0;
        int86(0x33,&r,&r);
        return(r.x.ax);
}
```

La función miembro *Activa_Raton*, activa y muestra el cursor del ratón utilizando la función 1, en el registro AX, de la interrupción 33h de DOS.

```
void Raton::Activa_Raton(void)
{
        r.x.ax=1;
        int86(0x33,&r,&r);
}
```

La función miembro *DesactivaRaton*, oculta y desactiva el cursor del ratón utilizando la función 1, en el registro *AX*, de la interrupción 33h de *DOS*. El cursor del ratón siempre debe de estar oculto cuando se escribe algo en pantalla, de lo contrario, si se abre una ventana o se captura un fragmento de imagen, el cursor afectará al objeto que se coloque en ese momento en la pantalla.

```
void Raton::DesactivaRaton(void)
{
        r.x.ax=2;
        int86(0x33,&r,&r);
}
```

MueveRaton es una función miembro de clase *Raton*, que mueve el cursor del ratón a otras coordenadas en pantalla que son indicadas por la variable LOCALIZA ***p1**, esto lo realiza sin necesidad de mover el ratón físicamente. Es utilizada la función 4, en el registro AX, de la interrupción 33h de DOS.

```
void Raton::MueveRaton(LOCALIZA *p1)
{
        r.x.ax=4;
        r.x.cx=p1->x;
        r.x.dx=p1->y;
        int86(0x33,&r,&r);
}
```

MueveRaton es la función miembro de la clase *Raton*, que regresa el estado de activación de los botones del dispositivo; utiliza la función 3, en el registro AX, de la interrupción 33h de DOS. Coloca también el estado de los botones del ratón en el registro BX (sólo dos bits, uno para el botón izquierdo y el otro para el derecho). Esta función regresa un valor verdadero, solo si el botón izquierdo ha sido presionado.

```
int Raton::BotonesRaton(void)
{
        r.x.ax=3;
        int86(0x33,&r,&r);
        return(r.x.bx & 0x03);
}
```

La función miembro *Det_Fuera_Raton*, regresa el estado de los botones del ratón y simula que el ratón se encuentra en unas coordenadas no válidas (-1,-1). Usa la variable LOCALIZA *RatonAfuera*, que se encuentra en el archivo VARS_IGU.H (el cual contiene todas las variables para la interfaz gráfica), estos valores son pasados a los registros CX y DX; utilizando la función 3, en el registro AX, de la interrupción 33h de DOS.

```
// Detecta la posición del ratón, si éste no apunta a nada
int Raton::Det_Fuera_Raton(LOCALIZA *p1)
{       if(RatonAfuera.x != -1 && RatonAfuera.y != -1) {
                memcpy((char *)p1,(char *)&RatonAfuera,sizeof(LOCALIZA));
                RatonAfuera.x=-1;
                RatonAfuera.y=-1;
                return(1);  }
        r.x.ax=3;
        int86(0x33,&r,&r);
        p1->x=r.x.cx;
        p1->y=r.x.dx;
        return(r.x.bx & 0x03);
}
```

Posicion_Raton es la función miembro que indica la posición del ratón en la pantalla; utiliza la función 3, en el registro AX, de la interrupción 33h de DOS. El controlador del ratón regresa la posición horizontal del ratón en el registro CX y la posición vertical en el registro DX.

```
void Raton::Posicion_Raton(LOCALIZA *p1)
{
        r.x.ax=3;
        int86(0x33,&r,&r);
        p1->x=r.x.cx;
        p1->y=r.x.dx;
}
```

El cursor del ratón esta definido por tres elementos: la máscara de la pantalla, la máscara del cursor y el punto de activación del cursor; que es el punto clave del cursor para señalar un objeto dentro de la pantalla. Ver figuras 5.1 y 5.2.

Figura 5.1. Elementos del cursor del ratón

Los valores constantes para los tipos de cursores del ratón, se definen en el archivo VARS_IGU; por ejemplo, para el cursor de flecha se tiene el siguiente código:

```
char flecha[] = {
// Los primeros 32 bytes definen la máscara del cursor
        0xFF,0x3F,0xFF,0x1F,0xFF,0x0F,0xFF,0x07,
        0xFF,0x03,0xFF,0x01,0xFF,0x00,0x7F,0x00,
        0x3F,0x00,0x1F,0x00,0xFF,0x01,0xFF,0x10,
        0xFF,0x30,0x7F,0xF8,0x7F,0xF8,0x7F,0xFC,

// Los siguientes 32 bytes definen la máscara de la pantalla ( el fondo que contraste con la pantalla )
        0x00,0x00,0x00,0x40,0x00,0x60,0x00,0x70,
        0x00,0x78,0x00,0x7C,0x00,0x7E,0x00,0x7F,
        0x80,0x7F,0x00,0x7C,0x00,0x6C,0x00,0x46,
        0x00,0x06,0x00,0x03,0x00,0x03,0x00,0x00 };
```

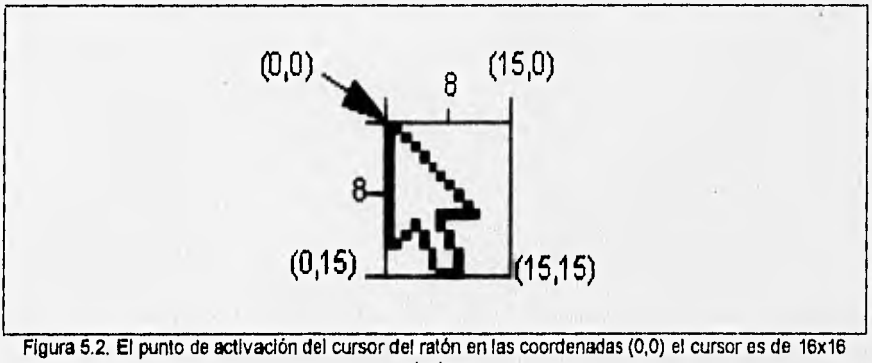


Figura 5.2. El punto de activación del cursor del ratón en las coordenadas (0,0) el cursor es de 16x16 pixeles.

El código de la siguiente función miembro, permite cambiar la apariencia del cursor del ratón; utiliza la función 9, en el registro AX, de la interrupción 33h de DOS. Coloca en los registros BX y CX, las coordenadas de punto de activación del cursor, las cuales permiten hacer una selección de un objeto en la pantalla. *CursorRaton* contiene el tipo de cursor para el ratón y las variables *x, y;* las coordenadas del punto de activación del cursor del ratón.

```
void Raton::Cursor(char *CursorRaton,int x, int y)
{
        struct SREGS sr;

        r.x.ax=0x0009;
        r.x.bx=x;
        r.x.cx=y;
        r.x.dx=FP_OFF(CursorRaton);
        sr.es=FP_SEG(CursorRaton);
        int86x(0x33,&r,&r,&sr);
}
```

En esta interfaz se manejarán tres tipos de cursores: el de reloj de espera (indica al usuario que se esta realizando alguna actividad), el de figura de mano (sirve para indicar que pueden ser seleccionar algunas opciones de algún menú) y el de flecha que es el indicador normal del ratón, ver fig. 5.3.

Fig. 5.3. Tipos de cursores para la interfaz

5.1.3. Definición de la clase *Recuadro*

La clase *Recuadro*, es utilizada como la clase base que será heredada a las clases: *Ventana, Menús, Boton, Lista, Texto, BarrasDesp, MapaBits*. Esta clase tiene como función principal, el definir el área rectangular sobre la pantalla utilizando una variable *struct* RECUADRO (definida en VARS_IGU.H). En este caso, la clase *Recuadro* se maneja de una forma polimórfica, dependiendo de la naturaleza de la clase a la que le son heredadas sus propiedades.

```
// struct para definir las coordenadas de un recuadro
typedef struct {
        int izq,arriba,der,abajo;
        }RECUADRO;
```

Definición de la clase *Recuadro*:

```
class Recuadro{
// Variables utilizadas por la clase Recuadro
protected:
RECUADRO *r;
int izq,arriba,der,abajo,l;
char *mb;
// Funciones miembro de la clase
public:
Recuadro(RECUADRO *r1,int izq1, int arriba1);
Recuadro(RECUADRO *r1,int izq1, int arriba1, int l1);
Recuadro(RECUADRO *r1,int izq1, int arriba1, char *mb1);
Recuadro(RECUADRO *r1,int izq1, int arriba1, int der1, int abajo1);
~Recuadro(){}; // Destructor de la clase Recuadro
};
```

5.1.3.1. Funciones miembro de la clase *Recuadro*

Constructores de la clase *Recuadro*:

1) Constructor *Recuadro* para la clase *BarrasDesp*; en este caso, sólo se necesita definir las coordenadas superiores del recuadro para su variable *struct* RECUADRO (RECUADRO **r1,int izq1, int arriba1)*; las coordenadas inferiores son calculadas posteriormente, en base a otros parámetros de la clase heredada.

```
Recuadro::Recuadro(RECUADRO *r1,int izq1,int arriba1)
{
  r=r1;
  r->izq=izq1;
  r->arriba=arriba1;
};
```

2) Constructor *Recuadro* para las clases: *CampoTexto y CampoEdicion*, se definen las coordenadas superiores del recuadro, para su variable struct RECUADRO (RECUADRO **r1,int izq1, int arriba1)*, así como, la longitud de la cadena de texto (*int l*), que permite calcular las dimensiones inferiores del recuadro.

```
Recuadro::Recuadro(RECUADRO *r1,int izq1, int arriba1,int l1)
{
  r=r1;
  l=l1;
  r->izq=izq1-3;
  r->arriba=arriba1-3;
  r->der=r->izq+(8*l)+2;
  r->abajo=r->arriba+PROFFONT+3;
}
```

3) Constructor *Recuadro* para la clase *MapaBits*; se definen las coordenadas superiores del recuadro para su variable *struct* RECUADRO (RECUADRO **r1,int izq1, int arriba1)*, así como, se calcula el tamaño que ocupa el mapa de bits (*char *mb1*), para definir las dimensiones inferiores del recuadro.

```
Recuadro::Recuadro(RECUADRO *r1,int izq1,int arriba1,char *mb1)
{
  r=r1;

// Define las coordenadas para la struct RECUADRO del objeto Mapabits
        mb=mb1;
        r->izq=izq1;
        r->arriba=arriba1;
        r->der=r->izq+AnchoImagen(mb);
        r->abajo=r->arriba+LargoImagen(mb);
};
```

4) Constructor *Recuadro*, define las coordenadas para el RECUADRO, que será usado en las clases *Ventana* y *Botones*; estas coordenadas formarán el área del recuadro, de su variable *struct* RECUADRO *(RECUADRO *r1,int izq1, int arriba1, int der1, int abajo1).*

```
Recuadro::Recuadro(RECUADRO *r1,int izq1, int arriba1, int der1, int abajo1)

{
  r=r1;
  r->izq=izq1;
  r->arriba=arriba1;
  r->der=der1;
  r->abajo=abajo1;
}
```

5.1.4. Definición de la clase *Ventana*

La clase *Ventana*, hereda las propiedades de la clase *Recuadro* (una área rectangular.) Entre las funciones de la clase *Ventana* tenemos que:

1) Definen el área que ocupa la ventana.

2) Colocan un *buffer* lo suficientemente grande para contener el área de la ventana, junto con los objetos que puedan estar colocados en su Interior (botones, iconos, texto, etc.)

3) Copian el contenido al área del *buffer*.

4) Dibujan una área blanca del tamaño de la ventana, así como, un cuadro negro alrededor de ésta.

5) Cierran la ventana, copiando la imagen desde el *buffer* a la pantalla y por último hacen la liberación del *buffer*.

```
#define EnVentana      0x0001 //Máscara para localizar el objeto ventana en la pantalla

// struct para definir una ventana
typedef struct {
        APUNTADOR apunta;   // struct de tipo APUNTADOR para la ventana
        RECUADRO rec;       // struct de tipo RECUADRO para la ventana
        unsigned char *reg; // Contiene el tamaño de la ventana
        }VENTANA;
```

```
class Ventana:public Recuadro {
// Variables utilizadas por la clase Ventana
protected:
VENTANA *w;
// Funciones miembro de la clase
public:
Ventana(VENTANA *w1,RECUADRO *r1,int izq1, int arriba1, int der1, int abajo1);
int AbreVentana(void);
void CierraVentana(void);
~Ventana(){};
};
```

5.1.4.1. Funciones miembro de la clase *Ventana*

El constructor de la clase *Ventana*, hereda las propiedades de la clase *Recuadro* e inicializa a la variable *struct* VENTANA *w* , coloca además los parámetros para el apuntador dentro de la lista ligada *(w->apunta.sig* y *w->apunta.tipo)*. SOMBRAVENT es una constante que se encuentra en el archivo VARS_IGU, y su valor es igual a 4.

```
Ventana::Ventana(VENTANA *w1,RECUADRO *r1,int izq1, int arriba1, int der1, int abajo1):
   Recuadro(r1,izq1,arriba1,der1,abajo1)
{
        w=w1;

// Inicializa la variable de tipo VENTANA
        w->rec.izq = r->izq-1;
        w->rec.arriba = r->arriba-1;
        w->rec.der = r->der+SOMBRAVENT+1;
        w->rec.abajo = r->abajo+SOMBRAVENT+1;

// Inicializa los parámetros del apuntador dentro de la lista ligada
        w->apunta.sig=NULL;
        w->apunta.tipo=EnVentana;
}
```

La función miembro *AbreVentana*, devuelve un valor verdadero si logró dibujar la ventana exitosamente, la función *Tam_Memoria* obtiene el tamaño que ocupará la ventana, en base al área de recuadro. La función *Obtiene_Imagen*, se usa para reemplazar a la función *getimage* de la librería gráfica de Borlandc C++, ya que ofrece mejores resultados al hacer la apertura de la ventana, ésta función se obtuvo de una librería gráfica realizada en Turbo C y que fue adaptada para ser usada en este programa.

```
int Ventana::AbreVentana(void)
{
        int tam;
        union REGS m1;
        Raton R(m1);

        if((tam=Tam_Memoria(&w->rec))== -1) return(0);
```

```
        // Obtiene el tamaño para la ventana y después la abre
        if((w->reg=(unsigned char *)malloc(tam)) != NULL) {
                R.DesactivaRaton();
                Obtiene_Imagen(w->rec.izq,w->rec.arriba,w->rec.der,w->rec.abajo,w->reg);
                setwritemode(COPY_PUT);
                setfillstyle(SOLID_FILL,getmaxcolor());
                setlinestyle(SOLID_LINE,0,NORM_WIDTH);
                setcolor(BLACK);
                bar(r->izq-1,r->arriba-1,r->der+1,r->abajo+1);
                rectangle(r->izq-1,r->arriba-1,r->der+1,r->abajo+1);
                setlinestyle(SOLID_LINE,0,THICK_WIDTH);
                line(r->der+2,r->arriba+SOMBRAVENT,r->der+2,r->abajo+3);
                line(r->izq+SOMBRAVENT,r->abajo+2,r->der,r->abajo+2);
                R.Activa_Raton();
                return(1);
        } else return(0);
}
```

La función miembro *CierraVentana*, cierra un objeto de tipo ventana, siempre y cuando contenga un valor diferente de NULL, es decir, que exista la ventana. Se usa la función *Coloca_Imagen* para reemplazar a la función *putimage* de la librería gráfica de Borland C++, debido a que ofrece mejores resultados al sobreescribir la ventana, *Coloca_Imagen* se obtuvo de una librería gráfica realizada en Turbo C. Por último, con *free(w->reg)* se libera al *buffer* que contiene a la ventana y *Activa_Raton* coloca nuevamente el cursor del ratón en la pantalla.

```
void Ventana::CierraVentana(void)
{
        union REGS m1;
        Raton R(m1);

        R.DesactivaRaton();
        // Verifica si hay una ventana abierta, si es así, la cierra
        if(w->reg != NULL) {
                Coloca_Imagen(w->rec.izq,w->rec.arriba,w->reg);
                free(w->reg);
                R.Activa_Raton();
        }
}
```

5.1.5. Definición de la clase *Menus*

La clase *Menus* hereda las propiedades de la clase *Recuadro*. La clase *Menus* permite crear los elementos necesarios para manejar menús en una aplicación, como son: la barra de menús, los títulos de los menús, los índices de cada menú colgante y el manejo de los mismos. En esta clase son utilizadas las variables *struct* LOCALIZA, MENU, VENTANA; así como, la máscara para la barra de menús (*EnBarraMenu)*, que indica si se encuentra posicionado el cursor en la barra de menús; estas variables se encuentran en el archivo VARS_IGU.H

```
#define EnBarraMenu    0x0002  // Máscara para localizar la barra de menús
#define INDMAXMENUS        16  // Máximo número de índices para un menú
#define TAMLINMENUS    18  // Máxima longitud para un índice de un menú
#define PROFLINMENUS   15  // Largo para la línea de menú
#define CONTMENUS      6  // Número de menús
#define TAMTITMENUS    12  // Máxima longitud para el título de un menú
#define PROFBARRMENU       13  // El largo para la barra de menús

// struct para localizar coordenadas en la pantalla
typedef struct {
        int x,y;  // Define coordenadas
        }LOCALIZA;

// struct para definir un índice de un menú
typedef struct {
        char nombre[TAMLINMENUS+1]; // Contiene el nombre de los índices de un menú
        int (*func)(); // La función es ejecutada, si se selecciona este índice
        }INDMENU;

// struct que contabiliza y contiene el título de un menú
typedef struct {
        int cont;          // Lleva la cuenta de los menús
        char titulo[TAMTITMENUS+1];  // Contiene los títulos para los menús
        }TITMENU;

// struct para definir un menú completo
// incluye título e índices
typedef struct {
        int cont;   // Lleva la cuenta de los índices de los menús
        char titulo[TAMTITMENUS+1];  // Contiene los títulos para los menús
        INDMENU indice[INDMAXMENUS]; // Almacena los índices de los menús
        }MENU;

// struct para definir una ventana
typedef struct {
        APUNTADOR apunta;   // struct de tipo APUNTADOR para la ventana
        RECUADRO rec;       // struct de tipo RECUADRO para la ventana
        unsigned char *reg; // Contiene el tamaño de la ventana
        }VENTANA;

MENU *ArregloMenu[CONTMENUS];      // almacena los menús usados en la aplicación
int IndiceMenu=0;
```

Definición de la clase *Menus*

```
class Menus:public Recuadro{
// Variables utilizadas por la clase Menus, definidas como protected
// que pueden ser heredadas a otra clase
protected:
int cont,i, m,n;
VENTANA w; // Variable  VENTANA  que sirve para almacenar los índices del menú
LOCALIZA pn,*p; // Variable para localizar un índice de menú
RECUADRO rm,lr,*r; // Variables recuadro para el manejo de los índices de menú
//  Funciones miembro de la clase
public:
Menus(RECUADRO *r1,int x1,int y1,int x2,int y2); // El constructor de la clase menús
int ColocMenus(MENU *m1);              // Coloca un menú en el arreglo de menús
void DibBarraMenus(void);              // Dibuja la barra de menús
int AgregaMenus(LOCALIZA *p1);         // Agrega los menús a la barra de menús
};
```

5.1.5.1. Funciones miembro de la clase *Menus*

A continuación se muestra el código para el constructor de la clase *Menus*, en éste se inicializan los arreglos que contendrán los diferentes menús colgantes.

```
//Inicializa constructor para la clase Menus
Menus::Menus(RECUADRO *r1,int x1,int y1,int x2,int y2)
:Recuadro(r1,x1,y1,x2,y2)
{
        int inc;
        r=r1;
        // Inicializa a NULL el arreglo de menús
        for(inc=0;inc<CONTMENUS;++inc)
            ArregloMenu[inc]=NULL;

        }
```

La función *ColocMenus*, contabiliza los menús que se van creando de acuerdo a las necesidades del usuario (hasta un máximo de 6 menús.)

```
int Menus::ColocMenus(MENU *m1)
{
        if(IndiceMenu < CONTMENUS) {
                ArregloMenu[IndiceMenu++]=m1;
                return(1);
        } else return(0);
}
```

La función *DibBarraMenus*, coloca la barra de menús en la parte superior de la pantalla, con sus títulos respectivos:

```
void Menus::DibBarraMenus(void)
{
        WhiteRule(PROFBARRMENU,0);     // Dibuja la barra blanca donde se colocan lo títulos
        for(cont=0;cont<IndiceMenu;++cont)  // Coloca los títulos en la barra de menús
        DrawString(cont*(8*(TAMTITMENUS)),2,ArregloMenu[cont]->titulo,ACTIVO); // Los títulos
}
```

La función *AgregaMenus*, permite las selección de un título en la barra de menús, abre el menú colgante (la ventana) debajo del título, para poder seleccionar alguno de los índices y actualizar el estado de éstos (activo o inactivo.)

```
int Menus::AgregaMenus(LOCALIZA *p1)
{
        union REGS m1;
        Raton R(m1);
        p=p1;
 // Localiza el menú dentro de la barra de menús y define las coordenadas
 // para abrir la ventana para dicho menú
        m=p->x / (TAMTITMENUS<<3);
        if(m >= IndiceMenu) return(0);
        rm.izq=(m*(TAMTITMENUS<<3))+8;
        rm.arriba=r->abajo;
        rm.der=rm.izq+(TAMLINMENUS<<3);
        rm.abajo=rm.arriba+(ArregloMenu[m]->cont*PROFLINMENUS);
        Ventana V(&w,&rm,rm.izq,rm.arriba,rm.der,rm.abajo); // Objeto Ventana para los índices

        // Coloca un menú colgante debajo del título respectivo, con los índices para el menú
        if(V.AbreVentana()) {
                for(l=0;l<ArregloMenu[m]->cont;++l) {
                        if(ArregloMenu[m]->indice[l].nombre[0]==' ')
                           DrawString(rm.izq+9,rm.arriba+(l*PROFLINMENUS)
                                +1,ArregloMenu[m]->indice[l].nombre+1,ACTIVO);
                        else if(ArregloMenu[m]->indice[l].nombre[0]==251)
                           DrawString(rm.izq,rm.arriba+(l*PROFLINMENUS)
                                +1,ArregloMenu[m]->indice[l].nombre,ACTIVO);
                        else
                           DrawString(rm.izq+9,rm.arriba+(l*PROFLINMENUS)
                                +1,ArregloMenu[m]->indice[l].nombre+1,INACTIVO);
                }
                i=-1;
                // Localiza si el ratón se encuentra en la barra de menús
                // y si esta seleccionando uno de sus índices
                while(R.Det_Fuera_Raton(&pn)) {
                        if(LocRecuadro(&pn,&rm)) {
                                n=(pn.y-rm.arriba)/PROFLINMENUS;
                                if(n != i) {
                                        if(l != -1) {
                                        // Si para el índice del menú, su primer caracter es un espacio en
                                        // blanco, coloca en color negro este índice (estará activo.)
                                        Recuadro REC1(&lr,rm.izq,rm.arriba+(l*PROFLINMENUS),
                                        rm.der,rm.arriba+(l*PROFLINMENUS)+PROFLINMENUS);
                                        if(ArregloMenu[m]->indice[l].nombre[0]==' ')
                                         InviertRec(&lr);
                                        }
```

```
                        Recuadro  REC2(&lr,rm.izq,rm.arriba +(n*PROFLINMENUS),
                        rm.der,rm.arriba+(n*PROFLINMENUS)+PROFLINMENUS);
                                if(ArregloMenu[m]->indice[n].nombre[0]==' ') InviertRec(&lr);
                                i=n;
                        }
                }
        }
        // Cierra el menú abierto
        V.CierraVentana();
        if(LocRecuadro(&pn,&rm) && ArregloMenu[m]->indice[n].nombre[0]==' ') {
                n=(pn.y-rm.arriba)/PROFLINMENUS;
                (ArregloMenu[m]->indice[n].func)();
        }
        return(1);
    } else return(0);
}
```

5.1.5.2. Funciones asociadas a la clase *Menus*

Funciones externas que se encuentran en la librería FUNCASM, que son usadas en la clase *Menus:*

ClearScreen(void) , limpia la pantalla y coloca el escritorio en color gris.

WhiteRule(int x,int y), coloca un rectángulo blanco en las coordenadas x, y; para dibujar la barra de menús.

DoTable(int x), selecciona de una tabla, el tipo de tarjeta de vídeo que se está utilizando.

*DrawString(int w,int x,char *y,int z),* coloca una cadena de texto en color negro o gris.

*int LocRecuadro(*LOCALIZA *p,RECUADRO *r): esta función regresa un valor de 1, si el apuntador se encuentra dentro del área de una variable struct RECUADRO.

5.1.6. Definición de la clase *BarrasDeDespl*

La clase *BarrasDeDespl* hereda las propiedades de la clase *Recuadro,* permite crear objetos de tipo barras de despliegue, tanto verticales, como horizontales. Utiliza la variable *struct* BARRASDESP, para almacenar los datos necesarios para su definición; así como, las máscaras para indicar que elemento de las barras esta siendo seleccionado.

```
// Máscaras para los objetos relacionados con las barras de despliegue
#define EnDespVert      0x0080
#define EnDespHor       0x0100
#define FlechaSup       0x0200
#define FlechaInf       0x0400
#define HaciaArriba     0x0800
#define HaciaAbajo      0x1000
#define EnArea                  0x2000

char Rell_Barra_Desp[]="\063\314\063\314\063\314\063\314"; // Coloca el relleno  para la barra de
                                                          // despliegue
```

```
typedef struct {
        APUNTADOR apunta;  // Variable para colocar el objeto dentro de la lista ligada
        int x,y;           // Coordenadas para colocar la barra
        int tam;           // La longitud de la barra de despliegue, largo (vertical) o ancho
(horizontal)
        RECUADRO rec,flecharr,flechab,area,despl; // Cinco elementos de la barra de despliegue
        unsigned int activo;    // Estado de selección de la barra de despliegue
        int min,max,posicion;
        }BARRASDESP;
```

Definición de la clase *BarrasDeDespl*

```
class BarrasDeDespl :public Recuadro {
// Variable utilizada por la clase BarrasDeDespl
protected:
        BARRASDESP *bd;
//  Funciones miembro de la clase
public:
        BarrasDeDespl(int largo1,int min1,int max1,int posicion1,int activo1,BARRASDESP
*bs1,VENTANA *w1,RECUADRO *r1,int x1,int y1);
        void Dib_Bar_Vert(void);
        void Dib_Bar_Hor(void);
        };
```

5.1.6.1. Funciones miembro de la clase *BarrasDeDespl*

Constructor de la clase *BarrasDeDespl* inicializa los datos para las barras de despliegue, horizontales y verticales; esta información es proporcionada por la clase *Recuadro*. Además, son colocados: el objeto dentro de la lista ligada asociada con un objeto de tipo ventana, las dimensiones para las barras (largo y ancho) y el estado de la barra (activa o inactiva).

```
BarrasDeDespl::BarrasDeDespl(int largo1,int min1,int max1,int posicion1,int activo1,BARRASDESP
*bs1,VENTANA *w1,RECUADRO *r1,int x1,int y1):Recuadro(r1,x1,y1)
{
        bd=bs1;
        APUNTADOR *oh;
        oh=(APUNTADOR *)w1;          // Asocia la lista ligada para la ventana correspondiente

        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig; // Hace el recorrido por la lista ligada
        oh->sig=(APUNTADOR *)bd;    // Apunta a un objeto de tipo barra de despliegue
        bd->tam=largo1;              // Longitud de la barra de despliegue
        bd->min=min1;                // Dimensiones de la barra de despliegue
        bd->max=max1;
        bd->posicion=posicion1;      // La posición del indicador de la barra
        bd->activo=activo1;          // Estado de actividad de la barra de despliegue


        bd->rec.izq=bd->x=r->izq;       // Coordenadas iniciales x,y; para colocar la barra de despliegue
        bd->rec.arriba=bd->y=r->arriba;
}
```

Dib_Bar_Vert dibuja una barra de despliegue vertical, formada por las dos flechas en los extremos de la barra (inferior y superior), un indicador de desplazamiento de la barra y las zonas de desplazamiento de la barra (recuadros arriba y abajo del indicador de desplazamiento.)

```
void BarrasDeDespl::Dib_Bar_Vert(void)
{
// Declaración de variables "flechas gráficas" para indicar dentro
// de la barra de despliegue vertical hacia donde moverse
        static char flecharriba[]={ 0x0f,0x00,0x0f,0x00,
                0x00,0x00,0x7E,0x7E,0x7D,0xBE,0x7B,0xDE,
                0x77,0xEE,0x6F,0xF6,0x5F,0xFA,0x03,0xC0,
                0x7B,0xDE,0x7B,0xDE,0x7B,0xDE,0x7B,0xDE,
                0x78,0x1E,0x7F,0xFE,0x7F,0xFE,0x00,0x00 };
        static char flechabajo[]={ 0x0f,0x00,0x0f,0x00,
                0x00,0x00,0x7F,0xFE,0x7F,0xFE,0x78,0x1E,
                0x7B,0xDE,0x7B,0xDE,0x7B,0xDE,0x7B,0xDE,
                0x03,0xC0,0x5F,0xFA,0x6F,0xF6,0x77,0xEE,
                0x7B,0xDE,0x7D,0xBE,0x7E,0x7E,0x00,0x00 };
        char *p;  // Apuntador para las flechas gráficas
        union REGS m;  // Define un objeto Raton
        Raton R(m);

        // Se definen las coordenadas para una barra de despliegue vertical
        bd->rec.der=bd->x+15;
        bd->rec.abajo=bd->y+bd->tam;

        // Coloca las dimensiones de los recuadros necesarios para
        // definir los elementos de la barra de despliegue
        Recuadro R1(&bd->flecharr,bd->rec.izq,bd->rec.arriba,bd->rec.der,bd->rec.arriba+15);
        Recuadro R2(&bd->flechab,bd->rec.izq,bd->rec.abajo-15,bd->rec.der,bd->rec.abajo);
        Recuadro R3(&bd->area,bd->rec.izq,bd->rec.arriba+16,bd->rec.der,bd->rec.arriba+32);
        Recuadro R4(&bd->despl,bd->rec.izq,bd->flecharr.abajo,bd->rec.der,bd->flechab.arriba);
        bd->apunta.tipo=EnDespVert;   // Apunta a un objeto de tipo barra de despliegue vertical
        bd->apunta.sig=NULL;          // apunta al siguiente objeto de la lista ligada
        R.DesactivaRaton();  // Desactiva el cursor del ratón para que no interfiera al colocar la barra

        // Dibuja la barra de despliegue vertical
        if(bd->activo==ACTIVO) setfillpattern(Rell_Barra_Desp,getmaxcolor());
        else setfillstyle(SOLID_FILL,getmaxcolor());
        setwritemode(COPY_PUT);
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        setcolor(BLACK);
        bar(bd->rec.izq,bd->rec.arriba,bd->rec.der,bd->rec.abajo);
        rectangle(bd->rec.izq,bd->rec.arriba,bd->rec.der,bd->rec.abajo);

        // Coloca los dos indicadores de dirección de la
        // barra de despliegue vertical, dependiendo del adaptador gráfico que se este utilizando

        if(Det_Modo_Monocr()) putimage(bd->flecharr.izq,bd->flecharr.arriba,flecharriba,COPY_PUT);
        else {
                if((p=Reg_Bmp_EGA(flecharriba)) != NULL) {
                        putimage(bd->flecharr.izq,bd->flecharr.arriba,p,COPY_PUT);
```

```
                free(p);
            } else MensajeError();
        }

        if(Det_Modo_Monocr()) putimage(bd->flechab.izq,bd->flechab.abajo-15,flechabajo,COPY_PUT);
        else {
            if((p=Reg_Bmp_EGA(flechabajo)) != NULL) {
                putimage(bd->flechab.izq,bd->flechab.abajo-15,p,COPY_PUT);
                free(p);
            } else MensajeError();
        }

        Dib_Area_Contr(bd);  // Dibuja el área donde se desplazará el cursor de posicionamiento de la
                             // barra
        R.Activa_Raton();
}
```

La función *Dib_Bar_Hor* dibuja una barra de despliegue horizontal, formada por las dos flechas en los extremos de la barra (izquierda y derecha), un indicador de desplazamiento de la barra y las zonas de desplazamiento de la barra (recuadros a la derecha e izquierda del indicador de desplazamiento.)

```
void BarrasDeDespl::Dib_Bar_Hor(void)
// Declaración de variables "flechas gráficas" para indicar dentro
// de la barra de despliegue horizontal hacia donde moverse
{
        static char flechaizq[]={ 0x0f,0x00,0x0f,0x00,
                0x00,0x00,0x7E,0x7E,0x7D,0x7E,0x7B,0x7E,
                0x77,0x7E,0x6F,0x06,0x5F,0xF6,0x3F,0xF6,
                0x3F,0xF6,0x5F,0xF6,0x6F,0x06,0x77,0x7E,
                0x7B,0x7E,0x7D,0x7E,0x7E,0x7E,0x00,0x00    };

        static char flechader[]={ 0x0f,0x00,0x0f,0x00,
                0x00,0x00,0x7E,0x7E,0x7E,0xBE,0x7E,0xDE,
                0x7E,0xEE,0x60,0xF6,0x6F,0xFA,0x6F,0xFC,
                0x6F,0xFC,0x6F,0xFA,0x60,0xF6,0x7E,0xEE,
                0x7E,0xDE,0x7E,0xBE,0x7E,0x7E,0x00,0x00 };

        char *p; // Apuntador para las flechas gráficas
        union REGS m; // Define un objeto Raton
        Raton R(m);

// Se definen las coordenadas para una barra de despliegue horizontal
        bd->rec.der=bd->x+bd->tam;
        bd->rec.abajo=bd->y+15;

        // Coloca las dimensiones de los recuadros necesarias para
        // definir los elementos de la barra de despliegue horizontal
        Recuadro R1(&bd->flecharr,bd->rec.izq,bd->rec.arriba,bd->rec.izq+15,bd->rec.abajo);
```

```
Recuadro R2(&bd->flechab,bd->rec.der-15,bd->rec.arriba,bd->rec.der,bd->rec.abajo);
Recuadro R3(&bd->area,bd->rec.izq+16,bd->rec.arriba,bd->rec.izq+32,bd->rec.abajo);
Recuadro R4(&bd->despl, bd->flecharr.der,bd->rec.arriba,bd->flechab.izq,bd->rec.abajo);
bd->apunta.tipo=EnDespHor; // Apunta a un objeto barra de despliegue horizontal
bd->apunta.sig=NULL;       // apunta al siguiente objeto de la lista ligada
R.DesactivaRaton();// Desactiva el cursor del ratón para que no interfiera al colocar la barra

// Dibuja la barra de despliegue horizontal
if(bd->activo==ACTIVO) setfillpattern(Relf_Barra_Desp,getmaxcolor());
else setfillstyle(SOLID_FILL,getmaxcolor());
setwritemode(COPY_PUT);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
setcolor(BLACK);
bar(bd->rec.izq,bd->rec.arriba,bd->rec.der,bd->rec.abajo);
rectangle(bd->rec.izq,bd->rec.arriba,bd->rec.der,bd->rec.abajo);

// Coloca los dos indicadores de dirección de la
// barra de despliegue horizontal dependiendo del adaptador gráfico que se este utilizando
if(Det_Modo_Monocr()) putimage(bd->flecharr.izq,bd->flecharr.arriba,flechaizq,COPY_PUT);
else {
        if((p=Reg_Bmp_EGA(flechaizq)) != NULL) {
                putimage(bd->flecharr.izq,bd->flecharr.arriba,p,COPY_PUT);
                free(p);
        } else MensajeError();
}

if(Det_Modo_Monocr()) putimage(bd->flechab.izq,bd->flechab.abajo-15,flechader,COPY_PUT);
else {
        if((p=Reg_Bmp_EGA(flechader)) != NULL) {
                putimage(bd->flechab.izq,bd->flechab.abajo-15,p,COPY_PUT);
                free(p);
        } else MensajeError();
}

Dib_Area_Contr(bd); // Dibuja el área donde se desplazará el cursor de posicionamiento de la
                    // barra
R.Activa_Raton();
}
```

5.1.6.2. Funciones asociadas a la clase *BarrasDeDespl*

*char *Reg_Bmp_EGA(char *version)*: regresa una versión EGA monocromática, de un mapa de bits.

int Det_Modo_Monocr(void): regresa un valor verdadero, si el modo gráfico es monocromático.

*void Dib_Area_Contr(*BARRASDESP **bs1)*: dibuja el área de control de la barra de despliegue, donde se mueve el cuadro indicador de desilizamiento.

5.1.7. Definición de la clase *MapaBits*

La clase *MapaBits* permite colocar un objeto mapa de bits en una ventana (iconos), ya sea para fines decorativos o para que éstos, al ser seleccionados, ejecuten alguna función. Esta clase requiere de la variable *struct* MAPABITS, la cual contiene a su vez variables *struct* APUNTADOR, para manejar su inclusión en la lista ligada y *struct* RECUADRO (que es el área correspondiente al mapa de bits.)

```
#define EnMapaBits    0x0008    // Máscara para indicar que se encuentra el cursor en un mapa de
                                // bits

typedef struct {
        APUNTADOR apunta;     // struct de tipo APUNTADOR para el mapa de bits
        int x,y;               // Coordenadas iniciales del mapa de bits
        RECUADRO rec;         // struct RECUADRO para el mapa de bits
        char sel;              // Almacena el estado de selección del MAPABITS
        unsigned int activo;  // Indica el estado del botón (ACTIVO/INACTIVO)
        char *mapabits;       // Contiene el mapa de bits
        int (*func)();        // Apuntador a una función que se llama cuando
                              // se selecciona un mapa de bits
        }MAPABITS;


class MapaBits: public Recuadro {
// Variable utilizada por la clase MapaBits
protected:
        MAPABITS *b;
// Funciones miembro de la clase
public:
        MapaBits(char *p1,int sel1,int (*func1)(),int activo1,MAPABITS *b1,VENTANA *w1,
        RECUADRO *r1,int x1,int y1);
        void Dib_Map_Bits(void);
        ~MapaBits() { };   // Destructor
};
```

5.1.7.1. Funciones miembro de la clase *MapaBits*

Constructor para la clase *MapaBits*, hereda las propiedades de la clase *Recuadro*; inicializa la información para la lista ligada, el estado de selección del mapa de bits y la función que le será asociada.

```
MapaBits::MapaBits(char *p1,int sel1,int (*func1)(),int activo1,MAPABITS *b1,VENTANA
*w1,RECUADRO *r1,int x1,int y1)
:Recuadro(r1,x1,y1,p1)
{
        b=b1;
        APUNTADOR *oh;
        oh=(APUNTADOR *)w1;

        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig;  // Navegación por la lista ligada
        oh->sig=(APUNTADOR *)b;                          // Apunta al mapa de bits
```

```
        // Inicializa la variable struct MAPABITS *b
        b->mapabits=mb;
        b->sel=sel1;                    // Estado de selección del mapa de bits
        b->func=func1;                  // Función asociada al mapa de bits
        b->activo=activo1;              // Estado de activación del mapa de bits
        b->apunta.tipo=EnMapaBits;      // Lo coloca en la lista ligada
        b->apunta.sig=NULL;             // Apunta al siguiente elemento de la lista
        b->rec.izq=b->x=r->izq;         // Caracteristicas de la clase Recuadro
        b->rec.arriba=b->y=r->arriba;
        b->rec.der=r->der;
        b->rec.abajo=r->abajo;
}
```

La función *Dib_Map_Bits*, dibuja el objeto mapa de bits en las coordenadas definidas en el constructor y dependiendo también del adaptador gráfico en el que se esta trabajando.

```
// Dibuja el mapa de bits, de acuerdo a las especificaciones
// del constructor
void MapaBits::Dib_Map_Bits(void)
{
        char *tam;  // Almacena el mapa de bits
        int n;
        union REGS m;  // Coloca un objeto de tipo Raton
        Raton R(m);

        if(b->sel) n=COPY_PUT;  // Dibuja el mapa de bits con sus colores invertidos dependiendo de
        else n=NOT_PUT;         // su estado

        if(b->activo==INACTIVO) n=NOT_PUT;
        R.DesactivaRaton();     // Desactiva el cursor de ratón para que no interfiera cuando es colocado
                                // el mapa de bits
        // Coloca el mapa de bits en las coordenadas x, y; dependiendo del adaptador gráfico
        if(Det_Modo_Monocr()) putimage(b->x,b->y,b->mapabits,n);
        else {
                if((tam=Reg_Bmp_EGA(b->mapabits)) != NULL) {
                        putimage(b->x,b->y,tam,n);
                        free(tam);
                } else MensajeError();
        }
        R.Activa_Raton();
}
```

5.1.8. Definición de la clase base *Texto*

La clase *Texto*, es la clase base que es heredada a las clases: *Obj_Texto, Campo_Texto, Campo_Edicion, Cuad_Dial, Boton y Lista*. Esta clase utiliza la variable *struct TEXTO* (se encuentra en el archivo VARS_IGU.H), que es aplicable a todas las clases de tipo texto mencionadas anteriormente.

```
typedef struct {
        int x,y;            // Coordenadas iniciales para la cadena de texto
        char *texto;        // Contiene la cadena de texto
        }TEXTO;


class Texto {
// Variable utilizada por la clase Texto
protected:
        TEXTO *t;
// Funciones miembro de la clase
public:
        Texto(char *p1,TEXTO *t1,int x1,int y1);
        ~Texto(){ delete t ;}; // Destructor para la clase Texto
};
```

5.1.8.1. Funciones miembro de la clase *Texto*

Constructor para la clase *Texto*, es utilizado para inicializar las coordenadas donde aparecerá la cadena de texto y el propio texto, éstos son heredados a las clases *Obj_Texto, Campo_Texto, Campo_Edicion, Cuad_Dial, Boton y Lista*.

```
Texto::Texto(char *p1,TEXTO *t1,Int x1,int y1)
{
        t=t1;
        t=new TEXTO;
        if (!t)

        {
        cout << "\n Corriendo fuera de memoria ";
        exit(1);
        }
        t->texto=p1;  // Contiene la cadena de texto
        t->x=x1;      // Coordenadas para colocar el texto
        t->y=y1;

}
```

5.1.9. Definición de la clase *Obj_Texto*

La clase *Obj_Texto*, hereda las características de la clase *Texto*, se usa para representar al tipo de control más simple en una interfaz gráfica (una cadena de texto fija dentro de una ventana.) La variable *struct* OBJ_TEXTO (contenida en el archivo VARS_IGU.H), es utilizada para crear objetos de tipo texto; contiene a su vez una variable *struct* APUNTADOR con los campos para especificar las coordenadas donde se debe colocar el texto en la pantalla, un apuntador al texto y una bandera que indica si esta o no activo el texto (aparecerá en color negro o gris respectivamente.)

```
#define EnTexto      0x0010    // Máscara para indicar si el cursor se encuentra
                               // posicionado en un objeto de tipo texto

typedef struct {
        APUNTADOR apunta;     // struct de tipo APUNTADOR para el texto
        TEXTO tx;
        unsigned int activo;          // Indica el color del texto (gris inactivo, negro activo)
        }OBJ_TEXTO;
```

Definición de la clase *Obj_Texto*:

```
class Obj_Texto : public Texto  {
// Variable utilizada por la clase Obj_Texto
protected:
        OBJ_TEXTO *ot;
//  Funciones miembro de la clase
public:
        Obj_Texto(char *p1,int a1,OBJ_TEXTO *ot1,TEXTO *t1,VENTANA *w1,int x1,int y1);
        void DibujaTexto(void);
        ~Obj_Texto(){delete ot ;};
};
```

5.1.9.1. Funciones miembro de la clase *Obj_Texto*

El constructor para la clase *Obj_Texto*, tiene como función principal inicializar un objeto de tipo texto. Además de las características heredadas por la clase *Texto*, el constructor coloca el objeto dentro de la lista ligada, que es asociada a la ventana donde será puesto y el estado para la cadena de texto en color gris o negro.

```
Obj_Texto::Obj_Texto(char *p1,int a1,OBJ_TEXTO *ot1,TEXTO *t1,VENTANA *w1,int x1,int y1)
:Texto(p1,t1,x1,y1)
{       ot=ot1;
        APUNTADOR *oh;
        oh=(APUNTADOR *)w1;          // Asocia la lista ligada para la ventana correspondiente
        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig; // Hace el recorrido por la lista ligada
        oh->sig=(APUNTADOR *)ot;  // Apunta a un objeto de tipo texto
        ot->tx.texto=t->texto;          // Cadena de texto
        ot->activo=a1; // Estado del texto, aparecerá en color negro si esta activo y gris si no
        ot->tx.x=t->x;                  // Coordenadas para colocar el texto
        ot->tx.y=t->y;
        ot->apunta.tipo=EnTexto;  // Apunta a un objeto texto
        ot->apunta.sig=NULL;     // apunta al siguiente objeto de la lista ligada
}
```

La función *DibujaTexto*, dibuja un objeto de tipo texto en una ventana, esto lo realiza a través de la función *DrawString*, la cual permite colocar una cadena de texto en una coordenada especifica y puede representarla en color negro o gris; esta función es utilizada principalmente en índices de menús y botones, para indicar su estado de actividad. *DrawString* forma parte de la librería gráfica FUNCSASM.

```
void Obj_Texto::DibujaTexto(void)
{
        union REGS m;           // Define un objeto Raton
        Raton R(m);

        R.DesactivaRaton(); // Desactiva el cursor del ratón para que no interfiera con el texto
        DrawString(ot->tx.x,ot->tx.y,ot->tx.texto,ot->activo); // Coloca la cadena de texto
        R.Activa_Raton();   // Restablece el cursor del ratón
}
```

5.1.10. Definición de la clase *Campo_Texto*

La clase *Campo_Texto*, define la longitud asociada con el texto y dibuja un recuadro alrededor de éste. El objeto *Campo_Texto* tiene como datos: las propiedades de la clase *Texto*, los elementos de longitud de la cadena y las coordenadas del recuadro en donde va colocado el texto (propiedades heredadas por la clase *Recuadro*.) Utiliza una variable *struct* CAMPOTEXTO, para almacenar la información.

```
#define EnCampoTexto 0x0020  // Máscara para localizar el objeto CampoTexto

typedef struct {
        APUNTADOR apunta;     // struct de tipo APUNTADOR para la lista ligada
        TEXTO tct;            // struct de tipo TEXTO para este tipo de objeto
        int largo;            // Indica la longitud de la cadena
        RECUADRO rec;         // struct RECUADRO para el objeto texto
        unsigned int activo;  // Indica el estado del  (ACTIVO/INACTIVO)
        }CAMPOTEXTO;
```

Definición de la clase *CampoTexto*:

```
class CampoTexto :public Recuadro,public Texto{
// Variable utilizada por la clase CampoTexto
protected:
        CAMPOTEXTO *ct;     // struct CAMPOTEXTO
// Funciones miembro de la clase
public:
        CampoTexto(char *p1,int a1,int l1,CAMPOTEXTO *ct1,
        TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1);
        void DibCampoTexto(void);
        ~CampoTexto(){}; // Destructor para la clase CampoTexto
};
```

5.1.10.1. Funciones miembro de la clase *CampoTexto*

El constructor para la clase *CampoTexto*, permite crear un objeto de este tipo en una ventana. Inicializa los datos para este tipo de objetos, como son: la cadena de texto, su longitud y su incorporación dentro de la lista ligada que lo asocia a una ventana específica. Se muestra a continuación el detalle de esta función:

```
CampoTexto::CampoTexto(char *p1,int a1,int l1,CAMPOTEXTO *ct1,
TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1)
:Recuadro(r1,x1,y1,l1),Texto(p1,t1,x1,y1)
{
        ct=ct1;
        APUNTADOR *oh;

// Inicia el recorrido por la lista ligada ( estructura de datos )
        oh=(APUNTADOR *)w1; // Asocia la lista ligada para la ventana correspondiente
        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig;
        oh->sig=(APUNTADOR *)ct; // Apunta a un objeto de tipo CampoTexto
        ct->tct.texto=t->texto; // Cadena de texto
        ct->activo=a1; // Estado del texto
        ct->tct.x=t->x; // Coordenadas para colocar el texto
        ct->tct.y=t->y;
        ct->rec.izq=r->izq;         // Coloca las coordenadas del recuadro donde va el texto
        ct->rec.arriba=r->arriba;
        ct->rec.der=r->der;
        ct->rec.abajo=r->abajo;
        ct->apunta.tipo=EnCampoTexto;    // Apunta a un objeto CampoTexto
        ct->apunta.sig=NULL;    // apunta al siguiente objeto de la lista ligada
}
```

La función *DibCampoTexto*, dibuja un objeto de tipo *CampoTexto* en una ventana, coloca la cadena de texto y dibuja alrededor de ella un recuadro.

```
void CampoTexto::DibCampoTexto(void)
{
        union REGS m;       // Define un objeto Raton
        Raton R(m);
        R.DesactivaRaton(); // Desactiva el cursor del ratón para que no interfiera con el CampoTexto
        setwritemode(COPY_PUT); // Se dibuja el CampoTexto
        setfillstyle(SOLID_FILL,getmaxcolor());
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        setcolor(BLACK);
        bar(ct->rec.izq,ct->rec.arriba,ct->rec.der,ct->rec.abajo);
        rectangle(ct->rec.izq,ct->rec.arriba,ct->rec.der,ct->rec.abajo); // Coloca el recuadro donde
                                                                         // aparecerá el texto
        DrawString(ct->tct,x+4,ct->tct.y,ct->tct.texto,ct->activo); // Coloca la cadena de texto
        R.Activa_Raton();                   // Restablece el cursor del ratón
}
```

5.1.11. Definición de la clase *CampoEdicion*

La clase *CampoEdicion* es parecida a la clase *Campo_Texto*, la diferencia radica en puede ser editado el contenido del campo. El campo puede ser seleccionado, si esto ocurre, un cursor vertical aparecerá dentro del recuadro para poder editar el texto. *CampoEdicion* hereda también las propiedades de las clases *Texto* y *Recuadro*. Adicionalmente, la variable *struct* CAMPOEDICION contiene la longitud del texto y la posición de la cadena que se esta editando.

```
#define EnCampoEdicion      0x8000 0x0010    // Máscara para indicar si el cursor se encuentra
                                             // posicionado en un objeto de tipo CampoEdición

typedef struct {
        APUNTADOR apunta;    // struct de tipo APUNTADOR para el texto
        TEXTO tce;
        int largo;           // Longitud del campo de edición
        int poscur;          // Posición actual del cursor en la edición de texto
        int (*func)();       // Función asociada con el campo de edición de texto
        char sel;            // Indica el estado de selección del objeto
        RECUADRO rec;        // Recuadro alrededor del texto
        unsigned int activo; // Indica el estado del objeto (ACTIVO/INACTIVO)
        }CAMPOEDICION;
```

Definición de la clase *CampoEdicion:*

```
class CampoEdicion : public Recuadro,public Texto {
// Variable utilizada por la clase CampoEdicion
protected:
        CAMPOEDICION *ce;
// Funciones miembro de la clase
public:
        CampoEdicion(char *p1,int a1,int l1,int (*func1)(),CAMPOEDICION *ce1,
        TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1);
        void DibCampoEdicion(void);
        ~CampoEdicion(){};
};
```

5.1.11.1. Funciones miembro de la clase *CampoEdicion*

El constructor para la clase *CampoEdicion*, inicializa la información para objetos de esta clase: la cadena de texto, su longitud, el estado de selección del objeto, la posición del cursor del campo de edición y la incorporación de dicho objeto a la lista ligada (estructura de datos asociada a la ventana.)

```
// Definición de las funciones miembro de la clase CampoEdicion
CampoEdicion::CampoEdicion(char *p1,int a1,int l1,int (*func1)(),
CAMPOEDICION *ce1,TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1)
:Recuadro(r1,x1,y1,l1),Texto(p1,t1,x1,y1)
{
        ce=ce1;
        APUNTADOR *oh;
```

```
    oh=(APUNTADOR *)w1;
    while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig;  // Asocia la lista ligada a la ventana
                                                     // correspondiente

    oh->sig=(APUNTADOR *)ce;          // Apunta a un objeto de tipo CampoEdicion
    ce->largo=l;                      // Longitud de la cadena de texto
    ce->tce.texto=t->texto;   // Cadena de texto
    ce->activo=a1;            // Estado del texto, aparecerá en color negro, si esta activo, y gris si no
    ce->rec.izq=r->izq;       // Coloca las coordenadas del recuadro donde va el texto
    ce->rec.arriba=r->arriba;
    ce->rec.der=r->der;
    ce->rec.abajo=r->abajo;
    ce->tce.x=t->x;           // Coordenadas iniciales para colocar la cadena de texto
    ce->tce.y=t->y;
    ce->sel=0x00;             // Estado de selección del objeto
    ce->poscur=0;             // Inicializa la posición del cursor
    ce->func=func1;           // Asocia una función al objeto
    ce->apunta.tipo=EnCampoEdicion; ;  // Apunta a un objeto CampoEdicion
    ce->apunta.sig=NULL;      // apunta al siguiente objeto de la lista ligada
}
```

La función *DibCampoEdicion*, dibuja un objeto de tipo *CampoEdicion* en una ventana, coloca la cadena de texto, alrededor de ella dibuja un recuadro y el cursor en forma de "I" para iniciar la edición.

```
void CampoEdicion::DibCampoEdicion(void)
{
    union REGS m;          // Define un objeto Raton
    Raton R(m);

    R.DesactivaRaton();   // Desactiva el cursor del ratón para que no interfiera con el CampoEdicion
    setwritemode(COPY_PUT);   // Se dibuja el CampoEdicion
    selfillstyle(SOLID_FILL,getmaxcolor());
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    setcolor(BLACK);
    bar(ce->rec.izq,ce->rec.arriba,ce->rec.der,ce->rec.abajo);
    rectangle(ce->rec.izq,ce->rec.arriba,ce->rec.der,ce->rec.abajo); // Coloca el recuadro donde
                                                                     // aparecerá el campo de edición
    DrawString(ce->tce.x,ce->tce.y,ce->tce.texto,ce->activo); // Coloca la cadena de texto
    if(ce->activo==ACTIVO && ce->sel) {
        setcolor(getmaxcolor());
        setwritemode(XOR_PUT);
        line(ce->rec.izq+(8*ce->poscur)+2,ce->rec.arriba+2,ce->rec.izq+(8*ce->poscur)
        +2,ce->rec.abajo-2);  // Coloca el cursor de edición
    }
    R.Activa_Raton();                  // Restablece el cursor del ratón
}
```

5.1.12. Definición de la clase *Cuad_Dial*

Esta clase nos permite crear objetos de tipo cuadro de selección o cuadro de diálogo, al igual que las clases anteriores, hereda las características de las clases *Recuadro* y *Texto* . Esta clase usa la variable *struct* CUADDIAL, para guardar su información relevante.

```
#define EnCuadDial     0x4000 // Máscara para localizar el objeto CuadDial

typedef struct {
        APUNTADOR apunta;   // struct de tipo APUNTADOR para la lista ligada
        TEXTO tcd;          // struct de tipo TEXTO para este tipo de objeto
        RECUADRO rec;       // struct RECUADRO para el objeto texto
        char sel;           // Indica el estado de selección del cuadro de diálogo
        unsigned int activo; // Indica el estado del (ACTIVO/INACTIVO)
        }CUADDIAL;
```

Definición de la clase *Cuad_Dial*

```
class Cuad_Dial: public Recuadro,public Texto {
// Variables utilizadas por la clase Cuad_Dial
protected:
        CUADDIAL *cb;          // Variable para el cuadro de diálogo

// Funciones miembro de la clase
public:
        Cuad_Dial(char *texto1,int sel1,int activo1,CUADDIAL *cb1,
        TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1);
        void Dib_Cuad_Dial(void);
        ~Cuad_Dial(){};
};
```

5.1.12.1. Funciones miembro de la clase *Cuad_Dial*

El constructor de la clase *Cuad_Dial*, inicializa los datos para el cuadro de selección o de diálogo (para la variable *struct* CUADDIAL); coloca las coordenadas para el texto, la cadena de texto y el propio objeto *Cuad_Dial*, en la lista ligada.

```
// Funciones usadas en los campos de edición
// Constructor para la clase Cuad_Dial, que crea un objeto cuadro de diálogo
// en una ventana
Cuad_Dial::Cuad_Dial(char *texto1,int sel1,int activo1,
CUADDIAL *cb1,TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1)
: Recuadro(r1,x1,y1),Texto(texto1,t1,x1,y1)
{
        cb=cb1;
        APUNTADOR *oh;

        // Hace el recorrido por la lista ligada
        oh=(APUNTADOR *)w1; // Asocia la lista ligada para la ventana correspondiente
```

```
        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig;
        oh->sig=(APUNTADOR *)cb; // Apunta a un objeto de tipo cuadro de diálogo

        // Inicializa los datos para una variable CUADDIAL
        // define las coordenadas del recuadro para desplegar el diálogo

        cb->tcd.texto=t->texto; // Contiene el texto que aparecerá en la ventana
        cb->sel=sel1;    // Bandera para indicar si el cuadro de diálogo
                         // ha sido seleccionado
        cb->activo=activo1;      // Estado del cuadro de diálogo, si puede ser seleccionado o no
        cb->rec.izq=r->izq-1;    // Coloca las coordenadas del recuadro donde va el texto
        cb->rec.arriba=r->arriba-1;
        cb->rec.der=r->izq+10;
        cb->rec.abajo=r->arriba+10;
        cb->tcd.x=t->x;    // Coordenadas iniciales para el cuadro de diálogo
        cb->tcd.y=t->y;
        cb->apunta.tipo=EnCuadDial; // Busca en la lista ligada si esta apuntando
                                    // dentro del cuadro de diálogo
        cb->apunta.sig=NULL; //Coloca el valor del apuntador sig = NULL
}
```

La función *Dib_Cuad_Dial*, dibuja un objeto de tipo cuadro de selección o de diálogo en una ventana; coloca la cadena de texto con el mensaje y a un lado de ésta, dibuja un recuadro que puede ser seleccionado, dependiendo del estado que se le defina (activo o inactivo), el recuadro aparecerá con una marca (una X en su interior.)

```
// Dibuja el cuadro de diálogo
void Cuad_Dial::Dib_Cuad_Dial(void)

{
        union REGS m;
        Raton R(m);   // Definición del objeto Raton

        R.DesactivaRaton(); // Desactiva el cursor del ratón para que no interfiera con el cuadro

        // Código para dibujar el cuadro de diálogo
        setwritemode(COPY_PUT);
        setfillstyle(SOLID_FILL,getmaxcolor());
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        setcolor(BLACK);
        bar(cb->rec.izq,cb->rec.arriba,cb->rec.der,cb->rec.abajo);
        rectangle(cb->rec.izq,cb->rec.arriba,cb->rec.der,cb->rec.abajo); // Coloca el recuadro donde
                                                                         // aparecerá el texto
        DrawString(cb->rec.der+16,cb->rec.arriba
        +2,cb->tcd.texto,cb->activo); // Coloca la cadena de texto
        // Si la opción fue seleccionada dibuja una X dentro del cuadro asociado al mensaje
        if(cb->sel) {
                line(cb->rec.izq,cb->rec.arriba,cb->rec.der,cb->rec.abajo);
                line(cb->rec.izq,cb->rec.abajo,cb->rec.der,cb->rec.arriba);
        }
        R.Activa_Raton();              // Restablece el cursor del ratón
}
```

5.1.13. Definición de la clase *Botones*

La clase *Botones*, crea objetos de tipo botón en una ventana. La información necesaria para definir este tipo de objetos se encuentra en la variable *struct* BOTON, que se encuentra en el archivo VARS_IGU.H; la clase *Botones* toma también los datos de las clases *Recuadro* y *Texto*.

```
#define EnBoton      0x0004 // Máscara para indicar si el cursor se encuentra
                            // posicionado en un objeto de tipo Boton

// struct para definir un botón
typedef struct {
        APUNTADOR apunta;   // struct de tipo APUNTADOR para el botón
        RECUADRO rec;       // Contiene una struct de tipo RECUADRO
        TEXTO tb;           // Contiene el texto para el botón
        unsigned int activo; // Indica el estado del botón (ACTIVO/INACTIVO)
        }BOTON;
```

Definición de la clase *Boton:*

```
class Boton: public Recuadro,public Texto {
// Variables utilizadas por la clase Boton
protected:
        BOTON *b;           // Estructura BOTON
//  Funciones miembro de la clase
public:
        Boton(char *tx1,int activo1,BOTON *b1,TEXTO *t1,VENTANA *w1,
        RECUADRO *r1,int izq1,int sup1,int ancho1,int largo1);
        void DibujaBoton(void);
         ~Boton(){};
};
```

5.1.13.1. Funciones miembro de la clase *Botones*

El constructor de la clase *Botones*, permite crear un objeto botón en una ventana, inicializa las coordenadas donde aparecerá el botón (su contorno), el texto que va en el interior del recuadro, su estado de selección y la inclusión del objeto en la lista ligada.

```
// Constructor para la clase Boton
Boton::Boton(char *tx1,int activo1,BOTON *b1,TEXTO *t1,
VENTANA *w1,RECUADRO *r1,int izq1,int sup1,int ancho1,int largo1)
:Recuadro(r1,izq1,sup1,ancho1,largo1),Texto(tx1,t1,izq1,sup1)
{
        b=b1;
        APUNTADOR *oh;

        oh=(APUNTADOR *)w1;
        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig;  // Asocia la lista ligada a la ventana
                                                         // correspondiente
```

```
        oh->sig=(APUNTADOR *)b; // Apunta a un objeto de tipo Boton
        // inicializa a la variable de tipo BOTON
        b->tb.texto=r->texto;   // Texto que se encuentra dentro del botón
        b->activo=activo1;      // Estado del texto, aparecerá en color negro si esta activo, y gris si no
        b->rec.izq= r->izq;     // Coloca las coordenadas del recuadro donde aparecerá
                                // el texto del botón
        b->rec.arriba= r->arriba;
        b->rec.der=r->der;
        b->rec.abajo=r->abajo;
        b->rec.izq &= 0xfff8;
        b->rec.der &= 0xfff8;
        b->apunta.tipo=EnBoton;     // Apunta a un objeto de tipo Boton
        b->apunta.sig=NULL;         // apunta al siguiente objeto de la lista ligada
}
```

DibujaBoton dibuja el objeto botón dentro de una ventana, coloca el recuadro con un contorno redondeado, alrededor del texto del botón.

```
void Boton::DibujaBoton(void)
{
        union REGS m; // Se define un objeto Raton
        Raton R(m);

        R.DesactivaRaton();   // Desactiva el cursor del ratón, para que no interfiera con el botón
        setwritemode(COPY_PUT); // Se dibuja el botón
        setfillstyle(SOLID_FILL,getmaxcolor());
        setlinestyle(SOLID_LINE,0,THICK_WIDTH);
        setcolor(BLACK);
        bar(b->rec.izq,b->rec.arriba,b->rec.der,b->rec.abajo);
        rectangle(b->rec.izq,b->rec.arriba,b->rec.der,b->rec.abajo);
        setlinestyle(SOLID_LINE,0,THICK_WIDTH);
        line(b->rec.der+2,b->rec.arriba+SOMBRAVENT,b->rec.der+2,b->rec.abajo+3);
        line(b->rec.izq+SOMBRAVENT,b->rec.abajo+2,b->rec.der,b->rec.abajo+2);
        DrawString(b->rec.izq+8,b->rec.arriba+4,b->tb.texto,b->activo); // Dibuja el texto del botón
        R.Activa_Raton();           // Restablece el cursor del ratón
}
```

5.1.14. Definición de la clase *Lista*

La clase *Lista*, crea varios objetos de tipo texto dentro de un recuadro. Por lo general, siempre es usada con las barras de despliegue; los datos necesarios para definir este tipo de objetos se encuentra en la variable *struct* LISTA. La clase *Lista* hereda además, los datos de las clases *Recuadro* y *Texto*. A continuación se muestra el código para definir los datos de esta clase.

```
#define EnLista         0x0040  // Máscara para localizar el objeto Lista

typedef struct {
        APUNTADOR apunta;  // struct de tipo APUNTADOR para la lista ligada
        TEXTO tl;          // struct de tipo TEXTO para este tipo de objeto
        int largo,ancho;   // Dimensiones de la lista
        int cont;          // Número de cadenas de texto que contendrá la lista
```

```
        int arriba;
        RECUADRO rec;       // struct RECUADRO para el objeto texto
        }LISTA;
```

Definición de la clase *Lista*:

```
class Lista: public Recuadro,public Texto {
// Variables utilizadas por la clase Lista
protected:
        LISTA *l;           // La estructura lista
        int cuenta;
// Funciones miembro de la clase
public:
        Lista(int largo1,int ancho1,int cont1,int arriba1,char *base1,LISTA *l1,TEXTO *t1,VENTANA
        *w1,RECUADRO *r1,int x1,int y1);
        void Dibuja_Lista(void);
        ~Lista();
};
```

5.1.14.1. Funciones miembro de la clase *Lista*

El constructor para la clase *Lista*, inicializa los siguientes datos: las coordenadas iniciales para la lista, el número de cadenas de texto, su longitud, un contador para las cadenas, un indicador de posición y el manejo del objeto dentro de la lista ligada para la ventana en la cual es colocado dicho objeto.

```
// Constructor de la clase lista
Lista::Lista(int largo1,int ancho1,int cont1,int arriba1,
char *base1,LISTA *l1,TEXTO *t1,VENTANA *w1,RECUADRO *r1,int x1,int y1)
: Recuadro(r1,x1,y1),Texto(base1,t1,x1,y1)
{
        l=l1;
        APUNTADOR *oh;

        // Empieza el recorrido por la lista ligada
        oh=(APUNTADOR *)w1; // Asocia la lista ligada para la ventana correspondiente
        while(oh->sig != NULL) oh=(APUNTADOR *)oh->sig;
                oh->sig=(APUNTADOR *)l; // Apunta a un objeto de tipo lista

        // Inicializa la variable struct LISTA
        l->largo=largo1;  // Las dimensiones de la lista
        l->ancho=ancho1;
        l->cont=cont1;          // Contador de las cadenas de texto
        l->arriba=arriba1;      // Recorrido de las listas
        l->tl.texto=t->texto;   // Contiene la cadena de  texto
        l->rec.izq=r->izq-2;  // Coloca las coordenades del recuadro donde va la lista
        l->rec.arriba=r->arriba-2;
        l->rec.der=r->izq+((l->largo+2)*8)+2;
        l->rec.abajo=r->arriba+(l->ancho*PROFLINMENUS)+2;
        l->tl.x=t->x;
        l->tl.x &= 0xfff8;
        l->tl.y=t->y;
```

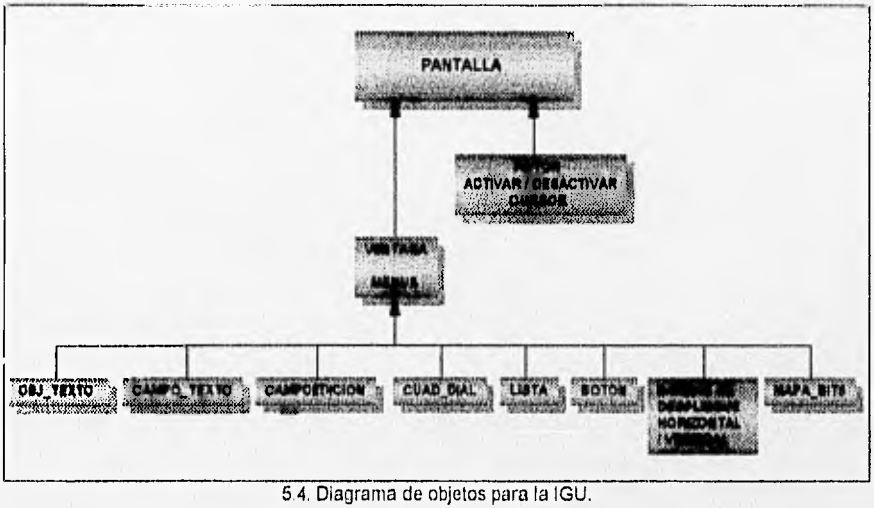
```
    l->apunta.tipo=EnLista;      // Busca en la lista ligada si esta apuntando
                                 // dentro del objeto Lista
      l->apunta.sig=NULL; //Coloca el valor del apuntador sig = NULL
}
```

La función *Dibuja_Lista*, dibuja varios objetos de tipo texto (cadenas de texto), dentro de un recuadro, en una ventana. Esta función esta estrechamente relacionada con las funciones para dibujar barras de despliegue.

```
void Lista::Dibuja_Lista(void)
// Función para dibujar el objeto LISTA
{
      union REGS m;        // Definición del objeto Raton
      Raton R(m);
      static int i=0;      // Contador para el número de cadenas de texto

      R.DesactivaRaton();// Desactiva el cursor del ratón, para que no interfiera con el dibujo de la lista

      // Código para dibujar la lista
      setwritemode(COPY_PUT);
      setfillstyle(SOLID_FILL,getmaxcolor());
      setlinestyle(SOLID_LINE,0,NORM_WIDTH);
      setcolor(BLACK);
      bar(l->rec.izq,l->rec.arriba,l->rec.der,l->rec.abajo);
      rectangle(l->rec.izq,l->rec.arriba,l->rec.der,l->rec.abajo); // Coloca el recuadro donde aparecerá
                                                                   // el texto

      // Coloca el contenido de la lista (las cadenas de texto)
      if(l->tl.texto != NULL) {
            for(i=0;i<l->ancho;++i) {
                  if((l->arriba+i) >= l->cont) break;
                  DrawString(l->tl.x+8,l->tl.y+(i*PROFLINMENUS)+2,l->tl.texto
                  +((l->arriba+i)*(l->largo+1)),ACTIVO); // Coloca la cadena de texto
            }
      }
    R.Activa_Raton();      // Restablece el cursor del ratón

}
```

5.2. ESQUEMA FINAL DE LAS CLASES PARA LA INTERFAZ GRAFICA DE USUARIO

De acuerdo al diseño anterior, se obtuvo el siguiente diagrama de objetos, que podemos manejar como herramientas para construir una interfaz gráfica y que pueden ser utilizadas en diferentes aplicaciones que así lo requieran.



5.4. Diagrama de objetos para la IGU.

5.3. DESCRIPCION DE OTRAS FUNCIONES QUE FORMAN PARTE DE LA LIBRERIA PARA CONSTRUIR UNA INTERFAZ GRAFICA

void InviertRec(RECUADRO *r1)

Invierte los colores de un objeto recuadro, es usada para indicar la selección de algunos objetos gráficos en pantalla.

void Inic_Contr_Vents(void)

Función para iniciar el control de las ventanas. Coloca la ventana inicial con las coordenadas del tipo de tarjeta de video (el escritorio), en la variable *Pantalla*.

MAPABITS *Loc_Map_Bits(VENTANA *w1,LOCALIZA *p1)

Regresa un mapa de bits localizado dentro de una ventana de trabajo.

void Sel_Map_Bits(MAPABITS *b1)

Localiza un mapa de bits, de acuerdo a la posición del cursor dentro del área de la ventana de trabajo.

void Redib_Map_Bits(MAPABITS *b)

Función que vuelve a dibujar el mapa de bits que haya sido seleccionado.

*void Dib_Area_Contr(BARRASDESP *bs1)*

Dibuja el área de control de la barra de despliegue. Dibuja el área de control de la barra de despliegue, donde se mueve el cuadro indicador de despliegue.

*void Ajuste_Area_Contr(BARRASDESP *bs1,Int n)*

Ajusta el área de control de la barra de despliegue. Hace el ajuste del movimiento y del efecto visual del indicador en el área de desplazamiento.

*void Borra_Area_Contr(BARRASDESP *bs1)*

Borra el área de control de la barra de despliegue. Esta función crea el efecto de borrado del indicador de desplazamiento, redibujando el cuadro indicador en otra área y rellenando esta área con el fondo que tiene.

*BARRASDESP *Loc_Bar_Desp(VENTANA *w1,LOCALIZA *p1)*

Regresa un apuntador, indicando la barra de despliegue seleccionada horizontal o vertical, en una ventana de trabajo.

*void Sel_Bar_Desp(BARRASDESP *bs1,Int rt1)*

De acuerdo a la posición del cursor, indica si esta apuntando a una barra de despliegue.

*CAMPOEDICION *Loc_Camp_Ed(VENTANA *w1,LOCALIZA *p1)*

Devuelve un campo de edición localizado en una ventana de trabajo.

*void Sel_Camp_Ed(CAMPOEDICION *f1)*

Localiza un campo de edición, de acuerdo a la posición del cursor dentro del área de la ventana de trabajo.

*void RedibCampoEdicion(CAMPOEDICION *f)*

Redibuja el campo de edición que haya sido seleccionado, es decir, actualiza su estado.

*CUADDIAL *Loc_Cuad_Dial(VENTANA *w1,LOCALIZA *p1)*

Función para localizar un cuadro de diálogo en una ventana.

*void Sel_Cuad_Dial(CUADDIAL *cd)*

Esta función selecciona el cuadro de diálogo.

*void Redib_Cuad_Dial(CUADDIAL *cb)*

Función utilizada para reconstruir un cuadro de diálogo seleccionado.

*BOTON *Loc_Boton(VENTANA *w1,LOCALIZA *p1)*

Regresa un apuntador de tipo BOTON, del botón seleccionado. Localiza si el cursor del ratón se encuentra dentro del área de un objeto *Boton*, regresa el botón seleccionado o *NULL* si no lo localiza.

*void Sel_Boton(BOTON *b1)*

Maneja la selección de un objeto BOTON, invierte el área definida por su elemento de tipo recuadro, si lo selecciona, invierte sus colores.

*void Redib_Boton(BOTON *b)*

Redibuja un objeto de tipo BOTON.

*LISTA *Loc_Lista(VENTANA *w1,LOCALIZA *p1)*

Regresa un apuntador del objeto LISTA, que haya sido seleccionado por el cursor del ratón.

*void Sel_Lista(LISTA *lp1,char *s1)*

Indica que ha sido seleccionado el objeto LISTA.

*void Redib_Lista(LISTA *l)*

Redibuja el objeto LISTA que haya sido seleccionado.

int Verif_RAM(unsigned int n)

Esta función prueba un segmento de memoria para verificar que este disponible, si lo esta regresa un valor verdadero (1).

*int Tam_Memoria(RECUADRO *r)*

Regresa el tamaño necesario, para la variable *struct* de tipo RECUADRO.

int Det_Modo_Monocr(void)

Regresa un valor verdadero, si el modo gráfico es monocromático.

int Sel_4Planos(void)

Selecciona los cuatro planos de los adaptadores EGA y VGA a colores. Regresa (0) si es monocromático, (1) si maneja colores, si es igual a 1 escribe a los cuatro planos para que aparezca en forma monocromática.

*void Obtiene_Imagen(int izq,int arriba,int der,int abajo,unsigned char *p)*

Función que utiliza la técnica de fragmentación de imágenes por alineación de bytes, para obtener una imagen.

*void Coloca_Imagen(int izq,int arriba,unsigned char *p)*

Función que utiliza la técnica de fragmentación de imágenes por alineación de bytes, para colocar una imagen.

*int Tam_Imagen(RECUADRO *r)*

Regresa el tamaño necesario para la variable RECUADRO.

long Contador(void)

Obtiene el contador del BIOS.

int Det_Tarj_Herc(void)
Regresa un valor verdadero, si se trata de una tarjeta Hércules.

int Detecta_Tarj(void)
Regresa el tipo de tarjeta que se está utilizando o (-1), si no hay tarjeta de gráficos disponible.

void Reg_Tarj_Herc(void)
Esta función, después de usar la tarjeta Hércules en modo gráfico, lo desactiva antes de salir a DOS.

char *Reg_Bmp_EGA(char *version)
Regresa una versión EGA monocromática de un mapa de bits

int LocRecuadro(LOCALIZA *p,RECUADRO *r)
Regresa un valor de (1), si localiza que el apuntador se encuentra dentro del área de una variable *struct* RECUADRO.

int Comp(const void *a,const void *b)
Función que permite comparar dos variables.

char *Reg_Apunt(char *p,long l)
Regresa un apuntador *far p+l*.

int Checa_Vol_Rect(int l,int t,int r,int b)
Regresa un valor verdadero, si el rectángulo tiene algún volumen.

void Interc_Valores(int *a,int *b)
Intercambia dos valores enteros.

void Despl_Recta(RECUADRO *r,LOCALIZA *p)
Desplaza el RECUADRO *r*, a la posición *p*.

void Def_Lim_Recuad(LOCALIZA *p,RECUADRO *r)
Limita la variable LOCALIZA *p*, a los límites de RECUADRO *r*.

int Checa_Var(int c)
Esta función regresa un valor verdadero, si *c* es un dígito

void LimpiaLinea(char *p,int bytes,int ancho)
Limpia cualquier bit extraño, del extremo derecho de la línea de datos del mapa de bits.

*void InvierteBytes(char *p,int n)*
Invierte *n* bytes contenidos en *p*.

*void DefineRecta(LOCALIZA *r1,LOCALIZA *r2)*
Verifica que LOCALIZA *r1*, se encuentre a la izquierda de *r2*.

int NoHaceNada(void)
Esta función es utilizada en los índices de menús, pero no realiza ninguna actividad.

int Tecla_Pres(void)
Función que detecta si una tecla ha sido presionada.

int Sel_Indice_Menu(void)
Función que detecta la actividad del teclado, para manejar las teclas alternas de los índices de los menús.

*void ActivaTeclado(VENTANA *w)*
La función *ActivaTeclado* detecta un caracter relacionado con alguno de los objetos en la pantalla, comienza a buscar en la lista ligada por el objeto específico. Esta función estará especialmente interesada en dos tipos de objetos, el objeto *Boton*, si éste aparece en la ventana y es presionada la tecla de retorno (por ejemplo: el botón de *Aceptar* o de *Sí)* tendrá el mismo efecto que si es seleccionado por el ratón. Si hay otro botón le será asociada la tecla de escape *Esc* (botón para *Cancelar* o de *No*), siempre y cuando los botones hayan sido agregados en este orden a la ventana. Por convención, al primer botón se le asociará la tecla de retorno y al segundo la de escape *Esc*. El otro objeto de interés para esta función es el *Campo_Edicion*, cuando localiza este tipo de objeto, la función permite el uso de las teclas de inicio, de fin, de retroceso, de borrado y las teclas de dirección pueden ser usadas para posicionar el cursor en el campo de edición.

*int Loc_Pos_Raton(LOCALIZA *p,VENTANA *w)*
Regresa un valor para indicar donde se localiza el cursor del ratón dentro de la pantalla.

CAPITULO 6. ANALISIS Y DISEÑO DE LA APLICACION

6.1. DEFINICION DE LA APLICACION

6.1.1. Herramienta de dibujo

El propósito de diseñar una aplicación, estará enfocado en mostrar al usuario de la librería de construcción de interfaces gráficas, la forma en que puede ser empleada en sus programas de aplicación.

En este caso, se desarrollará una sencilla herramienta de dibujo, la cual ejemplificará los posibles usos que pueden dársele a la librería gráfica, se pretende abarcar todos los objetos gráficos que contiene la librería. Por otra parte, el programa de dibujo manejará archivos en formato gráfico y estos podrán ser editados a nivel pixel.

6.1.2. Formato gráfico PCX

El formato PCX fue diseñado por *ZSoft*, creador de PC *Paintbrush*, que fue uno de los primeros programas primitivos de dibujo. PCX, es uno de los formatos más comunes para almacenar archivos de imágenes; entre sus características más importantes tenemos que ocupa menor espacio que otro tipo de formatos (BMP, TIF, GIF) y puede ser manejado muy fácilmente, ya que la compactación y descompactación de archivos PCX puede realizarse rápidamente; además, no es necesario escribir un programa muy grande para descomprimir una línea de un archivo PCX.

Los archivos PCX consisten de un encabezado de 128 bytes de longitud, que define parámetros tales como:

- Las dimensiones de la imagen contenida en el archivo PCX.
- El número de colores que contiene la imagen.
- La información del fabricante, tiene un valor constante de diez y sirve para reconocer que se trata de un archivo PCX.
- Datos de versión y codificación, le indican al programa decodificador como manejar la paleta de colores del dibujo (el esquema de colores.)
- Bits por pixel y planos de color, estos dos parámetros, en forma conjunta definen cuantos colores forman parte del dibujo.
- Tipo de paleta, indica la paleta de colores utilizada por el archivo PCX.
- Número de *bytes* por linea, es el valor que especifica el largo de una línea de la imagen.

Cuando un archivo PCX es abierto con la intención de cargarlo en el *buffer* o desplegarlo en la pantalla, se deberán de copiar los 128 *bytes* que forman parte de su encabezado, para que sus elementos sean accesados.

En un archivo PCX, la imagen puede contener múltiples líneas en blanco o varias líneas iguales, por lo que este tipo de líneas pueden ser reducidas, indicando solo su posición y el número de veces que aparecen repetidas en la imagen. De hecho, todas las líneas de la imagen pueden ser representadas de dos formas:

1) Como un campo duplicado o repetido: es una línea o porción de la línea, donde un *byte* se encuentra repetido un número fijo de veces.

2) Un campo de tipo cadena: es una línea o porción de la misma, donde dos o más *bytes* adyacentes son diferentes.

En una imagen compleja, las líneas son con frecuencia representadas como una combinación alternada de campos duplicados y campos cadena. La forma en la cual están codificados estos tipos de campos, permite reducir la cantidad de espacio que ocupa una imagen en un archivo PCX.

Los archivos PCX siempre son descomprimidos línea por línea. Se puede desempacar una línea empezando con el primer *byte* después del encabezado y leyendo cada campo en el archivo, hasta que se ha desempacado el número de *bytes* por línea especificados en el encabezado.

Cada línea de un archivo PCX es codificada como una serie de campos, el primer *byte* después del encabezado, es el inicio del primer campo. El código para leer un archivo PCX desempaca un campo y lee el primer *byte* de dicho campo, para verificar si sus dos bits superiores son iguales a uno, si no lo son, este *byte* es escrito directamente al *buffer* de la pantalla (el campo en cuestión es de un *byte* de longitud, el campo cadena.)

Si los dos *bits* superiores son iguales a uno, este *byte* corresponde a un campo duplicado, los dos *bits* superiores son eliminados y el valor resultante (contador) es almacenado. El siguiente *byte* es leído y escrito hacia el *buffer* de salida, el número de veces que especifique el contador de repeticiones. Una vez que el campo ha sido leído, se debe verificar si ha sido desempacada una línea completa (número de *bytes* por línea.) La fig. 6.1 muestra el diagrama de flujo para desempacar una línea de un archivo PCX.

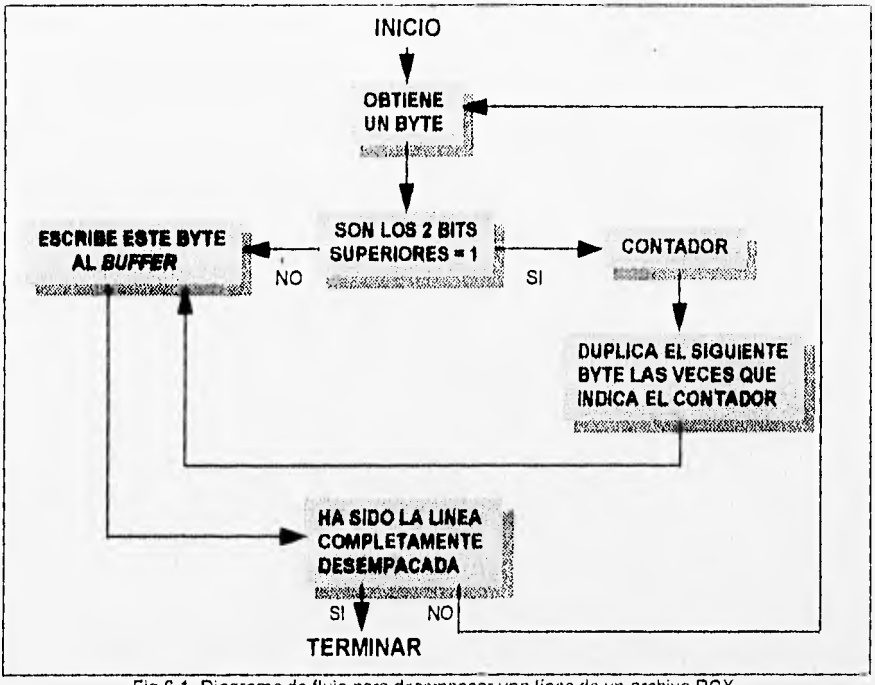


Fig.6.1. Diagrama de flujo para desempacar una línea de un archivo PCX.

6.1.3. Funciones de la herramienta de dibujo:

Las funciones que serían consideradas para la edición de un archivo en formato gráfico PCX monocromático, son las siguientes:

- Para dibujo de líneas.
- Dibujo de puntos o pixeles.
- Herramienta para borrar.
- Para dibujar cuadrados y rectángulos.
- Para dibujar elipses y círculos.
- Para dibujar múltiples pixeles (*spray*.)
- Deshacer una actividad realizada.
- Realizar la ampliación y edición por pixeles.
- Selección de tipo y grosor de línea.
- Selección de relleno de figuras (elipses y rectángulos.)

6.2. DISEÑO DEL PROGRAMA DE APLICACION

6.2.1. Definición de menús

Los menús a ser utilizados por el programa de aplicación, serán solo los necesarios para hacer el manejo de la edición de los archivos gráficos PCX. De esta manera, se consideran los siguientes menús:

1) Menú de escritorio: para mantener un estándar con otros programas de aplicación de este tipo, este menú contendrá información referente al nombre de la aplicación y el número de versión. La opción para este menú sería una opción de "referencia".

2) Menú para manejo de los archivos: será el menú más importante del programa de aplicación; en este caso, el menú permitirá realizar las actividades necesarias para la edición de los archivos PCX, que son: la creación de nuevos archivos, la apertura de un archivo existente, cerrar un archivo abierto, salvar un archivo en disco, salvar un archivo con otro nombre y una opción para salir del programa. De esta forma, las opciones del menú de archivo serían : "nuevo", "abrir", "cerrar", "salvar", "salvar como" y "salir".

3) Menú de opciones: este menú trabajará en combinación con el menú de manejo de archivos, para establecer algunas características de las herramientas de dibujo, como son: tipos de líneas, grosor de la línea y tipos de relleno para las herramientas de líneas y figuras sólidas. Las opciones del menú serían: tipo de línea, grosor de línea y relleno.

4) Menú de ayuda: este menú proporcionará la ayuda para el programa de aplicación, principalmente explicaría las funciones de los diferentes menús usados; de esta manera, la ayuda dará una breve descripción de los menús de: escritorio, archivo, opciones y de la ayuda misma.

En la figura 6.2, se muestra un diagrama general de los menús que serán considerados para el programa de aplicación:

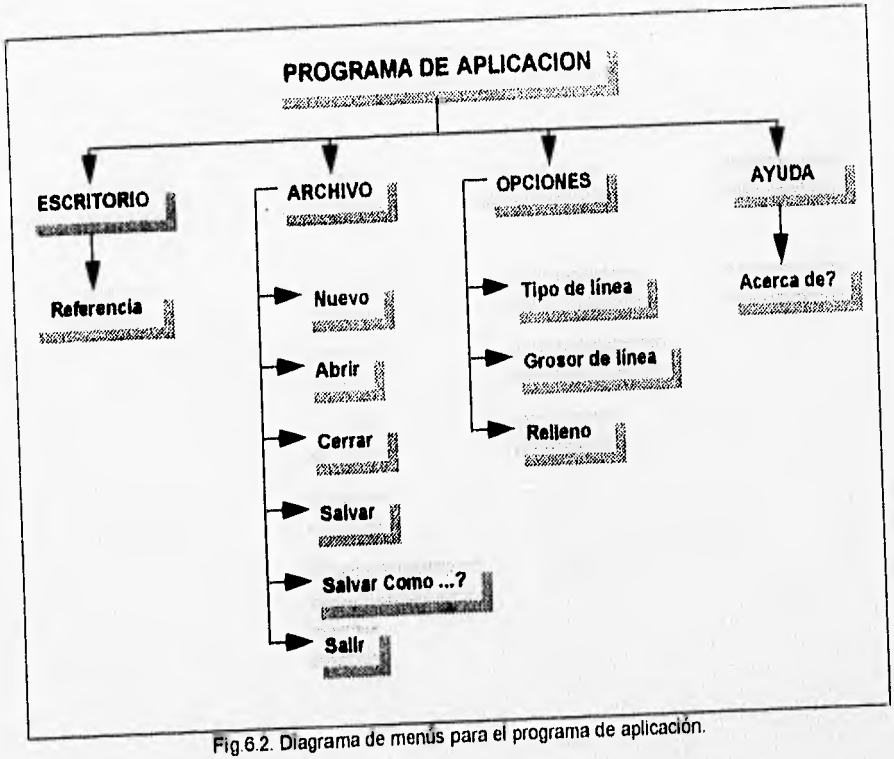


Fig.6.2. Diagrama de menús para el programa de aplicación.

6.2.2. Diseño del programa principal

De acuerdo a la información anterior y a los menús que se utilizarían en el programa de aplicación se tendría el siguiente diagrama de flujo para el programa principal:

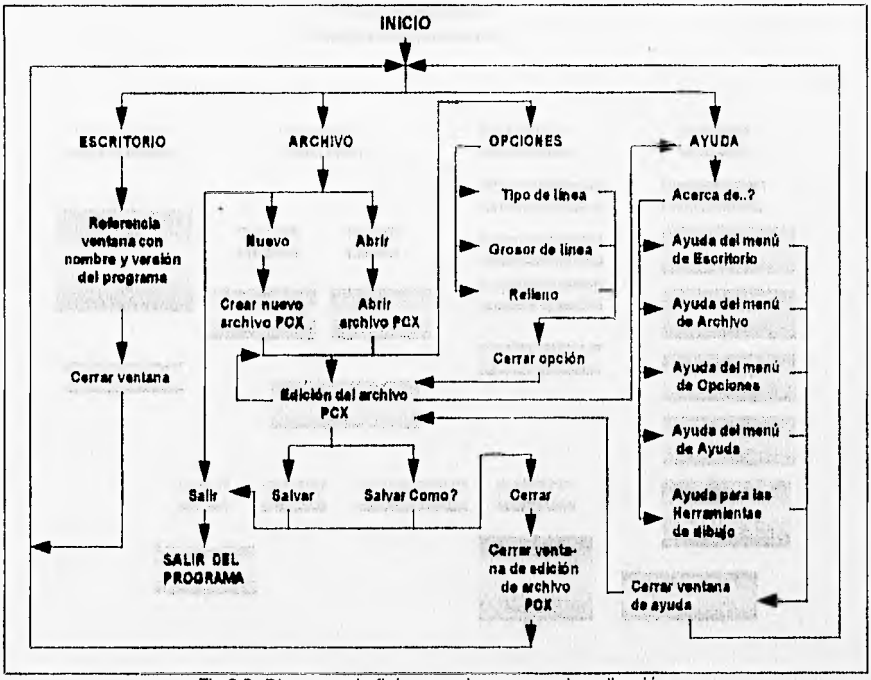


Fig.6.3. Diagrama de flujo para el programa de aplicación.

6.2.3. Definición de los objetos gráficos que se utilizarán en la aplicación.

En la siguientes tablas, se muestran los objetos que se consideran necesarios para el programa de aplicación.

Elemento del programa	Objetos gráficos
Pantalla	Objeto adaptador gráfico Barra de menús - Títulos de menús - Ventanas para menús colgantes - Objetos de tipo texto - Objeto ratón
Presentación	Ventana (1) (de inicio del programa) - Objetos de tipo texto - Botón - Objeto ratón
Menú: ESCRITORIO	Ventana (Menú colgante) - Objeto ratón

Elemento del programa	Objetos gráficos
indice de menú: Referencia	Ventana (1) - Objetos de tipo texto - Mapa de bits - Botón - Objeto ratón
Menú: ARCHIVO	Ventana (Menú colgante) - Objeto ratón
Indice de menú: Nuevo	Ventana (1) (información para el nuevo archivo PCX) - Objetos texto - Campo de texto - Campo de edición de texto - Botones - Objeto ratón Ventana (2) (ventana de trabajo para edición del archivo PCX) - Barras de despliegue - Mapas de bits (herramientas de dibujo) - Objeto ratón
Indice de menú: Abrir	Ventana (1) (para abrir un archivo existente) - Objetos texto - Campo de texto - Campo de edición de texto - Lista - Barra de despliegue - Botones - Objeto ratón Ventana (2) (mensaje para indicar si no se puede abrir el archivo PCX) - Objetos texto - Botones - Objeto ratón Ventana (3) (ventana de trabajo para edición del archivo PCX) - Barras de despliegue - Mapas de bits (herramientas de dibujo) - Objeto ratón
Indice de menú: Cerrar	Ventana (1) (mensaje para cerrar el archivo PCX) - Objetos texto - Botones - Objeto ratón

Elemento del programa	Objetos gráficos
Indice de menú: Salvar	Ventana (1) (mensaje para salvar el archivo PCX) - Objetos texto - Mapa de bits - Botones - Objeto ratón
Indice de menú: Salvar Como?	Ventana (1) (Para salvar el archivo con otro nombre) - Objetos texto - Botones - Campos de edición de texto - Objeto ratón
Indice de menú: Salir	Ventana (1) (mensaje para salir del programa de aplicación) - Objetos texto - Mapa de bits - Botones - Objeto ratón
Menú: OPCIONES	Ventana (Menú colgante) - Objeto ratón
Indice de menú: Tipo de linea	Ventana (1) (de selección de ayuda) - Cuadros de selección o diálogo para tipo de linea - Objeto ratón
Indice de menú: Grosor de linea	Ventana (1) (de selección de ayuda) - Cuadros de selección o diálogo para tipo de grosor - Objeto ratón
Indice de menú: Relleno	Ventana (1) (de selección de ayuda) - Cuadros de selección o diálogo para tipo de grosor - Objeto ratón
Menú: AYUDA	Ventana (Menú colgante) - Objeto ratón
Indice de menú: Acerca de...?	Ventana (1) (de selección de ayuda) - Cuadros de selección o diálogo - Objeto ratón Ventana (2) (para proporcionar la ayuda) - Objetos texto - Botones - Objeto ratón

CAPITULO 7. IMPLEMENTACION DEL PROGRAMA DE APLICACION

7.1. IMPLEMENTACION DE LOS MENUS

7.1.1. Menú de *Escritorio*

La función *Presentacion*, crea una ventana con los datos de referencia del programa de aplicación, como son el nombre y número de versión; la función es usada por el menú de *Escritorio*. *Presentacion* utiliza objetos gráficos de tipo: texto, ventana, botón, mapa de bits. El código para esta función, así como la salida que produce se muestran a continuación:

```
int Presentacion(void)
{       static char in[3][36] =    { "  FACULTAD DE INGENIERIA",
                                   "  HERRAMIENTA DE DIBUJO",
                                                "          Versión 1.0"   };
        union REGS m;
        Raton R(m);
        OBJ_TEXTO tx[3];
        BOTON ok,*bp;
        VENTANA w;
        RECUADRO r;
        LOCALIZA p;
        MAPABITS bmp;
        unsigned int rt;
        int a=0xff;
        static int i;
        Obj_Texto *TXT[3];
        Ventana V(&w,&r,30,36,280,200);

       // Intenta abrir la ventana
       if(V.AbreVentana()) {

              // Agrega el icono a la ventana de presentación
              MapaBits BMP(ICONO11,0x00,NULL,INACTIVO,&bmp,&w,&bmp.rec,
              r.izq+((r.der-r.izq)/2)-(AnchoImagen(ICONO11)/2),r.arriba+96);
              BMP.Dib_Map_Bits();

              // Agrega el botón de Aceptar
              Boton BOT("Aceptar",ACTIVO,&ok,&ok.tb,&w,
              &ok.rec,r.der-85,r.abajo-23,r.der-15,r.abajo-8);
              BOT.DibujaBoton();

              // Agrega el arreglo de cadenas de texto a la ventana
              for(i=0;i<3;++i)
              { TXT[i]=new Obj_Texto(in[i],ACTIVO,&tx[i],&tx[i].tx,&w,
                r.izq+((r.der-r.izq)/2)-((strlen(in[i])*8)/2),
                r.arriba+16+(i*20));
                TXT[i]->DibujaTexto();
              }
```

```
        // Detecta que ocurra alguna actividad en la ventana actual
        while(a) {
                ActivaTeclado(&w);
                if(R.Det_Fuera_Raton(&p)) {
                        rt=Loc_Pos_Raton(&p,&w);
                        if(rt & EnBoton) {
                                bp=Loc_Boton(&w,&p);
                                Sel_Boton(bp);
                                if(bp==&ok) a=0;
                        }
                        else MensajeError();
                }
        }
        V.CierraVentana();
    } else MensajeError();
        // Borra los objetos dinámicos
        delete TXT[0];
        delete TXT[1];
        return(1);
}
```

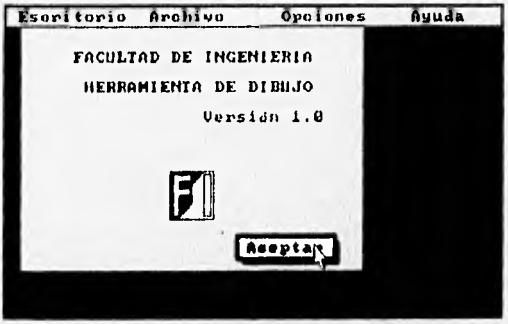


Fig. 7.1. Salida producida por la función *Presentacion*.

7.1.2. Menú de *Archivo*

7.1.2.1. Código para abrir un nuevo archivo PCX

La función *Arch_Nuevo* crea un nuevo archivo PCX, a través de un cuadro de diálogo solicita las dimensiones para el nuevo archivo; las dimensiones por *default* son 640 x 480. Esta función verifica también, que las dimensiones dadas sean las correctas y que exista la memoria necesaria para crear el archivo. Posteriormente, utiliza la función *AbreArchivo* para abrir el archivo nuevo, aparecerá una área en color blanco (área o ventana de trabajo) y habilitará las herramientas de dibujo para poder editarlo.

```
int Arch_Nuevo(void)
{
        union REGS in;
        Raton R(in);
        VENTANA w;
        LOCALIZA p;
        BOTON ok,can,*bp;
        OBJ_TEXTO tx,vertx,hortx;
        CAMPOEDICION ver,hor,*ep;
        RECUADRO r;
        char verstr[8],horstr[8];
        unsigned int rt,a=0xff;

        // Delecta si hay un archivo abierto
        if(Arch_Resid) return(0);

        Ventana V(&w,&r,48,48,270,140);

        // Abre la ventana
        if(V.AbreVentana()) {

                // Coloca las dimensiones por default
                ancho=Configuracion.anchodefault; // por default es 480
                largo=Configuracion.largodefault;    // por default se definió como 640

                // Los coloca en los buffers
                sprintf(horstr,"%d",ancho);
                sprintf(verstr,"%d",largo);

                // Añade un titulo
                Obj_Texto TXT1("Archivo nuevo",ACTIVO,&tx,&tx.tx,&w,r.izq+8,r.arriba+10);
                TXT1.DibujaTexto();
                // Agrega los nombres en el campo de edición
                Obj_Texto TXT2("Dim: Horizontal",ACTIVO,&hortx,&hortx.tx,&w,r.izq+8,r.arriba+30);
                TXT2.DibujaTexto();
                Obj_Texto TXT3("Dim: Vertical",ACTIVO,&vertx,&vertx.tx,&w,r.izq+8,r.arriba+46);
                TXT3.DibujaTexto();

                // Agrega los campos para edición
                CampoEdicion CED(horstr,ACTIVO,5,/*Variable*/NULL,
                &hor,&hor.tce,&w,&hor.rec,r.izq+160,r.arriba+30);
                CED.DibCampoEdicion();
                CampoEdicion CED2(verstr,ACTIVO,5,/*Variable*/NULL,
                &ver,&ver.tce,&w,&ver.rec,r.izq+160,r.arriba+46);
                CED2.DibCampoEdicion();

                // Agrega el botón para aceptar
                Boton BOT("Aceptar",ACTIVO,&ok,&ok.tb,&w,
                &ok.rec,r.der-180,r.abajo-23,r.der-105,r.abajo-8);
                BOT.DibujaBoton();

                // Agrega el botón de cancelar
                Boton BOT2("Cancelar",ACTIVO,&can,&can.tb,
                &w,&can.rec,r.der-90,r.abajo-23,r.der-10,r.abajo-8);
                BOT2.DibujaBoton();
```

```
        // Detecta alguna actividad
        while(a) {
                ActivaTeclado(&w);
                if(R.Det_Fuera_Raton(&p)) {
                        rt=Loc_Pos_Raton(&p,&w);

                        if(rt & EnCampoEdicion) {
                                ep=Loc_Camp_Ed(&w,&p);
                                Sel_Camp_Ed(ep);
                                RedibCampoEdicion(ep);
                        }
                        else if(rt & EnBoton) {
                                bp=Loc_Boton(&w,&p);
                                Sel_Boton(bp);
                                if(bp==&ok) a=0;
                                else if(bp==&can) a=0;
                        }
                        else MensajeError();
                }
        }
        V.CierraVentana();
        // Detecta si ha sido localizado el BOTON de aceptar
        if(bp==&ok) {
                // Redondea el ancho al byte más próximo
                ancho=pixels2bytes(atoi(horstr))<<3;
                // Obtiene la dimensión de largo
                largo=atoi(verstr);

                // checa los límites
                if(ancho > 31 && ancho < 32767 && largo > 31 && largo < 32767) {
                        bytes=pixels2bytes(ancho);

                        // Detecta si el dibujo permanece en memoria
                        if(Obt_Buffer((long)bytes*(long)largo)) {

                                // Carga el dibujo
                                if(AbreArchivo()) {
                                        Arch_Resid=0xff;
                                        nomb_arch[0]=0;
                                        LimpiaBuffer();
                                        Coloca_Frag_Img();
                                }
                                else {
                                        Cerrar();
                                        Desp_Mens("Error de asignación de memoria");
                                }
                        } else Desp_Mens("Error de asignación en el buffer");
                } else Desp_Mens("Error de asignación de valores");
        }
    } else MensajeError();
    return(1);
}
```

Fig. 7.2. Solicita las coordenadas para el nuevo archivo.

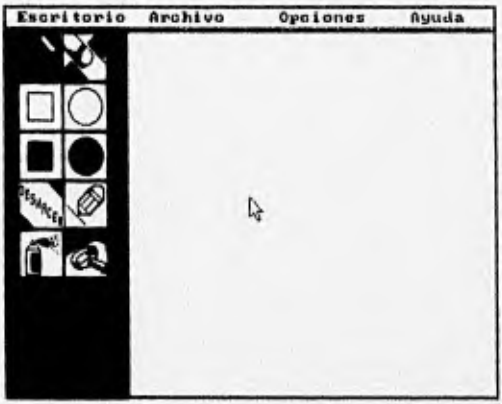


Fig. 7.3. Coloca el área de trabajo para la edición del archivo PCX.

7.1.2.2. Código para manejar los archivos PCX

El encabezado de un archivo PCX es de 128 bytes de longitud y puede ser expresado en una variable *struct*, como se muestra a continuación:

```
// struct para datos de un archivo PCX
typedef struct  {
        char fabricante;
char version
char Codificacion
char bitsPorPixel
int xmin,ymin                   // Coordenadas iniciales
int xmax,ymax                   // Coordenadas máximas
int Res_Hor                      // Resolución horizontal
int Res_Vert                     // Resolución vertical
char paleta[48]
```

```
char Der_Res            // Derechos de autor del archivo
char PlanosColor
int bitsPorLinea
int TipoDePaleta
char Relleno[58]
} DATOSPCX
```

donde:
fabricante: tiene un valor constante de diez y sirve para reconocer que se trata de un archivo PCX.
version y *codificacion:* indican al programa decodificador como manejar la paleta de colores del dibujo (el esquema de colores.)
bitsPorPixel y PlanosColor: juntos definen cuantos colores forman parte del dibujo.
TipoDePaleta: indica la paleta de colores utilizada por el archivo PCX.
Res_Hor y Res_Ver: definen la resolución vertical y horizontal del archivo PCX.
xmin y *xmax*: definen el ancho de la archivo de imagen PCX, es igual a *xmax - xmin*.
ymin y *ymax*: definen la dimensión vertical del archivo PCX, es igual *ymax - ymin*.
bytes_por_linea: es el valor que especifica el largo de una línea de la imagen, cuando cada línea del archivo es descomprimida.

La función *Lee_Lin_ArchPCX*, lee y decodifica una línea del archivo PCX hacia la variable *char p*. El código para leer un archivo PCX desempaca un campo y lee el primer byte de dicho campo, para verificar si sus dos bits superiores son iguales a uno, si no lo son, este byte es escrito directamente al *buffer* de la pantalla (el campo en cuestión es de un byte de longitud, el campo cadena.) Si los dos bits superiores son iguales a uno, este byte corresponde a un campo duplicado, los dos bits superiores son eliminados y el valor resultante (contador) es almacenado. El siguiente byte es leído y escrito hacia el *buffer* de salida, el número de veces que especifique el contador de repeticiones, la variable "*y*", en este caso. Una vez que el campo ha sido leído, se debe verificar si ha sido desempacada una línea completa (número de bytes por línea.)

```
int Lee_Lin_ArchPCX(char *p,FILE *fp,int bytes)
{       int n=0,c,i;

        memset(p,0,bytes);
        do {
                c=fgetc(fp) & 0xff;             // Lee el archivo PCX
                if((c & 0xc0) == 0xc0) {        // Verifica si los dos bits superiores son igual a 1
                        i=c & 0x3f;             // Indexa la variable c con 3F
                        c=fgetc(fp);
                        while(i--) p[n++]=c;    // Escribe al buffer el número de veces que indica i
                }
                else p[n++]=c;                  // Se escribe directamente al buffer
        } while(n < bytes);                     // Hace esto hasta que termine de leer la linea
        return(n);
}
```

7.1.2.3. Código para abrir un archivo PCX existente

La función *Abrir*, es usada para abrir un archivo en formato PCX contenido en algunas de las unidades de disco; la función detecta si es un archivo PCX válido y si hay memoria suficiente para cargarlo, si cumple con estas dos condiciones, es abierto para ser editado con las herramientas de dibujo.

```
int Abrir(void)
{
        char b[129],nombre[16];
        char drive[MAXDRIVE],dir[MAXDIR],ext[MAXEXT];

        // Detecta si ya se encuentra abierto otro archivo
        if(Arch_Resid) return(0);

        // Construye la ruta del archivo
        strcpy(b,ruta_arch);
        strcat(b,"*.*");
        strcat(b,"PCX");
        setwritemode(COPY_PUT);

        //Obtiene el nombre del archivo
        if(Sel_Archivo(b,nombre,48,48,Configuracion.Unidades)) {

                // reconstruye la ruta del archivo
                fnsplit(b,drive,dir,NULL,NULL);
                fnsplit(nombre,NULL,NULL,nomb_arch,ext);
                fnmerge(b,drive,dir,nomb_arch,ext);

                fnmerge(ruta_arch,drive,dir,NULL,NULL);

                //Carga el archivo
                if(CargaArchivo(b,Dib_Linea)) {

                        //Abre el archivo
                        if(AbreArchivo()) {
                                Arch_Resid=0xff; //indica que hay un archivo en memoria
                                Coloca_Frag_Img();
                        }
                        else {
                                Cerrar();
                                Desp_Mens("Error de asignación de memoria");
                        }
                } else Desp_Mens("Error al cargar el archivo");
        }
        return(1);
}
```

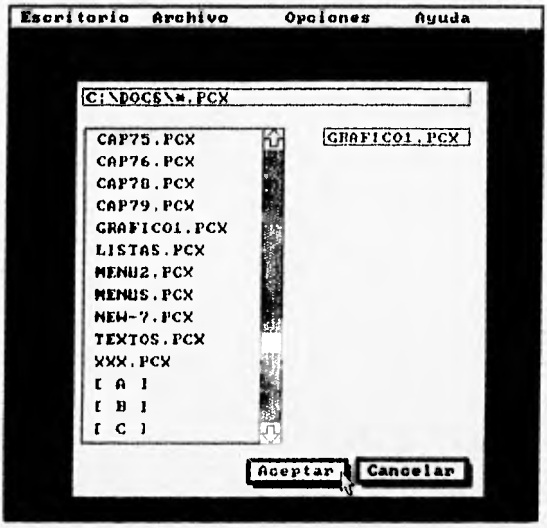


Fig. 7.4. Se indica el archivo que quiere abrir .

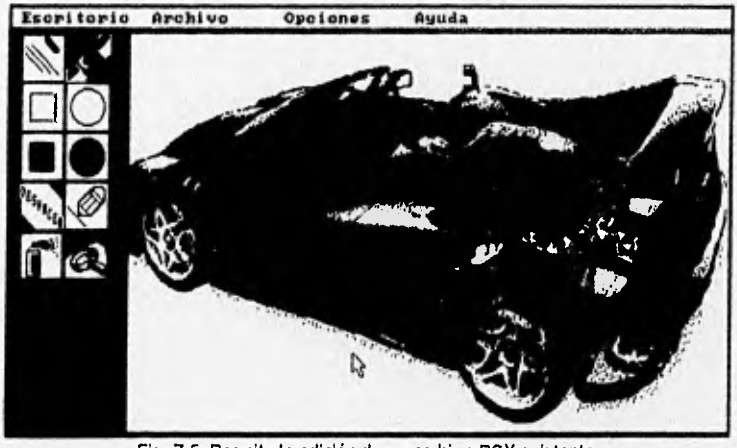


Fig. 7.5. Permite la edición de un archivo PCX existente.

7.1.2.4. Código para crear las herramientas de dibujo

La función *Dibujar*, es llamada cada vez que el ratón es presionado sobre los iconos de herramientas de dibujo; si son seleccionados dichos iconos, se pueden dibujar líneas, cuadros, círculos, puntos, realizar una ampliación, borrar y deshacer alguna operación realizada anteriormente.

```
Dibujar(LOCALIZA *inicio,int Det_Activ)
{
        union REGS m;
        Raton R(m);
        LOCALIZA p,p2,p3;
        unsigned int i,n;

        switch(Actual_Herr) {
                case Linea:
                        //Actualiza los cambios hechos en el buffer
                        if(Cambios) Obt_Frag_Img();
                        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
                        setcolor(getmaxcolor());
                        setwritemode(XOR_PUT);

                        //Mantiene el inicio dentro del área de dibujo
                        Def_Lim_Recuad(inicio,&dib_area);

                        // Si el ratón se encuentra en el Icono de linea, la dibuja
                        // desde el Inicio, a la posición actual del ratón
                        while(R.Det_Fuera_Raton(&p)) {
                                Def_Lim_Recuad(&p,&dib_area);
                                R.DesactivaRaton();
                                line(inicio->x,inicio->y,p.x,p.y);
                                R.Activa_Raton();
                                do {
                                        i=R.Det_Fuera_Raton(&p2);
                                        Def_Lim_Recuad(&p2,&dib_area);
                                } while(!i && p.x==p2.x && p.y==p2.y);
                                R.DesactivaRaton();
                                line(inicio->x,inicio->y,p.x,p.y);
                                R.Activa_Raton();
                        }

                        //Se verifica que se encuentre en el área de dibujo
                        Def_Lim_Recuad(&p,&dib_area);

                        // Coloca el tipo de estilo y grosor especificados desde el menú de Opciones
                        setlinestyle(estilo,0,grosor);
                        if(Det_Activ & 0x0001) setcolor(BLACK);
                        else setcolor(getmaxcolor());
                        setwritemode(COPY_PUT);
                        R.DesactivaRaton();
                        line(inicio->x,inicio->y,p.x,p.y);
                        R.Activa_Raton();
```

```
            //Indica que el dibujo ha sido cambiado
            Cambios=0xff;
            break;

    case Borrar:
            //Actualiza el dibujo que se encuentra en el buffer
            if(Cambios) Obt_Frag_Img();


            // Coloca el borrador
            if(Det_Activ & 0x0001) n=Configuracion.borramin;
            else n=Configuracion.borramax;
            setlinestyle(SOLID_LINE,0,NORM_WIDTH);
            setfillstyle(SOLID_FILL,getmaxcolor());
            setcolor(getmaxcolor());
            setwritemode(XOR_PUT);

            // Se verifica que el inicio se encuentre en el área de dibujo
            Def_Lim_Recuad(inicio,&dib_area);
            R.DesactivaRaton();

            // Si el ratón se encuentra en el icono para borrar, lo borra
            // desde el inicio, a la posición actual del ratón, con un cursor negro
            while(R.Det_Fuera_Raton(&p)) {
                    Def_Lim_Recuad(&p,&dib_area);
                    p3.x=p.x+n;
                    p3.y=p.y+n;
                    Def_Lim_Recuad(&p3,&dib_area);
                    bar(p.x,p.y,p3.x,p3.y);
                    rectangle(p.x,p.y,p3.x,p3.y);
                    do {
                            i=R.Det_Fuera_Raton(&p2);
                            Def_Lim_Recuad(&p2,&dib_area);
                    } while(i && p.x==p2.x && p.y==p2.y);
                    rectangle(p.x,p.y,p3.x,p3.y);
            }
            R.Activa_Raton();

            //Indica que el dibujo ha sido cambiado
            Cambios=0xff;
            break;
    case Cuadro:
            //Actualiza los cambios hechos en el buffer que almacena el dibujo
            if(Cambios) Obt_Frag_Img();
            setlinestyle(SOLID_LINE,0,NORM_WIDTH);
            setcolor(getmaxcolor());
            setwritemode(XOR_PUT);

            //Se verifica que el inicio se encuentre en el área de dibujo
            Def_Lim_Recuad(inicio,&dib_area);

        // Mientras el ratón seleccione esta opción, dibuja un recuadro
        // desde el inicio a la posición actual del ratón
            while(R.Det_Fuera_Raton(&p)) {
                    Def_Lim_Recuad(&p,&dib_area);
```

```
                R.DesactivaRaton();
                if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y))
                  rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
                do {
                        i=R.Det_Fuera_Raton(&p2);
                        Def_Lim_Recuad(&p2,&dib_area);
                } while(i && p.x==p2.x && p.y==p2.y);
                R.DesactivaRaton();
        if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
        }
        // Detecta que haya algún desplazamiento
        if(!Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) return(0);

        // Verifica que se encuentre en el área de trabajo
        Def_Lim_Recuad(&p,&dib_area);
        DefineRecta(inicio,&p);

        // Coloca el estilo y grosor especificados
        // desde el manú de Opciones
        setlinestyle(estilo,0,grosor);
        if(Det_Activ & 0x0001) setcolor(BLACK);
        else setcolor(getmaxcolor());
        setwritemode(COPY_PUT);
        R.DesactivaRaton();
        rectangle(inicio->x,inicio->y,p.x,p.y);
        R.Activa_Raton();
        // indica si el dibujo ha sufrido modificaciones
        Cambios=0xff;
        break;

case Elipse:
        // Actualiza el buffer del dibujo
        if(Cambios) Obt_Frag_Img();

        // Dibuja un recuadro XOR
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        setcolor(getmaxcolor());
        setwritemode(XOR_PUT);

        // Verifica que se encuentre en el área de dibujo
        Def_Lim_Recuad(inicio,&dib_area);

        // Mientras el ratón seleccione esta opción, dibuja un recuadro,
        // desde el inicio, a la posición actual del ratón
        while(R.Det_Fuera_Raton(&p)) {
                Def_Lim_Recuad(&p,&dib_area);
                R.DesactivaRaton();
                if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y))
                  rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
                do {
                        i=R.Det_Fuera_Raton(&p2);
                        Def_Lim_Recuad(&p2,&dib_area);
```

```
                } while(i && p.x==p2.x && p.y==p2.y);
                R.DesactivaRaton();
        if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
        }

        // Indica si hubo algún desplazamiento
        if(!Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) return(0);

        // Verifica que se encuentre en el área de dibujo
        Def_Lim_Recuad(&p,&dib_area);
        DefineRecta(inicio,&p);

        // Mientras el ratón seleccione esta opción, dibuja una elipse
        // desde el inicio a la posición actual del ratón
        setlinestyle(estilo,0,grosor); // Coloca el tipo de estilo y grosor, especificados
        // desde el menú de Opciones
        if(Det_Activ & 0x0001) setcolor(BLACK);
        else setcolor(getmaxcolor());
        setwritemode(COPY_PUT);
        R.DesactivaRaton();

        // Dibuja la elipse
        ellipse(inicio->x+((p.x-inicio->x)/2),inicio->y+((p.y-inicio->y)/2),0,360,
        (p.x-inicio->x)/2,(p.y-inicio->y)/2);
        R.Activa_Raton();

        // Indica si han ocurrido cambios en el dibujo
        Cambios=0xff;
        break;
case CuadRell:
        //Actualiza los cambios hechos en el buffer que almacena el dibujo
        if(Cambios) Obt_Frag_Img();
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        setcolor(getmaxcolor());
        setwritemode(XOR_PUT);
        setfillstyle(relleno,WHITE);

        //Verifica que el inicio se encuentre en el área de dibujo
        Def_Lim_Recuad(inicio,&dib_area);

      // Mientras el ratón seleccione esta opción, dibuja un recuadro
      // desde el inicio a la posición actual del ratón
        while(R.Det_Fuera_Raton(&p)) {
                Def_Lim_Recuad(&p,&dib_area);
                R.DesactivaRaton();
                if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y))
                  rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
                do {
                        i=R.Det_Fuera_Raton(&p2);
                        Def_Lim_Recuad(&p2,&dib_area);
                } while(i && p.x==p2.x && p.y==p2.y);
                R.DesactivaRaton();
```

```
        if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
        }

        // Detecta que haya algún desplazamiento
        if(!Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) return(0);
        // Verifica que se encuentre en el área de trabajo
        Def_Lim_Recuad(&p,&dib_area);
        DefineRecta(inicio,&p);

        // Dibuja un recuadro con el relleno especificado
        // anteriormente o el proporcionado en el menú de Opciones
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        if(Det_Activ & 0x0001) setcolor(BLACK);
        else setcolor(getmaxcolor());
        setwritemode(COPY_PUT);
        R.DesactivaRaton();
        bar(inicio->x,inicio->y,p.x,p.y);
        R.Activa_Raton();

        // Indica si el dibujo ha sufrido modificaciones
        Cambios=0xff;
        break;
case ElipsRell:
        // Actualiza el buffer del dibujo
        if(Cambios) Obt_Frag_Img();

        // Dibuja un recuadro XOR
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        setcolor(getmaxcolor());
        setwritemode(XOR_PUT);
        setfillstyle(relleno,WHITE);

        // Verifica que este en el área de dibujo
        Def_Lim_Recuad(inicio,&dib_area);

        // Mientras el ratón seleccione esta opción, dibuja un recuadro que contendrá
        // a la elipse, desde el inicio, a la posición actual del ratón
        while(R.Det_Fuera_Raton(&p)) {
                Def_Lim_Recuad(&p,&dib_area);
                R.DesactivaRaton();
                if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y))
                rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
                do {
                        i=R.Det_Fuera_Raton(&p2);
                        Def_Lim_Recuad(&p2,&dib_area);
                } while(i && p.x==p2.x && p.y==p2.y);
                R.DesactivaRaton();
                if(Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y))
                 rectangle(inicio->x,inicio->y,p.x,p.y);
                R.Activa_Raton();
        }
```

```
        // Indica si hubo algún desplazamiento
        if(!Checa_Vol_Rect(inicio->x,inicio->y,p.x,p.y)) return(0);

        // Verifica que se encuentre en el área de dibujo
        Def_Lim_Recuad(&p,&dib_area);
        DefineRecta(inicio,&p);

        // Mientras el ratón seleccione esta opción, dibuja una elipse
        // desde el inicio, a la posición actual del ratón
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        if(Det_Activ & 0x0001) setcolor(BLACK);
        else setcolor(getmaxcolor());
        setwritemode(COPY_PUT);
        R.DesactivaRaton();
        fillellipse(inicio->x+((p.x-inicio->x)/2),inicio->y+((p.y-inicio->y)/2),
           (p.x-inicio->x)/2,(p.y-inicio->y)/2);
        R.Activa_Raton();

        // Indica si han ocurrido cambios en el dibujo
        Cambios=0xff;
        break;

case Deshace:
        // Esta opción es para deshacer el último cambio realizado
        break;

case Pintar:
        // Actualiza el buffer que contiene el dibujo
        if(Cambios) Obt_Frag_Img();

        // Verifica que el inicio se encuentre en el área de dibujo
        Def_Lim_Recuad(inicio,&dib_area);

        // Se dibuja una línea
        setlinestyle(estilo,0,grosor);
        if(Det_Activ & 0x0001) setcolor(BLACK);
        else setcolor(getmaxcolor());
        setwritemode(COPY_PUT);

        // Cuando se esta seleccionando esta opción, se dibuja una línea
        while(R.Det_Fuera_Raton(&p)) {
                Def_Lim_Recuad(&p,&dib_area);
                do {
                        i=R.Det_Fuera_Raton(&p2);
                        Def_Lim_Recuad(&p2,&dib_area);
                } while(i && p.x==p2.x && p.y==p2.y);
                R.DesactivaRaton();
                line(p.x,p.y,p2.x,p2.y);
                R.Activa_Raton();
        }
        // Indica si han ocurrido cambios en el dibujo
        Cambios=0xff;
        break;
```

```
        case Spray:
                //Actualiza el dibujo que se encuentra en el buffer
                if(Cambios) Obt_Frag_Img();
                setfillstyle(INTERLEAVE_FILL,getmaxcolor());
                setwritemode(XOR_PUT);
                // Verifica que el inicio se encuentre en el área de dibujo
                Def_Lim_Recuad(inicio,&dib_area);
                R.DesactivaRaton();
                // Verifica los límites para el área donde es aplicado el spray, que estén dentro
                // del área de dibujo
                while(R.Det_Fuera_Raton(&p)) {
                        Def_Lim_Recuad(&p,&dib_area);
                        p3.x=p.x+16;
                        p3.y=p.y+16;
                        Def_Lim_Recuad(&p3,&dib_area);
                        do {
                                i=R.Det_Fuera_Raton(&p2);
                                Def_Lim_Recuad(&p2,&dib_area);
                        } while(i && p.x==p2.x && p.y==p2.y);
                        pieslice(p3.x-8,p3.y-8,0,360,8);
                }
                R.Activa_Raton();
                //Indica que el dibujo ha sido cambiado
                Cambios=0xff;
                break;

        case Zoom:
                //Actualiza el dibujo que se encuentra en el buffer
                if(Cambios) Obt_Frag_Img();
                setfillstyle(CLOSE_DOT_FILL,getmaxcolor());
                setwritemode(XOR_PUT);
                //Verifica que el inicio se encuentre en el área de dibujo
                Def_Lim_Recuad(inicio,&dib_area);
                R.DesactivaRaton();

                // Verifica los límites para el área que será ampliada, que estén dentro del área
                // de dibujo
                while(R.Det_Fuera_Raton(&p)) {
                        Def_Lim_Recuad(&p,&dib_area);
                        p3.x=p.x+31;
                        p3.y=p.y+31;
                        Def_Lim_Recuad(&p3,&dib_area);
                        do {
                                i=R.Det_Fuera_Raton(&p2);
                                Def_Lim_Recuad(&p2,&dib_area);
                        } while(i && p.x==p2.x && p.y==p2.y);
                }
                R.Activa_Raton();
                Ampliacion(p3.x-31,p3.y-31);
                //Indica que el dibujo ha sido cambiado
                Cambios=0xff;
                break;
    }
    return(1);
}
```

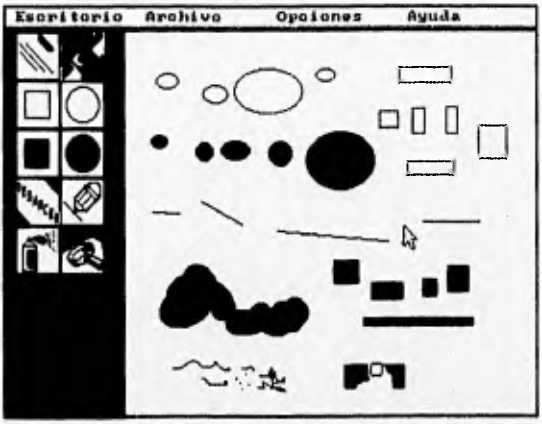


Fig. 7.6. La función *Dibujar* permite utilizar las herramientas de dibujo.

7.1.2.5. Código para cerrar un archivo PCX

La función *Cerrar*, permite cerrar un archivo PCX previamente abierto, envía un mensaje al usuario para indicarle si quiere continuar con esta operación y libera el *buffer* donde almacenaba el archivo. Utiliza objetos gráficos de tipo: texto, mapa de bits y botones.

```
int Cerrar(void)
{       // Si no hay dibujo residente, no hay nada que cerrar
        if(!Arch_Resid) return(0);
        if (Rec_Dialogo(" "Cerrar sin salvar...?","") == 0) return(0);
        // Borra el área de dibujo y los iconos de herramientas
        ClearScreen();
        Pantalla.apunta.sig=NULL;

        // Libera el buffer que contiene el dibujo
        LiberaBuffer();
        if(lineabuf != NULL) free(lineabuf);
        lineabuf=NULL;

        // Indica que no hay un archivo residente en memoria
        Arch_Resid=0x00;
        // Coloca las opciones del menú de Archivo
        archivo.indice[0].nombre[0]=' ';    //Activa la opción de "Nuevo"
        archivo.indice[1].nombre[0]=' ';    //Activa la opción de "Abrir"
        archivo.indice[2].nombre[0]='.';    //Desactiva la opción de "Cerrar"
        archivo.indice[3].nombre[0]='.';    //Desactiva la opción de "Salvar"
        archivo.indice[4].nombre[0]='.';    //Desactiva la opción de "Salvar como..."

        // Coloca las opciones del menú de Opciones
        opciones.indice[0].nombre[0]='.';  // Activa la opción de tipo de línea
        opciones.indice[1].nombre[0]='.';  // Activa la opción de grosor de línea
        opciones.indice[2].nombre[0]='.';  // Activa la opción de tipo de relleno
```

```
        // Coloca los iconos de herramientas para el siguiente dibujo
        Ult_Herr=0;
        Actual_Herr=-1;
        return(1);
}
```

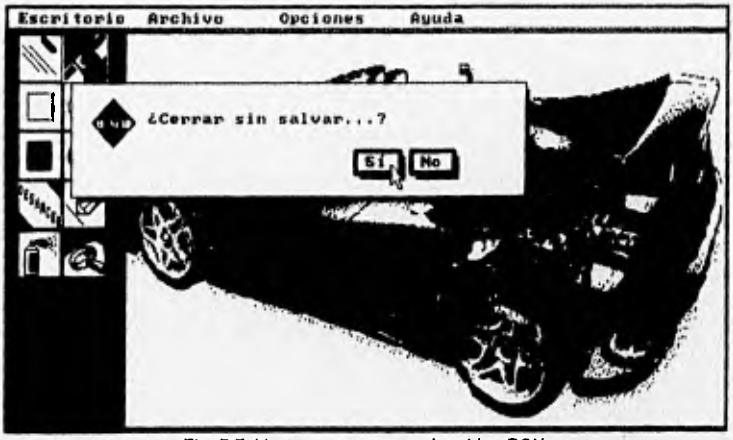


Fig. 7.7. Ventana para cerrar el archivo PCX.

7.1.2.6. Código para salvar un archivo PCX

La función Salvar, salva un archivo en una unidad de disco, como un archivo en formato **PCX** monocromático. *Salvar* verifica si el nombre del archivo ya existía previamente; si existe lo sobrescribe; si no, abre un cuadro de diálogo para salvarlo con el nombre y en la unidad de disco que le especifiquen (se trata de un archivo nuevo.)

```
int Salvar(void)
{
        INFOARCH fi;
        char b[129],fn[16];

        // No salva el archivo, si este no existe
        if(!Arch_Resid) return(0);

        // Si el dibujo ha sufrido cambios, actualiza el dibujo antes de salvarlo
        if(Cambios) {
                Obt_Frag_Img();
                Cambios=0x00;
        }
```

```
// Verifica que haya un nombre de archivo para salvarlo
if(!strlen(nomb_arch)) {
        Desp_Mens("- Archivo sin nombre...!");
        return(0);
}

// Construye la ruta y el nombre del archivo
strcpy(b,ruta_arch);
strcat(b,nomb_arch);
strcat(b,".");
strcat(b,"PCX");
strcpy(fn,nomb_arch);
strcat(fn,".");
strcat(fn,"PCX");

// Inicializa la variable struct INFOARCH, para almacenar la información del archivo
fi.ancho=ancho;
fi.largo=largo;
fi.bytes=bytes;
fi.bitsPorPixel=1;
fi.memoria=(long)bytes*(long)largo;

// Intente salvar el archivo
if(access(b,0)==0) {  // Verifica si ya existe el archivo
        if(Rec_Dialogo("¨Sobreescribo el archivo existente?",fn)) Salvar_Arch(b,&fi,Obt_Linea);
} else Salvar_Arch(b,&fi,Obt_Linea);
return(1);
}
```

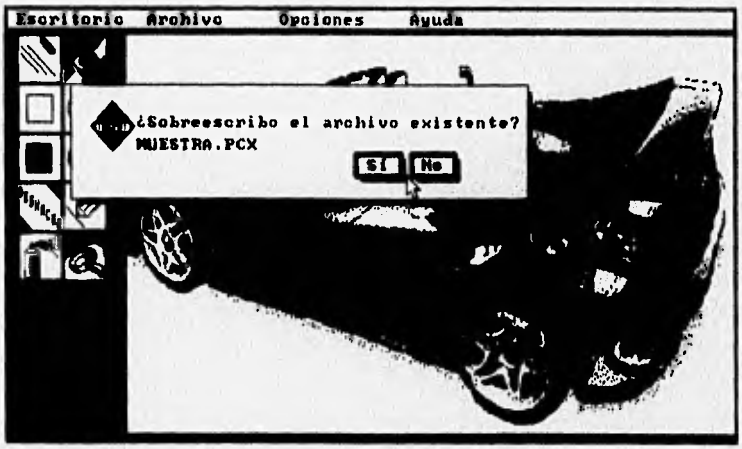


Fig. 7.8. Ventana para salvar el archivo PCX

7.1.2.7. Código para salvar un archivo PCX con un nombre diferente

La función *SalvarComo* permite salvar el archivo PCX, que esta siendo editado, con un nuevo nombre; llama a la función *Salvar* para guardar el dibujo con el nombre del nuevo archivo en la unidad de disco y directorio especificados.

```
int SalvarComo(void)
{
        char b[129],nombre[16];
        char drive[MAXDRIVE],dir[MAXDIR];

        // No salva el archivo, si este no existe
        if(!Arch_Resid) return(0);

        // Se crea el archivo especificado
        strcpy(b,ruta_arch);
        strcat(b,"*.");
        strcat(b,"PCX");

        // Se llama a la función Sel_Archivo
        if(Sel_Archivo(b,nombre,48,48,Configuracion.Unidades)) {

                // Reconstruye la ruta del archivo
                fnsplit(b,drive,dir,NULL,NULL);
                fnsplit(nombre,NULL,NULL,nomb_arch,NULL);
                fnmerge(ruta_arch,drive,dir,NULL,NULL);

                // Salva el archivo
                Salvar();
        }
        return(1);

}
```

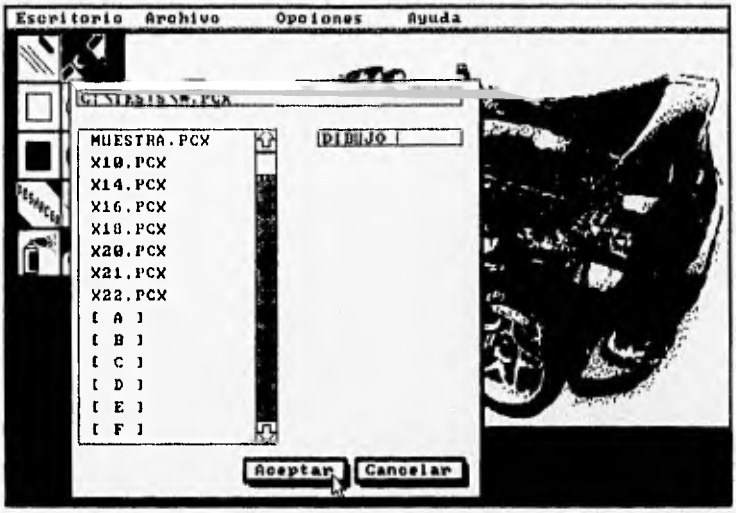


Fig. 7.9. Ventana para salvar el archivo con un nombre diferente

7.1.2.8. Código para salir del programa de aplicación

La función *Salir*, permite salir del programa de aplicación. Crea un cuadro de diálogo para prevenir al usuario y para confirmar su salida del programa. El código para esta función se muestra a continuación:

```
int Salir(void)
{
        if(Rec_Dialogo("Quieres salir ?","")) actividad=0;
    return(1);
}
```

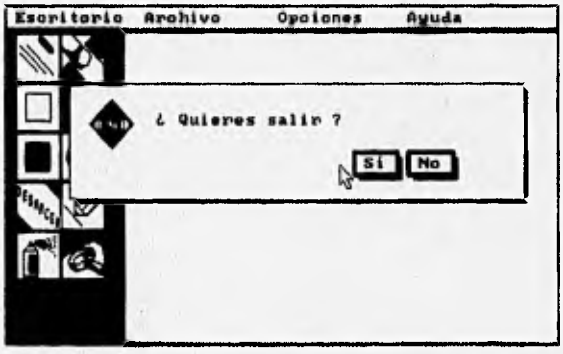


Fig. 7.10. Cuadro de diálogo para indicar si se quiere salir del programa.

7.1.3. Menú de *Opciones*

El menú de *Opciones* funciona en forma conjunta con el menú de *Archivo*, este menú permitirá colocar el grosor y estilo para las líneas; así como, el tipo de relleno para las figuras sólidas (rectángulo y elipses con relleno.)

La función *Grosor*, permite definir un grosor de la línea, de dos tipos de grosor posibles; utiliza objetos gráficos de tipo ventana, de tipo campo de texto, de tipo cuadro de selección y asociado a éste, se dibuja una línea ejemplificando el tipo de grosor que puede ser utilizado.

```
int Grosor(void)
{   // Definición de los objetos gráficos y variables locales utilizadas por la función
        VENTANA W;
        RECUADRO r;
        MAPABITS bmp;
        CUADDIAL cd[2],*ap;
        LOCALIZA p;
        static int i;
        unsigned int rt;
        int a=0xff;
        Cuad_Dial *C_D[2];
        union REGS m;
        Raton R(m);
        OBJ_TEXTO tx[4];

        // Titulo de la ventana
        char in[2][2] = { "", "" };
        char *in2[] =   { "      Grosor      ",
                          "                  ",
                        };
        Obj_Texto *TXT[4];
        Ventana VENT(&W,&r,200,13,370,90);

           R.Activa_Raton();
           if (VENT.AbreVentana()) {
               R.DesactivaRaton();

               // Dibuja los tipos de grosores
            for(i=0;i<2;i++) {
                C_D[i]=new Cuad_Dial(in[i],0x00,ACTIVO,&cd[i],&cd[i].tcd,
                &W,&cd[i].rec,r.izq+150,r.arriba+30+(i*20));
                C_D[i]->Dib_Cuad_Dial();
                TXT[i]=new Obj_Texto(in2[i],ACTIVO,&tx[i],&tx[i].tx,&W,
                r.izq+10,r.arriba+6+(i*10));
                TXT[i]->DibujaTexto();
                setlinestyle(SOLID_LINE,0,i*3);
                setcolor(getmaxcolor());
                setwritemode(XOR_PUT);
                line(r.izq+10,r.arriba+35+(i*20),r.izq+130,r.arriba+35+(i*20));
               }
           R.Activa_Raton();
           while (a) {
```

```
                ActivaTeclado(&W);
                if (R.Det_Fuera_Raton(&p)) {
                  rt=Loc_Pos_Raton(&p,&W);
                  if (rt & EnCuadDial) {
                    ap=Loc_Cuad_Dial(&W,&p);
                    Sel_Cuad_Dial(ap);
                    R.DesactivaRaton();

                // Maneja la selección de grosor
                    if (ap==&cd[0]) {
                      cd[0].sel=0xff;  C_D[0]->Dib_Cuad_Dial();    grosor=1;  }
                        else  if(ap==&cd[1]) {
                          cd[1].sel=0xff;  C_D[1]->Dib_Cuad_Dial(); grosor=3;  }
                            R.Activa_Raton();
                            delay(400);
                            a=0;
                           } else MensajeError();
                         }
                    // Borra los objetos creados dinámicamente
                    for(i=0;i<2;i++) {
                    delete C_D[i]; delete TXT[i];}
                    VENT.CierraVentana();
                  } else MensajeError();
        return(1);
}
```

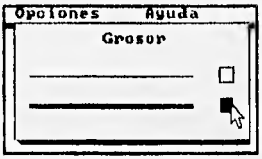


Fig. 7.11. Salida generada por la función *Grosor*.

La función *estilo_linea*, permite definir el estilo de línea; hay cuatro tipos de línea disponibles, la función *estilo_linea* utiliza una ventana, un objeto de tipo campo de texto, un objeto de tipo cuadro de selección, y asociado a éste, es dibujada una línea que muestra el estilo que puede ser seleccionado.

```
int estilo_linea(void)
{
        VENTANA W;
        RECUADRO r;
        MAPABITS bmp;
        CUADDIAL cd[4],*ap;
        LOCALIZA p;
        static int j;
        unsigned int rt,reg;
        int a=0xff;
        Cuad_Dial *C_D[4];
        union REGS m;
```

```
Raton R(m);
OBJ_TEXTO tx[4];

// Titulo de la ventana
char ln[4][2] = {"","","",""};
char *ln2[] =   {"    Selección     ",
                 "       de         ",
                 "     línea        ",
                 "                  "};
   R.Activa_Raton();
    Obj_Texto *TXT[4];
    Ventana VENT(&W,&r,200,13,370,135);
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    setcolor(getmaxcolor());
    setwritemode(XOR_PUT);
   if (VENT.AbreVentana()) {
    R.DesactivaRaton();

// Coloca los estilos de linea
     for(j=0;j<4;++j)
        {
         TXT[j]=new Obj_Texto(ln2[j],ACTIVO,&tx[j],&tx[j].tx,&W,
         r.izq+10,r.arriba+5+(j*10));
         TXT[j]->DibujaTexto();
         C_D[j]=new Cuad_Dial(ln[j],0x00,ACTIVO,&cd[j],&cd[j].tcd,
         &W,&cd[j].rec,r.izq+155,r.arriba+50+(j*17));
         C_D[j]->Dib_Cuad_Dial();
         setlinestyle(j,0,THICK_WIDTH);
         line(r.izq+10,r.arriba+55+(j*17),r.izq+145,r.arriba+55+(j*17));
        }
        R.Activa_Raton();
     while (a) {
         ActivaTeclado(&W);
         if (R.Det_Fuera_Raton(&p)) {
          rt=Loc_Pos_Raton(&p,&W);
          if (rt & EnCuadDial) {
           ap=Loc_Cuad_Dial(&W,&p);
           Sel_Cuad_Dial(ap);
           R.DesactivaRaton();

        // Maneja la selección del estilo de línea
           if (ap==&cd[0]) {
            cd[0].sel=0xff;  C_D[0]->Dib_Cuad_Dial();    estilo=0; }
               else if(ap==&cd[1]) {
                 cd[1].sel=0xff;  C_D[1]->Dib_Cuad_Dial(); estilo=1; }
               else if(ap==&cd[2]) {
                 cd[2].sel=0xff;  C_D[2]->Dib_Cuad_Dial(); estilo=2; }
               else if(ap==&cd[3]) {
                 cd[3].sel=0xff;  C_D[3]->Dib_Cuad_Dial(); estilo=3; }
               R.Activa_Raton();
                  delay(400);
                  a=0;
                 } else MensajeError();
               }
          }
```

```
        // Borra los objetos creados dinámicamente
        for(j=0;j<4;++j) {
        delete C_D[j]; delete TXT[j];}
        VENT.CierraVentana();
      } else MensajeError();
      return(1);

}
```

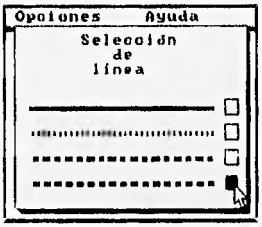


Fig. 7.12. Ventana de selección de línea, creada por la función *estilo_linea*.

La función *Relleno* coloca una ventana con 12 cuadros de selección, que pueden ser utilizados para definir el tipo de relleno de las figuras de rectángulos y elipses, de las herramientas de dibujo.

```
int Relleno(void)
{
        VENTANA W;
        RECUADRO r;
        MAPABITS bmp;
        CUADDIAL cd[12],*ap;
        LOCALIZA p;
        static int i,j;
        unsigned int rt,reg;
        int a=0xff;
        Cuad_Dial *C_D[12];
        union REGS m;
        Raton R(m);
        OBJ_TEXTO tx[4];

        // Titulo de la ventana
        char ln[12][2] = { "","","","","","",
                           "","","","","","",
                         };
        char *ln2[] =   {"   Selección    ",
                         "      de        ",
                         "   relleno      ",
                         "________________"};
        Obj_Texto *TXT[4];
        Ventana VENT(&W,&r,200,13,370,160);
          R.Activa_Raton();
          if (VENT.AbreVentana()) {
```

```
// Se coloca el título de la ventana y los 12 tipos de relleno
    for(j=0;j<4;++j)
        {
         TXT[j]=new Obj_Texto(In2[j],ACTIVO,&tx[j],&tx[j].tx,&W,
         r.izq+10,r.arriba+5+(j*10));
         TXT[j]->DibujaTexto();
        }
        R.DesactivaRaton();
    for(i=0;i<12;i++) {
        If(i>=0 && i<=5)
        { C_D[i]=new Cuad_Dial(In[i],0x00,ACTIVO,&cd[i],&cd[i].tcd,
          &W,&cd[i].rec,r.izq+20+(i*25),r.arriba+77);
          C_D[i]->Dib_Cuad_Dial();
          setfillstyle(i, getmaxcolor());
          // Dibuja una barra con relleno
          rectangle(r.izq+16+(i*25), r.arriba+57,r.izq+32+(i*25),r.arriba+74);
          bar(r.izq+17+(I*25),r.arriba+58,r.izq+31+(i*25),r.arriba+73);
        }
        else
        { C_D[i]=new Cuad_Dial(in[i],0x00,ACTIVO,&cd[i],&cd[i].tcd,
          &W,&cd[i].rec,r.izq+20+((i-6)*25),r.arriba+127);
          C_D[i]->Dib_Cuad_Dial();
          // Dibuja una barra con relleno
          setfillstyle(I, getmaxcolor());
          rectangle(r.izq+16+((i-6)*25), r.arriba+107,r.izq+32+((i-6)*25),r.arriba+124);
          bar(r.izq+17+((i-6)*25),r.arriba+108,r.izq+31+((i-6)*25),r.arriba+123);
        }
}
        R.Activa_Raton();
    while (a) {
        ActivaTeclado(&W);
        if (R.Det_Fuera_Raton(&p)) {
          rt=Loc_Pos_Raton(&p,&W);
          if (rt & EnCuadDial) {
            ap=Loc_Cuad_Dial(&W,&p);
            Sel_Cuad_Dial(ap);
            R.DesactivaRaton();

       // Maneja la selección del tipo de relleno
           If (ap==&cd[0]) {
             cd[0].sel=0xff;  C_D[0]->Dib_Cuad_Dial();    relleno=0;  }
                else  if(ap==&cd[1]) {
                  cd[1].sel=0xff;  C_D[1]->Dib_Cuad_Dial(); relleno=1;  }
                else  if(ap==&cd[2]) {
                  cd[2].sel=0xff;  C_D[2]->Dib_Cuad_Dial(); relleno=2;  }
                else  if(ap==&cd[3]) {
                  cd[3].sel=0xff;  C_D[3]->Dib_Cuad_Dial(); relleno=3;  }
                else  if(ap==&cd[4]) {
                  cd[4].sel=0xff;  C_D[4]->Dib_Cuad_Dial(); relleno=4;  }
                else  if(ap==&cd[5]) {
                  cd[5].sel=0xff;  C_D[5]->Dib_Cuad_Dial(); relleno=5;  }
                else  if(ap==&cd[6]) {
                  cd[6].sel=0xff;  C_D[6]->Dib_Cuad_Dial(); relleno=6;  }
                else  if(ap==&cd[7]) {
```

```
                cd[7].sel=0xff;  C_D[7]->Dib_Cuad_Dial(); relleno=7;  }
              else  if(ap==&cd[8]) {
                cd[8].sel=0xff;  C_D[8]->Dib_Cuad_Dial(); relleno=8;  }
              else  if(ap==&cd[9]) {
                cd[9].sel=0xff;  C_D[9]->Dib_Cuad_Dial(); relleno=9;  }
              else  if(ap==&cd[10]) {
                cd[1].sel=0xff;  C_D[10]->Dib_Cuad_Dial(); relleno=10;}
              else  if(ap==&cd[11]) {
                cd[11].sel=0xff; C_D[11]->Dib_Cuad_Dial(); relleno=11;}
                  R.Activa_Raton();
                  delay(300);
                  a=0;
                 } else MensajeError();
               }
          }
        // Borra los objetos dinámicos C_D
        for(i=0;i<12;i++)    delete C_D[i];
        // Borra los objetos creados dinámicamente
        for(j=0;j<4;++j) delete TXT[j];
        VENT.CierraVentana();
        } else MensajeError();
      return(1);
}
```

Fig.7.13. Resultado de la función *Relleno*.

7.1.4. Menú de *Ayuda*

7.1.4.1. Código para crear la ayuda del programa de aplicación

La función *Ayuda*, coloca la opción de ayuda para la aplicación. A través de cuadros de selección, permite elegir el tipo ayuda necesaria; una vez seleccionada, aparecerá una ventana con una breve descripción del menú en cuestión. Para más detalle, ver el código de la función y la salida que genera el programa, que a continuación se muestran.

```
int Ayuda(void)
{
        VENTANA W;
        RECUADRO r;
        MAPABITS bmp;
        CUADDIAL cd[5],*ap;
```

```
LOCALIZA p;
stalic int i;
unsigned int rt;
Int a=0xff;

// Texto para los cuadros de selección de ayuda
char In[5][23] = { "Menú de Escritorio    ",
                   "Menú de Archivo       ",
                   "Menú de Opciones      ",
                   "Menú de Ayuda         ",
                   "Herramientas de dibujo"
                 };

// Texto para las opciones de ayuda
char In2[11][50] ={"  AYUDA SOBRE: MENU DE ESCRITORIO",
                   "",
                   " Este menú muestra en su opción de referencia ",
                   " la presentación del programa; nombre y número",
                   "                de  versión."
                  };

char In3[11][50] ={      " AYUDA SOBRE: MENU DE ARCHIVO                ",
                   "                                                   ",
                   " La opción de :                                    ",
                   " Nuevo, permite crear un nuevo archivo PCX.        ",
                   " Abrir, permite editar un archivo PCX existente.",
                   " Cerrar, cierra el archivo PCX que  esta en uso.",
                   " Salvar, guarda el archivo actual con el mismo  ",
                   " nombre.                                           ",
                   " "Salvar Como?, guarda el archivo actual con    ",
                   " otro nombre.                                      ",
                   " Salir, permite salir de este programa.          "
                  };

char In4[11][50] ={"  AYUDA SOBRE: MENU DE OPCIONES",
                   " Las opciones de:                                  ",
                   " Tipo de linea, permite seleccionar el tipo de  ",
                   "     línea para: rectángulos y líneas.             ",
                   " Grosor, permite seleccionar el grosor de la    ",
                   "          línea (gruesa o delgada).                ",
                   " Relleno, 12 tipos de relleno para figuras como ",
                   "  elipses y rectángulos que llevan relleno.    "
                  };
char In5[11][50] ={"  AYUDA SOBRE: MENU DE AYUDA ",
                   "                                                   ",
                   " Proporciona información acerca de los menús  de ",
                   " de este programa. ",
                   "  ",
                   "  ",
                   "  "
                  };

Cuad_Dial *C_D[5];
union REGS m;
Raton R(m);
```

```
        Ventana VENT(&W,&r,295,13,515,140);
        if (VENT.AbreVentana()) {
        for(i=0;i<5;i++) {
            C_D[i]=new Cuad_Dial(In[i],0x00,ACTIVO,&cd[i],&cd[i].tcd,
            &W,&cd[i].rec,r.izq+15,(i*20)+30);
            C_D[i]->Dib_Cuad_Dial();
        }
        R.Cursor(mano,4,0); // Activa el cursor en forma de mano del ratón
            while (a) {
            ActivaTeclado(&W);
            if (R.Det_Fuera_Raton(&p)) {
            rt=Loc_Pos_Raton(&p,&W);

                    // Controla la selección de ayuda
                    if (rt & EnCuadDial) {
                       ap=Loc_Cuad_Dial(&W,&p);
                       Sel_Cuad_Dial(ap);
                       if (ap==&cd[0]) {
                         cd[0].sel=0xff; reg=0;
                         C_D[0]->Dib_Cuad_Dial();
                         a=0;
                       } else  if(ap==&cd[1]) {
                         cd[1].sel=0xff; reg=1;
                         C_D[1]->Dib_Cuad_Dial();
                         a=0;
                        } else  if(ap==&cd[2]) {
                         cd[2].sel=0xff; reg=2;
                         C_D[2]->Dib_Cuad_Dial();
                         a=0;
                        } else if(ap==&cd[3]) {
                         cd[3].sel=0xff; reg=3;
                         C_D[3]->Dib_Cuad_Dial();
                         a=0;
                        } else if(ap==&cd[4]) {
                         cd[4].sel=0xff; reg=4;
                         C_D[4]->Dib_Cuad_Dial();
                         a=0;
                         }
            } else MensajeError();
        }
        } // Sale de while
        delay(220);
        // Borra los objetos dinámicos C_D
        for(i=0;i<5;i++)  delete C_D[i];
        VENT.CierraVentana();
    // Coloca las ventanas de ayuda para las 5 opciones
        if(reg == 0)  Coloc_Ayuda(In2," Aceptar");  // Ayuda para el menú de Escritorio
         else if(reg == 1) Coloc_Ayuda(In3," Aceptar"); // Ayuda para el menú de Archivo
         else if(reg == 2) Coloc_Ayuda(In4," Aceptar"); // Ayuda para el menú de Opciones
         else if(reg == 3) Coloc_Ayuda(In5," Aceptar"); // Ayuda para el menú de Ayuda
         else if(reg == 4) Ayuda_Herr();  // Ayuda acerca de las herramientas de dibujo
      } else MensajeError();
      R.Cursor(flecha,0,0); // Coloca el cursor de flecha (cursor normal del ratón)
      return(1);
}
```

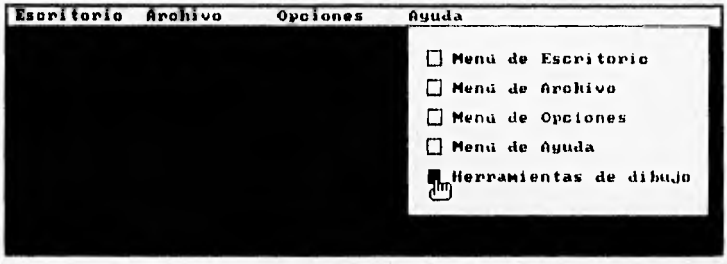


Fig. 7.14. Ventana que permite la selección de ayuda.

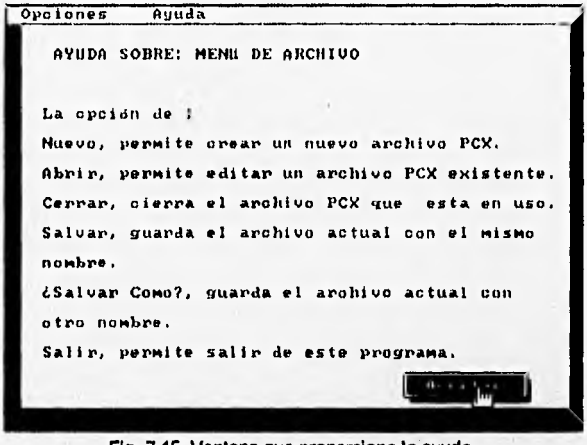


Fig. 7.15. Ventana que proporciona la ayuda.

7.2. DESCRIPCION DE OTRAS FUNCIONES UTILIZADAS EN EL PROGRAMA DE APLICACION

Int Apunt(void)
Esta función regresa el nombre de un archivo.

void Restaura_Dir(char *s)
Función que restaura el nombre de un directorio, a un nombre válido.

int Contr_Err_Disco(int errval,int ax,int bp,int si)
Función para controlar los errores de la unidad de disco.

int PruebaDisco(int n)
Función que prueba las unidades de disco.

Int Nombre_Arch(int c)
Función que verifica el nombre de un archivo.

*char *Obt_Archivos(char *especif,int *cont,char *MapaUnidades)*
Regresa un *buffer* conteniendo los nombres de los archivos.

*int Sel_Archivo(char *especif,char *nombre,int x,int y,char *unidades)*
Función que permite realizar la selección de un archivo desde una ventana.

*int Obt_Info_Archivo(INFOARCH *f,char *s)*
Obtiene la información del archivo PCX.

*int Carga_ArchPCX(char *s,INFOARCH *fi,int (*putl)(char *lb,int l))*
Carga los datos del archivo PCX a memoria.

*int Salvar_Arch(char *s,INFOARCH *fi,char *(*getl)(int l))*
Salva el dibujo a un archivo PCX.

*int Escr_Lin_ArchPCX(char *p,FILE *fp,int n)*
Comprime una línea original de un mapa de *bits*, codificándola a un archivo PCX.

*int Rec_Dialogo(char *l1,char *l2)*
Función para crear un cuadro de diálogo.

int Obt_Buffer(long n)
Coloca un *buffer* de n bytes de longitud, para el dibujo.

void LiberaBuffer(void)
Libera el *buffer* actual que contiene el dibujo.

void LimpiaBuffer(void)
Rellena el área de trabajo de color blanco, es usada cuando se crea un archivo nuevo.

*char *Obt_Linea(int n)*
Escribe una línea al *buffer*. Obtiene un apuntador a la línea *n*, del *buffer* que contiene el dibujo.

void Obt_Frag_Img(void)
Esta función copia al área de dibujo la parte apropiada del *buffer*.

void Coloca_Frag_Img(void)
Esta función copia una parte del dibujo, a la pantalla.

void Sel_Funcion(int n)
Función que selecciona una de las herramientas de dibujo (uno de los Iconos.)

int AbreArchivo(void)
Función que abre un archivo PCX para edición, regresa un valor de 1, si pudo abrir el archivo.

*Int CargaArchivo(char *s,Int (*putl)(char *,Int))*

Carga un archivo PCX. Coloca un apuntador que llama a la función *putl* para guardar las líneas del archivo.

Int Deshacer(void)

Función para deshacer la acción realizada, por la última herramienta de dibujo seleccionada.

*Int Coloc_Ayuda(char s[11][50],char *boton)*

Coloca las opciones de ayuda para los diferentes menús.

Int PantPres(void)

Función que coloca una ventana del tamaño de la pantalla, con la presentación de este programa.

Int Ampliacion(int x1,int y1)

Función que permite realizar la ampliación y edición de un fragmento de imagen pixel por pixel (una área de 32 x 32 pixeles)

Int Ayuda_Herr(void);

Esta función coloca la ventana de ayuda para las herramientas de dibujo.

Int Grosor(void);

La función *Grosor,* coloca una ventana con las dos opciones de grosor para líneas.

Int estilo_linea(void);

Coloca la ventana con los 4 estilos de línea disponibles para el programa.

Int Relleno(void);

Permite utilizar los tipos de relleno para figuras de tipo rectangular y elipsoidal (hay 12 tipos de relleno disponibles.)

7.3. PROCEDIMIENTO DE COMPILACION PARA UN PROGRAMA DE APLICACION

El siguiente diagrama muestra la forma en que fue compilado el programa de aplicación (la herramienta de dibujo), el mismo procedimiento es aplicable a cualquier otro programa que requiera de las herramientas de construcción de interfaces gráficas.

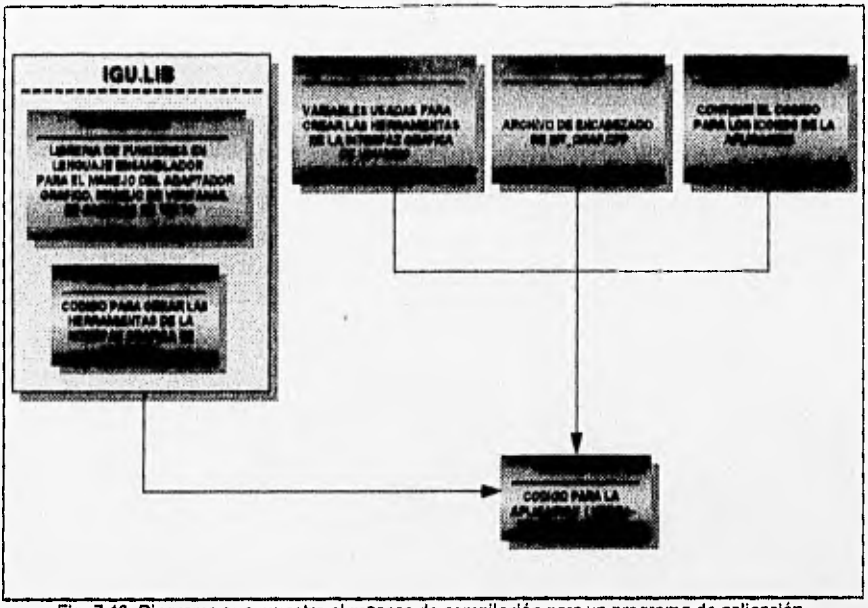


Fig. 7.16. Diagrama que muestra el proceso de compilación para un programa de aplicación.

CONCLUSIONES:

Durante el desarrollo de este trabajo pudimos darnos cuenta, de la importancia que tiene una interfaz gráfica de usuario en los programas de aplicación y porque debe ser considerada siempre en su diseño. Entre otras cosas, podemos mencionar, que cuando se tiene un adecuado diseño de la interfaz gráfica, ésta reditúa en un mejor aprovechamiento, familiarización y aprendizaje del programa de aplicación por parte del usuario final.

Nuestro objetivo fue desarrollar algunas herramientas para la creación de interfaces gráficas, basándonos en algunos de los principios más importantes con que debe contar una interfaz. En la realización de estas herramientas descubrimos todo lo que implica una interfaz gráfica, lo que hay detrás de cada objeto gráfico, la manera en que podemos interactuar con estos objetos, y la complejidad que la mayoría de veces pasa desapercibida cuando usamos interfaces como Windows, X-Window u otro tipo de interfaz gráfica.

Para lograr nuestra meta, se crearon rutinas para crear cuadros de diálogo, cuadros de selección, menús, ventanas de ayuda, cursores, iconos, ventanas de retroalimentación de información. De esta manera, se logró que las características anteriormente mencionadas pudieran ser plasmadas en una librería de funciones, con el fin de que todas ellas pudieran ser utilizadas para crear de forma rápida, una interfaz gráfica para programas de aplicación.

La librería de herramientas que fue desarrollada en el presente trabajo, no pretende ser la mejor solución para crear interfaces gráficas, sino una opción más que permita proporcionarle al programador un conjunto de herramientas que pueda integrar en sus programas, modificar y adaptar a sus necesidades.

El haber desarrollado parte de la librería gráfica en un lenguaje orientado a objetos (*Borland C++*), nos permitió darnos cuenta de las ventajas que este tipo de lenguajes proporciona. Además, el hecho de utilizarlos para representar objetos gráficos, permite observar de una manera más clara, las características de la programación orientada a objetos (el manejo de clases, herencia simple, herencia múltiple, polimorfismo.)

Algunas partes del código de la librería gráfica se encuentran en lenguaje C y en lenguaje ensamblador. El hecho de realizar una librería gráfica híbrida, nos permitió que se aprovecharan las ventajas que cada uno de estos lenguajes puede proporcionar; de esta manera, las funciones críticas como son: el manejo del adaptador gráfico, la escritura de texto y el manejo de ventanas; fueron desarrolladas en estos lenguajes. Podemos mencionar también que fue de gran experiencia el poder combinar tres diferentes clases de código en un programa (lenguaje C, lenguaje ensamblador y C++) para crear la librería gráfica.

Entre las facilidades que proporcionan estas herramientas, podemos mencionar que, de una forma rápida y sencilla, el programador puede incorporar en sus programas, los objetos gráficos de esta librería, como: ventanas, menús, botones, cuadros de diálogo, objetos de tipo texto, mapas de bits, listas, barras de despliegue y poder habilitar el uso del ratón. Es importante recalcar, que todos estos objetos son compatibles con los de otras interfaces gráficas (cumplen con los estándares de interfaces gráficas existentes), esto con el fin de que cualquier usuario que haya tenido contacto con otras interfaces gráficas, al ver estos objetos, le resulten igualmente familiares.

El programador que utilice esta librería, contará además con una mejor presentación de sus programas, haciendo uso del modo gráfico en los adaptadores de vídeo: EGA, VGA y Hércules. Asimismo, sus programas no estarán limitados por el adaptador gráfico, pudiéndose obtener la misma calidad de gráficos en los tres tipos de tarjeta de vídeo mencionados anteriormente. Además, el uso del ratón como dispositivo de posicionamiento en pantalla, permite minimizar el tiempo que el usuario emplea en la selección de objetos del programa de aplicación.

Otra de las características de las herramientas, es que pueden utilizarse en equipos de cómputo de bajo costo, sin que sea necesario que cuenten con grandes recursos de *hardware*, como espacio en disco duro, memoria, procesador y tipo de monitor.

En base a esta librería gráfica pueden desarrollarse otras herramientas que permitan hacer una interfaz gráfica más complicada. Las clases para los objetos gráficos están abiertas, para que a partir de ellas puedan generarse clases y funciones más complejas. El programa de aplicación que fue desarrollado (capítulos 6 y 7), nos permite mostrar el uso de todos los objetos gráficos que forman parte de la librería gráfica; sin embargo, no es más que un simple ejemplo de lo que puede hacerse con esta librería y estamos seguros que pueden dársele otros mejores usos, por parte de los programadores que la utilicen.

APENDICE A. ADAPTADOR GRAFICO HERCULES

TARJETA GRAFICA HERCULES

De los cuatro estándares de adaptadores gráficos, tres fueron introducidos por IBM; el adaptador MDA *(Monochrome Display Adapter)*, CGA *(Color Graphics Adapter)* y el adaptador EGA *(Enhanced Graphics Adapter.)*

De los cuatro estándares de vídeo, la tarjeta gráfica Hércules fue desarrollada por una compañía independiente fundada en 1982; Hércules, es esta compañía que ha establecido sus productos como estándares en un mercado dominado por la IBM.

Modelos de tarjetas gráficas Hércules

Después de la introducción de la tarjeta gráfica Hércules en 1982, fueron fabricados tres modelos de la tarjeta. Los modelos y fechas de producción, son los siguientes: GB100 (Agosto 1982 a Octubre 1983); GB101 (Noviembre 1983 a Junio 1984) y GB102 (Julio 1984 a la actualidad); sin embargo, todas estas tarjetas pueden ser consideradas como un sólo producto. Las características de la tarjeta, es el modo gráfico monocromático de 720 x 348 y el modo texto de 80 x 25, que emula el (MDA) de IBM.

Como el MDA, la tarjeta Hércules incluye un puerto paralelo que esta interconectado al puerto LPT1. La tarjeta Hércules esta diseñada para usarse con monitores monocromáticos con lógica transistor-transistor (TTL), como la del monitor monocromático de IBM.

MODOS DE VIDEO

La tarjeta gráfica Hércules, se caracteriza por 2 modos de vídeo distintos: uno de 80 columnas por 25 renglones (modo texto compatible con el adaptador monocromático de la IBM), y el otro 720 por 348 pixeles (modo gráfico de mapa de bits creado por Hércules). El modo es seleccionado, en base a una programación cuidadosa, de los puertos I/O de la tarjeta; éstos son: el *switch* de configuración, el modo de despliegue del puerto de control y varios registros del controlador 6845 CRT.

Para cualquiera de los dos modos texto o gráfico, el *software* de control será desplegado para introducir los datos apropiados dentro del *buffer* de vídeo de la tarjeta; un bloque de RAM, el cual se encuentra localizado en la tarjeta gráfica Hércules.

El modo texto es rápido y fácil de programar, pero su capacidad esta limitada solamente a los 256 caracteres ASCII extendidos de la PC. El modo gráfico, por otro lado, es completamente versátil, capaz de desplegar cualquier clase de imágenes que el programa pueda concebir y limitado solamente por la resolución disponible. El precio por agregar esta versatilidad, es que el modo gráfico es altamente demandante, igual para el programa, que para el CPU; ya que cada pixel que es activado, al desplegarse en pantalla, estará individualmente direccionado por el *software*.

En el modo texto, un número limitado de índices, llamados caracteres alfanuméricos estándares, pueden ser desplegados. El patrón de puntos, para estos caracteres, están almacenados permanentemente en un circuito integrado conocido como generador de caracteres. Un solo caracter es comúnmente un byte (ocho pixeles) y varias líneas de barrido (14.) Una consecuencia obvia de esto, es que el modo texto requiere menos memoria de video que el modo gráfico; solo 2000 bytes de memoria son necesarios para representar un texto de 80 columnas por 25 renglones.

El caracter en modo texto, está dividido lógicamente en tres partes:

- Caracteres de control: código ASCII 0-31 (0-1Fh). Códigos que le dicen a un dispositivo, como a una impresora o terminal, que realicen una actividad específica.
- Caracteres ASCII estándares imprimibles: código ASCII 32-127 (20h-7Fh.) Caracteres alfanuméricos y de puntuación (común en la mayoría de las computadoras.)
- Caracteres ASCII Extendidos: código ASCII 128-255 (80h-FFh.) Caracteres gráficos y otros caracteres especiales.

En la actualidad existen 7 atributos distintos para los caracteres:

- Caracter blanco.
- Caracter parpadeante.
- Fondo de alta intensidad.
- Caracter normal (caracter luminoso, sobre un fondo negro.)
- Video inverso (caracter negro, sobre un fondo luminoso.)
- Caracter subrayado.

Estos casos, pueden ocurrir en un número limitado de combinaciones: normal, video inverso, subrayado y blanco; todos son mutuamente exclusivos. Normal y subrayado, pueden ocurrir en combinación con cualquiera de los dos siguientes: alta intensidad, parpadeo, o con ambos. El video inverso puede ser combinado únicamente con el caracter parpadeante.

Esos atributos son generados por el *hardware* de la tarjeta. El *software* por necesidad, estará afectando solamente los códigos de los atributos deseados, almacenándolos en las localidades apropiadas del *buffer* de despliegue.

Los usos de los atributos son obvios, el atributo de subrayado es comúnmente usado en procesadores de palabra, indicando, en donde aparecerá subrayada una palabra cuando se imprima el texto. El atributo de alta intensidad es usado para simular texto "en negrita", en procesadores de palabras; llama más la atención en los indicadores del sistema (*prompts*) y en mensajes en toda clase de programas. El atributo de video inverso es frecuentemente usado para destacar bloques de datos, sobre los cuales, una operación de formateo será ejecutada. El atributo de parpadeo es usado para obtener la atención del usuario en mensajes importantes.

El concepto de mapa de bits o de direccionamiento de todos los puntos gráficos, es realmente simple. Cualquier pantalla de despliegue CRT, esta compuesta de un número finito de puntos, referidos como pixeles. En modo gráfico de mapa de bits, un bit específico en la RAM de vídeo corresponde a una localización específica en la pantalla, dentro de una área de 720 por 350 pixeles.

Todos los modelos de las tarjetas gráficas Hércules requieren un monitor monocromático TTL (con sincronía de 50 Hz vertical, 18.4 Khz horizontal) y la resolución de 720 por 350 pixeles.

La tarjeta gráfica Hércules esta diseñada para usarse en la PC IBM, PC XT y PC AT. Esta tarjeta es completamente compatible sobre todos los modelos de PC's.

ARQUITECTURA DE LA GRAFICA HERCULES

Mapeo de memoria de vídeo

Todos los diferentes tipos de adaptadores de vídeo de IBM, soportan el mapeo de memoria de vídeo. Esto es, que la imagen desplegada en el monitor, en cualquier instante, corresponde directamente a los datos almacenados en la RAM. IBM ha reservado una área en las PC´s , después de los 640 kbytes, que son usados por DOS y programas de aplicación, para usar el *buffer* de vídeo. Este bloque de memoria (*buffer*), se encuentra en la dirección absoluta A0000-BFFFF. El *buffer* de vídeo esta localizado sobre el adaptador gráfico, en lugar de la tarjeta madre.

Desde el punto de vista de programación, la tarjeta gráfica Hércules consiste de un *buffer* de despliegue (un bloque de RAM), que almacena directamente los datos responsables de desplegar la imagen sobre el monitor monocromático y el número del puerto I/O, éstos son usados para configurar la tarjeta o regresar información acerca del estado actual de la imagen.

El *buffer* de despliegue es en un bloque de 64 K byte de RAM, localizado sobre la tarjeta gráfica y ocupando la dirección absoluta B0000-BFFFF. El *buffer* de despliegue es utilizado, dependiendo cual de los modos de la tarjeta (texto o gráfico), se encuentre en uso.

El mapeo de memoria de vídeo de la tarjeta, indica la forma en que el *buffer* es particionado en los dos modos de vídeo. En modo texto, únicamente son 4K bytes del *buffer* de despliegue (B0000-B0FFF) los que son usados; mientras que en el modo gráfico, el *buffer* de despliegue es presentado como dos páginas gráficas de 32k bytes, conocidas como: página 0 (B0000-B7FFF) y página 1 (B8000-BFFFF). Cualquiera de las dos o ambas páginas, pueden usarse para una aplicación gráfica.

La tarjeta gráfica Hércules consta de siete puertos de entrada-salida, de 8-bits, que son usados para controlar o regresar información del *buffer* de despliegue. Las funciones y direcciones de estos puertos son las siguientes:

6845 Registro Indexado	03B4
6845 Registro de datos	03B5
Puerto de control del modo de despliegue	03B8
Flip-Flop de lápiz luminoso	03B9
Puerto de despliegue de estado	03BA
Switch de configuración	03BF

El puerto de impresión, sobre todos los modelos de la tarjeta gráfica Hércules, soporta el protocolo *"Centronics"*; usando el mismo *pin* asignado y niveles de señal, tal como ocurre en el adaptador de despliegue monocromático IBM.

En la figura siguiente se muestra la arquitectura de la tarjeta gráfica Hércules:

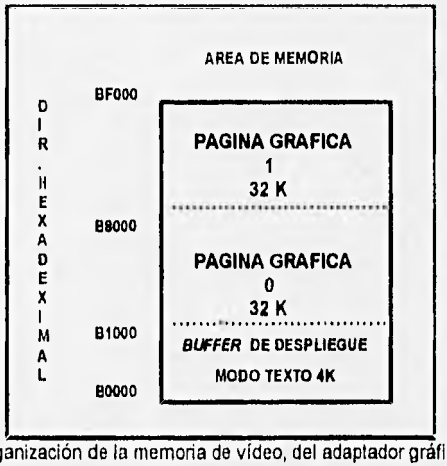


Fig. A.1. Organización de la memoria de vídeo, del adaptador gráfico Hércules

APENDICE B. ADAPTADOR GRAFICO EGA

EGA *(ENHANCED GRAPHICS ADAPTER)*

El adaptador gráfico EGA, fue el sucesor de los adaptadores MDA *(Monochrome Display Adapter)* y del CGA *(Color Graphics Adapter)*. EGA permitió tener gráficos en pantallas monocromáticas, incluyendo las de los primeros monitores ámbar, de las computadoras personales.

Con el EGA se lograron importantes mejoras en el mundo de las computadoras personales, se incrementó la resolución a 640 x 350 pixeles, formando caracteres en matrices de puntos de 8 x 14 pixeles. Además, todos los modos gráficos anteriores fueron incluidos en las características del EGA para hacerlo compatible.

MODOS DE VIDEO DISPONIBLES PARA EL ADAPTADOR GRAFICO EGA

El adaptador gráfico EGA puede ser programado en dos modos principales, que son: modo texto y modo gráfico.

El modo texto monocromático de 80 columnas por 25 renglones de MDA es soportado por EGA, así como, los dos modos gráficos de CGA (320 x 400 pixeles con cuatro colores y 640 x 200 pixeles con dos colores).

EGA introdujo tres nuevos modos gráficos, con más colores y mejor resolución que los anteriores modos gráficos del CGA, el de 320 x 200 pixeles con 16 colores, 640 x 200 con 16 colores y el modo de 640 x 350 con 16 colores. La tabla B.1, resume los modos gráficos utilizados por el adaptador EGA.

Número de modo del BIOS				
Hexadecimal	**Decimal**	**Modo**	**Resolución**	**Colores**
00H, 01H	0, 1	Texto	40 x 25	16
02H, 03H	2, 3	Texto	80 x 25	16
04H, 05H	4, 5	Gráfico	320 x 200	4
06H	6	Gráfico	640 x 200	Monocromático
07H	7	Texto	80 x 25	Monocromático
0BH, 0CH	11, 12			Usado Internamente por el BIOS EGA
0DH	13	Gráfico	320 x 200	16
0EH	14	Gráfico	640 x 200	16
0FH	15	Gráfico	640 x 350	Monocromático
10H	16	Gráfico	640 x 350	16

Tabla B.1. Modos gráficos utilizados por el adaptador EGA.

LOS COLORES EN EL ADAPTADOR EGA

Los modos gráficos más frecuentemente usados sobre el EGA, son los modos con 16 colores, los cuales requieren de 4 *bits* para definirlos. Cada *bit* se le asigna a un color en particular, el resultado es de 16 combinaciones, que corresponden a las 16 que se pueden tener con los 4 *bits*.

Los colores en las pantallas de video son producidos por combinaciones de 4 elementos: rojo, verde, azul (tres colores principales); y un nivel de intensidad o brillantez. En muchas aplicaciones, 16 colores son suficientes, ya que se puede seleccionar el rango de combinaciones que el *hardware* puede desplegar, 64 colores en este caso.

La tabla B.2 muestra los colores disponibles de los modos con 16 colores, para el modo gráfico y el modo texto.

Intensidad	Rojo	Verde	Azul	Binario	Hexadecimal	Descripción
0	0	0	0	0000	00H	Negro
0	0	0	1	0001	01H	Azul
0	0	1	0	0010	02H	Verde
0	0	1	1	0011	03H	(Azul-Verde)
0	1	0	0	0100	04H	Rojo
0	1	0	1	0101	05H	Magenta
0	1	1	0	0110	06H	Café
0	1	1	1	0111	07H	Gris claro o blanco ordinario
1	0	0	0	1000	08H	Gris oscuro, negro sobre muchas pantallas
1	0	0	1	1001	09H	Azul claro
1	0	1	0	1010	0AH	Verde claro
1	0	1	1	1011	0BH	(Azul-Verde) claro
1	1	0	0	1100	0CH	Rojo claro
1	1	0	1	1101	0DH	Magenta claro
1	1	1	0	1110	0EH	Amarillo claro
1	1	1	1	1111	0FH	Blanco

Tabla B.2. Colores disponibles en los modos de texto y gráficos de 16 colores.

En el EGA, cada valor de atributo del *buffer* de vídeo es asignado a uno de los 16 registros de paleta, cada uno de los cuales, contiene el valor que le corresponde a determinado color. El uso de paletas hace posible especificar un amplio rango de colores, usando relativamente unos cuantos bits de datos en el *buffer* de vídeo. Los 16 registros de paleta con que cuenta el EGA, permiten que se pueda seleccionar un color, de un conjunto de 64 colores.

BUFFER DE VIDEO EN EL EGA

En el modo de texto, el mapeo del *buffer* de vídeo se realiza de una manera similar en todos los adaptadores gráficos. En modos gráficos, se utiliza la técnica de mapeo paralelo que fue introducida por el EGA.

En la siguiente tabla se muestran todos los valores para los modos de vídeo:

Modo de video	Dirección de inicio (Hexadecimal)	Memoria usada (bytes)
00H, 01H	B800H	2000
02H, 03H	B800H	4000
04H, 05H	B800H	16000
06H	B800H	16000
07H	B000H	4000
0DH	A000H	32000
0EH	A000H	64000
0FH	A000H	56000
10H	A000H	112000

Tabla B.3. Direcciones del *buffer* de vídeo, en los diferentes modos de vídeo del EGA.

Por ejemplo, si se usa en: 640 x 350, modo de 16 colores, con 4 bits por pixel; se necesitan:
(640 x 350 x 4) / 8 = 112,000 bytes; para representar una pantalla completa de datos de vídeo.

En modo texto, sin embargo, si se despliegan 25 renglones de 80 caracteres con la misma resolución, se necesitarían:
80 x 25 x2 = 4000 bytes.

A continuación se resume en las tablas B.5 y B.6 las características de modo de vídeo y de las páginas para el adaptador EGA.

Modo de video	Número de páginas
00H, 01H	8
02H, 03H	8
04H, 05H	2 (no es soportado completamente por el ROM BIOS)
06H	1
07H	8
0DH	8
0EH	4
0FH	2
10H	2

Tabla B.4. Páginas disponibles en EGA.

Página	40x25, con 16 colores	80x25, con 16 colores	80x25, monocromático
0	B800:0000H	B800:0000H	B000:0000H
1	B800:0800H	B800:1000H	B000:1000H
2	B800:1000H	B800:2000H	B000:2000H
3	B800:1800H	B800:3000H	B000:3000H
4	B800:2000H	B800:4000H	B000:4000H
5	B800:2800H	B800:5000H	B000:5000H
6	B800:3000H	B800:6000H	B000:8000H
7	B800:3800H	B800:7000H	B000:7000H

Tabla B.5. Direcciones de inicio para las páginas en modo texto

APENDICE C. ADAPTADOR GRAFICO VGA

VGA *(VIDEO GRAPHICS ARRAY)*

La tendencia de los nuevos subsistemas de video de IBM, es proporcionar una mejor resolución vertical. Por ejemplo, el VGA en algunos modos de texto, tienen una resolución de 720 X 400, asimismo, los caracteres son cada uno de 9 pixeles de ancho por 16 pixeles de alto. El modo texto de 80 X 25 que se encuentra en CGA y MCGA, está también presente en el VGA.

La resolución de 640 X 480 pixeles, con 16 colores del VGA, tiene más del doble de pixeles en pantalla que el original del modo gráfico CGA (640 X 200 pixeles.) El adaptador gráfico VGA puede ser programado en modo texto y en modo gráfico. Las 80 columnas por 25 renglones del modo texto monocromático del adaptador MDA, es soportado por el VGA; el usuario puede ver con mejor resolución la pantalla, que en los adaptadores de vídeo antecesores. Similarmente, los dos modos de texto del CGA (40 X 25 y 80 X 25 pixeles), también pueden ser soportados por VGA.

Un cursor parpadeante, es una característica del modo texto que es usado para indicar la localización activa sobre la pantalla de despliegue. El cursor es actualmente un grupo de líneas que llenan completamente las dimensiones del caracter. El tamaño de sus dimensiones varía, de acuerdo con el *hardware* de vídeo y el modo de vídeo. El VGA usa 9 pixeles de ancho por 16 de alto.

Se puede cambiar el tamaño del cursor, así como, su localización en la pantalla, usando los servicios proporcionados por el ROM BIOS. La interrupción 10H y la función 01H permiten colocar el tamaño del cursor, la función 02H permite mover el cursor a cualquier posición en la pantalla. El ROM BIOS también proporciona una función (interrupción 10H, función 03H) que reporta el tamaño y localización del cursor.

En la tabla C.1 se resumen los modos de vídeo utilizados por el adaptador VGA.

Número de modo del BIOS				
Hexadecimal	Decimal	Tipo	Resolución	Colores
00H, 01H	0, 1	Texto	40 x 25	16
02H, 03H	2, 3	Texto	80 x 25	16
04H, 05H	4, 5	Gráficos	320 x 200	4
06H	6	Gráficos	640 x 200	2
07H	7	Texto	80 x 25	Monocromático
0DH	13	Gráficos	320 x 200	16
0EH	14	Gráficos	640 x 200	16
0FH	15	Gráficos	640 x 350	Monocromático
10H	16	Gráficos	640 x 350	16
11H	17	Gráficos	640 x 480	2
12H	18	Gráficos	640 x 480	16
13H	19	Gráficos	320 x 200	256

Tabla C.1. Modos de vídeo disponibles para el adaptador VGA.

En la tabla C.2, se muestran todos los valores para los modos de vídeo del adaptador VGA:

Modo de vídeo	Dirección de inicio (Hexadecimal)	Memoria usada (bytes)
00H, 01H	B800H	2000
02H, 03H	B800H	4000
04H, 05H	B800H	16000
06H	B800H	16000
07H	B000H	4000
0DH	A000H	32000
0EH	A000H	64000
0FH	A000H	56000
10H	A000H	112000
11H	A000H	38400
12H	A000H	153600
13H	A000H	64000

Tabla C.2. Direcciones del *buffer* de vídeo, en los diferentes modos de vídeo del VGA.

USANDO COLORES EN MODO GRAFICO

La diferencia más importante entre los atributos de modo texto y modo gráfico, es que, en modo gráfico se puede controlar el color de cada pixel, lo que permite usar colores mucho más efectivamente que en modo texto. Esto es totalmente evidente usando VGA.

El VGA es considerablemente más flexible en términos de administración de color, porque se puede asignar cualquier combinación de color, a cualquier paleta o registros de color de vídeo (DAC). De la misma manera, es importante el hecho de que se cuente con un pixel grande. Los modos gráficos usados más frecuentemente sobre VGA, son los modos de 16 colores, con pixeles que requieren 4 *bits* para definir los colores. En casi todas las aplicaciones, 16 colores son adecuados, porque se puede seleccionar éstos, desde el rango completo de combinación de colores que el *hardware* puede desplegar (64 colores sobre le EGA y 262,144 colores sobre el MCGA y VGA). Asimismo, el ROM BIOS proporciona servicios que permiten asignar arbitrariamente combinación de colores a la paleta y registros de color de vídeo (DAC), sobre el VGA.

PAGINAS DE DESPLIEGUE EN MODO GRAFICO

Para el VGA, el concepto de página esta disponible tanto en modo gráfico, como en modo de texto.

El principal beneficio de usar múltiples páginas de vídeo, en cualquiera de los dos modos, gráfico o texto, es la capacidad de cambiar instantáneamente desde una pantalla de despliegue a otra; siendo atractivo el tiempo que tarda en construir la información desplegada. En teoría, múltiples páginas de vídeo pueden ser usadas en modos gráficos, para producir suaves y finos efectos de animación.

MAPEO DE PIXELES EN MODO GRAFICO

Sobre el VGA, en modos gráficos de 16 colores, los pixeles están ordenados en cuatro mapas de memoria paralela. En efecto, algunos rangos de direcciones empiezan en A0000:0000H. Sin embargo, VGA tiene una circuitería especial que accesa los cuatro mapas en paralelo. En modos gráficos de 16 colores, cada pixel consta de 4 bits y es almacenado con un bit en cada mapa de memoria.

En las tablas C.3 y C.4 se muestran las características de las páginas de vídeo para el VGA.

Modo de vídeo	Número de páginas
00H, 01H	8
02H, 03H	8
04H, 05H	2 (no es soportado completamente por el ROM BIOS)
06H	1
07H	8
0DH	8
0EH	4
0FH	2
10H	2
11H	1
12H	1
13H	1

Tabla C.3. Páginas disponibles en VGA.

Página	40x25, con 16 colores	80x25, con 16 colores	80x25, monocromático
0	B800:0000H	B800:0000H	B000:0000H
1	B800:0800H	B800:1000H	B000:1000H
2	B800:1000H	B800:2000H	B000:2000H
3	B800:1800H	B800:3000H	B000:3000H
4	B800:2000H	B800:4000H	B000:4000H
5	B800:2800H	B800:5000H	B000:5000H
6	B800:3000H	B800:6000H	B000:6000H
7	B800:3800H	B800:7000H	B000:7000H

Tabla C.4. Direcciones de inicio para las páginas en modo texto

APENDICE D. ICONOS DE LA INTERFAZ GRAFICA

ICONOS

Los iconos son un popular medio para representar comandos en modo gráfico. Si se han usado programas como: Corel Draw, AutoCAD u otros programas de herramientas de dibujo; se pueden ver los beneficios que proporcionan los iconos.

Los iconos son utilizados en muchas aplicaciones gráficas, ya que éstos pueden representar comandos complejos como símbolos o figuras; y como resultado, las interfaces gráficas pueden ser diseñadas en torno a ellos, en lugar de tener que teclear el comando o seleccionar uno de una lista de menús. Un icono puede ser seleccionado para invocar el comando.

El tamaño de los iconos varía dependiendo del uso que se les pueda dar; sin embargo, el manejar iconos de gran tamaño, debido al consumo de memoria, puede volverse lento y tedioso de desplegar en la pantalla. Por las razones antes mencionadas se recomienda usar tamaños de iconos pequeños (tamaños típicos de iconos son: de 16 x 16 y 32 x 32 pixeles.)

FORMATO DE UN ARCHIVO DE ICONO

El formato de los archivos de iconos consta de dos partes: el encabezado y el cuerpo de archivo.

El encabezado se encuentra localizado en la parte superior del archivo y es una simple línea que especifica el ancho y largo del icono. A continuación se tiene el cuerpo del archivo del icono, el cual puede variar dependiendo de las dimensiones especificadas en el encabezado, por ejemplo: iconos de 16 x 16 pixeles, 32 x 32 pixeles son de los más comunes. El cuerpo del archivo se organiza de forma tal que, cada renglón representa a una línea del icono, como se muestra en el ejemplo siguiente expresado en formato hexadecimal:

No. línea	Código en hexadecimal
0	0x20,0x00,0x20,0x00, // Encabezado del icono 20H de largo por 20H de ancho
1	0xff,0xff,0xff,0xff,0xfd, // Comienza cuerpo del icono
2	0x80,0x00,0x00,0x00,0xcc,
3	0x80,0x00,0x70,0x00,0xfb,
4	0x80,0x00,0x78,0x00,0x8f,
5	0x80,0x00,0x7c,0x00,0x86,
6	0x80,0x00,0x7e,0x00,0xff,
7	0x80,0x00,0x3f,0x00,0x95,
8	0x80,0x00,0x1f,0x80,0x80,
9	0x90,0x80,0x0f,0xc0,0xfd,
10	0x88,0x40,0x07,0xe0,0xcc,
11	0x84,0x20,0x03,0xf0,0xfb,
12	0x82,0x10,0x01,0xf0,0x8f,
13	0x81,0x08,0x00,0xf0,0x86,
14	0x90,0x84,0x00,0x00,0xff,
15	0x88,0x42,0x00,0x00,0x95,
16	0x84,0x21,0x00,0x00,0x80,
17	0x82,0x10,0x80,0x00,0xfd,
18	0x81,0x08,0x40,0x00,0xcc,
19	0x80,0x84,0x20,0x00,0xfb,
20	0x80,0x42,0x10,0x00,0x8f,
21	0x80,0x21,0x08,0x00,0x86,
22	0x80,0x10,0x84,0x00,0xff,
23	0x80,0x08,0x42,0x00,0x95,
24	0x80,0x04,0x21,0x00,0x80,
25	0x80,0x02,0x10,0x80,0xfd,
26	0x80,0x01,0x08,0x40,0xcc,
27	0x80,0x00,0x04,0x20,0xfb,
28	0x80,0x00,0x00,0x10,0x8f,
29	0x80,0x00,0x00,0x00,0x88,
30	0x80,0x00,0x00,0x00,0xff,
31	0x80,0x00,0x00,0x00,0x85,
32	0x80,0x00,0x00,0x00,0x80,
33	0xff,0xff,0xff,0xff,0xfd

El código anterior representa al icono de *spray* de la figura D.1.

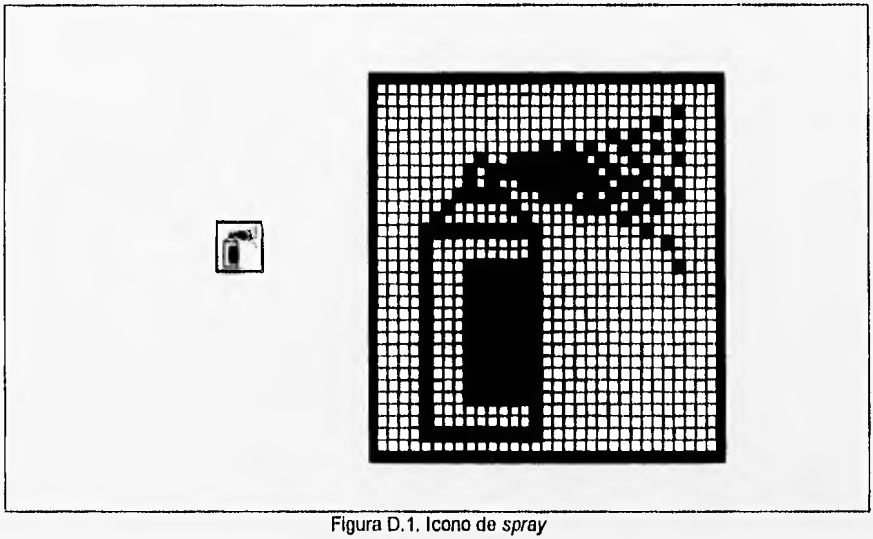


Figura D.1. Icono de *spray*

ICONOS UTILIZADOS EN LA APLICACION

Para las herramientas de dibujo, fueron utilizados los iconos que se muestran en la figura 2. Estos iconos tienen asociada una función especifica de dibujo (dibujar una línea, para borrar, dibujar rectángulos, círculos, elipses, herramienta de "*spray*", para deshacer alguna operación de dibujo realizada, ampliar un fragmento de imagen, dibujo a mano alzada, etc.)

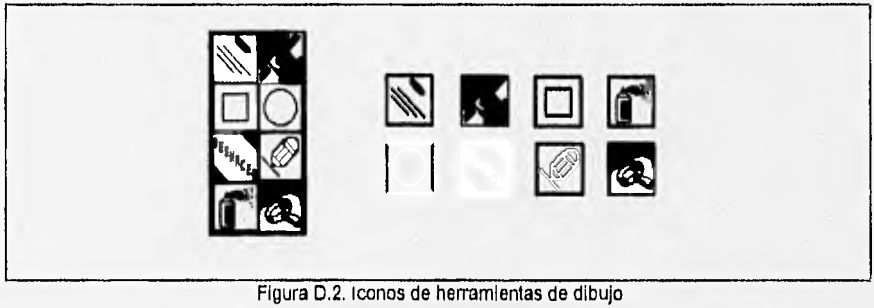


Figura D.2. Iconos de herramientas de dibujo

Los iconos utilizados dentro de cuadros de diálogo sirven para dar una idea de la función que se esta realizando.

En la siguiente figura, se muestra un icono que sirve para indicar al usuario que tenga precaución con la actividad que esta realizando.

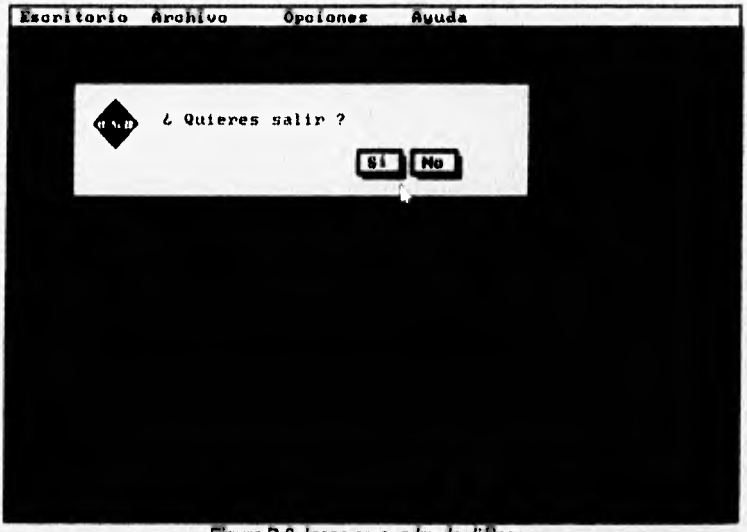


Figura D.3. Icono en cuadro de diálogo.

Los iconos o mapas de bits de gran tamaño tienen por objeto solo efectos decorativos en la pantalla, ya que éstos consumen mucha memoria y tardan demasiado tiempo en ser desplegados. En la siguiente figura, se muestra un ejemplo de este tipo de iconos, los cuales aparecen en la ventana de presentación del programa de aplicación que fue mostrado en el capítulo 7.

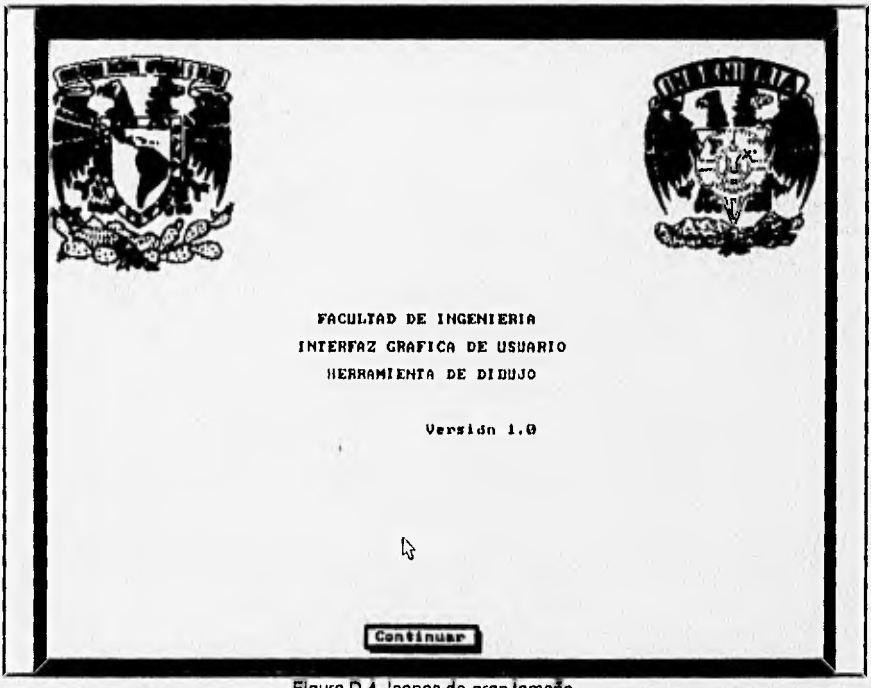


Figura D.4. Iconos de gran tamaño.

APENDICE E. MANUAL DE USUARIO DE LA IGU

PROGRAMAS PARA EJEMPLIFICAR EL USO DE LA LIBRERIA DE HERRAMIENTAS PARA INTERFACES GRAFICAS

Los programas que se muestran a continuación, pretenden ejemplificar el uso que puede dárseles a las herramientas de construcción de interfaces gráficas. Los programas fueron compilados con Borland C++ 3.1 y requieren de los archivos de encabezados VARS_IGU.H (que contiene las variables globales para la interfaz), del archivo INT_GRAF.H (declaración de las funciones usadas en la librería; mostradas en el capítulo 5) y del archivo ICONOS.H (que contiene el código para los iconos.) Para crear el código ejecutable, los programas necesitan ligarse con la librería IGU.LIB, la cual contiene: FUNCSASM.LIB e INT_GRAF.OBJ. Se creó un archivo PRJ, el cual ilustra la forma como deben de ligarse (Véase la fig. E.1.)

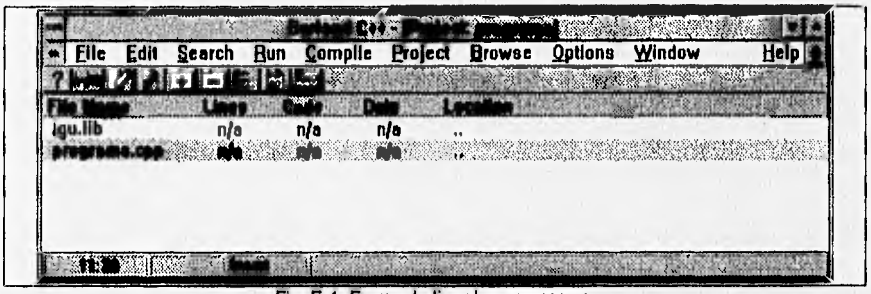


Fig. E.1. Forma de ligar los programas.

PROGRAMA DE PRUEBA PARA LA CLASE *RATON* Y *GRAFICOS*

El siguiente programa hace uso de las clases *Raton* y *Graficos*. Detecta el controlador del ratón, inicializa el modo gráfico y el manejo del ratón; muestra además, tres diferentes tipos de cursores (el de flecha, de reloj de arena y el de una mano.) Finalmente, el ratón es deshabilitado; así como, el modo gráfico. A continuación puede observarse tanto el código del programa, como la salida que genera dicho programa (Véase la fig. E.2.)

```
#define _DATOSIGU   1  // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>      // Contiene la definición de variables globales
#include <int_graf.h>      // Definición de funciones utilizadas en este programa

void main(void)
{
      union REGS m;        // Definición de variables necesarias para
      Raton R(m);          // crear el objeto Raton
      Graficos A;          // Definición del objeto Graficos
```

```
    if(R.Del_Contr_Raton()) {              // Busca el controlador del ratón y
      registerbgidriver(EGAVGA_driver);    // el adaptador gráfico utilizado
      registerbgidriver(Herc_driver);
      if(A.IniciaGraficos()) {             // Inicia el modo gráfico
        ClearScreen();                     // Limpia la pantalla y coloca el fondo en color gris
        R.Activa_Raton();                  // Activa el ratón y coloca el cursor del ratón
        R.Cursor(flecha,0,0);
        getch();
        R.Cursor(reloj,7,7);               // Cambia la apariencia del cursor, a la de
                                           // reloj de arena
        getch();
        R.Cursor(mano,1,1);                // Cambia la apariencia del cursor, a la de una mano
        getch();
        R.DesactivaRaton();                // Desactiva el ratón y desaparece el cursor
        getch();
        clrscr();                          // Limpia la pantalla
      } else cout << "No se puede establecer el modo gráfico";
    } else cout << "No se localizó el controlador del ratón";
}
```

Fig. E.2. El resultado del programa de las clases *Graficos* y *Raton*.

PROGRAMA DE PRUEBA PARA LA CLASE *VENTANA*

En el siguiente programa muestra el uso de la clase *Ventana*, se define un arreglo dinámico de 5 objetos *Ventana*, que son colocados a lo largo de la pantalla y después estos objetos van siendo borrados uno por uno, a continuación se muestra el código y el resultado del programa.

```
#define _DATOSIGU    1  // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>       // Contiene la definición de variables globales
#include <int_graf.h>       // Definición de funciones utilizadas en este programa

void main(void)
{       VENTANA W[5];       // Arreglo de 5 elementos VENTANA
        RECUADRO REC[5];    // Arreglo de 5 elementos de tipo RECUADRO
        union REGS m;       // Variable necesaria para el objeto Raton
        Graficos A;         // Objeto Grafico
        static int i;
        Raton R(m);         // Objeto Raton
        Ventana *V[5];      // Arreglo de 5 objetos Ventana

          if(R.Del_Contr_Raton()) {              // Busca el controlador del ratón y
            registerbgidriver(EGAVGA_driver);    // el adaptador gráfico utilizado
            registerbgidriver(Herc_driver);
            if(A.IniciaGraficos()) {             // Inicia el modo grafico
              ClearScreen();                     // Limpia la pantalla y coloca el
                                                 // fondo en color gris
              R.Activa_Raton();                  // Activa, el ratón y coloca el cursor de flecha
              Inic_Contr_Vents();      // Función que inicia el control de ventanas,
                                       // necesario definirla antes de abrir cualquier ventana
              for(i=0;i<5;i++)       // Coloca 5 objetos dinámicos de tipo Ventana, en la pantalla
               {
                V[i]=new Ventana(&W[i],&REC[i],5+(i*128),5+(i*96),
                ((i+1)*128)-10,((i+1)*96)-10);
                V[i]->AbreVentana();
               }
              getch();
              for(i=4;i>-1;i--)                  // Borra los 5 objetos Ventana definidos
               {
                V[i]->CierraVentana();
                delete V[i];
                delay(800);
               }
              getch();
              R.DesactivaRaton();                // Desactiva el ratón y desaparece el cursor
              clrscr();                          // Limpia la pantalla
            } else cout << "No se puede establecer el modo gráfico";
        } else cout << "No se localizo el controlador del ratón";
}
```

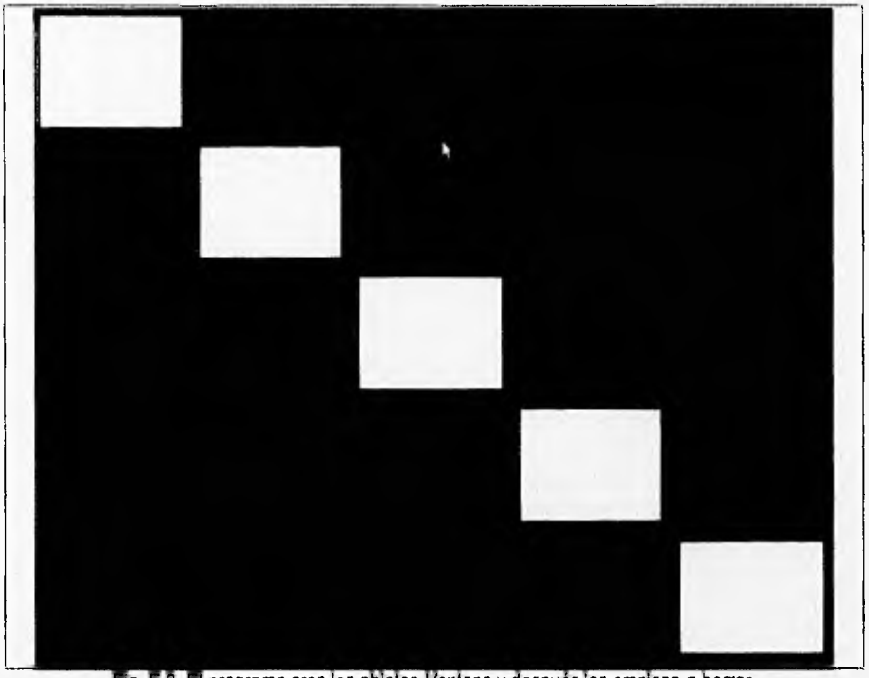


Fig. E.3. El programa crea los objetos *Ventana* y después los empieza a borrar.

PROGRAMA DE PRUEBA PARA LA CLASE *MENUS*

El programa que se muestra a continuación permite probar la clase *Menus*, coloca la barra de menús con sus títulos (6 objetos menú, fig. E.4); permite el uso de los índices de los menús y se pueden observar los estados que pueden tener éstos, activos o inactivos.

En este programa, se puede ver también como le es asociada una función a un determinado índice de menú y la actividad que realiza cuando es seleccionado.

```
#define _DATOSIGU    1  // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>       // Contiene la definición de variables globales
#include <int_graf.h>       // Definición de funciones utilizadas en este programa
// Definicion de prototipos de funciones usadas en el programa
int NoHaceNada(void);       //  Esta función no realiza ninguna actividad
int Salir(void);            // Función para salir del programa
int actividad=1;            // Bandera para indicar actividad en la pantalla
int rt,i;
```

```
LOCALIZA Loc;              // Variable LOCALIZA, para localizar objetos en la pantalla

// Definición de 6 menús, con índices variables de 1 a 5.
MENU menu1 = { 1," MENU1 ",
               " Salir   \004S",Salir,
             };
MENU menu2 = { 2," MENU2",
               ".indice 1 \004A",NoHaceNada,
               " Salir   \004S",Salir,
             };
MENU menu3 = { 3," MENU3",
               ".Indice 1 \004A",NoHaceNada,
               ".Indice 2 \004B",NoHaceNada,
               " Salir   \004S",Salir,
             };
MENU menu4 = { 4," MENU4 ",
               ".Indice 1 \004A",NoHaceNada,
               ".Indice 2 \004B",NoHaceNada,
               ".Indice 3 \004C",NoHaceNada,
               " Salir   \004S",Salir,
             };
MENU menu5 = { 5," MENU5",
               ".Indice 1 \004A",NoHaceNada,
               ".Indice 2 \004B",NoHaceNada,
               ".Indice 3 \004C",NoHaceNada,
               ".Indice 4 \004D",NoHaceNada,
               " Salir   \004S",Salir,
             };
MENU menu6 = { 6," MENU6",
               ".Indice 1 \004A",NoHaceNada,
               ".Indice 2 \004B",NoHaceNada,
               ".Indice 3 \004C",NoHaceNada,
               ".Indice 4 \004D",NoHaceNada,
               ".Indice 5 \004E",NoHaceNada,
               " Salir   \004S",Salir,
             };


void main(void)
{
        union REGS m;       // Definición de variables necesarias para
        Raton R(m);         // crear el objeto Raton
        Graficos A;         // Definición del objeto Graficos

          if(R.Del_Contr_Raton()) {              // Busca el controlador del ratón y
            registerbgidriver(EGAVGA_driver);    // el adaptador gráfico utilizado
            registerbgidriver(Herc_driver);
            if(A.InicIaGraficos()) {             // Inicia el modo gráfico
              ClearScreen();                     // Limpia la pantalla y coloca
                                                 // el fondo en color gris
              R.Activa_Raton();                  // Activa, el ratón y coloca el cursor de flecha
              Inic_Contr_Vents();
```

```
            Menus MEN(&BarraMenu,0,0,SCREENWIDE,PROFBARRMENU); // Inicializa
                                                                // objetos Menus
            MEN.ColocMenus(&menu1); // Coloca los 6 objetos de tipo menú en pantalla
            MEN.ColocMenus(&menu2);
            MEN.ColocMenus(&menu3);
            MEN.ColocMenus(&menu4);
            MEN.ColocMenus(&menu5);
            MEN.ColocMenus(&menu6);
            MEN.DibBarraMenus();          // Dibuja la barra de menús en la parte
                                          // superior de la pantalla
            ClearScreen();
            do {
              Sel_Indice_Menu(); //
              if (R.Det_Fuera_Raton(&Loc)) {      // Detecta si ha sido presionado un
                                                  // botón del ratón
                rt=Loc_Pos_Raton(&Loc,&Pantalla); // Localiza la posición del ratón en la
                                                  // pantalla
                if(rt & EnBarraMenu) {            // Si se localiza en un menú ?
                    MEN.AgregaMenus(&Loc);        // Coloca los indices del menú debajo del
                                                  // titulo
                 }
                 else MensajeError();     // Si no localiza nada, emite un sonido de error
              }
            } while(actividad);           // Realiza esto, mientras no se seleccione el índice
                                          // para salir del programa

            R.DesactivaRaton();           // Desactiva el ratón y desaparece el cursor
            clrscr();
          } else cout << "No se puede establecer el modo gráfico";
        } else cout << "No se localizo el controlador del ratón";
}

// Esta función no realiza ninguna actividad, mas que mostrar la selección de un índice
int NoHaceNada(void)
{
        return(0);
}

// Esta función permite salir del programa
int Salir(void)
{
        actividad=0;
        return(1);
}
```

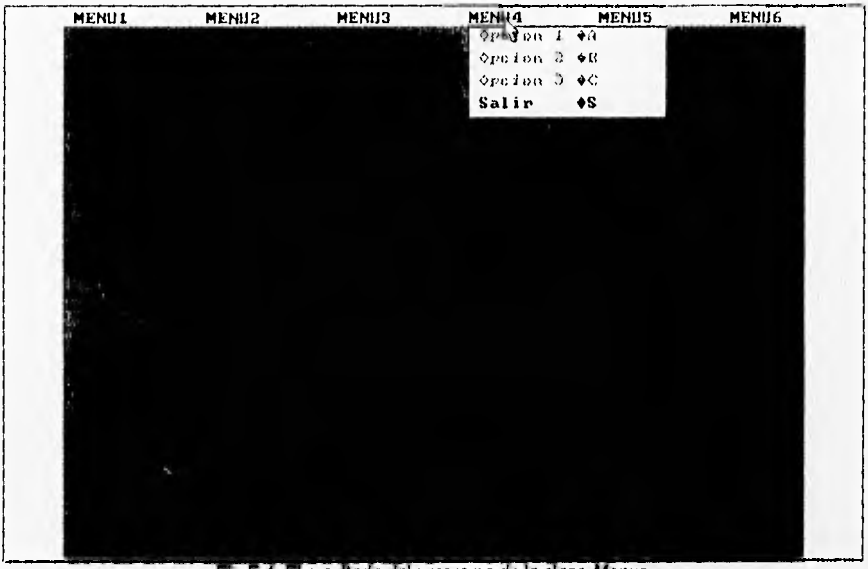


Fig.E.4. El resultado del programa de la clase *Menus*.

PROGRAMA DE PRUEBA PARA LA CLASE *OBJ_TEXTO*

El programa muestra el uso de la clase *Obj_Texto*. Crea cinco objetos dinámicos y los coloca en una ventana. El código y la salida del programa se ilustran a continuación:

```
#define _DATOSIGU    1  // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>       // Contiene la definición de variables globales
#include <int_graf.h>       // Definición de funciones utilizadas en este programa

void main(void)
{    // Se define el texto para los objetos texto
       static char ln[5][36] =   { " OBJETOS TEXTO DE PRUEBA ",
                                   "   TEXTO 1     ",
                                   "   TEXTO 2     ",
                                   "   TEXTO 3     ",
                                   "   TEXTO 4     "
                                 };
```

```
union REGS in;          // Definición de variables necesarias para
Raton R(in);            // crear el objeto Raton
OBJ_TEXTO tx[5];        // Arreglo de 5 elementos OBJ_TEXTO
VENTANA w;              // Define variable de tipo VENTANA
RECUADRO r;             // Define variable de tipo RECUADRO
LOCALIZA p;             // Define variable para localizar objetos en pantalla
static int i;           // Contador
Graficos A;             // Definición del objetos Graficos

if(R.Det_Contr_Raton()) {              // Busca el controlador del ratón y
 registerbgidriver(EGAVGA_driver);     // el adaptador gráfico utilizado
 registerbgidriver(Herc_driver);

 if(A.IniciaGraficos()) {              // Inicia el modo gráfico
   ClearScreen();                      // Limpia la pantalla coloca el fondo en color gris
   R.Activa_Raton();                   // Activa el ratón y coloca el cursor del ratón
   Inic_Contr_Vents();                 // Función que inicia el control de ventanas, necesario
                                       // definirla antes de abrir cualquier ventana
   Obj_Texto *TXT[5];                  // Definicion de 5 objetos dinamicos texto
   Ventana V(&w,&r,200,100,440,280);   // Construye el objeto Ventana

   //Intenta abrir la ventana
   if(V.AbreVentana()) {
     // Agrega los objetos de tipo texto
     for(i=0;i<5;++i)  // Coloca 5 objetos de tipo texto
         {
          TXT[i]=new Obj_Texto(in[i],ACTIVO,&tx[i],&tx[i].tx,&w,
          r.izq+((r.der-r.izq)/2)-((strlen(tn[i])*8)/2),
          r.arriba+50+(i*20));
          TXT[i]->DibujaTexto();
         }

         // Borra los objetos creados dinámicamente
         for(i=0;i<5;++i) delete TXT[i];
     }
     getch();
     R.DesactivaRaton();                    // Desactiva el ratón y desaparece el cursor
     V.CierraVentana();
     clrscr();                              // Limpia la pantalla
   } else cout << "No se puede establecer el modo gráfico";
 } else cout << "No se localizo el controlador del ratón";
}
```

Fig.E.5. Salida del programa de prueba de la clase *Obj_Texto*.

PROGRAMA DE PRUEBA PARA LA CLASE *BOTON*

El siguiente programa hace uso de la clase *Boton*. Coloca un botón, en estado inactivo/activo, dentro de una ventana; el botón permite cerrarla, así como, terminar con el programa (Véase figura E.6.)

```
#define _DATOSIGU   1  // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>      // Contiene la definición de variables globales
#include <int_graf.h>      // Definición de funciones utilizadas en este programa

int rt,actividad=0xff;     // Definición de banderas para el programa

void main(void)
{
        union REGS m;      // Definición de variables necesarias para
        Raton R(m);        // crear el objeto Raton
        Graficos A;        // Definición del objeto Graficos
```

```
// Definición de variables necesarias para los objetos
RECUADRO rec;
VENTANA W;
BOTON ok,*bot;
LOCALIZA Loc;
char cadena[12]="Aceptar"; // Definición de texto que va dentro del objeto Boton

        if(R.Det_Contr_Raton()) {              // Busca el controlador del ratón y
          registerbgidriver(EGAVGA_driver);    // el adaptador gráfico utilizado
          registerbgidriver(Herc_driver);
          if(A.IniciaGraficos()) {             // Inicia el modo gráfico
            ClearScreen();                     // Limpia la pantalla y coloca el
                                               // fondo en color gris
            R.Activa_Raton();                  // Activa el ratón y coloca el cursor del ratón
            inic_Contr_Vents();      // Función que inicia el control de ventanas, es
                                     // necesario definirla antes de abrir cualquier ventana
            Ventana VENT(&W,&rec,48,48,248,248); // inicializa objeto Ventana
            if (VENT.AbreVentana()) {
              Boton B(cadena,ACTIVO,&ok,&ok.tb,&W,  // tricializa el objeto Boton
              &ok.rec,170,220,170+(10*(strlen(cadena))),235);
              B.DibujaBoton();  // Coloca el objeto Boton
              while (actividad) {              // Mientras que no se seleccione el Boton
                  ActivaTeclado(&W);                  // Habilita el uso del teclado
                  if (R.Det_Fuera_Raton(&Loc)) {
                    rt=Loc_Pos_Raton(&Loc,&W);
                    if(rt & EnBoton) {                 // Si localiza el Boton, actividad=0,
                                                       // sale  del programa
                      bot=Loc_Boton(&W,&Loc);
                      Sel_Boton(bot);
                      if (bot==&ok) actividad=0;
                     }
                     else MensajeError(); // Si no esta seleccionando ningún objeto emite el
                                          // sonido de error
                  }
              }
              VENT.CierraVantana();
            } else MensajeError();
            R.DesactivaRaton();                // Desactiva el ralón y desaparece el cursor
            delay(2000);                       // Retardo para mostrar los objetos
            clrscr();                          // Limpia la pantalla
        } else cout <<  "No se puede establecer el modo gráfico";
      } else cout << "No se localizo el controlador del ratón";
}
```

Fig.E.6. Salida generada por el programa de la clase *Boton*.

PROGRAMA DE PRUEBA PARA LA CLASE *MAPABITS*

Este programa coloca 12 objetos de tipo mapa de bits en una ventana, en dos hileras de 6 objetos cada una. El código se muestra a continuación, así como, se ilustra en la fig. E.7 la salida del programa.

```
#define NUMMAXICONOS      12      // Número máximo de iconos definidos
#define _DATOSIGU    1            // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>             // Contiene la definición de variables globales
#include <int_graf.h>             // Definición de funciones utilizadas en este programa
#include <iconos.h>               // Contiene el código para los 12 iconos


char *ap_Iconos[NUMMAXICONOS]={ICONO1,ICONO2,    //Apuntador a los iconos definidos
                                                 // en el archivo iconos.h
                        ICONO3,ICONO4,
                        ICONO5,ICONO6,
                        ICONO7,ICONO8,
                        ICONO9,ICONO10,
                        ICONO11,ICONO12 };

void main(void)
{
        static int I;             // Variable utilizada como contador

// Variables utilizadas para crear los objetos
        VENTANA W;
```

```
RECUADRO r,r1[12];
MAPABITS bmp[12];      // Arreglo de 12 variables MAPABITS
LOCALIZA Loc;
union REGS m;          // Variable necesaria para el objeto Raton
Raton R(m);
Graficos A;            // Objeto Grafico
MapaBits *BMP[12];     // Arreglo dinámico de 12 objetos MapaBits
        if(R.Det_Contr_Raton()) {              // Busca el controlador del ratón y
          registerbgidriver(EGAVGA_driver);    // el adaptador gráfico utilizado
          registerbgidriver(Herc_driver);
          if(A.IniciaGraficos()) {             // Inicia el modo gráfico
            ClearScreen();            // Limpia la pantalla y coloca el fondo en color gris
            R.Activa_Raton();         // Activa, el ratón y coloca el cursor de flecha
            Inic_Contr_Vents();  // Función que inicia el control de ventanas, es necesario
                                 // definirla antes de abrir cualquier ventana
            Ventana VENT(&W,&r,37,38,350,250);
            if (VENT.AbreVentana()) {
              for(i=0;i<12;i++)  // Se colocan los 12 objetos MapaBits en la pantalla
              {
                  if(i<=5)  // Coloca la primera hilera de mapas de bits
                  {
                   BMP[i]=new MapaBits(ap_iconos[i],0x00,
                  NULL,INACTIVO,&bmp[i],&W,&r1[i],
                   r.izq+5,r.arriba+10+(i*33));
                   BMP[i]->Dib_Map_Bits();
                  }
                  else   // Coloca la segunda hilera de mapas de bits
                  {
                   BMP[i]=new MapaBits(ap_iconos[i],0x00,NULL,
                  INACTIVO,&bmp[i],&W,&r1[i],
                   (r.izq+39),r.arriba+10+((i-6)*33));
                   BMP[i]->Dib_Map_Bits();
                  }
              }
            getch();
            for(i=0;i<12;i++)                  // Borra los 12 objetos mapa de bits
             {
              delete BMP[i];
             }
      }

            VENT.CierraVentana();
            R.DesactivaRaton();                // Desactiva el ratón y desaparece el cursor
            getch();
            clrscr();                          // Limpia la pantalla
          } else cout << "No se puede establecer el modo gráfico";
      } else cout << "No se localizó el controlador del ratón";
}
```

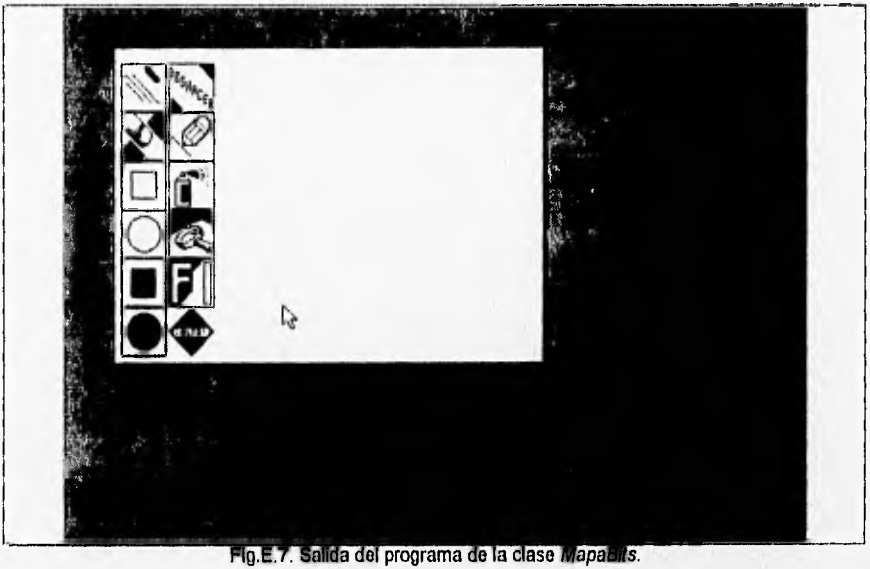


Fig.E.7. Salida del programa de la clase *MapaBits*.

PROGRAMA DE PRUEBA PARA OBJETOS DE TIPO: BARRAS DE DESPLIEGUE, LISTAS, CAMPO TEXTO Y CAMPO EDICION

El siguiente programa sirve para ejemplificar el uso de las clases: *BarrasDeDesp, Campo_Texto, Lista y Campo_Edicion.*

El programa realiza la búsqueda de archivos PCX en el directorio actual; permite la búsqueda en otros directorios y unidades de disco haciendo uso de listas y barras de despliegue, para visualizar toda la información contenida dentro de ese directorio.

```
#define _DATOSIGU   1                  // Usa las variables del archivo vars_igu.h

#include <vars_igu.h>                  // Contiene la definición de variables globales
#include <int_graf.h>                  // Definición de funciones utilizadas en este programa

void main(void)
{   // Definición de variables para crear los objetos
        BARRASDESP bd,*apb;
        LISTA ls,*lp;
        OBJ_TEXTO errt;
        CAMPOTEXTO fespecif;
        CAMPOEDICION nombarch,*ep;
        RECUADRO r,errr;
        VENTANA w,errw;
        LOCALIZA p;
        BOTON ok,can,errok,*boton;
```

```
char *IndEspecif,unidad[MAXDRIVE],nm[24];
char dir[MAXDIR],archivo[MAXFILE],ext[MAXEXT]; // Variables para almacenar la ruta de
                                                // un archivo
unsigned int rt;              // Variable utilizada como bandera de localización de objetos
int loc=0,interna,externa=0xff,erra,i; // Definición de banderas utilizadas por el programa
LOCALIZA Loc;
union REGS m;                 // Variable necesaria para el objeto Raton
Raton R(m);
Graficos A;
char ruta_arch[129],*Unidades="ABCDEFGHIJKLM"; // Almacenan ruta del archivo y
                                                // unidades de disco
char b[129],nombre[16];

// Se crea la ruta para el archivo especificado
getcwd(ruta_arch,128);
if(ruta_arch[strlen(ruta_arch)-1] != '\\')
strcat(ruta_arch,"\\");

// Se crea la ruta para el archivo, en este caso con la extensión PCX
strcpy(b,ruta_arch);
strcat(b,"*.");
strcat(b,"PCX");

r.izq=10;      // Se definen las coordenadas para la ventana
r.arriba=10;
r.der=r.izq+300;
r.abajo=r.arriba+280;

if(R.Det_Contr_Raton()) {                  // Busca el controlador del ratón y
 registerbgidriver(EGAVGA_driver);         // el adaptador gráfico utilizado
 registerbgidriver(Herc_driver);

 if(A.IniciaGraficos()) {                  // Inicia el modo gráfico
  ClearScreen();                           // Limpia la pantalla y coloca el fondo en color gris
  R.Activa_Raton();                        // Activa, el ratón y coloca el cursor de flecha
  Inic_Contr_Vents();                      // Función que inicia el control de ventanas, y es
                                           // necesario definirla antes de abrir cualquier ventana
  Ventana V(&w,&r,r.izq,r.arriba,r.der,r.abajo);
  nombre[0]=0;                             // Inicializa la cadena

  if(V.AbreVentana()) {
   // Se colocan los objetos que aparecerán en la ventana
   Boton B("Aceptar",ACTIVO,&ok,&ok.tb,&w,&ok.rec,
   r.izq+130,r.abajo-25,r.izq+200,r.abajo-10);
   B.DibujaBoton();

   Boton B2("Cancelar",ACTIVO,&can,&can.tb,&w,&can.rec,
   r.izq+210,r.abajo-25,r.izq+290,r.abajo-10);
   B2.DibujaBoton();

   if(strlen(b) <= 35) IndEspecif=b; // Coloca la ruta actual de archivos
   else IndEspecif=b+strlen(b)-35;
   CampoTexto CT(IndEspecif,ACTIVO,35,&fespecif,
   &fespecif.tct,&w,&fespecif.rec,r.izq+8,r.arriba+8);
   CT.DibCampoTexto(); // Coloca el campo de texto con la ruta del archivo
```

```
CD.DibCampoEdicion();                    // Coloca el campo de edición

Lista L(TAMLINARCH,14,0,loc,NULL,&ls,&ls.tl,&w,&ls.rec,r.izq+8,r.arriba+34);
L.Dibuja_Lista();                        // Coloca la lista

BarrasDeDespl BD(ls.rec.abajo-ls.rec.arriba,0,0,0,INACTIVO,&bd,&w,
&bd.rec,ls.rec.der,ls.rec.arriba);
BD.Dib_Bar_Vert();                       // Dibuja la barra de despliegue vertical

do {
    interna=0xff;       // Bandera que indica que se puede trabajar dentro de la ventana
    ls.cont=0;
    R.Cursor(reloj,7,7); // Coloca el cursor de reloj de arena para indicar que se esta
                         // esperando a que se realice una actividad
    ls.tl.texto=Obt_Archivos(b,&ls.cont,Unidades);  // Coloca las cadenas de texto para
                                                    // la lista
    R.Cursor(flecha,0,0);  // Terminada la actividad (lectura de archivos del directorio),
                           // coloca el cursor normal
    ls.arriba=bd.posicion=0;

    if(ls.cont > 14) {  // Calcula las dimensiones para la barra de despliegue vertical
                        // (número de cadenas de texto)
      bd.max=ls.cont-14;
      bd.activo=ACTIVO;
     }
     else {
      bd.max=0;             // Si es igual a cero, no coloca la barra de despliegue vertical
      bd.activo=INACTIVO;
     }
   Ajuste_Area_Contr(&bd,0);  // Ajusta la barra de despliegue para los primeros datos
    L.Dibuja_Lista();               // Coloca la lista
    BD.Dib_Bar_Vert();              // Coloca la barra de despliegue vertical

    while(interna) {      // Mientras que haya actividad en la ventana realiza lo siguiente
      ActivaTeclado(&w);                    // Habilita el uso del teclado para manejo de
                                            // botones y campos de edición

      if(R.Det_Fuera_Raton(&p)) {           // Detecta si se ha selecctonado algún
                                            // objeto con el ratón
        rt=Loc_Pos_Raton(&p,&w);        // Devuelve la posición del ratón en la pantalla

    // Detecta que tipo de objeto  se ha seleccionado con el ratón
      if(rt & EnCampoEdicion) {  // Si se seleccionó este objeto, permite la edición del
                                 // campo
        ep=Loc_Camp_Ed(&w,&p);
        Sel_Camp_Ed(ep);
        RedibCampoEdicion(ep);
       }
       else if(rt & EnBoton) {   // Si se seleccionó alguno de los botones
        boton=Loc_Boton(&w,&p);
        Set_Boton(boton);

        if(boton==&can) {           // Si se escogió este botón, sale del programa
            nombre[0]=0;            // Limpia esta cadena
```

```
            interna=externa=0;    // Sale del programa
          }
          else if(boton==&ok) {  // Si se escogió este botón, seleccionará un archivo,
                                 // unidad o directorio posteriormente
            if(nombre[0] != '\\' && nombre[0] != '[') externa=0;
              interna=0;       // Sale del "do"
            }
        }
        else if(rt & EnDespVert) { // Si esta en la barra, desplegará la información de la
                                   // lista
          apb=Loc_Bar_Desp(&w,&p);
          Set_Bar_Desp(apb,rt);

          if(apb==&bd) {
            is.arriba=apb->posicion;
            L.Dibuja_Lista(); // Actualiza la información de la lista en la ventana
          }
        }
        else if(rt & EnLista) { // Si esta en la lista, despliega su información
          strcpy(nm,nombre);
          lp=Loc_Lista(&w,&p);
          Set_Lista(lp,nombre);
          CD.DibCampoEdicion();
          if(!strcmp(nombre,nm)) {
            if(nombre[0] != '\\' && nombre[0] != '[') externa=0;
            interna=0; // Sale del "do"
            }
        }

      else MensajeError();
      }
    }

    if(ls.tl.texto != NULL) free(ls.tl.texto);  // Limpia el texto de la lista
      if(nombre[0]=='\\') {
        // Cambia de directorio
        fnsplit(b,unidad,dir,archivo,ext);
        if(strcmp(nombre+1,"..")) {
          // Agrega un nuevo directorio
          strcat(dir,nombre+1);
          strcat(dir,"\\");
        }
        else {
          // Respalda un directorio
          i=strlen(dir)-2;
          while(i>0 && dir[i] != '\\') --i;
          dir[i+1]=0;
        }
      nombre[0]=0;  // Limpia la cadena
      fnmerge(b,unidad,dir,archivo,ext); // Ordena el arreglo de cadenas
      CT.DibCampoTexto();
    }
    else if(nombre[0]=='[') {
      // Cambia las unidades de disco
      if(!PruebaDisco(nombre[2]-'A')) {     // Detecta si existe la unidad y si puede leerla
```

```
                fnsplit(b,unidad,dir,archivo,ext);
                unidad[0]=nombre[2];
                strcpy(dir,"\\");
                fnmerge(b,unidad,dir,archivo,ext);   // Ordena el arreglo de cadenas
                nombre[0]=0;                         // Inicializa la cadena
                CT.DibCampoTexto();
               }
               else {  // Coloca un objeto Ventana, para un cuadro de diálogo de error
                Ventana V2(&errw,&errr,r.izq+16,r.arriba+8,r.izq+240,r.arriba+72);

              // Coloca un cuadro de diálogo, indicando que hay un problema con la unidad de
              // disco
                if(V2.AbreVentana()) {
                 Obj_Texto TX("Error leyendo de la
                 unidad",ACTIVO,&errt,&errt.lx,&errw,errr.izq+10,errr.arriba+10);
                 TX.DibujaTexto();
                 Boton B2("Aceptar",ACTIVO,&errok,&errok.tb,
                 &errw,&errok.rec,errr.izq+130,errr.abajo-25,errr.izq+200,errr.abajo-10);
                 B2.DibujaBoton();
                 erra=1;

                 do {  // Maneja la ventana con el cuadro de diálogo de error en la unidad
                     if(R.Del_Fuera_Raton(&p)) {
                       ActivaTeclado(&errw);
                       rt=Loc_Pos_Raton(&p,&errw);
                       if(rt & EnBoton) {
                        boton=Loc_Boton(&errw,&p);
                        Sel_Boton(boton);
                        if(boton==&errok) erra=0;
                       }
                      }
                    } while(erra);              // Mientras no se seleccione el botón
                  V2.CierraVentana();           // Cierra la ventana de error
                 }
                }
               } else externa=0;                // Termina el programa
             } while(externa);
             V.CierraVentana();                 // Cierra la ventana de error
           } else MensajeError();
       R.DesactivaRaton();                      // Desactiva el ratón y desaparece el cursor
       getch();
       clrscr();                                // Limpia la pantalla
     } else cout << "No se puede establecer el modo gráfico";
   } else cout << "No se localizó el controlador del ratón";
}
```

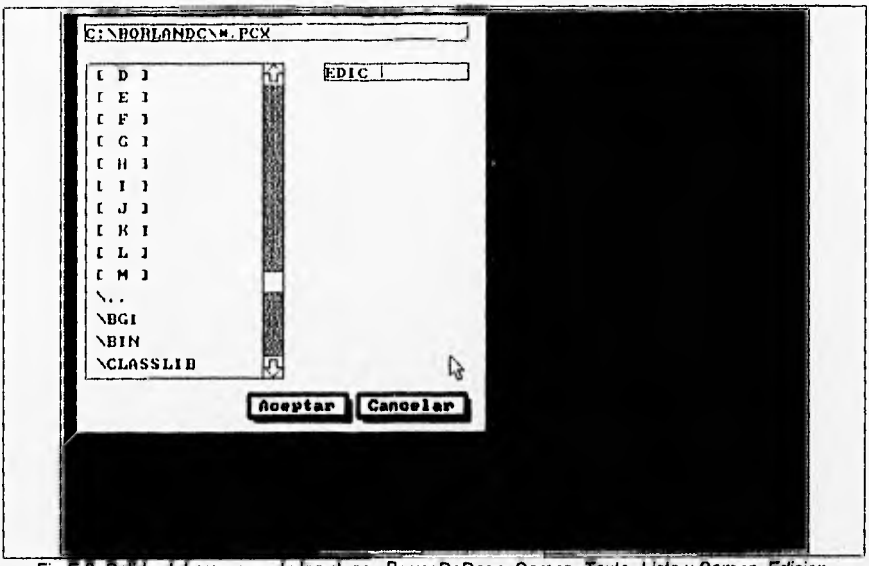


Fig.E.8. Salida del programa de las clases *BarrasDeDesp, Campo_Texto, Lista y Campo_Edicion.*

ARCHIVO VARS_IGU.H

```
#ifndef __VARS_IGU_H
#define __VARS_IGU_H
extern "C" { // Funciones externas contenidas en FUNCS_ASM
ClearScreen(void);  // Limpia la pantalla
WhiteRule(int x,int y); // Coloca una barra blanca para contener los titulos de los menús
DoTable(int x);         // Contiene las tablas de apuntadores para los adaptadores de vídeo
DrawString(int w,int x,char *y,int z); // Coloca una cadena de texto, en las coordenadas x,y
};
#include "iostream.h"
#define REGISTRADR 1   // Coloca con valor de verdadero la bandera para los
                       // manejadores de tarjetas de vídeo de la BGI
#define VENTANARAP 1   // Coloca con valor de verdadero la bandera para utilizar
                       // las funciones pera manejar ventanas rápidas

#define INDMAXMENUS        16 // Máximo número de índices para un menú
#define TAMLINMENUS    18 // Máxima longitud para un índice de un menú
#define PROFLINMENUS   15 // Largo para la línea de menú
#define CONTMENUS      7 // Número de menús
#define TAMTITMENUS    12 // Máxima longitud para el título de un menú
#define PROFBARRMENU       13 // El largo para la barra de menús
#define SOMBRAVENT          4 // Tamaño para el sombreado de la ventana
#define TAMLINARCH  14 // Indica el tamaño para una línea de archivo

// Definición de algunos caracteres extendidos del teclado
#define BORRAR             83 * 256
#define INICIO             71 * 256
```

```
#define CURSOR_IZQ          75 * 256
#define CURSOR_DER          77 * 256
#define FIN                 79 * 256

// Definición de teclas alternas
#define ALT_Q               0x1000
#define ALT_W               0x1100
#define ALT_E               0x1200
#define ALT_R               0x1300
#define ALT_T               0x1400
#define ALT_Y               0x1500
#define ALT_U               0x1600
#define ALT_I               0x1700
#define ALT_O               0x1800
#define ALT_P               0x1900
#define ALT_A               0x1E00
#define ALT_S               0x1F00
#define ALT_D               0x2000
#define ALT_F               0x2100
#define ALT_G               0x2200
#define ALT_H               0x2300
#define ALT_J               0x2400
#define ALT_K               0x2500
#define ALT_L               0x2600
#define ALT_Z               0x2C00
#define ALT_X               0x2D00
#define ALT_C               0x2E00
#define ALT_V               0x2F00
#define ALT_B               0x3000
#define ALT_N               0x3100
#define ALT_M               0x3200
#define ALT_1               0x7800
#define ALT_2               0x7900
#define ALT_3               0x7a00
#define ALT_4               0x7b00
#define ALT_5               0x7c00
#define ALT_6               0x7d00
#define ALT_7               0x7e00
#define ALT_8               0x7f00
#define ALT_9               0x8000
#define ALT_0               0x8100

// Máscaras para los objetos definidos en la interfaz
#define ANada               0x0000
#define EnVentana           0x0001
#define EnBarraMenu         0x0002
#define EnBoton             0x0004
#define EnMapaBits          0x0008
#define EnTexto             0x0010
#define EnCampoTexto        0x0020
#define EnLista             0x0040
#define EnDespVert          0x0080
#define EnDespHor           0x0100
#define FlechaSup           0x0200
#define FlechaInf           0x0400
```

```
#define HaciaArriba     0x0800
#define HaciaAbajo      0x1000
#define EnArea                  0x2000
#define EnCuadDial      0x4000
#define EnCampoEdicion          0x8000

#define VERIFICA        251


// Banderas para indicar el estado de los objetos
#define ACTIVO                  0xffff
#define INACTIVO        0xaa55

// Caracter para indicar el equivalente en el teclado para los menús
#define TECLAMENU   4

// Definición de macros para la interfaz
#define pixels2bytes(n) ((n+7)/8)
#define pixels2words(n)((n+15)/16)
#define AnchoImagen(p)          (1+p[0]+(p[1] << 8))
#define LargoImagen(p)          (1+p[2]+(p[3] << 8))
#define BytesImagen(p)(pixels2bytes((1+p[0]+(p[1] << 8))))

#define MensajeError()  cout << '\a' // El sonido de campana

/*****************************************************************/
// Definición de variables struct utilizadas en la interfaz
/*****************************************************************/

// struct para apuntador a objetos
typedef struct {
        unsigned int tipo;  // Contiene el tipo del objeto
        void *sig;  // apuntador al siguiente objeto
        }APUNTADOR;

// struct para localizar coordenadas en la pantalla
typedef struct {
        int x,y;  // Define coordenadas
        }LOCALIZA;

// struct para definir las coordenadas de un recuadro
typedef struct {
        int izq,arriba,der,abajo;
        }RECUADRO;

// struct para definir un índice de un menú
typedef struct {
        char nombre[TAMLINMENUS+1]; // Contiene el nombre de los índices de un menú
        int (*func)(); // Función que se ejecuta, si se selecciona este índice
        }INDMENU;


// struct que contabiliza y contiene el título de un menú
typedef struct {
        int cont;                       // Lleva la cuenta de los menús
```

```
        char titulo[TAMTITMENUS+1]; // Contiene los títulos para los menús
        }TITMENU;

// struct para definir un menú completo incluye título e índices
typedef struct {
        int cont;  // Lleva la cuenta de los índices de los menús
        char titulo[TAMTITMENUS+1]; // Contiene los títulos para los menús
        INDMENU indice[INDMAXMENUS]; // Almacena los índices de los menús
        }MENU;

// struct para definir una ventana
typedef struct {
        APUNTADOR apunta;  // struct de tipo APUNTADOR para la ventana
        RECUADRO rec;      // struct de tipo RECUADRO para la ventana
        unsigned char *reg;    // Contiene el tamaño de la ventana
        }VENTANA;

typedef struct {
        int x,y;              // Coordenadas iniciales para ta cadena de texto
        char *texto;          // Contiene la cadena de texto
        }TEXTO;

typedef struct {
        APUNTADOR apunta;     // struct de tipo APUNTADOR para el mapa de bits
        int x,y;              // Coordenadas iniciales del mapa de bits
        RECUADRO rec;         // struct RECUADRO para el mapa de bits
        char sel;             // Almacena el estado de selección del MAPABITS
        unsigned int activo;  // Indica el estado del botón (ACTIVO/INACTIVO)
        char *mapabits;       // Contiene el mapa de bits
        int (*func)();        // Apuntador a una función que se llama cuando se selecciona un mapa
                              // de bits
        }MAPABITS;


typedef struct {
        APUNTADOR apunta;     // struct de tipo APUNTADOR para el texto
        TEXTO tx;             // struct conteniendo el texto
        unsigned int activo;  // Indica el estado del objeto texto (ACTIVO/INACTIVO)
        }OBJ_TEXTO;

// struct para definir un botón
typedef struct {
        APUNTADOR apunta;  // struct de tipo APUNTADOR para el botón
        RECUADRO rec;      // Contiene una struct de tipo RECUADRO
        TEXTO tb           // Contiene el texto para el botón
        unsigned int activo;   // Indica el estado del botón (ACTIVO/INACTIVO)
        }BOTON;

typedef struct {
        APUNTADOR apunta;     // struct de tipo APUNTADOR, para el campo de texto
        TEXTO tcl;            // struct conteniendo el texto
        int largo;            // Indica la longitud de la cadena
        RECUADRO rec;         // struct RECUADRO para el campo de texto
        unsigned int aclivo;  // Indica el estado del campo de texto (ACTIVO/INACTIVO)
        }CAMPOTEXTO;
```

```
typedef struct {
        APUNTADOR apunta; // struct de tipo APUNTADOR, para el objeto lista
        TEXTO tl; // struct conteniendo el texto
        int largo,ancho; // Dimensiones para la lista
        int cont;       // Contador de cadenas de texto
        int arriba;     // Inicio de la lista
        RECUADRO rec; // struct RECUADRO para la lista
        }LISTA;

typedef struct {
        APUNTADOR apunta; // struct de tipo APUNTADOR, para el objeto barras de despliegue
        int x,y;   // Coordenadas iniciales de la barra de despliegue
        int tam;  // Tamaño de la barra de despliegue
        RECUADRO rec,flecharr,flechab,area,despl;  // 5 variables RECUADRO para las barras
                                                   // de despliegue
        unsigned int activo;   // Estado de selección de la barra de despliegue
        int min,max,posicion; // Posición del cursor de la barra de despliegue
        }BARRASDESP;

typedef struct {
        APUNTADOR apunta; // struct de tipo APUNTADOR, para el objeto CUADDIAL
        TEXTO tcd;   // struct conteniendo el texto del CUADDIAL
        RECUADRO rec;  // Variable RECUADRO que contiene el cuadro de diálogo
        char sel;           // Estado de selección del cuadro de diálogo
        unsigned int activo;  // Indica el estado del cuadro de diálogo (ACTIVO/INACTIVO)
        }CUADDIAL;

typedef struct {
        APUNTADOR apunta; // struct de tipo APUNTADOR, para el objeto CAMPOEDICION
        TEXTO tce; // struct conteniendo el texto del CAMPOEDICION
        int largo;     Largo del recuadro de edición
        int poscur;   // Posición del cursor para la edición
        int (*func)();  // Función que se ejecuta si es seleccionado el campo de edición
        char sel; // Estado de selección del CAMPOEDICION
        RECUADRO rec;
        unsigned int activo;  // Indica el estado del CAMPOEDICION (ACTIVO/INACTIVO)
        }CAMPOEDICION;

typedef struct {          // La configuración de los datos para la interfaz
        char sig[4];
        int version;
        int borramin;
        int borramax;
        int anchodefault;
        int largodefault;
        char Unidades[27];
        } CONFIG;
```

```
/*****************************************************************************/
//                  Cursores para el ratón, 4 tipos: de flecha, reloj, mano.
/*****************************************************************************/

char flecha[] = {
0xFF,0x3F,0xFF,0x1F,0xFF,0x0F,0xFF,0x07,0xFF,0x03,0xFF,0x01,0xFF,0x00,0x7F,0x00,
0x3F,0x00,0x1F,0x00,0xFF,0x01,0xFF,0x10,0xFF,0x30,0x7F,0xF8,0x7F,0xF8,0x7F,0xFC,
0x00,0x00,0x00,0x40,0x00,0x60,0x00,0x70,0x00,0x78,0x00,0x7C,0x00,0x7E,0x00,0x7F,
0x80,0x7F,0x00,0x7C,0x00,0x6C,0x00,0x46,0x00,0x06,0x00,0x03,0x00,0x03,0x00,0x00 };

char reloj[] = {
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x80,0x03,0xC0,0x07,0xE0,0x0F,0xF0,0x07,0xE0,
0x03,0xC0,0x01,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,
0x00,0x00,0xFE,0x7F,0x06,0x60,0x0C,0x30,0x18,0x18,0x30,0x0C,0x60,0x06,0xC0,0x03,
0x60,0x06,0x30,0x0C,0x98,0x19,0xCC,0x33,0xE6,0x67,0xFE,0x7F,0x00,0x00,0x00,0x00 };

char mano[] = {
0xff,0xf3,0xff,0xe1,0xff,0xe1,0xff,0xe1,0x01,0xe0,0x00,0xe0,0x00,0xe0,0x00,0xe0,
0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x80,0x03,0xc0,
0x00,0x00,0x00,0x0C,0x00,0x0C,0x00,0x0C,0x00,0x0C,0x00,0x0C,0xB6,0x0D,0xB6,0x0D,0xB6,0x0D,
0xB6,0x0D,0xFE,0x6F,0xFE,0x6F,0xFE,0x6F,0xFE,0x7F,0xFE,0x7F,0xFC,0x3F,0x00,0x00 };

// Variables externas contenidas en FUNCS_ASM (librería en ensamblador)
extern unsigned int SCREENTBL[];
extern unsigned int SCREENSEG,SCREENBYTES;
extern unsigned int SCREENWIDE,SCREENDEEP;

#if _DATOSIGU // Inicialización de variables
int Modografico,Tarj_Herc,IndiceMenu=0;
LOCALIZA RatonAfuera={-1,-1};
MENU *ArregloMenu[CONTMENUS];
VENTANA Pantalla;
RECUADRO BarraMenu;        // Contiene la barra de menús
int Prob_Disco;            // Variable para detectar problemas en el disco
char Error_Mem[129];  // Variable para almacenar un directorio equivocado
char Rell_Barra_Desp[]="\063\314\063\314\063\314\063\314"; // Relleno de la barra de despliegue

// Teclas alternas
unsigned int Tab_Tec_Alt[]= {
        ALT_Q,'Q',ALT_W,'W',ALT_E,'E',ALT_R,'R',ALT_T,'T',ALT_Y,'Y',ALT_U,'U',ALT_I,'I',
        ALT_O,'O',ALT_P,'P',ALT_A,'A',ALT_S,'S',ALT_D,'D',ALT_F,'F',ALT_G,'G',ALT_H,'H',
        ALT_J,'J',ALT_K,'K',ALT_L,'L',ALT_Z,'Z',ALT_X,'X',ALT_C,'C',ALT_V,'V',ALT_B,'B',
        ALT_N,'N',ALT_M,'M',ALT_1,'1',ALT_2,'2',ALT_3,'3',ALT_4,'4',ALT_5,'5',ALT_6,'6',
        ALT_7,'7',ALT_8,'8',ALT_9,'9',ALT_0,'0',0xffff
                };

#endif
#endif //VARS_IGU_H
```

ARCHIVO ICONOS.H

```
// ***************************************************************
//        Contiene los iconos para la interfaz gráfica
// ***************************************************************

// ICONO1, es el icono para seleccionar la herramienta que permite dibujar lineas rectas
char ICONO1[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x00,0x00,0x00,0xcc,0x80,0x00,0x70,0x00,0xfb,0x80,0x00,0x78,0x00,0x8f,0
x80,0x00,0x7c,0x00,0x86,0x80,0x00,0x7e,0x00,0xff,0x80,0x00,0x3f,0x00,0x95,0x80,0x00,
0x1f,0x80,0x80,0x90,0x80,0x0f,0xc0,0xfd,0x88,0x40,0x07,0xe0,0xcc,0x84,0x20,0x03,0xf0,0xfb,
0x82,0x10,0x01,0xf0,0x8f,0x81,0x08,0x00,0xf0,0x86,0x90,0x84,0x00,0x00,0xff,0x88,0x42,0x00,
0x00,0x95,0x84,0x21,0x00,0x00,0x80,0x82,0x10,0x80,0x00,0xfd,0x81,0x08,0x40,0x00,0xcc,0x80,0x84,0
x20,0x00,0xfb,0x80,0x42,0x10,0x00,0x8f,0x80,0x21,0x08,0x00,0x86,0x80,0x10,0x84,0x00,
0xff,0x80,0x08,0x42,0x00,0x95,0x80,0x04,0x21,0x00,0x80,0x80,0x02,0x10,0x80,0xfd,0x80,0x01,
0x08,0x40,0xcc,0x80,0x00,0x04,0x20,0xfb,0x80,0x00,0x00,0x10,0x8f,0x80,0x00,0x00,0x00,0x86,
0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x95,0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO2, es el icono para seleccionar la herramienta que permite borrar
char ICONO2[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x40,0x00,0xd5,0xcc,0x80,0x20,0x01,0xaa,0xfb,0x80,0x10,0x03,0x55,0x8f,0
x80,0x08,0x06,0xab,0x86,0x80,0x04,0x0d,0x57,0xff,0x80,0x02,0x1a,0xaf,0x95,0x80,0x01,
0x35,0x5f,0x80,0x80,0x02,0x9a,0xbf,0xfd,0xc0,0x05,0x4d,0x7f,0xcc,0xa0,0x0a,0xa6,0xff,0xfb,
0xd4,0x15,0x13,0xff,0x8f,0xaa,0xaa,0x09,0xff,0x86,0x95,0x54,0x04,0xff,0xff,0x8a,0xac,0x04,0x7f,0x95,
0x85,0x5c,0x02,0x3e,0x80,0x82,0xbc,0x02,0x1c,0xfd,0x81,0x78,0x04,0x08,0xcc,0x80,0xf8,
0x04,0x04,0xfb,0x82,0x78,0x08,0x02,0x8f,0x87,0x3e,0x08,0x01,0x86,0x8d,0x9f,0xf0,0x00,0xff,
0x9a,0xcf,0xc0,0x00,0x95,0xb5,0x63,0x00,0x00,0x80,0xea,0xb0,0x00,0x00,0xfd,0xd5,0x78,0x00,
0x00,0xcc,0xaa,0xfc,0x00,0x00,0xfb,0xd5,0xfe,0x00,0x00,0x8f,0xab,0xff,0x00,0x00,0x86,0xd7,
0xff,0x80,0x00,0xff,0xaf,0xff,0x40,0x00,0x95,0xdf,0xfe,0x20,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO3, es el icono para seleccionar la herramienta que permite dibujar cuadros
char ICONO3[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x00,0x00,0x00,0xcb,0x80,0x00,0x00,0x00,0xfb,0x80,0x00,0x00,0x00,0x8f,0
x80,0x00,0x00,0x00,0x86,0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x95,0x81,0xff,
0xff,0xc0,0x80,0x81,0x00,0x00,0x40,0xfd,0x81,0x00,0x00,0x40,0xcb,0x81,0x00,0x00,0x40,0xfb,
0x81,0x00,0x00,0x40,0x8f,0x81,0x00,0x00,0x40,0x86,0x81,0x00,0x00,0x40,0xff,0x81,0x00,0x00,
0x40,0x95,0x81,0x00,0x00,0x40,0x80,0x81,0x00,0x00,0x40,0xfd,0x81,0x00,0x00,0x40,0xcb,0x81,0x00,0
x00,0x40,0xfb,0x81,0x00,0x00,0x40,0x8f,0x81,0x00,0x00,0x40,0x86,0x81,0x00,0x00,0x40,
0xff,0x81,0x00,0x00,0x40,0x95,0x81,0x00,0x00,0x40,0x80,0x81,0x00,0x00,0x40,0xfd,0x81,0xff,
0xff,0xc0,0xcb,0x80,0x00,0x00,0x00,0xfb,0x80,0x00,0x00,0x00,0x8f,0x80,0x00,0x00,0x00,0x86,
0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x95,0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO4, es el icono para seleccionar la herramienta que permite dibujar circulos y elipses
char ICONO4[]={
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x00,0x00,0x00,0xca,0x80,0x00,0x00,0x00,0xfb,0x80,0x00,0x00,0x00,0x8f,0
x80,0x07,0xf0,0x00,0x86,0x80,0x18,0x0c,0x00,0xff,0x80,0x60,0x03,0x00,0x95,0x80,0x80,
```

```
0x00,0x80,0x80,0x81,0x00,0x00,0x40,0xfd,0x82,0x00,0x00,0x20,0xca,0x82,0x00,0x00,0x20,0xfb,
0x84,0x00,0x00,0x10,0x8f,0x84,0x00,0x00,0x10,0x86,0x88,0x00,0x00,0x08,0xff,0x88,0x00,0x00,
0x08,0x95,0x88,0x00,0x00,0x08,0x80,0x88,0x00,0x00,0x08,0xfd,0x88,0x00,0x00,0x08,0xca,0x88,0x00,0
x00,0x08,0xfb,0x88,0x00,0x00,0x08,0x8f,0x84,0x00,0x00,0x10,0x86,0x84,0x00,0x00,0x10,
0xff,0x82,0x00,0x00,0x20,0x95,0x82,0x00,0x00,0x20,0x80,0x81,0x00,0x00,0x40,0xfd,0x80,0x80,
0x00,0x80,0xca,0x80,0x60,0x03,0x00,0xfb,0x80,0x18,0x0c,0x00,0x8f,0x80,0x07,0xf0,0x00,0x86,
0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x95,0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO5, es el icono para seleccionar la herramienta que permite  dibujar cuadros con relleno
char ICONO5[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x00,0x00,0x00,0xcb,0x80,0x00,0x00,0x00,0xfb,0x80,0x00,0x00,0x00,0x8f,0
x80,0x00,0x00,0x00,0x86,0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x95,0x81,0xff,
0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,
0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,
0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80
0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,
0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x81,0xff,0xff,0x80,0x80,0x80,0x00,
0x00,0x00,0xfb,0x80,0x00,0x00,0x00,0x8f,0x80,0x00,0x00,0x00,0x86,0x80,0x00,0x00,0x00,0xff,
0x80,0x00,0x00,0x00,0x95,0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO6, es el icono para seleccionar la herramienta que permite
// dibujar círculos y elipses con relleno
char ICONO6[]={
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x00,0x00,0x00,0xca,0x80,0x00,0x00,0x00,0xfb,0x80,0x00,0x00,0x00,0x8f,0
x80,0x07,0xf0,0x00,0x86,0x80,0x1f,0xfc,0x00,0xff,0x80,0x7f,0xff,0x00,0x95,0x80,0xff,0xff,
0x80,0x80,0x81,0xff,0xff,0xc0,0xfd,0x83,0xff,0xff,0xe0,0xca,0x83,0xff,0xff,0xe0,0xfb,0x87,0xff,0xff,0xf0,
0x8f,0x87,0xff,0xff,0xf0,0x86,0x8f,0xff,0xff,0xf8,0xff,0x8f,0xff,0xff,0xf8,0x95,0x8f,0xff,0xff,
0xf8,0x80,0x8f,0xff,0xff,0xf8,0xfd,0x8f,0xff,0xff,0xf8,0xca,0x8f,0xff,0xff,0xf8,0xfb,0x8f,0xff,0xff,
0xf8,0x8f,0x87,0xff,0xff,0xf0,0x86,0x87,0xff,0xff,0xf0,0xff,0x83,0xff,0xff,0xe0,0x95,0x83,0xff,0xff,
0xe0,0x80,0x81,0xff,0xff,0xc0,0xfd,0x80,0xff,0xff,0x80,0xca,0x80,0x7f,0xff,0x00,0xfb,0x80,0x1f,
0xfc,0x00,0x8f,0x80,0x07,0xf0,0x00,0x86,0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x95,
0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO7, es el icono para seleccionar la herramienta que permite
// deshacer la última operación realizada
char ICONO7[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x00,0x00,0x03,0xff,0xff,0x00,0x00,0x01,0xff,0xff,0x60,0x00,0x00,0xff,0xff,
0x50,0x00,0x00,0x7f,0xff,0x50,0x00,0x00,0x3f,0xff,0x57,0x00,0x00,0x1f,0xff,0x55,0x00,0x00,0x0f,0xff,0
x54,0x00,0x00,0x07,0xff,0x67,0x70,0x00,0x03,0xff,0x04,0x50,0x00,0x01,0xff,0x85,0x40,
0x00,0x00,0xff,0x87,0x75,0x00,0x00,0xff,0x80,0x15,0x00,0x00,0xff,0x80,0x55,0x00,0x00,0xff,
0x80,0x77,0x70,0x00,0xff,0x80,0x05,0x50,0x00,0xff,0x80,0x05,0x50,0x00,0xff,0x80,0x05,0x77,
0x00,0xff,0x80,0x00,0x55,0x00,0xff,0x80,0x00,0x54,0x00,0xff,0x80,0x00,0x54,0x70,0xff,
0xc0,0x00,0x04,0x50,0xff,0xe0,0x00,0x05,0x40,0x00,0xf0,0x00,0x07,0x77,0x00,0xf8,0x00,0x00,
0x45,0x00,0xfc,0x00,0x00,0x55,0x00,0xfe,0x00,0x00,0x77,0x00,0xff,0x00,0x00,0x06,0x00,0xff,0x80,0x0
0,0x05,0x00,0xff,0xc0,0x00,0x05,0x00,0xff,0xe0,0x00,0x00,0x00,
0xff,0xff,0xff,0xff,0xff
```

```
};

// ICONO8, es el icono para seleccionar la herramienta que permite dibujar
char ICONO8[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x07,0xf0,
0xff,0x80,0x00,0x0c,0x18,0xff,0x80,0x00,0x38,0x0c,0xff,0x80,0x00,0x74,0x04,0xff,0x80,0x00,0xf2,0x04,
0xff,0x80,0x01,0x91,0x04,0xff,0x80,0x03,0x18,0x84,0xff,0x80,0x06,0x24,0x44,0xff,0x80,
0x0c,0x42,0x2c,0xff,0x80,0x18,0x83,0x1c,0xff,0x80,0x31,0x04,0x98,0xff,0x80,0x22,0x08,0x70,
0xff,0x80,0x34,0x10,0x60,0xff,0xa0,0x28,0x20,0xc0,0xff,0x90,0x24,0x41,0x80,0xff,0x88,0x22,
0x83,0x00,0xff,0x84,0x21,0x06,0x00,0xff,0x82,0x30,0x8c,0x00,0xff,0x81,0x38,0x58,0x00,0xff,
0x80,0xbf,0xf0,0x00,0xff,0x80,0x40,0x00,0x00,0xff,0x80,0x20,0x00,0x00,0xff,0x80,0x10,0x00,
0x00,0xff,0x80,0x08,0x00,0x00,0xff,0x80,0x04,0x00,0x00,0xff,0x80,0x02,0x00,0x00,0xff,0x80,
0x01,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0xff,
0xff,0xff,0xff,0xff,0xff
};

// ICONO9, es el icono para seleccionar la herramienta que permite pintar con spray
char ICONO9[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0x80,0x80,0x00,0x00,0x00,0x80,0x80,0x00,0x00,0x00,0x80,0x80,0x00,0x00,
0x08,0x80,0x80,0x00,0x00,0x20,0x80,0x80,0x00,0x02,0x88,0x80,0x80,0x00,0xa9,0x20,0x80,0x80,0x27,
0xf4,0x88,0x80,0x80,0x5f,0xfa,0x40,0x80,0x80,0x53,0xfd,0x90,0x80,0x80,0xf8,0xea,0x48,
0x80,0x81,0x04,0x1c,0xa0,0x80,0x82,0x02,0x01,0x00,0x80,0x87,0xff,0x00,0x40,0x80,0x84,0x01,
0x00,0x10,0x80,0x84,0x01,0x00,0x00,0x80,0x84,0x7f,0x00,0x08,0x80,0x84,0x7f,0x00,0x00,0x80,
0x84,0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,
0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,
0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,0x7f,0x00,0x00,0x80,0x84,0x01,0x00,0x00,
0x80,0x84,0x01,0x00,0x00,0x80,0x87,0xff,0x00,0x00,0x80,0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xfd
};

// ICONO10, es el icono para seleccionar la herramienta que permite
// hacer una ampliación de un fragmento del dibujo, y editarlo pixel por pixel
char ICONO10[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0x80,0xff,0xff,0xff,0xff,0x80,0xff,0xff,0xff,0xff,0x80,0xff,0xff,0xff,0xff,0x80,
0xff,0xff,0xff,0xff,0x80,0xff,0xff,0xff,0xff,0x80,0xff,0xff,0xfe,0xff,0x80,0xff,0xdf,0xf8,0x3f,0x80,
0xff,0xaf,0xc0,0x0f,0x80,0xfd,0x5f,0x00,0x03,0x80,0xf2,0xbc,0x00,0x01,0x80,0xd7,0xf1,0xfc,0x01,0x80,
0xab,0xcf,0x07,0x01,0x80,0xdf,0x10,0x7b,0x01,0x80,0xbe,0x20,0x7b,0x81,0x80,0xf8,0xc3,
0x87,0x80,0x80,0xe1,0x0c,0xfb,0x80,0x80,0xc2,0x10,0x10,0x80,0x80,0x83,0x20,0x21,0x00,0x80,0x84,0
x40,0x6f,0x00,0x80,0x84,0x41,0x93,0x80,0x80,0x84,0x02,0x38,0x70,0x80,0x87,0xfc,0x7c,
0x0c,0x80,0x83,0xc0,0x3f,0x03,0x80,0x81,0xf0,0x6f,0xc0,0x80,0x80,0xf7,0x83,0xe7,0x80,0x80,
0x3c,0x00,0xfc,0x80,0x80,0x00,0x00,0x3f,0x80,0x80,0x00,0x00,0x1f,0x80,0x80,0x00,0x00,0x00,
0x80,0x80,0x00,0x00,0x00,0x80,0x80,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xff
};

// ICONO11, es el icono creado para representar a la Facultad de Ingeniería
char ICONO11[] = {
0x20,0x00,0x20,0x00,
0xff,0xff,0xff,0xff,0xfd,0xff,0xff,0xff,0x80,0xff,0xff,0xff,0xfe,0x7e,0xff,0xe0,0x00,0xfe,0x42,0xff,
0xe0,0x00,0xf8,0x42,0xff,0xe0,0x00,0xf6,0x42,0xff,0xe0,0x00,0xe0,0x42,0xff,0xe1,0xff,0xe0,0x42,0xff,0
xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,
0x42,0xff,0xe0,0x01,0x80,0x42,0xff,0xe0,0x01,0x80,0x42,0xff,0xe0,0x01,0x80,0x42,0xff,0xe0,
```

```
0x01,0x80,0x42,0xff,0xe0,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,0xff,
0xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,0xff,0xe1,0xff,0x80,0x42,
0xff,0xe1,0xfe,0x00,0x42,0xff,0xe1,0xfe,0x00,0x42,0xff,0xe1,0xf8,0x00,0x42,0xff,0xe1,0xf8,0x00,
0x42,0xff,0xe1,0xe0,0x00,0x42,0xff,0xe1,0xe0,0x00,0x42,0xff,0xe1,0x80,0x00,0x42,0xff,0xff,0x80,0x00,
0x7e,0xff,0xfe,0x00,0x00,0x00,0xff,
0xff,0xff,0xff,0xff,0xff
};

// ICONO12, es el icono que indica se tenga precaución para realizar la siguiente actividad del
// sistema
char ICONO12[] = {
0x20,0x00,0x20,0x00,
0x00,0x00,0x80,0x00,0x00,0x00,0x01,0xc0,0x00,0x00,0x00,0x03,0xe0,0x00,0x00,0x00,0x07,0xf0,0x00,0
x00,0x00,0x0f,0xf8,0x00,0x00,0x00,0x1f,0xfc,0x00,0x00,0x00,0x3f,0xfe,0x00,0x00,0x00,
0x7f,0xff,0x00,0x00,0x00,0xff,0xff,0x80,0x00,0x01,0xff,0xff,0xc0,0x00,0x03,0xff,0xff,0xe0,0x00,
0x07,0xff,0xff,0xf0,0x00,0x0f,0xff,0xff,0xf8,0x00,0x1f,0xff,0xff,0xfc,0x00,0x38,0x00,0x00,0x0e,
0x00,0x72,0x74,0xee,0x47,0x00,0xe0,0x54,0x4a,0x43,0x80,0x72,0x74,0x4a,0x07,0x00,
0x32,0x57,0x4e,0x46,0x00,0x18,0x00,0x00,0x0c,0x00,0x0f,0xff,0xff,0xf8,0x00,0x07,0xff,0xff,0xf0,
0x00,0x03,0xff,0xff,0xe0,0x00,0x01,0xff,0xff,0xc0,0x00,0x00,0xff,0xff,0x80,0x00,0x00,0x7f,0xff,
0x00,0x00,0x00,0x3f,0xfe,0x00,0x00,0x00,0x1f,0xfc,0x00,0x00,0x00,0x0f,0xf8,0x00,0x00,0x00,
0x07,0xf0,0x00,0x00,0x00,0x03,0xe0,0x00,0x00,0x00,0x01,0xc0,0x00,0x00,
0x00,0x00,0x80,0x00,0x00
};
```

APENDICE F. MANUAL DE LA APLICACION

Presentación del programa de aplicación

La parte inicial de este programa de aplicación, muestra una ventana con la presentación del programa y habilita el uso del ratón (Ver figura F.1.) Si es seleccionado con el cursor el botón de *Continuar*, que se encuentra en la parte inferior de la ventana, permitirá proseguir con el programa de edición de archivos PCX (herramienta de dibujo.)

Fig. F.1. Pantalla de presentación del programa de edición de archivos *PCX*

Una vez presionado el botón de *Continuar*, permite pasar a la siguiente pantalla (ver fig. F.2.), desde la cual se puede elegir con el cursor del ratón, alguno de los cuatro menús que se encuentran en la barra de menús *(Escritorio, Archivo, Opciones o Ayuda.)*

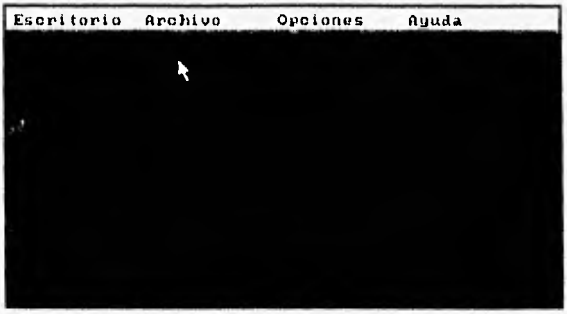


Fig. F.2. Barra de menús y títulos para el programa de aplicación.

Menús del programa

Si es elegido el menú de *Escritorio*, aparecerá un menú colgante debajo de éste, mostrando la opción de *Referencia*

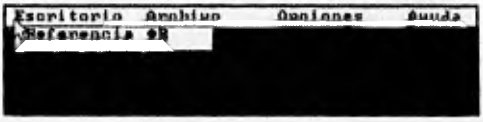


Fig. F.3. Selección del menú de *Escritorio*.

Si se mantiene presionado el botón del ratón sobre la opción de *Referencia* o desde el teclado con Alt-R, aparecerá la siguiente ventana indicando la referencia del programa (su nombre y versión.)

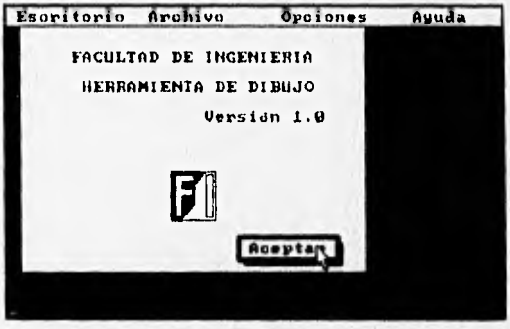


Fig. F.4. Ventana de referencia del programa (nombre y número de versión.)

Si se elige el menú de *Archivo*, éste abrirá un menú colgante, del cual aparecerán como opciones activas solamente *Nuevo, Abrir y Salir*, que podrán ser seleccionadas. Estas opciones podrán ser seleccionadas también a través de sus teclas alternas: *Alt-N* para *Nuevo*, *Alt-A* para *Abrir* y *Alt-L* para *Salir*.

Fig. F.5. Menú de *Archivo* y opciones activas.

La opción *Nuevo* presenta la siguiente pantalla (Fig. F.6), en la cual se pueden definir las dimensiones para el nuevo archivo PCX; las dimensiones por *default* son 640 x 480 pixeles, pero pueden ser cambiadas si se posiciona el cursor del ratón sobre estos cuadros y son seleccionados, se podrán editar estos valores con los que prefiera el usuario. Una vez que son colocados los nuevos valores, se selecciona el botón de *Aceptar* para proseguir con el programa, o el botón de *Cancelar* para salir de esta ventana y regresar a la ventana anterior.

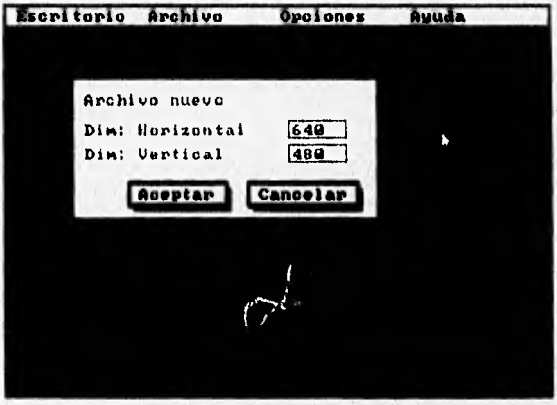


Fig. F.6. Ventana para especificar un archivo nuevo

Si se seleccionó el botón de *Aceptar* y las dimensiones fueron válidas, aparecerá la siguiente ventana, la cual permite empezar la edición del archivo PCX por medio de las herramientas de dibujo que se encuentran en la parte izquierda de la ventana.

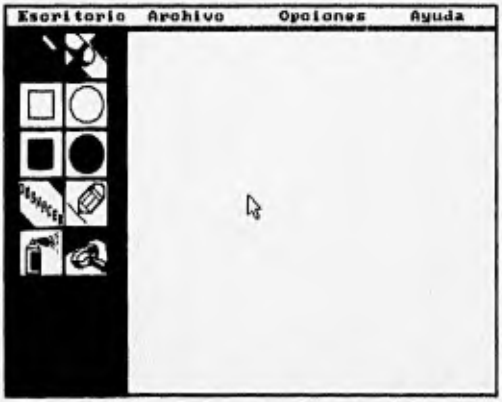


Fig. F.7. Ventana para iniciar la edición de un nuevo archivo PCX.

Si se seleccionó la opción de *Abrir* del menú colgante, permitirá dar el nombre de un archivo PCX existente en alguna de las unidades de disco; por medio de una lista que se muestra en la parte izquierda y seleccionándolo con el ratón. En la parte superior aparecerá un recuadro mostrando el directorio actual de trabajo **C:\DOCS*.PCX** (para el ejemplo mostrado en la fig. F.8.) Una vez seleccionado desde la lista, aparecerá en el recuadro situado a la derecha de la lista (en este caso **GRAFICO1.PCX**, es el nombre del archivo). Una vez realizado lo anterior, si se presiona el botón de *Aceptar*, permitirá la edición de este archivo, siempre y cuando no exista error al cargarlo desde la unidad de disco (ver fig. F.9); o bien, si no es un archivo PCX válido (ver fig. F.10.) Si es presionado el botón de *Cancelar*, lo regresará a la pantalla anterior (Fig. F.7.)

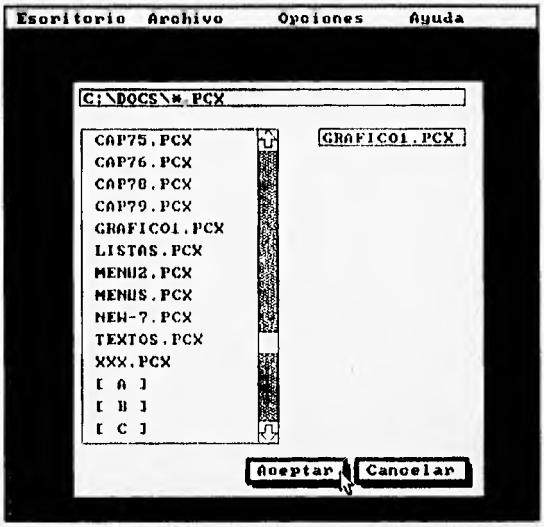


Fig. F.8. Ventana para abrir un archivo PCX existente.

Fig. F.9. Ventana que muestra que existió error al intentar abrir el archivo desde la unidad de disco.

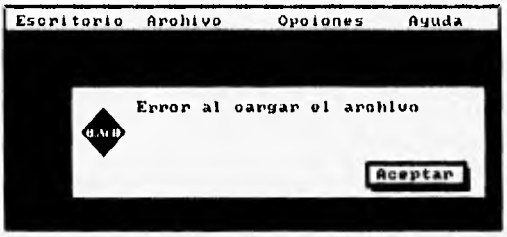


Fig. F.10. Ventana que indica que hubo un error al tratar de abrir un archivo PCX.

Si se seleccionó el botón de *Aceptar*, aparecerá la siguiente pantalla, desde la cual podrá empezar la edición del archivo.

Fig. F.11. Ventana de edición de un archivo PCX seleccionado.

Para iniciar la edición de un archivo, podrán ser utilizadas las herramientas de dibujo que tienen la siguiente función (de acuerdo a la figura F.12):

1. Herramienta para dibujar líneas dentro del área de trabajo.
2. Herramienta para borrar dentro del cuadro o área de trabajo.
3. Herramienta para dibujar rectángulos dentro del área de trabajo.
4. Herramienta para dibujar elipses y círculos.
5. Herramienta para dibujar rectángulos con relleno.
6. Herramienta para dibujar elipses y círculos con relleno dentro del área de trabajo.
7. Herramienta para deshacer la última operación de dibujo llevada a cabo.
8. Herramienta para dibujar pixel por pixel.
9. Herramienta de *spray* que permite dibujar múltiples pixeles a la vez.

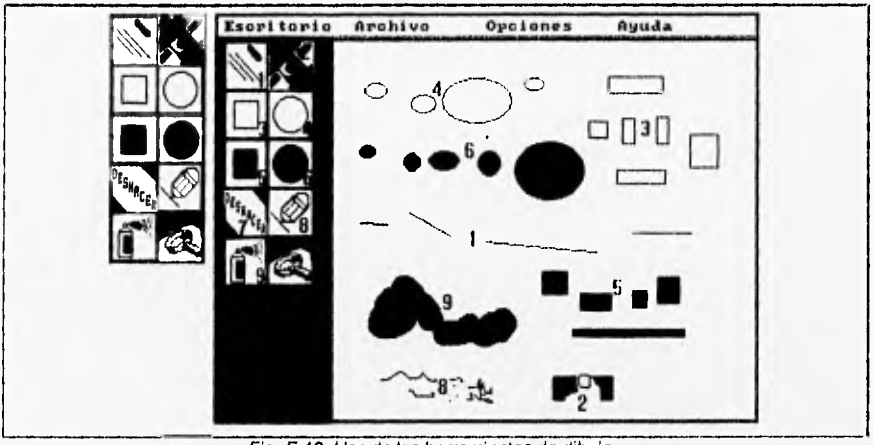


Fig. F.12. Uso de las herramientas de dibujo.

10. Esta herramienta permite hacer una ampliación de una área de 32 x 32 pixeles, y su edición pixel por pixel. (Ver fig. F.13).

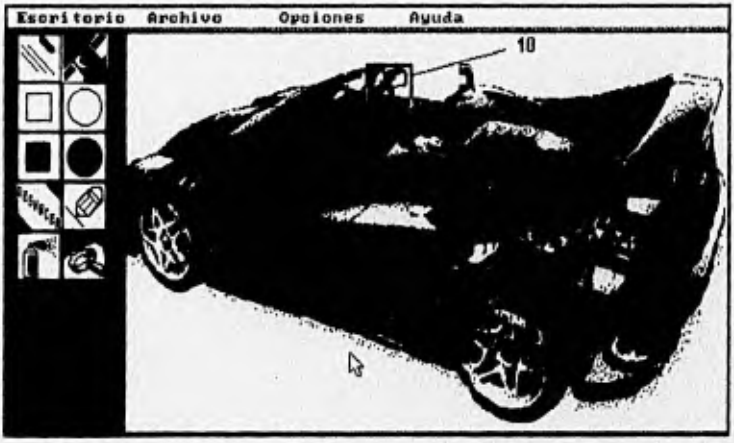


Fig. F.13. Uso de la herramienta de ampliación y edición por pixel.

Si es seleccionada la herramienta 10 (de ampliación), aparecerá la siguiente pantalla (fig. F.14), la cual permitirá la edición del fragmento de imagen por medio del ratón. Cada cuadro representa a un pixel, si un cuadro es seleccionado, éste aparecerá con su color invertido; es decir, si esta en color negro; aparecerá como blanco y viceversa.

Fig. F.14. Edición de un fragmento de imagen de 32 x 32 pixeles.

Una vez que ha sido editado el archivo, se activarán las opciones de: *Cerrar, Salvar, ¿Salvar como?* y *Salir;* del menú colgante de *Archivo,* como se muestra en la figura F.15.

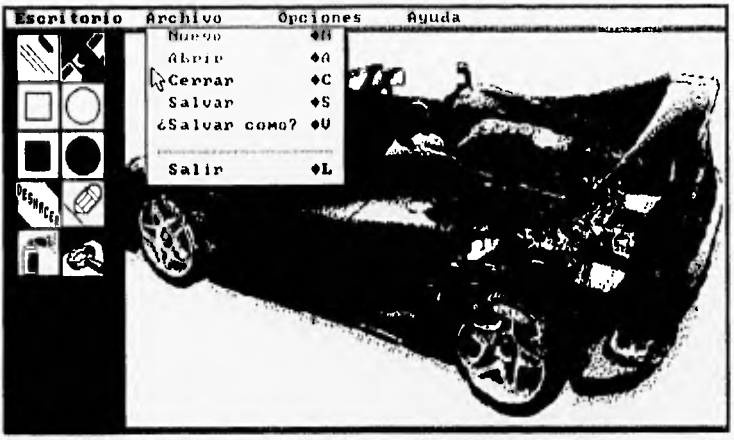


Fig. F.15. Opciones activas del menú de *Archivo* durante la edición de un archivo.

Si se selecciona la opción de *Cerrar*, aparecerá la siguiente pantalla que solicita la confirmación para cerrar el archivo; si se presiona el botón de *Sí*, el archivo será cerrado; con el botón de *No*, se cancelará esta orden

Fig. F.16. Opción para cerrar el archivo PCX que se edita actualmente.

Si se selecciona la opción de *Salvar* y el archivo ya existía, aparecerá la siguiente pantalla para confirmar si se salva o no con el mismo nombre. El botón *Sí*, lo guarda con el mismo nombre; *No* cancela la operación.

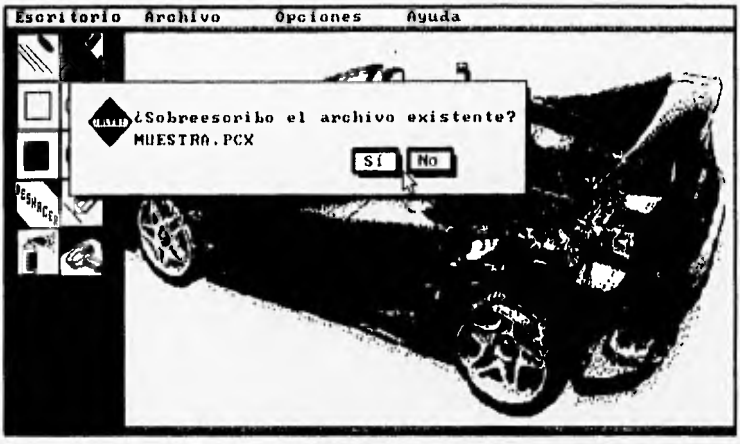


Fig. F.17. Opción para salvar el archivo PCX

Si se selecciona la opción de *¿Salvar como?*, permitirá ponerle un nombre al archivo; ya sea escogiendo uno desde la lista de la izquierda, o bien, escribiéndolo en el cuadro de edición situado a la derecha de la lista (**DIBUJO**, en el ejemplo). El directorio actual de trabajo aparece en la parte superior, éste puede ser cambiado desde la lista y por medio de la barra de despliegue, si se selecciona con el cursor del ratón sobre las flechas de la barra. Una vez que se escribió el nombre, se presiona el botón de *Aceptar* para guardar el archivo con ese nombre, o *Cancelar* para abortar esta operación.

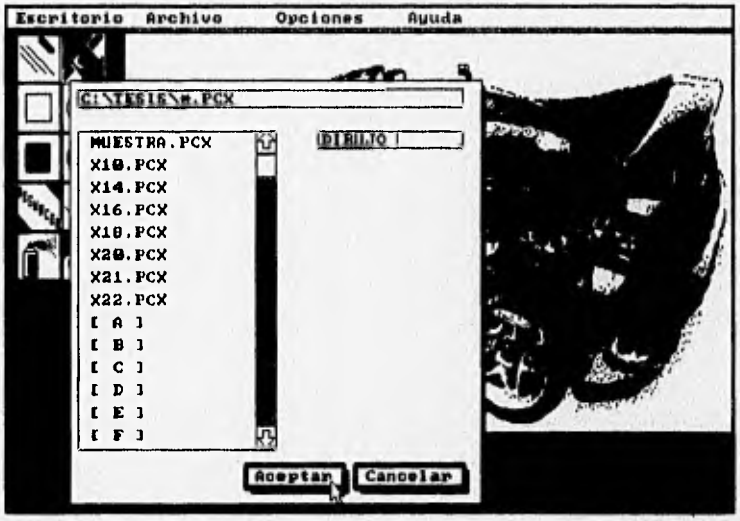


Fig. F.18. Opción para salvar con otro nombre un archivo PCX.

Si se selecciona la opción de *Salir*, del menú de *Archivo*, le permitirá salir del programa. Esta pantalla solicitará su confirmación de salida, el botón de *Si*, le permitirá salir del programa y *No* cancelará la operación.

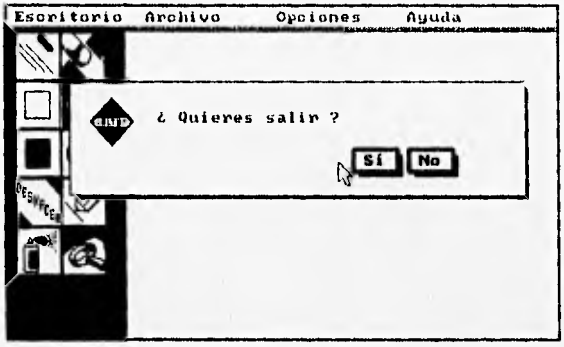


Fig. F.19. Opción para salir del programa.

Menú de *Opciones*

Una vez que ha sido abierto un archivo PCX (con las opciones de *Nuevo* o *Abrir*, del menú de *Archivo*), todos los índices del menú de *Opciones*, son habilitadas para poder ser usadas (*Tipo de línea, Grosor de línea y Relleno.*) Estas opciones pueden ser seleccionadas por medio del ratón, o por medio de las teclas, *Alt-T:* para tipo de línea, *Alt-G:* para grosor de línea y *Alt-E:* para colocar el relleno.

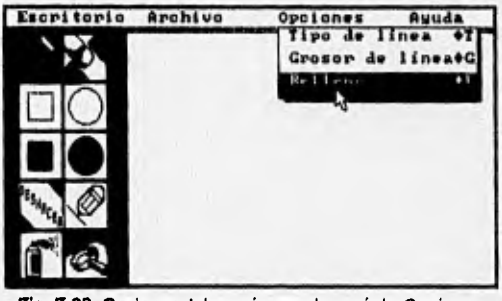


Fig. F.20. Opciones del menú para el menú de *Opciones*.

Las opciones de tipo y grosor de línea son colocadas para las herramientas de dibujo de líneas; éstas permiten colocar cuatro tipos de línea y dos tipos de grosor, pueden ser usadas para dibujar rectángulos y para dibujar a mano alzada. (Ver fig. F.21.)

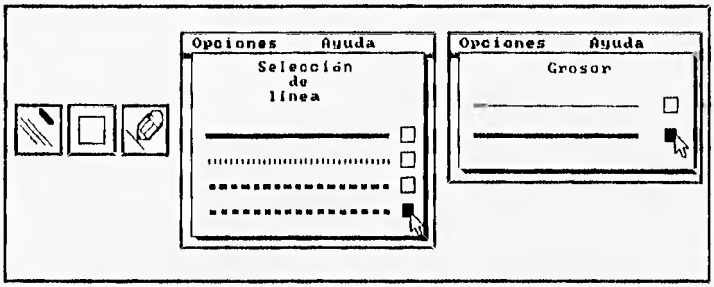


Fig. F.21. Opciones para colocar el tipo y el grosor de línea.

La opción de relleno permite escoger uno, de doce tipos de relleno disponibles, para las herramientas de dibujo de elipses y rectángulos.

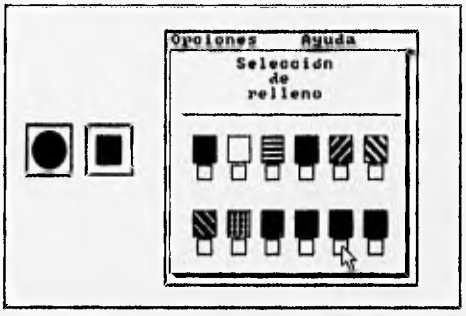


Fig. F.22. Opción para colocar el tipo de relleno.

Al cerrarse el archivo que esta siendo editado, las opciones del menú de *Opciones* son desactivadas.

Menú de *Ayuda*

Si se selecciona el menú de *Ayuda*, le presentará la opción de *¿Acerca de?*, que podrá ser seleccionada por medio del cursor del ratón o con la tecla alterna *Alt-Y*.

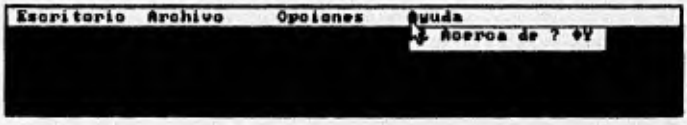


Fig. F.23. Menú de Ayuda.

Al ser seleccionada la opción *¿Acerca de?*, se abrirá un cuadro de selección y cambiará el cursor del ratón al de forma de mano, el cursor permitirá seleccionar una de las 5 opciones. Las opciones mostrarán las descripciones de las funciones de los menús, así como, de las herramientas de dibujo.

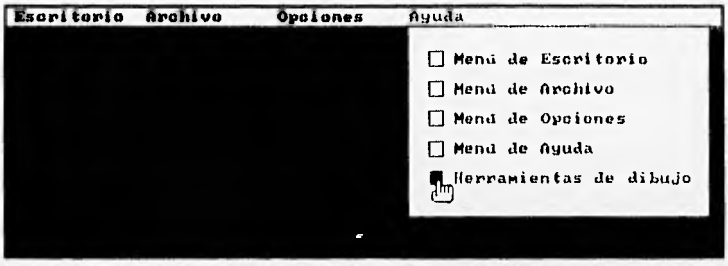


Fig. F.24. Ventana para selección de una opción de ayuda.

Al seleccionar una de las opciones, se abrirá una ventana con la ayuda correspondiente. Para cerrar esta ventana, presione el botón de *Aceptar*, con el cursor del ratón.

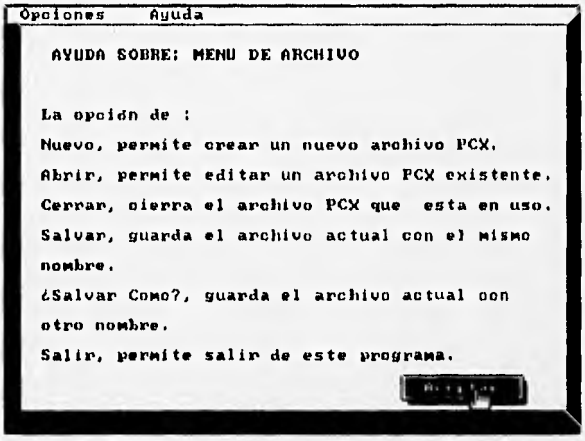


Fig. F.25. Ventana que proporciona la ayuda del menú.

Glosario

Adaptador gráfico: Es una tarjeta de expansión que se conecta en una computadora y que genera el texto e imágenes gráficas en la pantalla. Define además, la resolución y la cantidad de colores en pantalla. También es conocido como adaptador o tarjeta de vídeo, tarjeta de gráficos y controlador de vídeo.

Apuntador: Es una variable que se utiliza como una referencia al elemento actual en una tabla (arreglo) o a algún otro tipo de dato como un registro o un archivo.

Arreglo: Es un conjunto ordenado de datos, si es un arreglo unidimensional, es llamado también vector y si es un arreglo de dos o más dimensiones, es llamado matriz.

Cociente de aspecto: Relación entre el largo y ancho de la pantalla o imagen. Cuando se transfieren imágenes de un sistema a otro, la proporción del aspecto debe mantenerse para proveer una representación adecuada del fragmento original.

Barra de despliegue: Barra horizontal o vertical que contiene un recuadro que funciona como controlador de posición de la barra. Este recuadro puede ser desplazado hacia arriba o abajo, o hacia la izquierda o derecha.

Barra de menús: Es la lista de menús, que aparece por lo regular en la parte superior de la pantalla.

BIOS (*Basic Input/Output System*): Sistema básico de entrada/salida, es un conjunto detallado de instrucciones que activan los dispositivos periféricos de la computadora. Este sistema incluye rutinas para el teclado, la pantalla, los puertos paralelos y seriales; así como, para servicios internos como la hora y fecha.

Botón: Un pequeño recuadro en pantalla, con información en su interior, correspondiente a una actividad específica y que se acciona si es seleccionado con el cursor del ratón.

Buffer: Segmento reservado de memoria, que se utiliza para almacenar datos mientras se procesan.

Campo de edición: Es un recuadro en el que puede ser editado su contenido, si éste es seleccionado por el cursor del ratón.

Caracteres alfanuméricos: combinación de letras del alfabeto, con números y caracteres especiales.

Clase: En programación orientada a objetos, es un conjunto de características que se utilizan para definir a un objeto.

Clase base: Son llamadas también clases padre y son las clases primarias que pueden ser utilizadas por otras clases, las "clases heredadas".

Clase heredada: Es una clase que hereda las propiedades de otras clases.

Codificación: Escritura del programa.

Comando: Orden dada por el usuario a la computadora.

Constructor: Es una función usada para construir un objeto de una clase dada. Pueden ser definidos más de un constructor para una clase.

Cuadro de diálogo: Es una ventana que aparece en la pantalla como respuesta a alguna solicitud. Contiene las opciones que están actualmente disponibles para el usuario.

Cursor: Símbolo utilizado para indicar la posición en la pantalla o para indicar las selecciones en un menú.

Destructor: Es una función dentro de una clase que es necesaria cuando un objeto ya no esta siendo usado. Realiza la operación inversa del constructor.

***Font*/fuente:** Conjunto de caracteres de un diseño y tamaño particular. (Ejemplos de *fonts* tenemos: *Times Roman, Helvetic*, etc.)

Formato: Estructura o disposición de un campo en pantalla

Función externa: Es una función que es invocada desde otro programa o librería.

Funciones miembro: Son una serie de funciones contenidas en una clase.

GUI (*Graphical User Interface*): Interfaz gráfica de usuario, basada en gráficos, que incorpora objetos tales como: iconos, botones, menús desplegables y algún dispositivo de posicionamiento.

Hardware: Son los recursos físicos que forman un equipo de cómputo.

Herencia: En programación orientada a objetos, es la técnica de tomar clases existentes y extender sus propiedades a otras clases.

Icono: Representación gráfica de un objeto en pantalla, que tiene una función específica si es seleccionado. Este tipo de objetos es ampliamente utilizado en interfaces gráficas.

Indice de menú: Es una de las opciones de menú disponibles para el usuario.

Lápiz óptico: Dispositivo de posicionamiento en forma de lápiz, sensible a la luz, que esta conectado a través de un cable a una terminal de video. El lápiz óptico permite introducir escritura, seleccionar menús y colocar marcas en la pantalla de la computadora en forma externa.

Librería: Es una colección de funciones, programas o archivos, que se unen al programa principal cuando éste es compilado.

Lista: Es un objeto gráfico que contiene una serie de cadenas de texto, dentro de un recuadro, este tipo de objetos por lo general, va siempre asociado con barras de despliegue.

Lista ligada: Es una estructura de datos que contiene un campo liga y que apunta al siguiente dato de la lista.

Mapeo de memoria: Es la localización específica de datos, en las direcciones de memoria.

Menú: Lista de opciones y comandos disponibles en pantalla. La selección se efectúa resaltando la opción con el ratón o teclas del cursor y presionando el botón del ratón o la tecla de *Enter*.

Modo gráfico: Es el modo del adaptador gráfico, que nos permite obtener la mayor resolución de la tarjeta gráfica.

Palanqueta/*joystick*: Dispositivo que se utiliza como apuntador, para mover un objeto en la pantalla en cualquier dirección. Posee una palanca vertical montada sobre una base, que tiene uno o dos botones. Se usa ampliamente en los juegos de vídeo y en algunos sistemas de CAD.

Pin: Es un pequeño conector o clavija, utilizado normalmente en puertos seriales, cables de monitor, conectores de teclados.

Pixel (*PIX picture Element*): Es el elemento más pequeño de una pantalla de vídeo. Un pixel es uno de los puntos que se tratan como una unidad.

Polimorfismo: Es una propiedad de una clase base, que sirve para derivar diferentes características a otras clases.

Protocolo: Es un conjunto de reglas que permiten el intercambio de información entre dos entes informáticos (dos computadoras, por ejemplo)

Prototipo: Modelo de un nuevo sistema generado por analistas de sistemas y usuarios.

Puerto *Centronics:* Interfaz estándar para el puerto paralelo, usado en computadoras personales. *Centronics Corp.* es el fabricante de este tipo de interfaz.

Puerto de impresión: Conector de entrada/salida que se utiliza para conectar una impresora u otro dispositivo de interfaz paralelo. Comúnmente se utiliza un conector hembra DB-25, de 25 *pins*.

RAM (*Random Access Memory*): Memoria de acceso aleatorio, este tipo de memoria puede tener acceso directo a el contenido de cada byte que se encuentra almacenado sin necesidad de buscarlo secuencialmente.

Ratón/*Mouse:* Es un dispositivo de posicionamiento en pantalla, que se usa para apuntar y dibujar. A medida que se hace rodar el ratón en una superficie, su cursor se mueve en forma correspondiente en la pantalla.

Registro: Es un grupo de campos relacionados, que almacenan datos acerca de una actividad o tópico.

Resolución: En un adaptador gráfico es el largo por ancho de la pantalla, que se expresa como una matriz de puntos.

Retícula: Es una casilla o recuadro en el cual puede ser incluido texto o algún otro tipo de objeto.

Sistema: Grupo de componentes relacionados, que interactúan para realizar una tarea determinada, una computadora es un sistema que consta de CPU, sistema operativo y de dispositivos periféricos.

Software: Una serie de instrucciones que realiza una tarea en particular. Existen dos clases de *software*: de sistemas y de aplicación.

Tableta óptica: Tableta de diseño gráfico que puede ser utilizada para diseñar nuevas imágenes o para trazar otras existentes, y sirve además para seleccionar elementos de menús.

Tecla alterna: Es una tecla utilizada normalmente para reemplazar el uso del ratón en la selección de índices de menús. Se usa la tecla *Alt* en combinación con alguna otra tecla.

Variable *private*: Este tipo de variable es definida como el acceso por *default* en una clase. Solo la misma clase puede tener acceso a esta variable, así como clases declaradas como *friend*.

Variable *protected* : Cuando se definen este tipo de variables en una clase base, solo las clases derivadas pueden utilizarlas en sus funciones miembro.

Variable *public*: Este tipo de variable puede ser utilizado como un objeto, por cualquier otra clase.

Vídeo inverso: El vídeo inverso consiste en intercambiar los colores del fondo, con los del texto.

Zoom/ampliación: Ampliación de un fragmento de imagen. Usada por lo general, para poder editar con más precisión una área de la imagen.

Bibliografía:

- Computer Graphics
  Donald Hearn / M. Pauline Baker
  Prentice Hall

- Graphical User Interface Programming
  Steve Rimmer
  Windcrest/McGraw-Hill

- Graphics File Formats
  John R. Levine
  Windcrest/McGraw-Hill

- Diccionario de computación
  Alan Fredman
  Mc Graw Hill

- Manual del usuario Microsoft MS-DOS
  Microsoft Corporation

- Manual del usuario Microsoft Windows 3.1
  Microsoft Corporation

- Algoritmos y Estructuras de Datos
  Nicklaus Wirth
  Prentice Hall

- Estructuras de Datos
  Jorge Eúan/ Luis G. Cordero
  LIMUSA

- Introducción a los Sistemas Operativos
  Harvey M. Deitel
  Adison Wesley Iberoamericana.

- Power Graphics using Turbo C++
  Keith Weiskamp / Loren Heiny
  Wiley

- Tom Swan´s C++ Primer
  Tom Swan
  Sams Publishing

- Borlandc C++3.1 Object-Oriented Programming
  Ted Faison
  Sams

- Programación Orientada a Objetos con Turbo C++
  Weiskamp / Hein / Flamig
  Megabyte, Noriega Editores

- Turbo C/C++: The Complete Reference
  Herbert Schildt
  McGraw-Hill

- Turbo C programación y manejo de archivos
  J. Javier García-Badell
  Macrobit

- C manual de bolsillo
  Alan C. Plantz
  Addison-Wesley Iberoamericana