UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

METODOLOGIA DE PROGRAMACION EN AMBIENTE WINDOWS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

ANTONIO VALLEJO PADILLA

DIRECTOR:

ING. ALBERTO TEMPLOS CARBAJAL

MEXICO, D. F. OCTUBRE DE 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

1

Una Introducción a Windows

Windows es una interface grafica diseñada por la compañía Microsoft para un mejor control de programas y periféricos en computadoras personales basadas en la arquitectura del 80x86. En Windows se utiliza la estructura ventana-menú-icono controlada preferentemente por ratón y combinaciones de teclas de control. En la actualidad la mayor parte de las computadoras personales son vendidas incluyendo tanto el MS-DOS como el Windows, lo cual esta causando más y más adeptos a esta interface gráfica.

1.1. La Historia de Windows

Para comprender mejor el origen y evolución de Windows se deben considerar los adelantos más notorios en software y en hardware acontecidos en los últimos veinticinco años.

1.1.1. La Herencia del PARC

A finales de los 60's, la compañía Xerox fundó el PARC (Palo Alto Research Center -Centro de Investigaciones de Palo Alto-) en California, Estados Unidos. El objetivo del PARC fue y es, pues continua funcionando hasta la fecha, el de crear nuevas tecnologías en software y hardware, de tal forma que se permita el uso de las computadoras a cualquier clase de persona.

En los primeros días en que inicio sus labores el PARC, la mayor parte de la programación se realizaba por medio de tarjetas perforadas; algunos equipos contaban con terminales que permitían una mejor interacción hacia el usuario o el programador; pero, para tal comunicación, el operador de computadoras debía aprender los complejos comandos del sistema operativo y de los compiladores que existían dentro de un sistema de cómputo.

El PARC atacó este problema desarrollando la filosofía WYSIWYG (What You See Is What You Get -Lo Que Tú Ves es lo Que Tú Obtienes-). La base fundamental de esta filosofía era que "una imágen dice más que mil palabras", por lo que se fijo el objetivo de crear sistemas de software usando símbolos que significaran comandos y procesos. Estos símbolos no necesitaban ser parte de un idioma, podrían ser señales de tránsito o figuras de objetos; lo cual implicó el desarrollo de hardware para permitir gráficos complejos. A este sistema de software se le llamó GUI (Graphic User Interface -Interface Gráfica para el Usuario-).

Durante el desarrollo de los primeros sistemas GUI, los investigadores del PARC encontraron las deficiencias en el uso del teclado para controlar los procesos que requirieran supervisión del usuario en forma muy directa. Por ello, entre sus primeros desarrollos en hardware, el PARC rediseño el dispositivo conocido como ratón, inventado por Doug Engelbart en 1963.

Otras de las deficiencias que el PARC atacó fue la forma de programar de su época. A principios de los 70's comenzaba a generalizarce el concepto "programa estructurado y modular". Pero este concepto soló permitia un mejor uso de las rutinas de control de un lenguaje de programación y una forma primitiva de dividir los procesos internos.

Un investigador del PARC, Alan Kay, analizó que las GUIs necesitaban ser programadas en una forma en la que cada parte, procesos internos y generados por el usuario, se comportarán como un individuo. Este concepto llevó a Kay a reevaluar un viejo lenguaje de programación llamado Simula; en el que se podian definir estructuras de datos que se comportaran como un objeto individual. Como resultado de todo esto, Kay fundamento el concepto de *programación orientada a objetos*; y desarrollo Smalltalk, un lenguaje de objetos orientados y primero en usar GUI.

Los avances en GUI y en la programación orientada a objetos continuaron durante toda la década de los 70's. En forma paralela a la evolución que sufrio Smalltalk en su interface hacia el usuario y en su estructura de programación, el PARC desarrollo otros sistemas de software con GUI, tales como Interlisp, Informat y Bravo, este ultimo antecesor de los actuales procesadores de palabras.

Entre otros desarrollos importantes en hardware, el PARC generó el principio de las redes locales y la impresora de sistema laser.

En abril de 1981, la Xerox lanzó al mercado el resultado concreto de casi diez años de trabajo ininterrumpido del PARC: la estación de trabajo Star 8010, figura 1-1. La Star 8010, fue la primera computadora comercial con la capacidad de conectarse en red, manipular GUI por medio de ratón y permitir la programación orientada a objetos en Smalltalk. La aparición de la Star 8010 fue oportuna para que el PARC obtuviera un lugar preferencial en la historia de las computadoras, antes de que esta fuera opacada por la rivalidad de Apple e IBM.

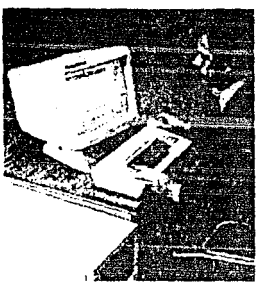


Figura 1-1: Demostracion del Sistema Star 8010.

1.1.2. Apple

En 1976, Steve Jobs, un exdiseñador de circuitos de la naciente compañía Atari, radicado en la Ciudad de San Francisco, había observado que la tecnología de la microelectrónica no había logrado brindar aún al "hombre de la calle" una computadora personalizada y de bajo costo. Por lo que Jobs se dió a tal tarea en sociedad con Steve Wozniak, un diseñador de calculadoras de la Hewlet-Packard. Hasta ese momento no existían computadoras personales en el mercado, las computadoras más pequeñas eran del tamaño de un escritorio y tenía un costo promedio de 11000 dólares. Algunos intentos para lograr computadoras pequeñas se habían realizado por entusiastas en electrónica, pero los resultados no pasaban de ser simples calculadoras grandes, caras y difíciles de programar, figura 1-2.


Popular Electronics

PROJECT BREAKTHROUGH!

World's First Minicomputer Kit to Rival Commercial Models...

"ALTAIR 8800" SAVE OVER $1000

Figura 1-2: Un anuncio para construir una computadora "casi" personal.

Entre 1976 y 1977, Jobs y Wozniak diseñaron una computadora compacta usando el microprocesador de 8 bits MOS Tecnology 6502. La computadora fue bautizada como Apple, y poseía una RAM de 4 Kbytes; un interprete de BASIC en ROM; el teclado formaba parte del módulo principal de la computadora, a este módulo se le podían conectar como periféricos una grabadora de cassets y un pequeño monitor monocromático. Las primeras computadoras Apple fueron manufacturadas en forma casera en el garage de Jobs. El impacto de aceptación de las computadoras Apple fue tremendo, y puesto que Jobs y Wozniak vivían cerca de Silicon Valley, les fue posible conseguir equipo electrónico a precio de mayoreo, con lo que lograron bajar el costo de su computadora de 2000 a 1000 dólares. En 1978, Jobs y Wozniak fundaron formalmente la empresa Apple Computer. Para 1979, ya establecida en talleres especiales, Apple Computer lanzó al mercado mundial la Apple II+. La Apple II+ usaba el mismo microprocedasor de la primera Apple, pero se le había agregado una unidad de disco flexible, aumentado la memoria a 64 Kbytes e instalado un BASIC de punto flotante para realizar programas que requirieran operaciones complejas.

Muy pronto surgieron competidores y aliados de Apple. El nuevo mercado de las computadoras personales comenzó a afectar el mercado de las grandes computadoras, sobre todo el de IBM.

Temiendo que la grandes compañías, como IBM, Hewlett-Packard y AT&T, poseedoras y creadoras de la tecnología electrónica mundial, bloquearan el futuro de Apple, Steve Jobs decidió buscar una fuente de inovación tecnológica para incorporarla a su empresa, por lo que visitó el PARC a finales de 1979. Fue tanto su asombro hacia los avances en hardware y, sobre todo, en software que ahí se habían alcanzado, que decidio formar un grupo con parte de los mismos investigadores del PARC, para que transladaran todos sus conocimientos al diseño de las futuras computadoras Apple. Algunos aceptaron, y con ellos: Jobs inicio el desarrollo de una computadora que conjuntara todos los años de investigación del PARC: la computadora Apple Lisa, el primer sistema personalizado con GUI.

1.1.3. IBM, Microsoft y las Computadoras Personales

A principios de 1980, IBM tuvo que reconocer la supremacía de Apple como el mayor fabricante de computadoras personales en el mundo, y sabedora de los planes de Jobs, comprendió que pronto Apple tendría una tecnología propia que la haría una rival muy poderosa. Por ello, la IBM inicio el diseño de una computadora que compitiera con el de la Apple. Así, en agosto de 1981, la IBM lanzo al mercado una computadora basada en el microprocesador Intel 8086, la computadora personal XT, figura 1-3. La XT de IBM contaba con un microprocesador 8088 de 8 bits a 4.77 MHz de velocidad de reloj, una memoria RAM de 64 Kbytes, dos unidades de disco flexible y una tarjeta de video de texto que a futuro podría ser sustituida por otra que permitiera gráficos y cinco canales expasión (slots). La computadora incluía como software el sistema operativo PC-DOS (Personal Computer Disk Operating System), desarrollado por una compañía de software desconocida hasta ese momento: Microsoft.

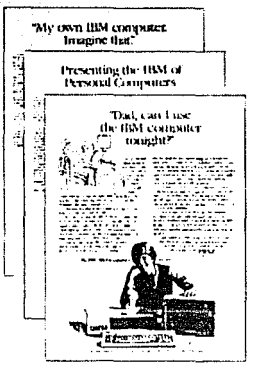


Figura 1-3: Primer anuncio oficial de la computadora XT de IBM.

Microsoft habia sido fundada en 1975 por Bill Gates y Paul Allen, dos extraordinarios programadores. Sus primeros trabajos fueron el de adaptar interpretes de Basic para las nacientes microcomputadoras. Mucho antes de que IBM convocara a concurso al fabricante del software para la XT; Gates y Allen habían trabajado en una versión del sistema operativo CP/M para computadoras basadas en el 8086, pero esta no era muy adecuada para los lineamientos que pedía IBM. Por lo cual decidieron comprar los derechos sobre un sistema operativo llamado 86-DOS desarrollado para tarjetas de control industrial que usaran el 8086. Modificando y uniendo rutinas, Gates y Allen crearon el PC-DOS que derrocó al rey de los sistemas operativos a nivel personal de aquel momento CP/M. A pocas semanas de ser lanzada comercialmente la XT, Microsoft firmó un contrato, casi en exclusividad, con IBM para el desarrollo de cualquier proyecto relacionado con sus computadoras personales.

La XT fue muy bien recibida y pronto ganó adeptos. Microsoft había incorporado, como software adicional al PC-DOS, cuatro compiladores, un ensamblador y un ligador de código; con lo que se ofrecia al usuario la oportunidad de crear programas de aplicación. Sin embargo, ni toda la publicidad a nivel mundial, ni la planta robotizada que IBM contruyó para ensamblar una XT por minuto, lograron derrocar a la Apple del liderato del mercado de la computadoras personales. Para 1982, IBM se enteró de los detalles sobre el diseño de GUI en la siguiente generación de computadoras Apple, por lo que solicitó a Microsoft el desarrollo de un sistema de software semejante para su modelo XT. Bill Gates se negó, alegando que el hardware que conjuntaba al modelo XT no era lo suficientemente favorable para implantar un sistema de tal índole.

En 1983, la nueva computadora Apple Lisa hizó su aparición en el mercado mundial con su atractiva GUI y su exorbitante precio de 10000 dólares, figura 1-3.

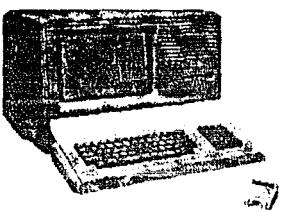


Figura 1-4: Computadora Apple Lisa.

Con el lanzamiento de la Lisa, IBM comprendió que su XT nunca alcanzaría en el momento inmediato la tecnología que ofrecía Apple, así que optó por liberar la licencia de fabricación de su computadora personal. Dado que la XT había sido diseñada en forma modular, al liberar la licencia permitiría que una buena parte de los fabricantes de equipo de cómputo, incluyendo los competidores de IBM y de la misma Apple, desarrollaran productos y computadoras basándose en la arquitectura de la XT, usando sus propios recursos y tecnologías. El resultado de esta estrategia fue inundar el mercado de computadoras personales tipo IBM XT y dando origen a lo que hoy se conoce como computadoras clones. El beneficio económico de IBM fue la reducción del costo de investigación y fomento al desarrollo de elementos tales como los discos duros y las tarjetas gráficas de video de alta resolusión. Sin embargo, el verdadero efecto a nivel mundial, fue la caída del precio de las computadoras personales.

De igual forma, IBM liberó el sistema operativo PC-DOS, bajo el acuerdo con Microsoft de conocerlo comercialmente como MS-DOS. Antes de finalizar 1983, IBM logró convencer a Microsoft para el desarrollo de una interface gráfica para la XT (que para entonces ya se vendía con disco duro). La razón por la cual aceptó Microsoft, fue el peligro que corría de ser desplazada del favoritismo de IBM por la compañía VisiCorp, la cuál, si había logrado desarrollar una modesta interface gráfica llamada VisiOn. El desarrollo de Windows estaba en marcha.

1.1.4. Surgimientos y Caídas

A mediados de 1983, la Apple comenzó a sentir los temibles efectos de la liberación de la licencia de la XT. Por ello, previendo la caída de la computadora Lisa, empezó trabajar en el desarrollo de una nueva computadora; no tan compleja como la Lisa, pero sí, de una inovación tecnológica igualmente comparable y de bajo costo: La Macintosh, figura 1-5.

Figura 1-4: Macintosh.

La computadora Macintosh apareció comercialmente en febrero de 1984 a un precio de 2500 dólares. Usando el microprocesador Motorola 68000 a 7.8 Mhz de velocidad de reloj, con una memoria RAM de 128 Kbytes, una unidad de disco de 3½" con capacidad de 400 Kbytes, un monitor monocromático de fondo blanco y con una resolución de 512 x 312 pixeles (todo esto integrado en una sola unidad), la Macintosh fue anunciada como el "Hermano Mayor" de los sistemas personales. Al igual que la Lisa, la Macintosh contaba con GUI controlada plenamente por ratón. La aparición de Macintosh marca el inicio de los "Apple-Brands", es decir, hardware y software diseñados exclusivamente por Apple. Entre estos productos cabe resaltar el HyperCard, una plataforma de programación para el ambiente gráfico de la Macintosh.

IBM contratacó con un nuevo modelo de computadora personal, el modelo AT, en diciembre de 1984. El nuevo modelo AT fue diseñado para trabajar con el microprocesador 80286, que tiene un bus de datos de 16 bits y la capacidad de accesar 16 MegaBytes de RAM y totalmente compatible con el software de la "vieja" XT. En noviembre de 1985, IBM y Microsoft anunciaron la salida comercial de la versión 1.01 de Windows. Está primera versión formal de Windows fue diseñada para correr en computadoras con microprocesador 8088, una memoria RAM de 256 Kbytes y dos unidades de disco flexible. En Windows 1.01 se podía tener varias ventanas translapadas y accesarlas en cualquier momento por medio del ratón; las ventanas contaban con botones deslizables, que permitían mover los contenidos en alguna ventana específica arriba-abajo o izquierda-derecha; los iconos eran muy llamativos y cumplían con el objetivo de simbolizar una acción, los menús permitían un control más eficientes de las acciones. Se proveía al usuario de programas tales como Paint, Clipboard, Write y Clock. Una característica interesante de esta versión, es el hecho de funcionar para la gran mayoría de periféricos comerciales. Sin embargo, en Windows 1.01 no existía ninguna forma viable de desarrollar software para este entorno gráfico, por lo que muchos usuarios no lo consideraron un digno rival para el software de la Macintosh.

Quizás, el único efecto relevante que ocacionó Windows 1.01, fue el estusiamo de usar ventanas y menús controlados por ratón. Por lo que a partir de 1985, la mayor parte de los fabricantes de software para computadoras personales IBM comenzaron a incluir en su software ventanas y menús como parte de su medio ambiente, figura. 1-5. Dado que el manejo de un entorno en modo gráfico ocupa demasiada memoria, la mayor parte de los fabricantes desarrollaron su software en modo texto, explotando los novedosos monitores y tarjetas gráficas, productos recien lanzados al mercado, y que junto con las primeras tarjetas

para conectar las computadoras personales en red, causaron toda una expansión en los mercados.

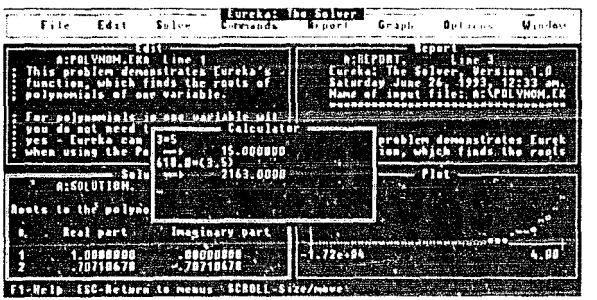


Figura 1-5: Eureka, de Borland, uno de los primeros paquetes en usar ventana.

Durante 1986 se fortalecieron muchas compañías norteamericanas de hardware y software, tales como Borland, Lotus, Compaq y Dell. Microsoft lanzó la versión 3.2 de DOS, el procesador de palabras WORD y un compilador de C. En aquel momento, todo el software diseñado para la AT no sobrepasaba los 512 Kbytes de RAM; el máximo de RAM permitido en la computadora era 640 Kbytes. Por la importancia que estaban tomado las computadoras AT, pronto sería imperativo crear programas más complejos y que sobrepasaran los 640 Kbytes.

Aunque el 80286 por su diseño permite un acceso a RAM de 16 Megabytes, IBM e Intel no consideron que se alcanzaría a romper la "barrera de los 640" en un corto plazo; por lo que el microprocesador fue diseñado para trabajar de dos formas, Modo Real y Modo Protegido. El modo real permite el acceso de memoria no mayor de los 640 Kbytes, para que cualquier programa diseñado para un 8088, pueda correr normalmente en un 80286. El modo protegido permite el uso de RAM ubicada más allá del 1 Mbytes y el multiprocesamiento. Para lograr pasar al modo protegido es necesaria la intervención del sistema operativo, y por supuesto, el DOS, en aquel momento, carecía de tal función. Esta fue la razon por la que la compañía Compaq inicio el desarrollo independiente de una computadora personal basada en el microprocesador Intel 80386, sin tomar en cuenta la "supervisión creativa" de IBM. Otras compañías, en especial las japonesas, siguieron el ejemplo de Compaq.

Con un objetivo parecido al de Compaq, la compañía Lotus presionó a Intel y a Microsoft para generar un medio que facilitara el uso de la memoria RAM más alta de 640 Kbytes. Como resultado se creó el LIM EMS (Lotus/Intel/Microsoft Expanded Memory Specification - Especificación de Memoria Expandida-), una serie de interrupciones que permiten al software, diseñado bajo ciertas normas, correr en una memoria RAM de hasta 32 Mbytes en computadoras AT. Microsoft tomó muy encuenta el problema de la "barrera de

los 640" para sus futuras versiones de Windows. Mientras ocurrían estas "microconspiraciones" en contra de los lineamientos de IBM; la Compañía Apple enfrentó una crisis más severa que la competencia con IBM.

Hasta 1986, Apple logró recuperarse del fracaso de la computadora Lisa. La nueva Macintosh Plus, con 1 megabyte de RAM y capacidad para conectarse en red, tuvo un buen éxito comercial. Sin embargo, estadísticamente había 12 computadoras personales tipo IBM por una de Apple; por lo que Steve Jobs pidió la adición de nuevos recursos de investigación para el desarrollo de una computadora que desplazará de una vez por todas a la IBM y fabricantes seguidores del mercado de las computadoras personales. El entonces presidente de la compañía, John Sculley, consideró que la petición de Jobs era totalmente descabellada y ordenó que fuera despedido de Apple. Ironicamente, Sculley ingresó a Apple por recomendación de Jobs. Despedido Jobs, Apple se estanco en la producción de la Macintosh Plus y de Apple II por dos años. IBM no perdio el tiempo.

1.1.5. Extendiendo el Imperio

En 1987, IBM lanzó al mercado un nueva serie de computadoras personales: la serie PS/2 (Personal System/2 -Sistemas personales 2-). La serie PS/2 constó de seis modelos, vease tabla 1-1. En forma paralela, IBM desarrollo una nueva gama de software para una gran variedad de aplicaciones, desde programas para usarse como agendas, hasta paquetes complejos de diseño asistido por computadora. Por su parte, Microsoft también desarrollo una nueva generación de productos, entre los cuales estaba la versión 3.3 del DOS.

Tabla 1-1.

MODELO PS/2	MICROPROCESADOR	VELOCIDAD	RAM MAXIMA
25	8086	8MHz	640 KB
30	8086	8MHz	640
50	80286	10MHz	7 MB
60	80286	16MHz	15 MB
70	80386	16MHz	16 MB
80	80386	20MHz	16 - 115 MB

La característica principal de la serie PS/2, es la adición del Micro Canal, un sistema en hardware que permite conectar periféricos de 16 ó 32 bits; es decir que el tamaño del bus de datos es variable. Puesto que el microprocesador 80386 fue diseñado con un bus de datos de 32 bits; el enfoque de IBM fue que las futuras computadoras diseñadas con la familia 80x86 fueran compatibles en sus periféricos sin importar el tamaño del bus de datos a usar.

Como parte de esta idea de compatibilidad, IBM anunció el desarrollo de un nuevo sistema operativo compatible con cualquier sistema de cómputo, que permitiera el uso normal de la memoria RAM de cualquier tamaño, implementar multiprocesamiento y llevar a cabo control de acceso a red. Este sistema operativo seria realizado en conjunto por IBM y Microsoft y su nombre seria OS/2 (Operating System/2 - Sistema Operativo 2-). Sin embargo, para lograr el pleno desarrollo del OS/2, Micorsoft debía lanzar primero su nueva versión de Windows y tener éxito. Windows versión 2.0, conocido mejor como Windows 2, apareció en septiembre de 1987. Esta vez, Microsoft se esforzó en proveer al sistema de una interface de aplicaciones para que programadores independientes pudierán crear sus

propios programas explotando las caracteristicas de la GUI de Windows. Una nueva caracteristica de Windows fue el concepto Intercambio de Datos Dinámico; el cual permite compartir los mismo datos entre diferentes programas de aplicación.

Para demostrar las ventajas de trabajo en Windows 2, Microsoft rediseño su procesador de palabras WORD y creó la hoja de cálculo Excel. La aceptación de Windows 2 durante un mes en el mercado, alentó a Microsoft a introducir una versión de Windows conocida como Windows/386. Esta versión solamente podía funcionar en computadoras basadas en el microprocesador 80386, su característica principal es el permitir multiprocesamiento tanto en programas desarrollados para Windows, como aquellos desarrollados en DOS. Windows/386 asombró tanto, que incluso compañías que diseñaban software para Macintosh, decidieron rediseñar sus productos para correr en Windows.

El lazo entre Windows 2 y OS/2 se debe a la introducción de las normas SAA (Systems Application Architecture -Arquitectura de Aplicación de Sistemas-) y CUA (Common User Acces -Acceso Común al Usuario-). La SAA permite la comunicación entre las computadoras personales tipo IBM con maintrames. El CUA marca el establecimiento de bibliotecas conteniedo funciones y estructuras comunes entre los sistemas para permitir una estandarización entre el diseño y programación de sistemas. Microsoft uso una parte de las normas para la creación de Windows/386 partiendo de la estructura de Windows 2, con lo que se comprobaba, en parte, la eficacia de la SAA y del CUA para el translado de software "simple" a más "complejo". Sin embargo, IBM no planeo que Windows fuera la interface principal para OS/2, y por ende de la SAA, y basandose en la estructuración de Windows'386 inició el desarrollo del Presentation Manager, el "Windows" de OS/2.

El aparente desinteres de IBM hacia Windows 2, inició la separación de Microsoft de los proyectos del OS/2 y del Presentation Manager. Microsoft inició la nueva y titánica tarea de hacer de Windows el verdadero y único sistema operativo compatible para todas las computadoras.

Mientras tanto, en una forma de recuperar mercado, la Apple lanzó al mercado nuevos modelos de Macintosh usando los microprocesadores 68020 y 68030, y con puerto de bus abierto, es decir un puerto para conectar periféricos que no necesariamente fueran "Apple-Brands", con este nuevo diseño Apple pretendió iniciar la liberalización de una parte de su arquitectura. Entre los diversos productos que se derivaron de la arquitectura de bus abierto fueron los sistemas, tanto en hardware como software, para permitir correr programas de computadoras personales tipo IBM. Para muchos seguidores de Apple, la aparición al mercado de tales productos "compatibles" significo la bandera de tregua entre IBM y Apple.

1.1.7. Nuevos Competidores

Microsoft colaboró con IBM para generación de la primera versión de OS/2, quedando lista para ser presentada a mediados de 1987. Al principio, OS/2 no sorprendió mucho, y algunos la consideraron demasiado lenta. Igualmente, en la mayor parte de las presentaciones que se realizaron en Estados Unidos, IBM nunca pudo demostrar la compatibilidad del OS/2 con otros sistemas. Microsoft aprendió bastante de esta experiencia, y prácticamente fue el ultimo trabajo formal con IBM. Irónicamente, aún una buena parte de las publicaciones se refieren a OS/2 como MS-OS/2, a pesar que la más reciente versión del sistema operativo esta prácticamente diseñada por IBM.

Para lograr el objetivo de convertir a Windows en un medio ambiente poderoso, Microsoft no solamente debía enfrentarse a IBM y Apple, sino también a otros fabricantes de computadoras que nunca habían considerado al MS-DOS como una plataforma lo bastante eficiente para programar. Desde la aparición de la Star 8010, varias compañías se dedicaron a la creación de estaciones de trabajo cuyo sistema operativo era el legendario UNIX. Y, al igual que Apple e IBM, habían diseñado GUI's para un mejor control de los procesos. El GUI más difundido entre los fabricantes de la estaciones de trabajo era, y aun lo es, X Windows.

El X Windows había sido creado en el MIT a mediados de 1984, posteriormente fue adoptado y estandarizado por las empresas diseñadoras de grandes equipos, como AT&T, DEC y Sun Microsystems. Sus primeros usos estuvieron ligados al control de CAD/CAM vía mainframes. Gracias a X Windows las nuevas estaciones de trabajo pudieron evolucionar hasta convertirse en extensiones de los mainframes a los que se conectaban; y, de alguna forma, habían mantenido con vida el interés por el uso de UNIX.

Por lo tanto, para que Microsoft pudiera alcanzar tales terrenos debía de desarrollar, primero que nada, un DOS tan eficiente como UNIX; y, partiendo de ese punto, desarrollar toda una nueva filosofía sobre el uso de Windows.

1.1.8. Alianzas

La década de los 90's se inició con grandes sorpresas en la industria de las computadoras. IBM y Apple firmaron acuerdos de colaboración e investigación para el desarrollo de nuevas tecnologías en hardware y software; la meta principal: computadoras personalizadas "inteligentes" usando multimedia. Posteriormente, a este acuerdo se les unió Motorola.

Hicieron su aparición nuevas computadoras basadas en los microprocesadores 80486 (IBM) y 68040 (Apple), cuyas arquitecturas permitieron alcanzar velocidades entre los 40 y 50 MHz. Sin embargo, el surgimiento de las computadoras con 80486 no se debe a IBM, sino a un grupo independiente de fabricantes de computadoras personales. Este grupo, usando sus propios recursos, creó las normas de diseño ISA (Industry Standard Architecture) y EISA (Extended Industry Standard Architecture), en una forma de competir con el Micro Canal. El éxito de EISA ocasionó una nueva baja en el precio de las computadoras personales.

Microsoft lanzó Windows 3.0 y MS-DOS versión 5.0 en 1991. En ambos sistemas se presentaban nuevas características para poder usar los modos real y protegido sin afectar considerablemente la forma de trabajo de la computadora. En Windows 3.0, Microsoft construyó un medio para correr programas que hubieran sido diseñados para MS-DOS; lo cual significaba que un usuario podría conservar el software "viejo" sin tener la necesidad inmediata de buscar programas para Windows. Por otra parte, Windows 3.0 fue diseñado para trabajar tanto en el modo protegido, como en modo normal.

Nuevas versiones de OS/2 y Program Manager también hicieron su aparición, aunque no con mucha aceptación. De hecho la poca aceptación a OS/2, motivo a Microsoft a generar la versión 3.1 de Windows a menos de un año de haber aparecido la 3.0.

Un gigantesco trecho separa a Windows 3.0 de 3.1. En Windows 3.1 se han agregado partes para controlar periféricos tales como aparatos de sonido e imagen, con lo

cual se introduce a Windows en plano de Multimedia. Gracias a este aspecto, el medio ambiente es menos suceptible a los cambios e inovaciones tecnológicas que puedan llevarse a cabo de aquí a dos años. En su arquitectura principal, Windows 3.1 permite el uso del concepto de objetos ligados, por medio de los cuales dos o más aplicaciones pueden actuar sobre un conjunto de datos específicos.

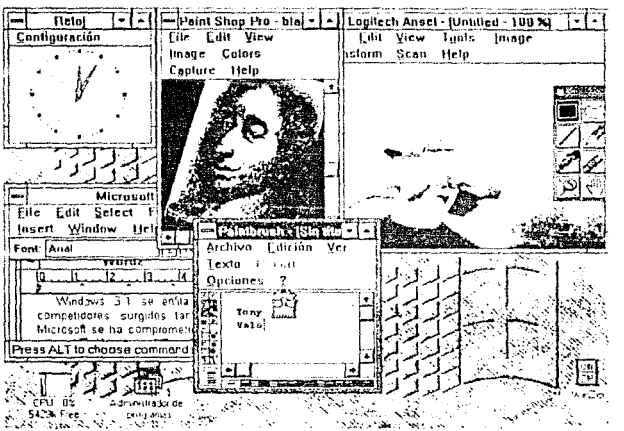


Figura 1-6: Windows 3.1.

1.1.9. Windows Hoy y Mañana

Windows 3.1 se enfila a derrocar a OS/2 y a Program Manager, así como a otros competidores surgidos tanto en Estados Unidos como en Europa. Para lograr esto, Microsoft se ha comprometido en lanzar pronto al mercado una versión de Windows conocida como NT. Windows NT podría correr en sistemas personales y en mainframes; con lo que se podría llegar a una estandarización del software y la portabilidad eficaz de los programas (algo con lo que IBM ha soñado desde los 60's). Recientemente, en el primer semestre de 1992, Microsoft lanzó al mercado mundial dos nuevos elementos para complementar el medio ambiente Windows: *Pen* y *Windows for Workgroups*.

Pen es un dispositivo semejante a los lectores ópticos que se utilizan para el diseño asistido por computadora. Con Pen, un usuario puede "escribir" un texto sobre la pantalla de un monitor de una computadora, en la cual se este corriendo Windows, y el mismo

sistema es capaz de reconocer los caracteres y pasarlos a cualquier tipografía del ambiente. El uso de Pen esta enfocado a funcionar en la captación de información de las nuevas computadoras Laptops y Notebooks, y con la ligera posibilidad de eliminar el teclado en futuro cercano, figura 1-7.

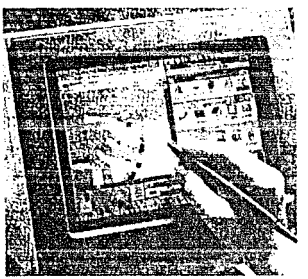


Figura 1-7: Una forma de usar Pen en monitores convencionales.

Windows for Workgroup es una extensión de las características de Windows de hacer funcionar a una ordinaria computadora como terminal inteligente. Windows for Workgroups esta diseñado para trabajar en una ambiente interactivo de red; donde los usuarios compartan todos los recursos de la red por medio de un ambiente Windows. En Windows for Workgroups dos utilerías, Mail y Schedule+, permiten el lazo entre dos o mas usuarios de la red, con lo que varias personas pueden estar realizando un trabajo en forma conjunta y lejana al mismo tiempo.

Por otra parte: algunos fabricantes de hardware han iniciado la comercialización de tarjetas de video y de incremento de velocidad con la capacidad de hacer a Windows más vistoso y más veloz, respectivamente. Con ello, el impacto de Windows hacia el hardware, podría llegar al extremo de hacer a todas las computadoras personales, por lo menos las aparecidas en los últimos dos años, en escalables y prácticamente eternas en su vida útil.

1.2. Una Visión de Windows

1.2.1. Ventanas

Una ventana es una zona reservada de trabajo que puede contener información relacionada a procesos tales como la edición de texto e imágenes, al igual que información documental; es decir, datos referentes a otras ventanas de trabajo o tales ventanas. Una ventana en Windows, en su formato más simple, consta de una zona de trabajo llamada área de trabajo, un marco limitador y la barra de título, figura 1-8.

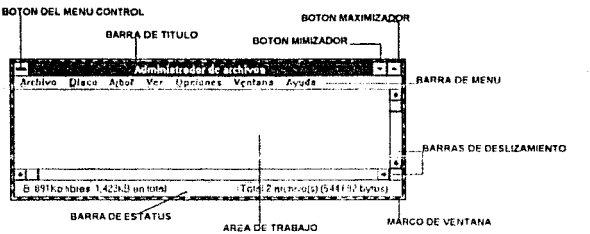


Figura 1-8: Una ventana.

En la barra del título se encuentra el escrito el nombre de referencia que recibe la ventana, según su función y contenido. En la mayoría de las ventanas, por sus características funcionales dentro de Windows, pueden aparecer en el extremos superior izquierdo un botón llamado menú control, igualmente dos botones en la parte superior derecha, el maximizador (flecha hacia arriba) y el minimizador (flecha hacia abajo).

Algunas ventanas pueden presentar una segunda barra en la parte baja del área de trabajo. Esta barra puede indicar alguna característica especial del área de trabajo.

1.2.2. Botones

Un botón es un elemento de control para invocar alguna acción o comando. Windows presenta diferentes tipos de botones depediendo del proceso que se requiera, figura 1-9.

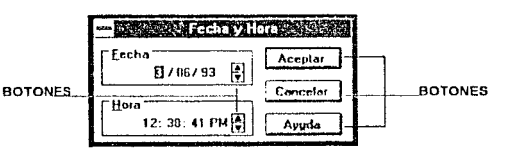


Figura 1-9: Botones.

Algunos botones pueden salirse un poco del formato clásico, figura 1-10. Los botones se activan por medio del ratón, aunque también puede ser activados por secuencias de teclas.

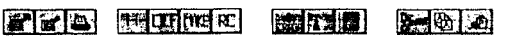


Figura 1-10: Otros tipos de botones.

1.2.3. Menús

Un menú es una lista comandos o de procesos disponibles en una ventana. Los menús pueden presentarse en forma horizontal y vertical. Pero, al igual que con un menu se pueden activar procesos, tambien se puede llamar a otro menu, al cual se le conoce como menú anidado, figura 1-11

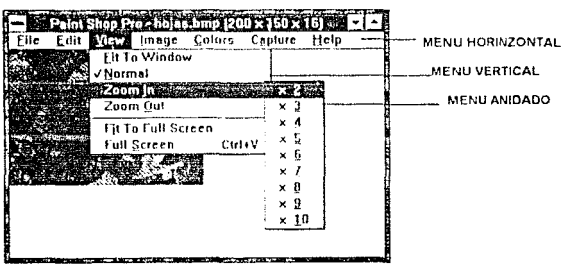


Figura 1-11: Menús.

Algunos menús pueden presentarse como un conjunto de botones, en tales caso al menú se le da el nombre paleta de opciones o toolbox, figura 1-12

Figura 1-12: Paleta de opciones o toolbox.

1.2.4. Barras Deslizables

Una barra deslizable es una especie de botón que aparece cuando el contenido de una ventana no es visible en su totalidad. Dependiendo de la cantidad y tipo de información

que se dezpliegue en una ventana podremos tener barras delizables horizontales y/o verticales. En los extremos de la zona donde se desliza la barra se encuentran dos pequeños botones con flechas, que permiten desplazamientos más lentos al ser colocado el cursor del ratón , figura 1-13

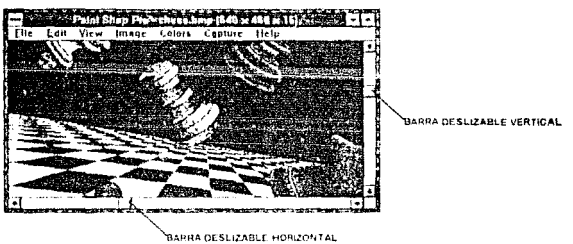


Figura 1-13.

También pueden usarse barras deslizables para controlar algún proceso como busqueda, figura 1-14 o regulación de alguna característica de trabajo, figura 1-15.

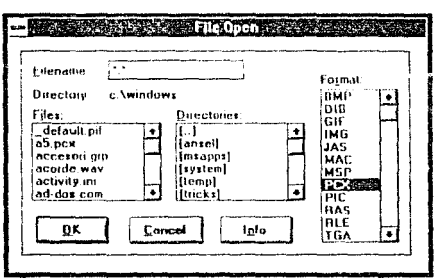


Figura 1-14: Busqueda de archivos según formato.

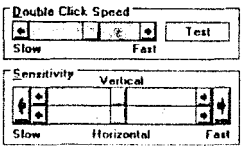


Figura 1-15: Control de la sensibilidad y velocidad de captación para el ratón.

1.2.5. Iconos

Un icono es una representación gráfica de 32 x 32 pixeles de un elemento del medio ambiente. Un icono puede representar un programa ó un grupo de programas, figura 1-16. Igualmente, como se observa en la figura 1-16, se pueden usar iconos sobre un botón.

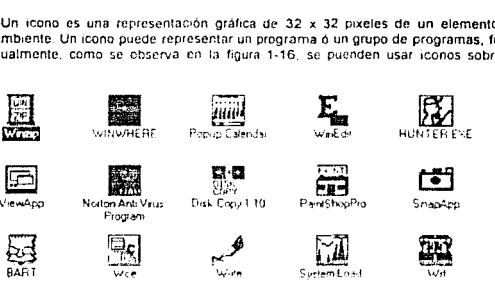


Figura 1-16: Programas representados por iconos.

Cuando se presiona el botón minimizador de una ventana, esta se reduce a un icono que el sistema o el usuario asignaron para referirse a él. A parte de presentarse el dibujo del icono, se presenta un letrero en la parte inferior del mismo que se refiere al nombre del icono o programa al que representa. Cuando el cursor del ratón se posiciona sobre el icono, no sobre el letrero inferior, y se presiona dos veces su botón izquierdo, se activa el menú control. Entre los procesos a elegir del menú control se encuentra el de ***"Restaurar"***, es decir, abrir nuevamente la ventana tal y como estaba antes de pasar al icono; y el de *"Cerrar"*, o sea, cerrar el programa sin restaurar la ventana. Cuando se cierra una ventana (un proceso) desde un icono, este desaparece de la pantalla.

1.2.6. Las Cajas

A pesar de la tremenda sistematización que pueden presentar los procesos en Windows por medio del ratón, es un hecho que algunas acciones requieren la intervención

directa de palabras escritas por el usuario. Los elementos usados para tales situaciones se agrupan en un conjunto de partes conocidas en forma general como las cajas. Por ejemplo, en la inserción de datos se utiliza el elemento conocido como caja de diálogo, donde el usuario puede escribir la información que requiera en el proceso, figura 1-17.

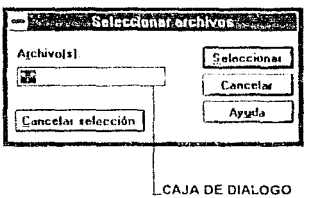


Figura 1-17.

Por otro lado, algunas aplicaciones requieren comunicar al usuario sobre una posible acción a realizar, o bien, enviar un mensaje de éxito de algún proceso. Cuando se trata de alguno de los casos anterior se abre una ventana especial la cual recibe el nombre de caja de mensajes, figura 1-18.

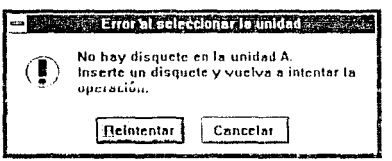


Figura 1-18.

Tanto las cajas de diálogo, como las cajas de mensajes, pueden presentar elementos adicionales como botones, barras deslizables e incluso menús. Un elemento interesante, y que es propio de estas cajas, es la llamada caja combo.

La caja combo es una caja de diálogo asociada a un botón de su lado izquierdo, figura 1-19.

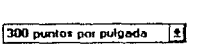


Figura 1-19.

Desde la caja combo un usuario puede escribir un dato, pero si "aprieta" el botón asociado por medio del ratón, a continuación aparecerá una especie de menú con opciones a elegir sobre el tipo de dato necesario, figura 1-20.

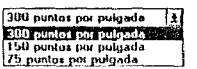


Figura 1-20.

Otros elementos que podemos encotrar dentro de las cajas son los botones de radio y las cajas de chequeo, figura 1-21.

Figura 1-21.

Los botones de radio son una especie de botones que hacen una alegoria de este aparato. Por medio de tales botones es posible indicar alguna caracteristica deseada de la información. En los botones de radio mostrados en la figura 1-21, se tienen dos opciones, **Centrado** o **Mosaico**, solo se puede optar por una; para el caso mostrado se ha elegido **Mosaico**, esto se hace "presionando" la zona circular por medio del cursor del ratón.

Las cajas de chequeo son semejantes a los botones de radio, pero en ellas si se puede elegir más de una opción. Para la figura 1-21, el botón de chequeo, **Ajuste de título**, esta activo, dato que esta marcado con una X.

1.2.7. El Ratón

En el ambiente Windows, al igual que otros en sistemas competidores, la mayor parte de los procesos que involucran el control de información en las ventanas se ejecuta por medio de un dispositivo conocido como ratón. En el mercado es común encontrar ratones con dos y tres botones. La mayor parte del software desarrollado para computadoras personales y que requiere ratón, ha sido diseñado para trabajar con el de dos botones. Windows no es la excepción, y aunque permite dentro de su configuración interna trabajar con ratón de tres botones, por lo regular se usan únicamente los botones de los extremos.

Por medio del ratón, el usuario puede seleccionar botones, ventanas, iconos y menús que se presenten en la pantalla. Igualmente, por medio del ratón, el usuario puede reducir y mover a sus necesidades las ventanas. Todas estas acciones se llevan a cabo presionado el botón izquierdo del ratón una o dos veces rápidamente. En algunos paquetes, como los procesadores de palabras y de imágenes, se puede llegar a utilizar el botón derecho. Por lo tanto, durante el presente trabajo al indicar "presione el botón del ratón", la referencia se enfocara a presicionar el botón izquierdo a menos que se indique lo contrario.

1.2.8. El Teclado

Se puede controlar algunos apectos de las ventanas y menús por medio del teclado, la tabla 1-2 presenta una relación de teclas y sus funciones.

Tabla 1-2.

TECLA	FUNCION
F10	Regularmente usada para pasar al menú horizontal de la ventana en uso.
Alt	Permite seleccionar una opción de menú; el sobre de cada opción presenta un caracter subrayado, cuando se presiona Alt y la tecla correspondiente a ese caracter se activa la la opción.
Flecha izquierda Flecha derecha Flecha arriba Flecha abajo	Para moverse a través de las diferentes opciones de un menú. Cuando se trabaja una ventana de grupo también pueden usarse estas teclas para seleccionar el icono deseado.
Enter	Para confirmar la elección de la opción o icono que se encuentre recalcado su nombre
Esc	Para cerrar un menú.

1.2.9. El Administrador de Programas

El Administrador de Programas, es la herramienta por medio de la cual Windows permite manejar archivos dentro de su medio ambiente, de hecho es el primer y único programa que siempre permance activo mientras Windows este activo, denotándose por el icono de la figura 1-22

Figura 1-22.

Puesto que cada programa se representa por iconos, el Administrador de Programas los agrupa en conjuntos conocidos como grupo. Un grupo es en si una ventana, con su respectivo icono, contenida dentro de la ventana del Administrador de programas, figura 1-23. es importante a señalar que el contenido y organización de los grupos nada tiene que ver con la organización de archivos y directorios que se haya dentro del MS-DOS.

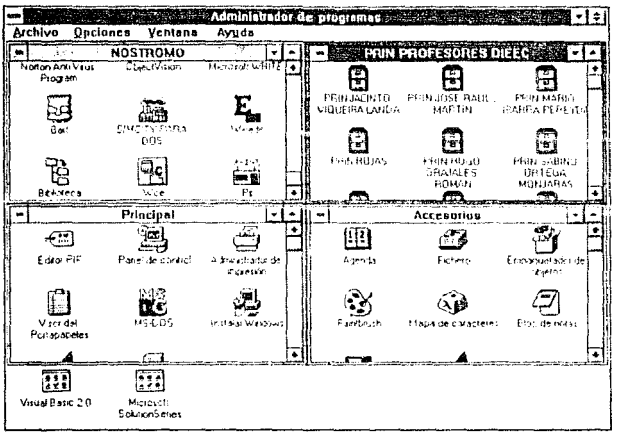


Figura 1-23.

Por medio del menú que ofrece el Administrador de Programas, se puede manejar la presentación de las ventanas e iconos de los diferentes grupos, figura 1- 24

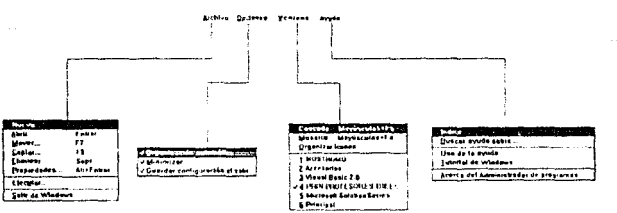


Figura 1-24.

Se pueden realizar algunos efectos interesantes por medio del ratón al estar abierta la ventana del Administrador de Programas. Por ejemplo, si se desea mover un programa de un grupo a otro, basta con colocar el cursor del ratón sobre el icono correspondiente y dejar presionado el botón del raton, después simplemente dirigir el cursor, que ahora arrastrará el

icono al grupo deseado. Este efecto solo se puede realizar entre grupos, y en ningún momento se afecta las estruturas de organización archivo-directorio en el MS-DOS.

1.3.10. El Grupo de Accesorios

El grupo de accesorios es un conjunto de programas proporcionados por Microsoft para permitir al usuario realizar tareas en Windows, como escribir textos, hacer calculos o incluso grabar sonidos, figura 1- 25

Figura 1-25.

La tabla 1-3 enlista los nombres de los programas accesorios y su respectivas funciones.

Tabla 1-3.

ACCESORIO	FUNCION
Write	**Es un procesador de palabras que permite crear documentos.**
Paintbrush	**Es un programa para crear dibujos sencillos o elaborados.**
Terminal	**Permite la emulación de una terminal y de esa forma intercambiar información a otros sistemas via modem.**
Calculadora	**Por medio de este programa se simula una calculadora en modo estandar o cientifico.**
Agenda	**Este programa permite al usuario programar citas por fecha y hora.**
Fichero	**Un emulador de un tarjetero para guardar relaciones de nombre, teléfonos y direcciones. Si se tiene el hardware adecuado se puede efectuar el marcaje automático de los números telefónicos hacia la línea.**

Reloj	**Emula un reloj digital o analógico.**
Transmisor de Medios	**Permite controlar la reproducción de archivos del tipo multimedia (CD-ROM y videodiscos).**
Block de Notas	**Es un procesador de textos sencillo.**
Empaquetador de objetos	**Cuando dos aplicaciones, como Write y Paintbrush, se intercambian información, esta puede ser totalmente dezplegada, o bién, empacada por este programa para usarse posteriormente.**
Grabadora de Macros	**Permite crear al usuario sus propias acciones por medio de secuencia de teclas.**
Mapas de caracteres	**Muestra diferentes tipos de caracteres para ser insertados en otras aplicaciones**
Grabador de Sonido	**Permite reproducir, grabar y editar sonidos si se tiene el hardware adecuado para hacerlo.**

1.3.10. Grupo Principal

El grupo principal representa a varios programas especiales por medio de los cuales el usuario mantiene control y acceso a las cualidades gráficas y administrativas de periféricos y de la información de Windows figura 1-26. La tabla 1-4 enlista los nombres de los programas del grupo principal y sus respectivas funciones.

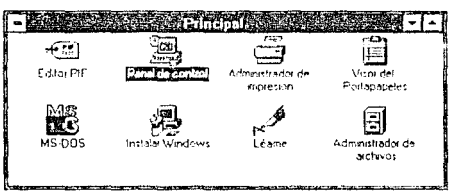


Figura 1-26.

Tabla 1-4.

PROGRAMA	FUNCION
Administrador de archivos	**Semejante a un shell de MS-DOS, Por medio de este programa se pueden ver todos los archivos y directorios dentro**

	de las unidades de disco.
Administrador de impresión	**Cuando se manda a impresión desde Impresión cualquier aplicación, este programa se hace cargo de la comunicación directa entre la computadora y el programa.**
Ejecución de aplicaciones no Windows (MS-DOS)	**Por medio de este programa se puede entrar a MS-DOS y correr cualquier aplicación.**
Editor PIF	**Permite crear archivos .PIF, los cuales permiten adecuar programas No-Windows para ser corridos desde Windows.**
Instalar Windows	**Colocado de forma opcional, permite al usuario instalar aplicaciones dentro de Windows.**
Panel de Control	**Permite configurar el medio ambiente.**
Clipboard	**Este programa guarda tanto texto como imágenes provenientes de alguna aplicacion.**

Una mención especial merece el panel de control pues en si no es un programa, sino un subgrupo de programas que controlan directamente los periféricos de la computadora

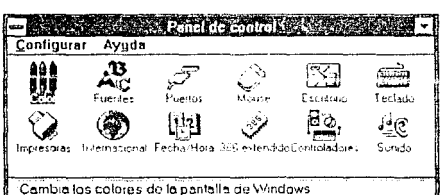


Figura 1-27.

La tabla 1-5 enlista los nombres de los programas que comprenden el panel de control y sus respectivas funciones.

Tabla 1-5.

PROGRAMA	FUNCION
Escritorio	**Permite al usuario selecionar diseños y dibujos para el fondo del medio ambiente,elegir protectores de pantalla, separación de los iconos, ajuste de**

	títulos y establecer velocidad de intermitencia del cursor.
Colores	Permite definir los colores del medioambiente.
Fuentes	Permite establecer o eliminar tipos de caracteres a usarse dentro de las aplicaciones, en especial las de procesadores de texto.
Puertos	Permite definer los parámetros de los puertos de comunicación.
Mouse	Permite establecer la sensibilidad del ratón.
Teclado	Permite ajutar la velocidad de repetición y tiempo de retardo
Impresoras	Permite instalar, configurar y quitar los diferentes manejadores de impresoras usadas por las aplicaciones.
Internacional	Permite configurar aspectos tales como el idioma, formatos numéricos, tipo de cambio de monedas, fecha y hora de acuerdo a las necesidades del usuario
Fecha/Hora	Permite establecer la hora y la fecha.
Red	Permite el control de la conexión en red, si la hay.
386 Extendido	Permite la configuración del medioambiente para usarse con microprocedor 80386.
Controladores	Instala y configura los manejadores de dispositivos especiales como las tarjetas de sonidos.
Sonido	Pemite asignar sonidos a las diferentes aplicaciones del medioambiente.
Mapeador Midi	Para usarse exclusivamente en la configuración de las tarjetas MIDI (Musical Instrumental Digital Interface).

1.3.11. Multiprocesamiento

Windows presenta la caracteristica de multiprocesamiento, es decir, poder correr dos o más programas al mismo tiempo. Por supuesto que decir "al mismo tiempo" solo es una suposición, pues Windows, por medio del MS-DOS, obliga al microprocesador a asignar tiempos de procesamientos a los programas. El multiprocesamiento en windows esta en función del tiempo de trabajo del microprocesador y de la cantidad de memoria principal de la computadora, figura 1-28.

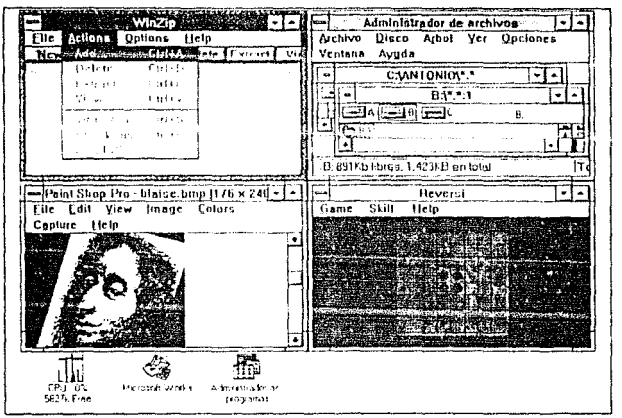


Figura 1-28.

Llegado el momento, un usuario puede tener demasiados procesos corriendo en Windows; y sin darse cuenta desperdiciando tiempo del microprocesador y memoria. Para lograr un control ante tales situaciones, Windows cuenta con la ventana Lista de tareas, la cual permite enlistar los diferentes procesos que estan corriendo en el medioambiente y ejecutar acciones tales como hacer que termine de ejecutarse un programa (Finalizar Tareas) y ordenar las ventanas o iconos en la pantalla (Cascada, Mosaico y Organizar Iconos), figura 1- 29

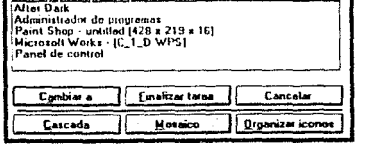


Figura 1-29.

Para activar la ventana de Lista de tareas basta con presionar dos veces el botón izquierdo del ratón en una área fuera de las ventanas e iconos.

1.4. La Necesidad de Programar en Windows

Por lo anteriormente expuesto se subraya la importancia que tiene Windows en el mundo de la computación. Actualmente, la tendencia del cambio de MS-DOS a Windows es total, ningún buen producto de software puede ofrecerse sin su versión para este medioambiente. Es muy probable que en unos cinco años todas las computadoras Windows hayan sustituido al MS-DOS en el uso cotidiano, y aunque Windows tolera aplicaciones de MS-DOS, no cabe duda que las ventajas dentro del medioambiente son más atractivas.

Por ello se hace necesario formalizar la enzeñanza de los principios básicos para programar en Windows. Es común que los estudiantes a nivel universitario se les dificulte la compresión de un sistema operativo como MS-DOS, y la situación empeora cuando se intenta programar; tanto en modo texto como en modo gráfico. Así, en los siguientes capítulos se pretende formalizar una metodología de programación para Windows esperando le sea útil a futuros programadores universitarios.

2

Hardware y Software

Para poder comprender mejor el funcionamiento y programación en el medio ambiente Windows en las computadoras personales es necesario presentar algunos aspectos de hardware y de sofware usados en las computadoras personales.

2.1. La Familia 80x86

La familia de microprocesadores 80x86, diseñada por la compañía Intel, ha sido el corazón de las computadoras personales tipo IBM por más de 10 años. Una gran parte de su evolución ha estado ligada a resolver los diversos problemas y deficiencias que encontraron los usuarios de las computadoras personales.

2.1.1. El 8086

El microprocesador 8086 tiene un bus de datos de 16 bits y un bus de direcciones formado por 20 lineas. Así, por medio del bus direcciones el 8086 es capaz de manejar una memoria máxima de 1,048,576 direcciones. Una parte importante a considerar en el 8086, es que las 16 lineas correspondientes al bus de datos comparten las primeras 16 patas que pertenecen al bus de datos; es decir, en el 8086 el bus de datos y el bus de direcciones estan multiplexados.

Se podria pensar que la memoria al conectarse al 8086 tiene en cada una de sus direcciones 16 bits; pero no es asi. Cuando el 8086 accesa una dirección, en realidad accesa dos; una que corresponderia al primer byte del bus de datos y la otra, que es la continua de la primera, para completar el segundo byte de los datos. De esta forma, dependiendo del tipo información a manejar en la memoria, el 8086 es capaz de optar por manejar datos de 8 ó 16 bits; con un mismo tiempo de acceso en ambos casos. Por lo tanto

se puede afirmar que la memoria máxima en las computadoras basadas en el 8086 es de un 1 Megabytes (1,048,576 bytes).

En su parte interna, el 8086 esta dividido en dos unidades de proceso: la unidad de ejecución (EU), la cual lleva a cabo la ejecución de las instrucciones, y la unidad del bus de interface (BIU), que es la responsable de la comunicación del microprocesador con el exterior, figura 2-1. La EU contiene ocho registros de 16 bits de longitud cada uno para llevar a cabo las operaciones lógicas y aritméticas del microprocesador. A cuatro de estos registros se les denomina como grupo de registros de datos, los cuales son: el registro acumulador (AX), el registro base (BX), el registro de conteo (CX) y el registro de datos (DX). Los cuatro registros de este grupo pueden ser manipulados como datos de 16 bits o dos datos separados de 8 bits cada uno. Los primeros 8 bits de cada uno de los registros se le llama parte baja, y a los ocho restantes parte baja. Así, por ejemplo, podemos tener un acumulador "alto" (AH) y un acumulador "bajo" (AL) dentro del acumulador general (AX).

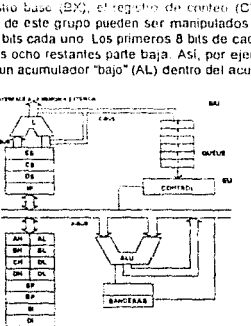


Figura 2-1: Arquitectura del 8086.

Otros dos registros en la EU son el apuntador de pila (SP) y el apuntador de base (BP); constituyen el llamado grupo apuntador. Los registros del grupo apuntador manipulan la pila de memoria que permiten las llamadas de rutinas en los programas. Un tercer grupo esta formado por el registro de índice destino (DI) y el registro de índice fuente (DS), los cuales se encargan de operaciones relacionadas con las cadenas de caracteres. Los tres grupos pasan a formar lo que en la arquitectura de computadoras se le conece como registros de uso general.

Dentro de la EU también encontramos la sección conocida como la unidad aritmética-lógica (ALU) ligada al registro de banderas (FLAGS) y operandos (OPERANDS); usados para indicar las condición de los resultados de la ALU.

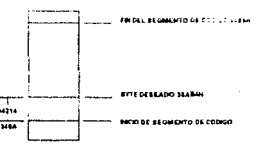
En la sección correspondiente a la BIU ecntramos cinco registros especiales. Ellos son el registro de segmento de código (CS), el registro de segmento de pila (SS), el registro de segmento extra (ES), el registro de segmento de datos (DS) y el registro apuntador de instrucciones (IP); todos ellos de 16 bits. La función de los registros de segmento es administrar la ubicación de los datos y programas usados por el 8088. Igualmente dentro de

la zona de la BIU, esta la cadena de encolamiento (QUEUE) y una unidad sumadora (Σ). La cadena de encolamiento es un grupo de seis registros de 1 byte cada uno, puesto la BIU es la encargada de las comunicaciones con el exterior, cuando se lleva la carga de información esta se va almacenado en los registros de la cola. Dado que siguen una política de pila, cuando se ha completado la información a procesar (una intrucción), la BIU la pasa al sistema de control , en EU, para que se ejecute. La unidad sumadora de la BIU se usa para calcular la dirección de la memoria a accesar.

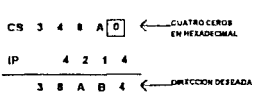
Cuando el 8086 funciona, organiza la memoria exterior en cuatro zonas de 65,536 bytes (64 Kbytes) cada una. Estas zonas reciben el nombre de segmentos. Cada uno de estos segmentos representa un lugar para almacenar información relacionada con un mismo programa, las variables usadas por el programa, la pila (stack) para permitir el llamado de subrutinas y variables extras para cálculos. La dirección de inicio de estas partes son almacenadas por los registros de segmentos según sea el caso.

En algunas ocasiones, un programa para el 8086 puede llegar a ser tan pequeño que no sobrepase los 64 Kbytes, en casos como estos, el microprocesador permite el translape de los segmentos.

La forma en que los registros apuntadores y los registros de segmentos trabajan para poder accesar la información de alguna dirección en especial de la memoria esta dada por la suma de los valores contenidos en ellos. Pero esta suma es muy especial. Supongamos que pretendemos obtener 1 byte ubicado en la dirección 38ABAH, dentro del segmento de código, figura 2-2 (a).



(a)



(b)

Figura 2-2.

El registro apuntador que usamos para este caso es el IP y el registro de segmento CS. El IP contiene la distancia, en hexadecimal, que guarda la dirección del dato entre la dirección de inicio del segmento, cuyo valor es almacenado en CS. La suma de los valores de IP y CS, se realiza considerando el contenido de CS corrido cuatro bits a la izquierda.

Como resultado obtenemos el valor de la dirección del byte, figura 2-2 (b). A este forma de accesar memoria se le llama *modo real* y es común en toda la familia 80x86.

El modo real funciona por igual para el resto de los registros de segmentos, siendo normal que se presente la dirección de la forma *xxxx:yyyy*; donde *xxxx* representa la dirección en hexadecimal del inicio del segmento (valor de algún registro de segmento) y *yyyy* la distancia entre valor deseado y el inicio del segmento. A la parte *xxxx* se le llama segmento y a la parte *yyyy* ajuste; al formato general *xxxx:yyyy*, se le conoce como *dirección absoluta*.

La ventaja del Modo Real es el ahorro de cuatro líneas internas; de no ser así, a la estrutura de los registros apuntadores y de datos se le haría una adición de cuatro líneas extras a cada uno.

2.1.2. El 8088

El microprocesador 8088 es idéntico en funcionalidad al 8086, pero varía en ciertos aspectos de su arquitectura. Así, el 8088 posee un bus de datos separado del bus de direcciones; lo que permite una configuración más sencilla en sus conexiones a los elementos exteriores. Sin embargo, su bus de datos es únicamente de 8 bits, por lo que presenta el doble de tiempo de acceso que el 8086; por lo tanto, debe realizar dos veces los accesos de 16 bits. Para complementar todavía su "lentitud", el 8088 posee una cadena de encolamiento de 4 bytes.

A pesar de este retroceso en comparación con 8086, el 8088 fué el microprocesador que eligió IBM para el desarrollo de sus primeras computadoras personales por su fácil configuración y su bajo costo.

2.1.3. Los Microprocesadores 80186 y 80188

A principios de los ochentas, Intel decidió crear una nueva clase de microprocesadores que incluyeran controladores de memorias y periféricos en una misma pastilla. A la naciente familia 80x86 le correspondieron el 80186 y 80188. Prácticamente son los mismos microprocesadores 8086 y 8088; con la única diferencia de contar con sus controladores integrados a sí mismos y de agregar un grupo intrucciones especiales para realizar las conexiones con el exterior. Los primeros usos de 80186 y del 80188 dentro de las computasdoras personales, fue el de ser el circuito controlador de las primeras tarjetas de red diseñadas para las AT.

2.1.4. El 80286

El microprocesador 80286 presenta grandes avances en su arquitectura interna y externa en comparación con el 8086 y el 8088. El 80286 tiene un bus de datos de 24 líneas, lo que le permite un acceso a memoria de 16 MBytes. Su bus de datos sigue siendo de 16 bits, pero al igual que el 8088 se encuentra separado del bus de direcciones; permitiendo una mejor conectividad con la memoria. En su arquitectura fundamental conserva los mismos registros que el 8086, figura 2-3, pero a pesar de presentar el modo real, el 80286 posee otro modo de direccionamiento muy adecuado para direccionar la memoria superior a 1 Mbyte. A esta nueva forma de direccionamiento se le llama *modo protegido*.

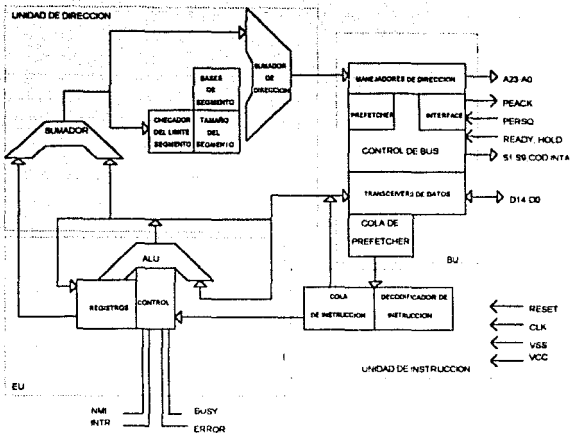


Figura 2-3: Arquitectura del 80286.

El modo protegido es la respuesta de Intel de dotar a la familia 80x86 de un miembro capaz de soportar multiprocesamiento. Para que el 80286 pase del modo real al modo protegido, este debe ser activado por medio del sistema operativo en que se encuentre funcionando el microprocesador. El mismo sistema operativo debe garantizar al 80286 que al pasar al modo protegido encontrará una zona de la memoria reservada para llevar un registro de los posibles programas a correr. A esta zona se le llama *segmento de tablas descriptoras*; y a su vez se divide en otras tres zonas: *tabla descriptora global* (GDT), *tabla descriptora local* (LDT) y *tabla descriptora de interrupciones* (IDT).

Las tablas descriptoras pretenden dar al microprocesador una descripción y prioridad de uso de cualquier programa a correr. Cualquier programa en 80x86 posee, aún en modo protegido, segmentos de datos, de código y de pila. Pueden existir varias tablas descriptoras locales para igual número de programas contenidos en la memoria. La GDT lleva el registro de las diferentes tablas locales y es única. La IDT se encarga de evaluar la estrategia a seguir cuando dos o más programas solicitan servicio de interrupciones al sistema operativo. La idea del modo protegido es permitir que varios programas se difundan en diferentes partes de la memoria con ciertos privilegios de corridas y estancias; las zonas de privilegio se pueden imaginar como círculos concéntricos, figura 2-4.

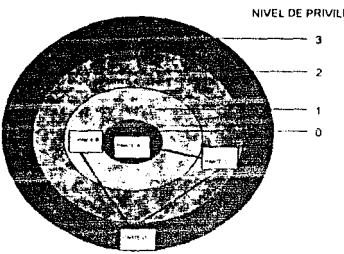


Figura 2-4.

Para llevar a cabo el acceso de alguna dirección en el modo protegido, el microprocesador se vale de un registro especial llamado *apuntador*. Este registro es de 32 bits, y esta dividido en dos partes de 16 bits cada una, figura 2-5. Estas partes son el *selector* y el *ajustador* y poseen la dirección de las tablas descriptoras de la parte que se pretende accesar. Dependiendo de la circunstancia de la tablas descriptoras, se enviará un valor a un registro conocido como *dirección de segmento de base*. El valor contenido la dirección del segmento de base le indica al microprocesador el inicio de segmento a que debe accesar y este se suma, en forma directa, con el contenido en la parte de ajuste del puntero se suma.

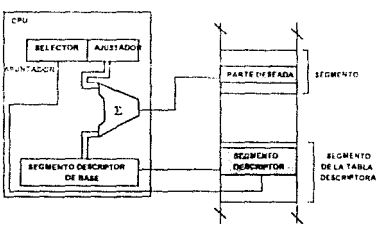


Figura 2-5.

A pesar del muy bien intencionado objetivo de dotar a las computadoras personales de un microprocesador con multiprocesamiento, al 80286 le toco una etapa de transición para lograr cambiar los viejos y muy bien fundamentados patrones del 8086. Por otra parte, el modo protegido fué planeado para trabajar con sistema operativo con las características

de UNIX, por lo que resultó poco práctico para el MS-DOS. La aceptación del 80286 al medio fué el hecho de presentar velocidades de hasta 25 Mhz..

2.1.5. El 80386 y el 80486

Los microprocesadores 80386 y 80486 son un punto y aparte dentro de la familia 80x86. Ambos microprocesadores han formado un subgrupo dentro de la familia con características totalmente diferentes al 8086 y 80286. Estos microprocesadores tienen un bus de datos y direcciones de 32 bits de longitud, por lo que son capaces de accesar memoria de hasta 4 Gigabytes (4,294,967,296 bytes). Quizás la única diferencia severa entre el 80386 y el 80486, en lo que respecta a su arquitectura interna, es que 80486 posee dentro de su misma pastilla un coprocesador matemático 80387, figuras 2-6, 2-7 y 2-8. Del 80386 y del 80486 se han derivado los grupos SX y DX, lo cuales son variaciones en tecnología de los mismo procesadores. Así, mientras un 80386 ó un 80486 "normales" pueden alcazar velocidades entre los 25 y 33 MHz, un 80486 DX2 puede funcionar de 33 hasta 66 MHz.

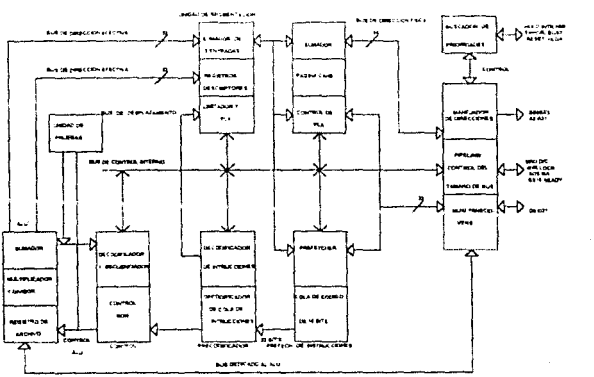


Figura 3-6: Arquitectura del 80386.

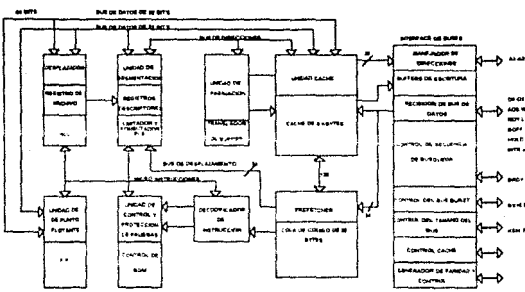


Figura 2-7: Arquitectura del 80486.

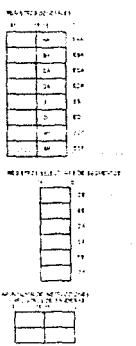


Figura 2-8: Registros típicos del 80386 y del 80486.

Una variación del 80386, es el llamado 386SX, que es una versión corregida y aumentada del 80286. El 386SX posee un bus de datos de 16 bits, pero conserva el bus de direcciones de 32 bits. La estructura de direcionamiento también a cambiado radicalmente, aunque se siguen conservado los modos real y protegido; siendo interesante recalcar que tanto el 80386, como el 80486, funcionan siempre en modo protegido, pero poseen la

cualidad de fragmentar zonas de la memoria para "simular" el modo real, figura 2-9. De esta forma, cualquier tipo de software elaborado para el 8088, el 8086 y el 80286 puede correr sin ninguna dificultad en computadoras con 80386 y 80486. Esta caracteristica de simular un 8086 recibe el nombre de *modo virtual 86*. Dentro del mismo concepto de "virtual", el 80386 y 80486 pueden crear una memoria RAM virtual usando las unidades de disco. La potencialidad de esta caracteristica permite correr aplicaciones de tamaños severamente mayores de 1 megabytes. Ver tabla 2-1

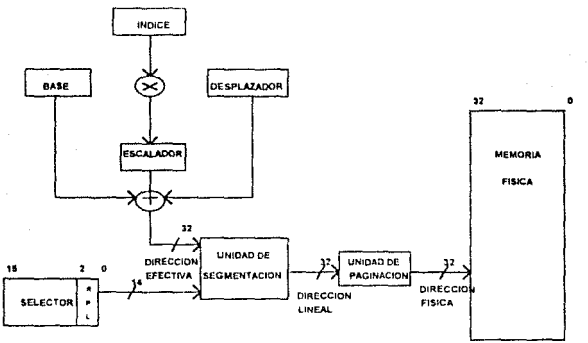


Figura 2-9:Estructura para trabajar el modo protegido y el modo virtual 8086.

TABLA 2-1.

LOS CUATRO MODOS DE TRABAJO DEL 80386 Y DE 80486	
Modo Real	Igual que el 8086/88, pero más rápido.
Modo Protegido (16 bit's)	Igual que el 80286.
Modo Protegido (32 bit's)	Transmisión de la información duplicada en comparación con el 80286.
Modo Virtual 86	Corridas múltiples, aparenta estar en modo real pero se encuentra en modo protegido.

2.2. La Memoria

La primera computadora personal IBM, la XT, tenia una memoria RAM de 64 Kbytes, versiones posteriores permitieron un crecimiento de 256 a 640 Kbytes. El surgimiento del modelo AT permitio abrir un nuevo horizonte de megabytes y gigabytes, sin

embargo, la arquitectura de los microprocesadores 80286, 80386 y el 80486 así como la misma a arquitectura de la AT, a limitado en muchos aspectos la ruptura de la barrera de los 640 K.

2.2.1. Esquema General de la Memoria

Por las secciones anteriores sabemos que que un 80x86 puede accesar un bloque de memoria de 1 Mbyte, figura 2-10 (a). Sin embargo dentro de la arquitectura de la Computadora Personal IBM el acceso a la memoria queda fragmentada en dos partes. La primera parte abarca desde la dirección 0000H hasta la dirección FFFFH totalizando 640 Kbytes; a esta zona se le llama *memoria convencional* y es usada para programas del sistema operativo y del usuario. La otra parte que abarca de la dirección A000H hasta la dirección 10000H, recibe el nombre de *memoria superior* y usada para direccionar la ROM, las tarjetas de video y otros elementos de hardware.

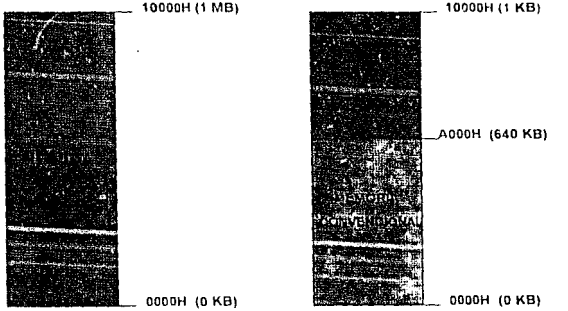


Figura 2-10.

2.2.2. Memoria Extendida y Memoria Expandida

En las computadoras AT que usan 80286, 80386 y 80486 el esquema de la memoria varia muy poco. La arquitectura de la AT y su diseño les puede permitir el acceso a memoria situada mas alla del 1 Mbytes, donde recibe el nombre de *memoria extendida*, figura 2-11. Sin embargo para manejar la RAM mas alla del 1 Mbyte, los microprocesadores deben de pasar al modo protegido, y una gran parte del software no puede trabajar en este modo.

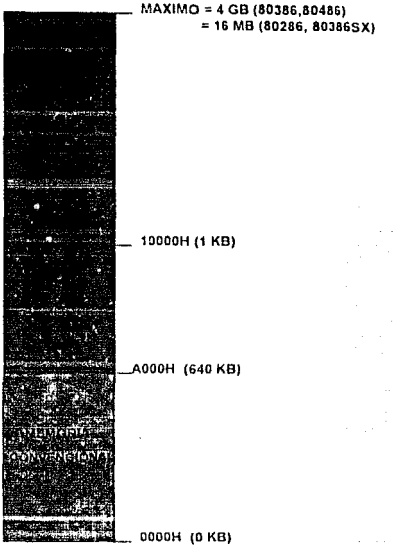


Figura 2-11.

Otra forma de manejar más RAM, sin tener que pasar al modo protegido, es empleando la norma de trabajo LIM-EMS (Lotus-Intel-Microsoft Expanded memory System). La LIM-EMS se funda en el hecho de que dentro de la memoria superior existen zonas no ocupadas por los buffers de video y/o red; por lo tanto es posible colocar una tarjeta diseñada con un buffer que mapee zonas de memoria totalmente ajenas a la RAM original de la arquitectura, a esta zona se le llama *memoria expandida*.

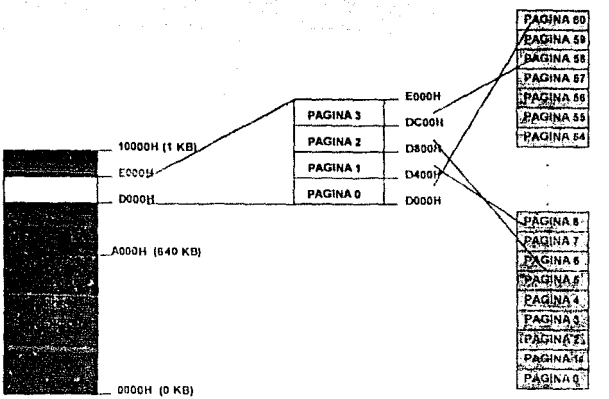


Figura 2-12.

En la figura 2-12 se muestra un ejemplo de funcionamiento de la memoria expandida por medio del LIM-EMS. Un segmento de 64 Kbytes ubicado en la memoria superior sirve al LIM-EMS para mapear la memoria expandida, que es colocada como una tarjeta adicional en las ranuras de la tarjeta madre de una computadora AT. La zona de 64 Kbytes de se (presupone que no es ocupada por ningun otro elemento de hardware y/o software) es particionada por el LIM-EMS en páginas de 16 Kbytes cada una. Igualmente, el LIM-EMS debe haber localizado la memoria expandida y particionarla en páginas de 16 Kbytes. En nuestro ejemplo, suponemos que la memoria expandida se particionado en 64 páginas, lo que nos da un total de 1 Mbyte. El LIM-EMS mapea las páginas de la memoria expandida a las páginas ubicadas en la memoria superior; y por medio una tabla dentro de la memoria convencional, el sistema operativo puede mapear la páginas que se requieran. La LIM-EMS versión 4.0 permite mapear páginas que se incluyan dentro de la memoria convencional.

La memoria expandida manejada por LIM-EMS puede ir desde lo 8 Mbytes (versión 3.2) hasta los 32 Mbytes (versión 4.0). Una norma competidora de la LIM-EMS es la norma EEMS (Enhanced Expanded Memory Specification) de las compañías Ashton-Tate y Quadram. La EEMS fué la primera en presentar el mapeo dentro de la memoria convencional y páginas de longitud variable, pero, prácticamente a sido derrocada por los manejadores de memoria que emplean LIM-EMS. Un manejador de memoria es un programa encargado de supervisar el control de la memoria expandida en forma casi invisible para el usuario. Entre los dos manejadores de memoria más empleados estan el QEMM/386 de Quarterdeck y EMM386 de Microsoft. Este ultimo usado dentro del MS-DOS y Windows.

2.2.3. Programas en Memoria

Como anteriormente se trato, un programa esta divido en partes conocida como segmentos. En la figura 2-13 se muestra un esquema idealizado de un programa.

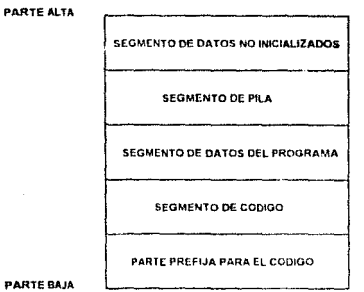


Figura 2-13.

El segmento de datos no inicializado es conocido como ***Heap*** y es usado para las variables dinámicas. Durante los primeros años de vida del microprocesador 8086, el esquema de memoria de la figura 2-13 se fijo en 64 Kbytes, fragmentados en segmentos cuya máxima logitud podia alcanzar 16 Kbytes. Por herencia del CP/M y sistemas operativos similares; al entrar el MS-DOS, los programas de un tamaño de 64 Kbytes obtuvieron la extensión .COM. Al ir aumentando las necesidades de memoria de los programas se plantearon nuevos esquemas de tamaño para los segmentos.

Los conceptos de ***paginación*** y ***overlays*** eran conocidos y usados desde mediados de los 70's en microprocesadores tanto de intel como de sus competidores. Intel y Microsoft replantearon su tales conceptos y como resultado se establecieron los llamados modelos de memoria.

Los modelos de memoria son las formas en que un programa puede poseer sus segmentos de un tamaño más grande que los 64 Kbytes. Para la familia 80x86 se tienen 6 diferentes modelos de memoria que afectan de alguna forma la velocidad de un programa, tabla 2-2.

Tabla 2-2.

MODELO	CARACTERISTICA	VELOCIDAD
TINY	LOS SEGMENTOS DE CODIGO Y DE DATOS COMPARTEN UN SEGMENTO NO MAYOR DE 64 KB.	RAPIDA

SMALL	**LOS SEGMENTOS DE CODIGO Y DE DATOS COMPARTEN UN SEGMENTO NO MAYOR DE 128 KB.**	**RAPIDA**
MEDIUM	**UN SEGMENTO DE CODIGO TANTO PARA LOS DATOS COMO PARA EL CODIGO; SIN EMBARGO, ESTE PUEDE POSSER SEGMENTOS EXTRAS SI ES NECESARIO.**	**REGULAR**
COMPACT	**SOLO SE TIENE UN SEGMENTO DE CODIGO, PERO PUEDE HABER VARIOS DE DATOS**	**REGULAR**
LARGE	**LOS SEGMENTOS DE CODIGO Y DATOS PUEDEN SER VARIOS, PERO INDIVIDUALMENTE SOLO PUEDEN SER DE 64 KB.**	**LENTA**
HUGE	**LOS SEGMENTOS DE CODIGO Y DATOS PUEDEN SER VARIOS, LOS SEGMENTOS PUEDEN SOBREPASAR LOS 64 KB.**	**LENTA**

De esta forma; usando el modelo de memoria adecuado, el programador puede asegurar que su programa no desperdiciara memoria. Los programas que usen los modelos Tiny y Small al ser compilados pueden poseer la extensión .COM y una longitud máxima de 256 kbytes, pero al usar alguno de los modelos posteriores se obtiene la extensión .EXE y las longitudes pueden sobrepasar los 1 Megabytes. Los modelos Large y Huge son usados para programas que corran en modo protegido.

2.3. El MS-DOS

A la par de las computadoras personales, ha surgido y evolucionado el MS-DOS. Actualmente las versiones 5.0 y 6.0 han sido bien recibidas al incorporar manejadores de memoria extendida y expandida, contraladores de periféricos (como impresoras laser) y una interface gráfica.

2.3.1. Lo Que Hace un Sistema Operativo

Un sistema operativo es un conjunto de programas supervisores que controlan los procesos de una computadora. Las características generales de un sistema operativo son:

- Almacenamiento de información.
- Control de procesos.
- Seguridad.
- Interface usuario-computadora

Los sistemas operativos pueden clasificados en tres categorías: *monitores, sistemas operativos tradicionales* y *medioambientes operativos*.

Un monitor es un programa simple que se coloca en la ROM de una computadora, igualmente simple, para el control de acceso de periféricos. El monitor es muy usado en los aparatos que requieren el uso de microcontroladores, tales como válvulas, hornos de microondas, máquinas de escribir eléctricas, cámaras fotográficas automáticas, etc.

Un sistema operativo tradicional se construye como una parte envolvente de un programa monitor, adhiriendo nuevas características de carga, ejecución y almacenamiento de datos a la memoria principal del sistema y a sus periféricos. El MS-DOS es un sistema operativo tradicional.

El medioambiente operativo, también se construye como una parte envolvente del programa de sistema operativo tradicional. La característica escencial del medioambiente operativo es de proveer al usuario de una interface más eficiente. Windows es un medioambiente operativo.

Por lo anterior podemos imaginar la estructura de cualquier sistema operativo como círculos concéntricos que van creciendo en complejidad y en funciones dirigidas al usuario, figura 2-14.

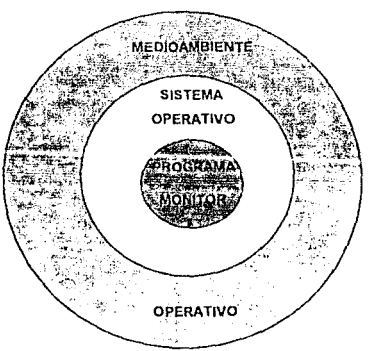


Figura 2-7.

2.3.2. Las partes del MS-DOS

Los componentes del MS-DOS son las siguientes:

- Rutina cargadora
- El BIOS (Basic Input/output System -Sistema Basico de entradas/Salidas)
- El núcleo (Kernel)
- La interface al usuario (Shell)
- Archivos de soporte.

La rutina cargadora es un pequeño programa al nivel del BIOS, su trabajo el de cargar a la memoria RAM de la computadora las partes superiores del sistema operativo: el núcleo y la interface al usuario. Puesto que los primeros sistemas operativos tradicionales se encontraban cargados directamente en la memoria ROM, esto hacia complejo su cambio a nuevas y diferentes versiones. Así, por medio de la rutina cargadora, la computadora es capaz de cambiar la versión del MS-DOS leyendo partes del mismo desde disco flexible o duro. Estas partes son dos archivos, invisibles al usuario, llamados MSDOS.SYS e IO.SYS (o IBMDOS e IBMIO.SYS, respectivamente, en caso de PC-DOS)

El BIOS es el programa monitor contenido en la ROM de toda computadora personal, sin embargo, se encuentra incompleto. La parte complementaria del BIOS es el archivo IO.SYS. Cuando la rutina cargadora encuentra y situa en la memoria RAM al IO.SYS, este se enlaza al BIOS y lo dota de una nueva rutina cargadora, el SYSINIT, que será usada para cargar la parte del núcleo. Al unirse el BIOS con el IO.SYS se establecen los manejadores para los diferentes periféricos y puertos de la computadora, de tal forma que el SYSINIT pueda solicitar la ruta de búsqueda del nucleo.

El núcleo se encuentra contenido en el archivo MSDOS.SYS. Al ser cargado por el SYSINIT a RAM, el MS-DOS es completado en los aspectos de control de procesos, manejo de memoria, interface entre programas y control de archivos. Un segundo archivo es cargado por SYSINIT a continuación del MSDOS.SYS, el CONFIG.SYS. Por medio del CONFIG.SYS se le comunica a SYSINIT donde buscar el archivo COMMAND.COM, que es la interface al usuario e instalar manejadores adicionales para el control de los periféricos y la memoria, figura 2.15. El archivo CONFIG.SYS es visible y modificable por el usuario.

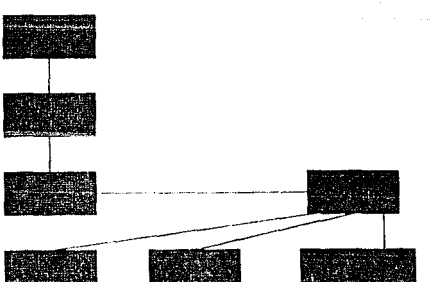


Figura 2.15

La función del COMMAND.COM es la comunicación del usuario con el sistema operativo por medio de una serie de comandos propios del MS-DOS. El COMMAND.COM tiene tres partes la porción residente, la porción en tránsito y la porción de inicialización. La porción residente se encarga de recibir las señales de "rompimiento de programa" tales como Ctrl-C y Ctrl-BREAK. La porción en tránsito es la encarga de recibir y vigilar que se lleven a cabo los comandos como DIR, TIME, DATE y PROMP. La porción de inicialización se carga cuando la computadora inicia su funcionamiento, tiene como función llamar al archivo AUTOEXEC.BAT para cargar atributos predefinidos por el usuario.

Los archivo de soporte son programas y tablas de información diseñados para colaborar con el MS-DOS en control de periféricos y la memoria. Entre los programas se encuentran se encuentran el CHKDSK, el FORMAT y DISKCOPY. Las tablas de información, mejor conocidas como manejadores, se reconocen por la extensión SYS. Así, CONFIG.SYS es la primera tabla de información que accesa el MS-DOS, permitiéndole saber las características de trabajo que desea el usuario.

2.3.3. Interrupciones y Programas

Cuando un programa corre en el MS-DOS, puede solicitar una serie de servicios como acceso de periféricos y control de archivos. El MS-DOS puede establecer una comunicación directa con el programa para permitir la ejecución de comandos o el uso de un conjunto de programas conocido como servicios del MS-DOS y BIOS, figura 2-16.

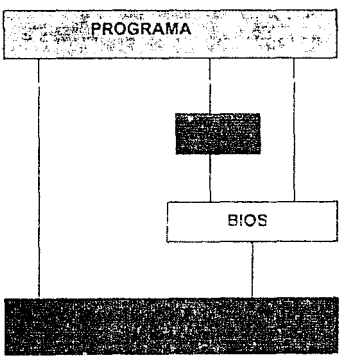
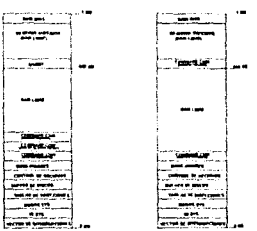


Figura 2-16

Los servicios de MS-DOS y del BIOS usan interrupciones, es decir, un conjunto de programas contenidos en ROM y que interactuan directamentre con el hardware del sistema. por medio las interrupciones se pueden atender los entradas y salidas de las unidades disco; las comunicaciones vía modem de las computadoras; los colores de caracteres, generación de efectos de sonido y control de la impresora; entre otras cosas.

2.3.3. MS-DOS a Memoria

La distribucion del MS-DOS dentro de la RAM se muestra en la figura 2-17. como se cargan las diferentes partes del sistemas operativo, en forma transistiva, figura 2-17(A), y el estado final figura 2-17(B).



(A) (B)

Figura 2-17.

Algunas computadoras pueden poner modulos de memoria en la zona superior a 640 KBytes, por lo que algunos programas, comerciales o del mismo sistema operativo, pueden transladar una parte del sistema operativo a esa zona y permitir más RAM libre. figura 2-18.

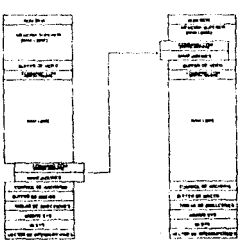


Figura 2-18.

2.3.4. MS-DOS a Windows

Para permitir que Windows se envuelva al MS-DOS, este le proporciona el acceso a cuatro archivos de soporte. HIMEN.SYS, EMM386.EXE, SMARTDRV.EXE y RAMDRIVE.SYS.

HIMEN.SYS es un manejador de memoria extendida que permite, entre otras cosas, pasar una parte del MS-DOS de la memoria convencional a la memoria alta.

EMM386.EXE es un manejador de memoria expandida que es usado para ubicar programas en esa zona de memoria.

SMARTDVR.EXE es un emulador de memoria cache, lo que permite la adquisión de datos desde las unidades de disco más rápido de lo normal.

RAMDRIVE.SYS permite emular una unidad disco extra en RAM, tanto en la zona extendida como en la expandida.

Windows a su vez, explotando el concepto de máquina virtual, permite invocar al DOS desde el medioambiente gráfico y proporcionarle una ventana. Desde esa ventana el usuario puede correr programas del DOS, aunque no es muy recomendable y funcional al corre programas en modo gráfico, figura 2-19.

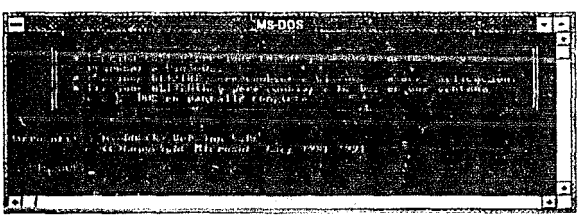


Figura 2-19.

2.4. Metodologías de Programación

2.4.1. Definición de Metodología

Una metodología es un conjunto de procesos y técnicas aplicadas en el ciclo de vida de un sistema de programación. Una metodología se basa en una "filosofía de trabajo" con la que se familiarizan un grupo de programadores durante el período de desarrollo de algún programa; si la filosofía es realmente buena (desde el punto de vista del grupo) esta podrá ser usada en futuros proyectos y pasar a otros grupos de programadores. En la ingeniería de software la filosofía de trabajo recibe el nombre de *modelo*. En los últimos 30

años una gran variedad de metodologías se han desarrollado, algunas por accidentes y otras más derivadas de metodologías anteriores, tabla 2-3.

Tabla 2-3: Algunas Metodologías.

Metodología de Acceso Orientado
Metodología Orientada a la Estructura de Datos
Metodología de Flujo de Datos
Metodología Funcional
Metodología Imperativa
Metodología Orientada a Objetos
Metodología Paralela
Metodología para Tiempo Real
Metodología Orientada a Reglas

Muchas metodologías se ocupan dentro de áreas específicas de la Ciencia de la Computación, lo cuál las convierte en inadecuadas para otras áreas. Sin embargo, algunos autores en ingeniería de software consideran que las metodologías son comunes unas a otras, y que por lo tanto solo son variaciones sobre el mismo tema (modelo). Esto los ha llevado a considerar solo tres "únicas" metodologías: *Metodología de Estructuras*, *Metodología de Datos a Manejar* y *Metodología de Objetos Orientados*.

Esta clasificación también considera las siguientes características:

Notación: Toda metodología debe tener una notación particular en forma escrita y/o gráfica, que permita establecer una abstracción del problema y la solución a seguir.

Arquitectura de Soluciones: Cualquier metodología debe establecer una o varias rutas para resolver un problema. El concepto clásico para poder resolver un problemas es dividirlo y obtener un grupo de soluciones, los que reciben el nombre de módulos.

Arquitectura de datos: Una metodología debe formalizar el uso de las diferentes estructuras de datos a usar en un programa y de ahí partir para formalizar, si es necesario, sus propias estructuras de datos.

Herramientas: El éxito de la aplicación de una metodología dependerá del uso de las herramientas con las que se cuente para resolver un problema. Las herramientas pueden ser programas y lenguajes de programación que permitan usar las características antes mencionadas.

Modelo: Todas las características antes mencionadas deben de girar alrededor de un modelo único y bien definido.

2.4.2. Modelo y Estilo

El modelo (filosofía de Trabajo) es la base por medio del la cual el programador procederá a resolver un problema, y plantear tal solución en instrucciones para la computadora. El modelo es una forma de interpretar el medio ambiente. Todos los individuos entendemos nuestro entorno de una forma diferente, por lo que es muy difícil que un modelo, y por lo mismo una metodología, sea aceptado para reemplazar otro. Por ello,

muchos programadores que se ven forzados a aceptar un nuevo modelo, lo hacen pero conservando parte del *estilo* del modelo anterior.

El estilo es un conjunto de reglas simples e individuales que ha concebido el programador para realizar mejor su trabajo. Un estilo puede estar limitado a funcionar dentro de la codificación de un programa, sin embargo, puede tener repercusiones en etapas posteriores como es la prueba o el mantenimiento. Algunos estilos pueden llegar a ser tan eficientes que son adoptados por otros programadores, y con el tiempo pueden pasar a ser parte de algun modelo y/o metodología. Entre algunos de los estilos más comunes se encuentra el de colocar comentarios antes de alguna función, procedimiento o definición de variables; el de usar nombre cortos para la definición de variables; y el de escribir las líneas de un programa con sangrías que denoten su ubicación profunda dentro del proceso.

2.4.3. Zonas de Trabajo de una Metodología

Aunque una metodología puede influenciar todo el ciclo de vida de un programa, la parte que por lo general es la más afectada es la de diseño. Y, esto se debe a que dentro del ciclo de vida, la fase de diseño de un programa es la que más tiempo de lleva. Sin embargo, dentro de una metodología pueden encontrarse procesos que se usan en las estapas de análisis, codificación, prueba e integración.

Así, por ejemplo, en la metodología de objetos orientados encontramos el análisis orientado a objetos (AOO), el diseño orientado a objetos (DOO) y la programación orientada a objetos (POO), tres fases de trabajo distintas, pero un mismo modelo.

2.4.4. Metodología de Estructuras

La metodología de estructuras es un gigantesco grupo que abarca las siguientes metodologías:

- Programación estructurada.
- Programación Modular.
- Diseño Top-Down.
- Diseño Botton-Up.
- El Método de Michel Jackson.

El modelo en las que se fundamentan estas metodologías es *"divide y venceras"*. Así, cualquier problema a resolver por medio de un programa es dividido en pequeños problemas; y de esa forma se les da una solución particular, figura 2-20. La forma en hacer la división puede ir de lo más simple a lo más complejo (Botton-Up), o de lo complejo a lo más simple (Top-Down).

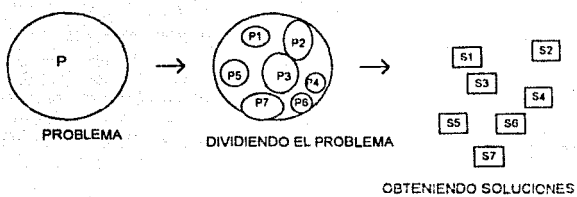


Figura 2-20.

De igual forma, la manera de relacionar las soluciones puede ser muy variada dependiendo del tipo de problema a tratar. El método de Michel Jackson trata este tema, considera que es lo que entra y que es lo que sale, figura 2-21.

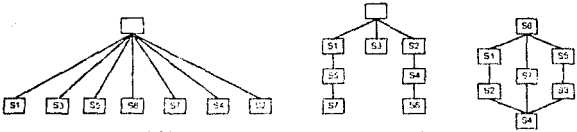


figura 2-21: Algunas formas de relacionar las soluciones.

Una vez que se ha construido la relación de soluciones, y otro pasos de diseño, se puede proceder a la implementación del programa. Para esta fase, la programación estructurada y la programación modular aportan eficiencia en la codificación del programa. Así, la programación estructurada aporta el uso eficiente de las estructuras de repetición, selección y proceso. La programación modular establece la creación de módulos que sean capaces de usarse después en futuras implementaciones. Tales módulos son procedimientos y funciones, que a su vez pueden ser contruidos con otros módulos. Partiendo del concepto de crear y usar módulos, la misma programación modular a generado otra metodología conocida como programación recusiva, muy conveniente para procesos repetitivos.

2.4.5. Metodología de Objetos Orientados

En la metodología de objetos orientados el modelo se basa en tratar de crear una estructura que represente características y funciones de algún objeto. Así , por ejemplo, el objeto auto, podría tener como caraterísticas:

-Número de puertas.
-Número de pasajeros.
-Capacidad de gasolina.
-Precio.

-Capacidad de carga.
-Modelo.
-Color y diseño de las vestiduras.
-Caballos de fuerza desarrollados por el motor.
-Velocidad máxima a alcanzar.

Entre la funciones, donde intervienen en forma directa las características, podríamos establecer:

-Gasto de gasolina en un recorrido con duración de x tiempo.
-Depreciación del precio en función del tipo de modelo.

Por supuesto, que el desglose de características y funciones de un objeto estará en función de lo que pretende alcanzar el programador. Así, el desglose de un objeto auto para un programa de juego podría ser muy simple en comparación para un programa de simulación de rendimiento. De igual forma puede intervenir una muy particular visión del programador respecto al objeto, pudiendo limitar el resultado funcional. A esta unión de características y funciones de un objeto se le llama *encapsulado*.

Para sobrellevar tales visiones y limitaciones, dentro del mismo modelo de los objetos orientados se incluyen los conceptos de *clase* y *herencia*. En nuestro ejemplo anterior, el objeto auto cuenta con características y funciones que podríamos usar para otro objeto; el objeto camión. Y, si pretendiéramos introducir otro objeto como el objeto bicicleta y el objeto motocicleta, concluiríamos que pretendemos manejar un conjunto de objetos correspondientes al tipo vehículo. La mejor táctica para la creación de objetos con características comunes es el de crear una estructura general, la clase; de la cual se derivarán subclases que hereden una parte de las características de la clase general.

El lazo de herencia entre las clases y las subclases se puede llevar hasta varias generaciones de objetos diferentes e incluso con la agregación de características y funciones de otras clases distintas. Así, la clase vehículo, para lograr más detalle en las descripción de sus objetos derivados, puede heredar características y funciones de las clases correspondientes a motores y sistemas eléctricos, las cuales a su vez también podrían tener subclases. figura 2-22.

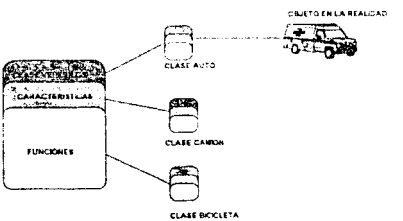


Figura 2-22.

Formadas las clases y subclases del objeto, se procede a crear variables y/o constantes que representen en el programa al objeto u objetos; a esto se le llama *instancia*. El manejo de las instancias dentro de la estructura de un lenguaje no debe ser muy diferente del manejo de variables y constantes que en lenguajes tradicionales como BASIC o FORTRAN; sin embargo algunos lenguajes orientados a objetos implementan mecanismos especiales para lograr un mejor control de las instancias.

Dado que los objetos pretenden ser una representación del mundo real, y que todo en la realidad esta conectado de alguna forma, se establece el concepto de *mensaje*. Un mensaje es un medio por el cual un objeto se comunica con otros. El mecanismo de mensaje es armado por el programador por medio de funciones muy ligadas a las caracteristicas del objetos, para de esta forma tener una evaluación de las instancias.

Otro concepto de los objetos orientados es el de *polimorfismo*, por medio del cual se establece la contrucción de funciones con un mismo nombre pero con diferente estructura de trabajo. De tal forma, que en las ocasiones en que el programador debe hacer referencia a procesos semejantes, pero con datos de entrada distintos, en lugar de asignarle a cada proceso un nombre diferente, puede escribir el mismo nombre para cada proceso. Por supuesto, la idea es facilitar al programador la escritura de su programa para evitar equivocaciones en las llamadas de procesos comunes.

La metodologia de objetos orientados en los ultimos ocho años ha tomado una fuerza tremenda en la creación de los programas, apesar que para muchos programadores la consideren demasiado compleja en uso de grandes sistemas. El mayor atrayente de los objetos orientados son los mecanismos de clase y herencia, lo que permite a un programador el acceso de bibliotecas, formadas por clases, de las que puede tomar todo o sólo lo que necesite sin hacer grandes cambios, lo cual no se podia hacer en las bibliotecas modulares. Varios autores sobre el tema, como Niklaus Wirth, opinan que los objetos orientados son una continuación de las metodologias de estructuras; y que por lo tanto es lógico que la ingeniería de software actual se enfoque a ellos.

2.4.6. Definición de Plataforma de Programación

Una plataforma de programación se define como el medioambiente, hardware y software, en que se pretende crear un programa. Algunas plataformas presentan la caracteristica de portabilidad, es decir que puede desarrollarse un programa capaz de correrse en computadoras totalmente distintas. Esto se debe, esencialmente, a que el software presente en la plataforma original esta estandanzado para diferentes equipos; sin embargo, en la realidad es casi imposible encontrar un software totalmente estandar, por lo que algunas plataformas estan dotadas de programas convertidores para el caso de importación y exportación de programas a otros sistemas.

Entre los aspectos importantes a considerar en hardware en una plataforma son:

- Memoria RAM para el usuario.
- Velocidad del sistema.
- Periféricos
- Forma de procesamiento (monousuario o multiusuario).
- Capacidad en red.

Y en software son:

- Sistema operativo.
- Lenguajes de programación.
- Herramientas de programación

La relación entre los apectos de una plataforma y los programadores pueden dar origen a toda una gama de comportamiento y características para el software a crear. Esta relación haya su enlace principal en las funciones y servicios que proporciona el sistema operativo. Analicemos el caso clásico de programar un sistema mainframe, figura 2-23.

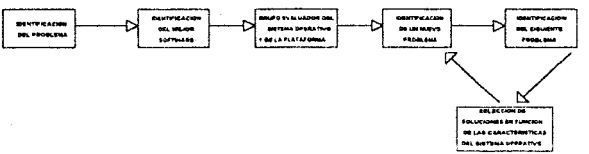


Figura 2-23.

Aquí, la selección de elegir el software adecuado solo puede pasar a la siguiente etapa si el sistema operativo y las condiciones de la plataforma son favorables. Para la solución de nuevos problemas a presentarse en el sistema, como correcciones y actualizaciones, seguirá existiendo una dependencia hacia las caracteristisca del sistema operativo.

Para el caso de las computadoras personales, el programador debe de conocer a fondo el hardware a usar y las características del sistema operativo. Dado la mediana complejidad que puede encerrar en su arquitectura las computadoras personales es la responsabilidad del programador establecer limitaciones y variaciones para el software a crear, figura 2-24.

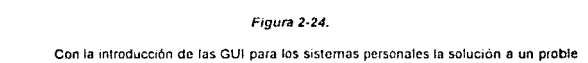


Figura 2-24.

Con la introducción de las GUI para los sistemas personales la solución a un proble puede tener una variación, figura 2-25.

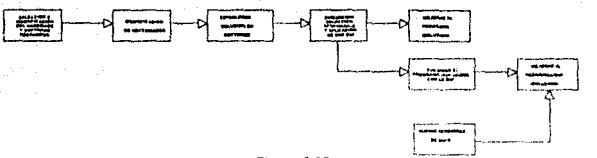


Figura 2-25.

Uno de lo ejemplos más interesantes sobre el caso mostrado de la figura 2-25 lo es multieditor WordPerfect. Durante casi cuatro años este programa fue considerado de los mejores paquete de media y alta edición, casi comparable con el Ventura de Xerox. WordPerfect 5.1 resultó la versión más exitosa para MS-DOS, con la tendencia que sufrió el mercado hacia Windows, la compañía desarrolladora del paquete decidio sacar una versión para el medio ambiente gráfico. Muy pocos usuarios de la version 5.0 y 5.1 se pasaron a Windows para ganar las características de la GUI. Sin embargo, otros grupos de usuarios que buscaban un editor de texto poderoso dentro de Windows, encontraron respuestas en la versión de WordPerfect para Windows. Como una forma de complacer a ambos bandos, la compañía WordPefect ha anunciado la salida de la version 6.0 de su editor. Esta vez el multieditor proporciona su propia GUI, capaz de correr tando en ambiente MS-DOS, como en Windows.

2.5. Lenguajes de Programación

2.5.1. El Lenguaje de Programación C

El lenguaje de programación C fué desarrollado por Dennis Ritchie en el año de 1972. C fué el resultado del trabajo de Ritchie con Brian Kernighan en desarrollo del sistema operativo UNIX. La característica especial de C es el de combinar estructuras y procesos complejos de los lenguajes de alto nivel, como FORTRAN y BASIC, con instrucciones de lenguaje ensamblador. Así, el programador puede usar intrucciones repetitivas y selectivas, que en los lenguajes ensambladores suelen ser difíciles de implementar; y operaciones directas con los registros de trabajo de la computadora, que en los lenguajes de alto nivel suelen ser muy lentas.

C presenta un conjunto de palabras reservadas muy pequeño en comparación con otros lenguages, sin embargo, posee la capacidad de llamar a funciones y procedimientos de librerías creadas por el usuario, y que junto con la potencialidad de sus características de lenguaje ensamblador, logran que se vaya incrementando sus aspectos de programación.

Por lo anterior, C se ha convertido en lenguaje favorito para los grandes y complejos sistemas de programación; lo cual permite, hasta cierto punto, portabilidad de programas entre computadoras que usen el mismo lenguaje.

Sin embargo, es necesario recalcar que el hecho de la portabilidad se ha ido perdiendo debido al surgimiento comercial de una variedad interminable de compiladores de C.

2.5.2. El Lenguaje de Programación Pascal

El lenguaje de programación Pascal, en honor del filosofo y matemático francés Blaise Pascal, fué desarrollado a finales de los 60's por el científico suizo Nicklaus Wirth. Pascal surgio como el intento de presentar la programación estructurada como una metodología ideal para programar. Durante toda la década de los 70's, Pascal fué adoptado en muchas universidades del mundo como un lenguaje para programar sistemas de mediano tamaño.

El conjunto de instrucciones de Pascal es relativamente mayor al de C, pero su sintáxis es mucho más flexible. Durante la década de los 80's, varios grupos de usuarios de Pascal trataron de imponerlo como el lenguaje ideal para las computadoras personales, esto ocasionó una invasión de todo tipo de compiladores de Pascal, lo que provoco que fuera imposible su portabilidad.

Actualmente Pascal se continua usando para hacer pequeños y medianos sistemas de programación.

2.5.3. El Lenguaje de Programación C++

A principios de los 80's, Bjarne Stroustrup (un investigador de los laboratorios AT&T Bell, en Estados Unidos) realizó una investigación sobre la forma de programar de entonces. Concluyó que en un lapso de 20 años los programas de uso medio habían crecido de 10 Kbytes (1960) a 500 Kbytes (1980); y que de continuar el desarrollo del hardware al mismo ritmo de ese momento, se alcanzaría para 1990 programas de 1 a 10 Mbytes. Por lo tanto, el programar tales sistemas implicaría una gran inversión, por lo que Stroustrup se dió a la tarea de desarrollar un lenguaje de programación que permitiera solucionar tal problema.

Despues de hacer una evaluación de los lenguajes y metodologías que en aquel mometo imperaban; Stroustrup consideró crear un lenguaje híbrido, tomando en cuenta las características del lenguaje C y la metodología de objetos orientados. El resultado fué C++.

El lenguaje C++ apareció comercialmente en 1986, causando una auténtica revolución sobre el software. C++ abrió las puertas, para muchos programadores deconocidas, de los objetos orientados. Practicamente con el mismo número de palabras reservadas y características del C original de Dennis Ritchie, pero conteniendo las estructuras necesarias para soportar la creación de los objetos orientados; C++ ha sido absorbido por la mayor parte de los programadores de C.

Al igual que C, C++ a comenzado a sufrir los extragos de la comercialización de los compiladores que ofrecen al usuario "el mejor C++". Sin embargo, los fabricantes de tales compiladores han tratado de mantener una forma estándar.

Para un gran sector de programadores, el lenguaje C++ pueder ser el único lenguaje que sobreviva a los grandes cambios de las interfaces en computación como serían los reconocedores de voces y sistemas semejantes al Pen. En la figura 3.26 se presenta una reseña, según Stroustrup, de los aspectos que han acompañado ha cada generación de lenguajes de programación.

SOFTWARE DE LOS 50'S EXTRAÑO Y EXOTICO
SOLO PARA MENTES PRIVILEGIADAS

SOFTWARE DE LOS 60'S MAS ACCESO A LA GENTE NORMAL
COMIENZA A VOLVERSE COMUN

SOFTWARE DE LOS 70'S LLEGA A TODOS A LAS PRINCIPALES ACTIVIDADES
DE LA VIDA COTIDIANA LA DIVERSION HA EMPEZADO.

SOFTWARE DE LOS 80'S PARA TODOS LOS GUSTOS SIES QUE LO PUEDES PAGAR
SE VUELVE NECESARIO Y A SU VEZ ES UN ESTORBO

SOFTWARE DE LOS 90'S MULTIFUNCIONAL UN PROGRAMA
HACE MARAVILLAS VARIOS HACEN MILAGROS

Figura 3-26.

3

Conceptos Avanzados

El presente capítulo explicará algunos aspectos avanzados de la estructura y funcionamiento interior de Windows, al igual que se establecerán los primeros conceptos básicos para comenzar a programar.

3.1. Dentro de la Estructura de Windows

3.1.1. Directorios y Archivos

Cuando se instala Windows en una computadora personal, se crea el subdirectorio principal WINDOWS. Dentro de este, se crearan los subdirectorios SYSTEM y TEMP. En el directorio WINDOWS se alojarán los archivos que conforman los programas accesorio, así como archivos complementarios.

En el subdirectorio SYSTEM se encuentran archivos medulares del medioambiente, los cuales poseen las extenciones EXE y DLL. Un segundo grupo, igual de importante, se aloja en subdirectorio, los archivos con extenciones CPL y DRV. Un tercer grupo de archivos es el formado por aquellos con las extenciones FOT y TTF; estos archivos corresponden a los diferentes tipos de letras a usar dentro del medio ambiente.

El subdirectorio TEMP es usado para guardar archivos de memoria virtual. Cuando un proceso sobrepasa la cantidad de memoria disponible; Windows puede usar un archivo temporal como memoria virtual; Sin embargo, si en una computadora se ejecutan muchos programas que sobrepasen la capacidad de la memoria podria llegarse el caso de saturar el subdirectorio de archivos de este tipo.

3.1.2. La Escencia de Windows

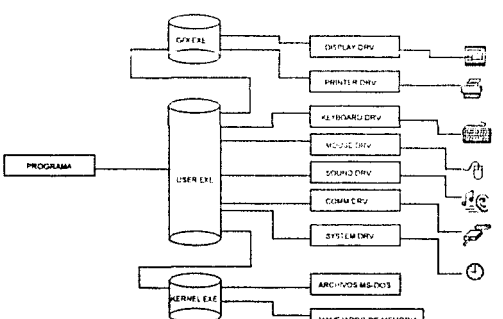
Cuando se corre Windows, se invoca desde el DOS el programa WIN. El programa WIN es del tipo COM y se encuentra en directorio WINDOWS. La función de este programa es el de verificar el tipo de microprocesador y caracteristicas de la memoria. Los tipos de microprocesador viables para el uso de Windows 3.0/3.1 son 80286, 80386 y 80486, para la verificación de la memoria, WIN se vale del HIMEM.SYS.

Enterado del tipo de microprocesador y del tipo de memoria, WIN activa DOSX.EXE, modo estándar para el microprocesador 80286, o WIN386.EXE, modo extendido para microprocesadores 80386 y 80486. Tanto DOSX.EXE, como WIN386.EXE, son manejadores de memoria que explotan las caracterisiticas de los microrprocesadores por lo cual se activan. A continuación se activan los programas KERNEL , USER y GDI.

El KERNEL es el corazón de Windows, controla los manejadores de memoria y del DOS a los que puede tener acceso una aplicación sobre el medio ambiente. El KERNEL se puede presentar en dos archivos, KRNL286.EXE o KRNL386.EXE según el microprocesador.

El GDI.EXE, acrónimo de Graphics Device Interface, crea la interface gráfica del medioambiente. Para ello, GDI hace contato con los archivos DRV correspondientes a la tarjeta de video y a la impresora conectadas a la computadora.

El USER.EXE proporciona la comunicación directa entre el usuario y la aplicación activa en el medioambiente. USER maneja el resto de los archivos DRV correspondientes a teclado, ratón, puertos de comunicaciones, tarjetas de sonidos, etc. Vease figura 3-1.



Figuras 3-1.

3.1.3. Los Archivos .INI

Así como se aplicó el termino "escencia" para los programas Kernel, GDI y User, para el caso de los archivos INI se podría aplicar el término "conciencia". Pues son gracias a estos archivos por medio de los cuales Windows se rige y establece su presentación y comportamiento en cada nueva sesión.

Dentro de un archivo INI se almacena información sobre la presentación, tamaño, colores, tipos de letras a usar, tipos y formas de comunicación hacia los periféricos. Al instalarse Windows por primera vez, son grabados cuatro archivos INI, ver tabla 3-1. Durante la vida útil del sistema los archivos cambian de acuerdo a la información recibida por el panel de control o por programas especiales.

Tabla 3-1.

Nombre del archivo	función
WIN.INI	Configuración total del medioambiente.
SYSTEM.INI	Dertermina aquellos procesos que será ejecutados primero.
CONTROL.INI	Guarda información sobre patrones de colores a usarse en el medioambiente.
PROGMAN.INI	Guarda información sobre el estado del medioambiente a ser terminada una sesión.

Algunos programas puede ser instalados junto con sus propios archivos INI, indicándole al medioambiente que hacer para efectuar un inicio de corrida controlada. La estructura de un archivo INI es muy fácil de entender. Cada una de las partes a controlar, se encuentran escrita dentro de paretesis cuadrados ([]), variables como coordenadas o colores manejan valores numéricos los cuales son asignados por medio del signo igual (=) Los comentarios se escriben después de un punto y coma. El listado 3-1 muestra una parte del archivo WIN.INI.

Listado 3-1.

```
[windows]
spooler=yes
run=
Beep=yes
NullPort=None
BorderWidth=5
KeyboardSpeed=31
CursorBlinkRate=620
DoubleClickSpeed=452
Programs=com exe bat pif
Documents=
DeviceNotSelectedTimeout=15
TransmissionRetryTimeout=45
swapdisk=
CoolSwitch=1
ScreenSaveActive=0
ScreenSaveTimeOut=300
MouseTrails=-7
```

```
KeyboardDelay=0
DosPrint=no
load=c:\mouse\pointer.exe
device=HP LaserJet III,HPPCL5A,LPT1:

[Extensions]
doc=C:\WINWORD\winword.exe ^.doc
dot=C:\WINWORD\winword.exe ^.dot
crd=cardfile.exe ^.crd
trm=terminal.exe ^.trm
txt=notepad.exe ^.txt
ini=notepad.exe ^.ini
pcx=pbrush.exe ^.pcx
bmp=pbrush.exe ^.bmp
wri=write.exe ^.wri
rec=recorder.exe ^.rec
ico=icondraw.exe ^.ico
dsk=desktop.exe ^.dsk
dnd=desknav.exe ^.dnd
dpn=dialer.exe ^.pho
pho=phones.exe ^.pho
ZIP=C:\SAMS\WINZIP.EXE ^.ZIP
LZH=C:\SAMS\WINZIP.EXE ^.LZH
ARC=C:\SAMS\WINZIP.EXE ^.ARC
HLP=winhelp.exe ^.hlp
rtf=C:\WINWORD\winword.exe ^.rtf
wdb=C:\MSWORKS\msworks.exe ^.wdb
wps=C:\MSWORKS\msworks.exe ^.wps
wks=C:\MSWORKS\msworks.exe ^.wks

[FontSubstitutes]
Helvetica=Arial
Times=Times New Roman
Tms Rmn=MS Serif
Helv=MS Sans Serif
```

3.2. Adornos del Medio Ambiente

La mayor parte de los medioambientes tratan de presentar aspectos que permitar trabajar mejor al usuario. En muchas ocaciones estos aspectos son meros adornos que crean una situación ergonomica. El mejor ejemplo para tales caso es el uso de los colores en imágenes. A cuntinuación se reseña una parte de estas cracteristicas dentro de Windows.

3.2.1. Mapas de Bits y Colores

Los mapas de bits (bitmap's) son imágenes formados por la agrupación masiva de pixeles. Los mapas de bits pueden tener diferentes usos dentro de Windows; desde la manipulación de información de imágenes en bases de datos; hasta el mero elemento decorativo como el que se usa como fondo del medio ambiente, figura 3-2.

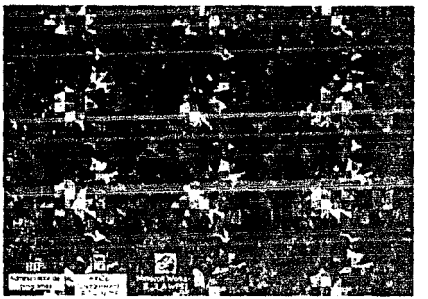


Figura 3-2: Un mapa de bits usado como fondo (papel tapiz).

Es lógico que para el mejor lucimiento de los mapas de bits y de las ventanas tiene una gran influencia la cantidad de colores que maneje el sistema. Por regular Windows maneja 16 colores básicos, formados por diferentes tonalidades de tres colores: rojo, verde y azul. Por medio de la opción Color , del panel de control, el usuario puede manejar las diferentes tonalidades y combinaciones de estos colores, al igual que hacer las propias, figura 3-3.

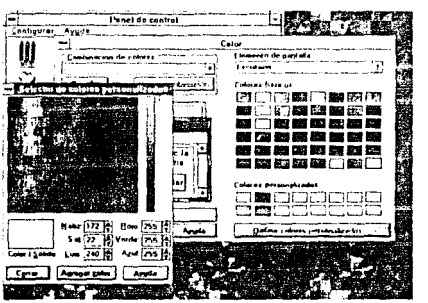


Figura 3-3.

3.2.2. Tipos de Letras

Sin duda uno de los elementos que causa más admiración en un medioambiente gráfico son los tipos de letras que se pueden manejar. Windows maneja tres categorías de tipos de letras para computadora: Raster, Vector y True Type.

La categoría raster se refiere a letras creadas a partir de mapas de bits. La cualidad esencial de los tipos raster es que pueden ser rápidamente escalados por estar formados por pixeles, figura 3-4.

Figura 3-4.

Los tipos de letra vector son aquellos formados por líneas y puntos, lo cual permite una mejor presentación, pero son más lentos al ser escalados, figura 3-5.

Figura 3-5

Los True Type son tipos de letras que usan características de raster y vector, los cual les permite ser más rápidamente escalables (como los raster) y tener más finura en el trazado de los caracteres (como los vector), figura 3-6.

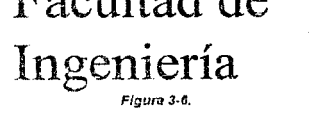


Figura 3-6.

de letras de las tres categorías; aunque la mayor parte de los usuarios se están inclinado por preferir los True Type. Un usuario puede adquirir más tipos de letras y agregarlos a su sistema por medio de la opción del panel de control; sin embargo se debe tener cuidado en lo que respecta al tipo de tablas de caracteres que manejan los tipos de letras, pues puede limitar los textos en lenguajes como español, francés y alemán. Las tablas que se manejan dentro de Windows son la OEM de IBM y la ANSI, siendo esta última la que contiene el conjunto de caracteres ajenos al alfabeto inglés.

3.2.3. La Ayuda

Uno de los elementos más sorprendentes de Windows es la parte correspondiente a la ayuda. A diferencia de los despliegues de los archivos de ayuda clásicos en DOS, la ayuda en Windows son programas individuales de aquellos que los mandan a llamar cuya complejidad de presentación puede superar a uno que otro programa similar.

Una ventana de ayuda posee un menú de opciones para editar e imprimir las partes interesantes. Además, posee una barra de botones que permiten al usuario avanzar o retroceder, mostrar un glosario palabras reservadas sobre el texto, mostrar un historial de los temas vistos en la presente consulta y un botón para regresar al punto de partida inicial (el índice), Figura 3-7.

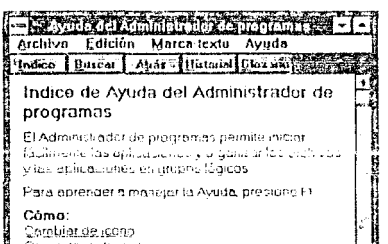


Figura 3-7.

Por si fuera poco, la misma ventana de ayuda proporciona su propia ventana de ayuda. Sobre el texto mismo se pueden encontrar ciertas partes de color distinto, y una vez sobre ellas el cursor cambia su forma de flecha y pasa a convertirse en una figura similar a la de una mano; lo cual significa que existe una sección que explica en profundidad esa parte y sólo se debe presionar el ratón para accesarla, a este concepto se le llama hypertexto.

Por último, la programación de una ventana de ayuda es tan diferente y compleja que incluso posee su propio compilador comercial de programas de ayuda, totalmente separado de los que se podrían usar para crear un programa en Windows.

3.2.4. Guarda Pantallas

Los guarda pantallas son efectos animados programados por medio del la opción escritorio del panel de control, el objetivo de estos programas es el de activar una animación después de un tiempo de no estar usando el sistema, de esa forma se evita el marcaje del medioambiente sobre la pantalla del tubo de rayos catódicos del monitor.

Windows posee, en la opción de escritorio del panel de control, su propios guada pantalla. También existen paquetes comerciales, figura 3-1, que al ser instalados pasan a formar parte de las opciones de Escritorio

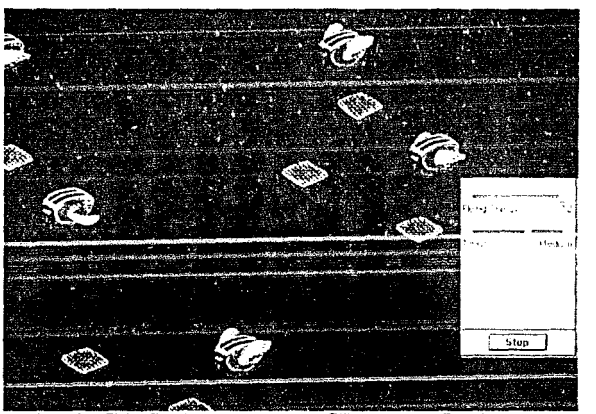


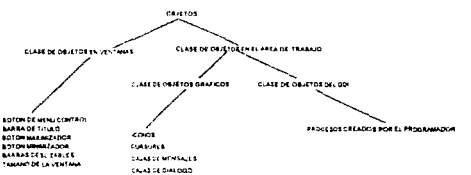
Figura 3-8: Flying Toasters, programado por Bill Stewart para Berkeley System.

Como parte del mismo concepto de protección, los guarda pantallas pueden ser programados para desactivarse por medio de una clave. Así, si un usuario deja su máquina "protegida" por un guarda pantalla, y una persona ajena trata de usar la computadora, el programa no lo dejara entrar y mucho menos le permitira ver que se esta procesando en ese momento.

3.3. La Dinámica de los Objetos

3.3.1. Objetos en Windows

A partir de Windows 2.0, Microsoft introdujo el concepto de los objetos orientados en sus sistemas. Microsoft define para los programas en Windows dos tipos de clases de objetos: La clase ventana principal y la clase proceso de trabajo. La clase proceso de trabajo se divide en dos subclases: la clase de gráficos y la clase generada por el programador, figura 3-9.



Figuras 3-9.

La clase ventana principal, como su nombre lo indica, es donde se lleva acabo el desarrollo de la aplicación. La clase ventana principal cuenta con una serie de características tales como el botón de menú control, la barra de título, el botón maximizador, el botón minimizador, las barras deslizables y capacidad de variación de tamaño por medio del cursor del ratón. Otras ventanas podrían abrirse heredando las características del ventana principal, pero no así el control del programa. A estas ventanas se le llama ventanas hijas (child windows), figura 3-10.

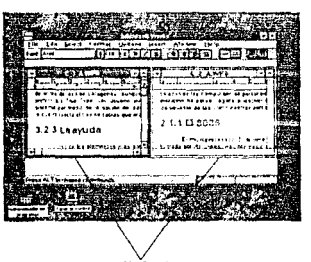


Figura 3-10.

La clase proceso de trabajo representa los efectos de los objetos a deplegarse en el área de trabajo de la ventana. La subclase de objetos gráficos engloba a elementos tales como iconos, cajas de mensaje, cajas de dialogos y formas de presentar el cursor del ratón. La subclase generada por el usuario, puede llegar a englobar más de una clase, pues es la parte donde el usuario desarrollará procesos totalmente ajenos a windows.

3.3.2. Mensajes en Windows

Cuando se corre un programa en Windows, los objetos que forman la ventana principal y los procesos de trabajo establecen una línea de comunicación hacia el USER. A esta línea se le llama canal de mensajes y es bidireccional, y es procesada por el DESPACHADOR.

El DESPACHADOR tiene como función enrutar los mensajes hacia los correspondientes periféricos, servicios al MS-DOS y zonas de memoria a los que el programa requiera. Así, al iniciarse la corrida de un programa, la parte de la ventana principal envía un mensaje hacia el GDI, para que le indique bajo que normas pretende dezplegarse en la pantalla (colores, tamaño, etc.). La misma ventana principal también comunica los mensajes generados por los elementos en ella contenidos, menú control, barra título, botones, menú de procesos,etc, figura 3-11.

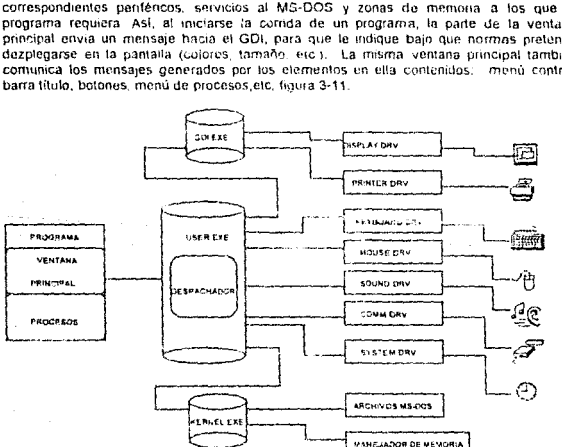


Figura 3-11.

Usando la misma línea de comunicación los procesos de trabajo diseñados por el programador también se comunican con el DESPACHADOR. El DESPACHADOR tiene una tremenda responsabilidad, pues al estar trabajando Windows en multiprocesamiento debe

enrutar correctamente los mensajes de las diferentes ventanas. Para lograr esto, Windows mantiene un estricto contato con el MS-DOS y este a su vez con las tablas descriptoras del modo protegido del microprocesador. Sin embargo, si puede ocurrir errores cuando dos o más programas requieren el uso de un periferico o área de memoria. Para solucionar este problema, Windows procede a eliminar a alguno de los procedimientos, Figura 3-13.

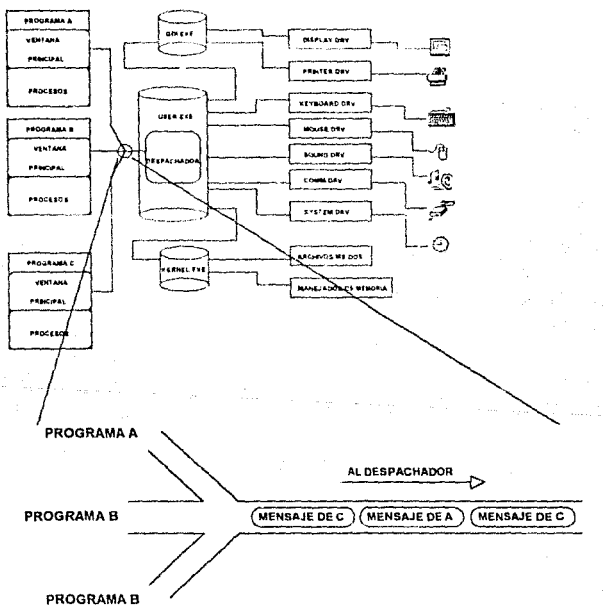


Figura 3-13.

3.3.3. Usando Tecnología de Objetos Ligados

Microsoft dotó a Windows de complejas formas de enlazar información entre aplicaciones. En el sentido más estricto, este tipo de enlaces no es más que el hecho de compartir un conjunto de datos, por dos o más aplicaciones, a esto se le llama objetos. Dentro de Windows se manejan tres estilos de compartir datos: objetos por liga, intercambio dinámico de datos (DDE), vinculación e incrustación de objetos (OLE -Object Linking and Embedding-).

El estilo de objetos por liga es una forma primaria de transportar datos de una aplicación a otra por medio del programa clipboard. El clipboard sirve como una memoria temporal cuando se realizan en algún programa instrucciones copia o corte por zonas. En clipboard se pueden almacenar tanto caracteres como imágenes. Cuando desde otra aplicación se invoca la instrucción pegar, es decir, colocar lo que hay en el clipboard, este descargará en la aplicación su contenido, Figura 3-14

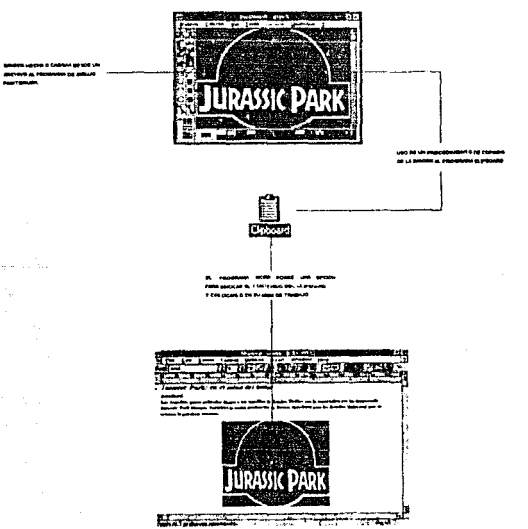


Figura 3-14.

Para el caso del intercambio dinámico de datos también se utiliza el estilo de los objetos por liga; sin embargo, la gran diferencia es que los objetos al ser cambiados desde la aplicación que la originó, pueden pasar dichos cambios a la nueva aplicación donde fueron transferidos, figura 3-15.

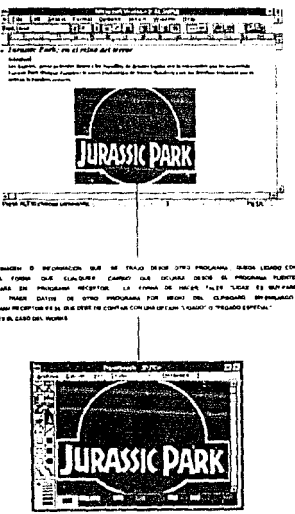


Figura 3-15.

Ahora bien, para el caso vinculación e incrustación de objetos, el estilo empieza donde acaban los DDE, pero ahora se puede llamar desde la aplicación receptora, en cualquier momento que se necesite, a la aplicación que generó el objeto y hacer los cambios necesarios, figura 3-16

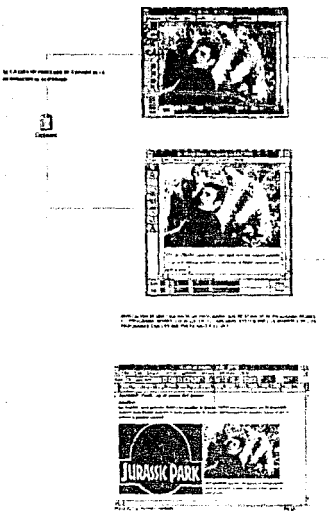


Figura 3-16.

El uso de DDE y OLE permite hacer que programas básicos como el Write (procesador de texto) y Paintbrush (editor de imágenes) se conviertan en avanzadas herramientas de edición, en programas especializados. OLE permite el intercambio y elección de nuevas partes del editor más avanzadas o que ocupen menos memoria. En este último punto, la memoria, es el único problema que presentan la tecnología de los objetos dinámicos; pues al cargar en el clipboard aplicaciones se comienza a saturar la memoria, lo cual puede limitar las intenciones de manejar DDE y OLE.

3.4. Programación

Se presenta a continuación los principios de la filosofía para programar en Windows, lo cual será el punto de partida de para entrar al capítulo 4.

3.4.1. La plataforma de Microsoft

Para construir programas que corran en Windows, Microsoft propuso la creación de tres archivos:

•Archivo fuente.
•Archivo de recursos.
•Archivo de definición.

El archivo fuente puede ser un archivo escrito, escencialmente, en C. Este archivo contiene los procesos para la ventana principal y los procesos del usuario. El archivo de recursos contiene instruciones especiales para permitir la aparición y manejo de elementos tales como iconos, cajas de mensaje, cajas de diálogo, menús de procesos y formas del cursor del ratón. El archivo de definición contiene un registro de información sobre las caracteristicas de memoria y nombre con que se llamará a la ventana principal.

El archivo fuente es procesado por un compilador correspondiente al lenguaje usado; el resultado es un archivo objeto, el cual es ligado junto con el archivo de definición. El resultado del ligador es un archivo .EXE, que a pesar de ser funcional para Windows podría presentar problemas de control al no tener los recursos. El archivo de recursos es procesado por el llamado compilador de recursos (RC); el resultado de esta compilación es un archivo con extensión .RES, el cual se compila nuevamente con el RC, pero ahora uniendo el archivo .EXE, obtenido con los archivos objeto y de definiciones. El resultado del RC sera un programa plenamente ejecutable en Windows. La figura 3-17 muestra lo anteriormente expuesto.

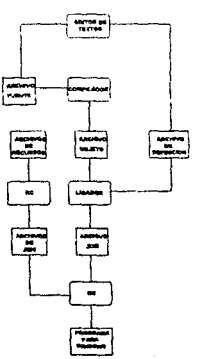


Figura 3-17.

Los compiladores de los archivos fuente y de recursos, ligador y otros programas de apoyo fueron desarrollados por Microsoft y comercializados en un paquete llamado Software Develoment Kit (SDK) desde la aparición del Windows 2.0. Microsoft eligió el lenguaje C como el lenguaje fuente para el desarrollo de programas en Windows por su versatilidad en el manejo de bibliotecas y por ser el más difundido en desarrollo de programación de sistemas, además, como una características adicional, los primeros compiladores desarrollados para el SDK permitían ligarse con programas objetos escritos en pascal o lenguaje ensamblador para ser más atractivos para los programadores. Esta catacteristica aún persite en el actual compilador C/C++ versión 7.0 de Microsoft, pero, como su nombre lo indica, a dado paso al uso de las ventajas de los objetos orientados permitiendo el uso opcional de estructuras de C++

Esta idea de "opcional" para los objetos orientados es verdadera, pues la biblioteca que se usa para la creación de ventanas, WINDOWS.H, es usada como una biblioteca normal de C, lo que ayuda al programador inexperto en objetos orientados a mantener un estilo de forma estruturada y de fácil entendimiento

En los manuales del SDK, el cual , también, se comercializa como parte del compilador C/C++ version 7.0, se asegura que puede usarse cualquier compilador de C para desarrollar los archivos fuentes. Sin embargo, tanto el ligador, como el compilador de recursos deben de seguir siendo los que se ofrecen en el SDK.

3.4.2. La Plataforma de Borland

La compañía Borland International, el competidor más severo en Estados Unidos de Microsoft, también desarrollo una plataforma de programación a raíz de la aparición de Windows 3.0, el nombre de esta plataforma es Borland C++. Literalmete, Borland conservó el mismo esquema de programación del SDK; pero, ofrece a los programadores varias herramientas como el uso de un medioambiente integrado (editor, compilador, ligador y compilador de recursos, todo en uno) y un depurador de programas en Windows, para usarse mientras estos corren en el medioambiente. Al igual que Microsoft, Borland uso los lenguajes C y C++ para diseñar los programas fuentes a los que adhirió una serie de bibliotecas de (en objetos y sin objetos) que lo convierten en una de las plataformas de programación más poderosas y baratas en el medio comercial

Una de estas bibliotecas es la conocida como OWL.H, que contiene un conjunto de clases que trabajan directamente con la biblioteca de WINDOWS.H (copia corregida y aumenta por Borland de la biblioteca que ofrece el SDK). Usando OWL.H el programador se ve forzado a usar la programación orientado a objetos y, en cierta forma, a seguir un nuevo estilo de programación (el que propone Borland).

Pero, sin usar la biblioteca OWL.H, se pueden obtener trabajos de igual calidad al de SDK-C/C++ y permitir la portabilidad de los archivos fuente, de recursos y de definiciones entre ambas plataformas.

3.4.3. Otros Lenguajes para Windows

Usando SDK-C/C++ y Borland C++, el programador debe hacer sus desarrollos en MS-DOS y después pasarse a Windows para probar los resultados. Al usar un lenguaje y,

mejor aún, una plataforma diseñada para operar dentro del ambiente Windows, el programador obtiene un gran número de elementos para mejorar su producto, tales como:

- Uso de la memoria y tipos de impresora
- Tener integrado editor, compilador, ligador y RC en una misma ventana.
- Poder trabajar con varios archivos fuentes al mismo tiempo
- Depuración en el medioambiente

Actualmente existen muchos lenguajes y compiladores diseñados para el ambiente Windows tales como Actor, Smalltalk V, Quick C para Windows, Visual Basic para Windows, Turbo C++ para Windows, Turbo Pascal para Windows y Realizer Unlimited .

Actor es un exótico lenguage de objetos orientados que explota al máximo esta metodología. Smalltalk V es una versión desarrollada para Windows del legendario Smalltalk del PARC, se distribuye en varias versiones según la complejidad del desarrollo a la que se desee llegar. Quick C para Windows es desarrollado por Microsoft, es una combinación del SDK y un compilador de C. Visual Basic para Windows también es desarrollado por Microsoft, es lenguaje BASIC, pero usando un enfoque de objetos orientados; es muy fácil de aprender y de usar. Turbo C++ y Turbo Pascal, ambos para Windows, es la respuesta de Borland hacia el Quick C y Visual Basic; ambos incluyen la biblioteca OWL, pero de igual forma que el BORLAND C++, permiten programar en un estilo estructurado. Realizer Unlimited, diseñado por la compañía Whitin Technologies, es un lenguaje semejante a BASIC, pero que combina aspectos estructurales semejantes a los de C, y con una sintáxis mucho más flexible que la de Visual Basic.

3.4.4. Creando y Editando Archivos de Recursos

Los archivos de recursos proporcionan al programador una serie de elementos muy necesarios para elevar la calidad de presentación y, muchas veces, de control. Un archivo de recursos puede ser escrito desde un simple editor de palabras para después ser compilado; por supuesto que para lograr tal tarea es deber del programador aprender un nuevo lenguaje y su sintáxis. Un ejemplo de un archivo de recursos aparece en el listado 3.2.

Listado 3.2.

```
/*-------------------------------------------------------
//
// TODOWIN.RC
//
//   Copyright (c) 1991 by Borland International
//   All Rights Reserved.
//
//-------------------------------------------------------*/

#include <windows.h>
#include "tododefs.h"

TodoMenu MENU
BEGIN
    POPUP "&File"
```

```
    BEGIN
        MENUITEM "&New",      IDM_NEW_LIST,
        MENUITEM "&Open...",  IDM_OPEN,
        MENUITEM "&Save",     IDM_SAVE,
        MENUITEM "Save &As...", IDM_SAVEAS,
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_QUIT
    END

    POPUP "&Edit"
    BEGIN
                MENUITEM "&Edit Entry...",  IDM_EDIT
                MENUITEM "&Insert Entry...", IDM_NEW_ENTRY
                MENUITEM "&Delete Entry...", IDM_DEL_ENTRY
    END

    POPUP "&Help"
    BEGIN
        MENUITEM "&About Todo List", IDM_ABOUT
    END

END

AboutBox DIALOG 22, 17, 144, 75
CAPTION "About Todo"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
        CTEXT "Borland International", -1, 0, 5, 144, 8
        CTEXT "Todo List", -1, 0, 14, 144, 8
        CTEXT "Version 1.0", -1, 0, 34, 144, 8
        CONTROL "OK", IDOK, "BUTTON", WS_GROUP, 56, 51, 32, 14
END

OpenFile DIALOG 10, 10, 148, 116
CAPTION "Select File"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
        CONTROL "File Name:", -1, "static", SS_LEFT | WS_CHILD, 2, 4, 76, 10
        CONTROL "", IDD_FNAME, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 2, 18, 144,
12
        CONTROL "Files in", -1, "static", SS_LEFT | WS_CHILD, 2, 40, 38, 10
        CONTROL "", IDD_FPATH, "static", SS_LEFT | WS_CHILD, 44, 40, 98, 12
        CONTROL "", IDD_FLIST, "listbox", LBS_NOTIFY | WS_BORDER | WS_VSCROLL | WS_CHILD, 2, 54,
70, 58
        DEFPUSHBUTTON "OK", IDOK, 88, 62, 50, 14
        PUSHBUTTON "Cancel", IDCANCEL, 88, 86, 50, 14
END

TodoEdit DIALOG 66, 35, 159, 143
CAPTION "To Do"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
```

```
BEGIN
    CONTROL "Date Entered:", -1, "static", SS_LEFT | WS_CHILD, 14, 8, 46, 8
        CONTROL "", IDE_DATEENT, "EDIT", ES_LEFT | WS_BORDER | WS_TABSTOP, 14, 20, 68, 12
        CONTROL "Date Due:", -1, "static", SS_LEFT | WS_CHILD, 15, 38, 44, 11
        CONTROL "", IDE_DATEDUE, "EDIT", ES_LEFT | WS_BORDER | WS_TABSTOP, 14, 51, 68, 12
    CONTROL "Priority:", -1, "BUTTON", BS_GROUPBOX | BS_LEFTTEXT, 101, 9, 45, 58
    CONTROL "Low", IDE_LOW, "BUTTON", BS_RADIOBUTTON | WS_GROUP | WS_TABSTOP, 106, 20, 28, 12
    RADIOBUTTON "Medium", IDE_MEDIUM, 106, 35, 37, 12
    RADIOBUTTON "High", IDE_HIGH, 106, 50, 28, 12
    CONTROL "What to do:", -1, "static", SS_LEFT | WS_CHILD, 13, 76, 40, 8
    CONTROL "", IDE_TEXT, "edit", ES_LEFT | ES_AUTOHSCROLL | WS_BORDER | WS_GROUP |
WS_TABSTOP, 13, 90, 134, 12
    DEFPUSHBUTTON "OK", IDOK, 29, 115, 43, 14, WS_TABSTOP
    PUSHBUTTON "Cancel", IDCANCEL, 89, 115, 42, 14, WS_TABSTOP
END
```

El formato se asemeja mucho al de un programa en Pascal, sin embargo, una serie de complicaciones pueden presentarse cuando el archivo pretende ser utilizado en un sistema complejo. Para resolver este problema, Microsoft introdujo los creadores y editores de recursos, es decir, una serie de programas herramienta por medio de los cuales el programador puede diseñar, desde Windows, el tipo de recurso que desea. Por lo tanto para la clase de recursos descritos anteriormente, como iconos, cajas de diálogo, cajas de mensaje y formas del cursor del ratón: existirán su correspondientes programas para crearlos y editarlos. Al ser salvado cada uno de los diferentes recursos, generaran archivos .RC; los cuales serán procesados por el compilador de recursos, de la misma forma que lo hace un compilador con los programas fuente. Para evitar manejar varios programas, los archivos .RC pueden fucionarse en uno solo.

El SDK-C/C++ y Borland proporcionan programas para crear y editar recursos por separado. Igualmente, algunas compañías de software independientes han lanzado al mercado una serie de programas que proporcionan al usuario la creación de varios tipos de recursos. Algunos ejemplos de estos paquetes son el Resource Toolkit de Whitewalter Group y el Resourse Worshop de Borland, figura 3-18.

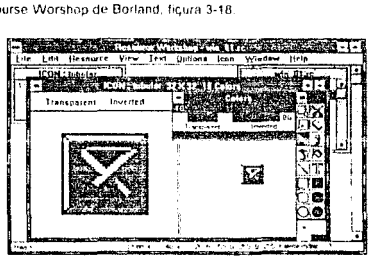


Figura 3-18.

Este tipo de programas han adherido la creación y edición de otros elementos ajenos al grupo de recursos como menús y tipos de letras para edición. Incluso dentro del SDK-C/C++ existe un editor de ventanas, por medio del cual el programador puede diseñar sus propias ventanas, lo que le permite salirse un poco del estándar ofrecido por Windows.

3.5. Formalizando una Metodología de Trabajo para Programar en Windows

El problema, desde un punto de vista personal, para programar computadoras es el hecho de elegir correctamente un lenguaje adecuado para hacerlo. Un sin número de lenguajes van y vienen entre los grupos de programadores, y cuando algún lenguaje es aceptado, se inicia una explosión de versiones "corregidas y aumentadas" que terminan generando algún otro lenguaje.

Para el presente trabajo se ha decido seguir los lineamientos porpuestos por Microsoft; esto es, usar un compilador de lenguaje C para realizar los archivos fuente. Las plataformas de Microsoft y Borland son compatibles en muchos aspectos, por lo que un programador puede construir programas que puedan ser movidos entre ambas plataformas sin preocupación de incompatibilidad. la figura 3-17 se modifica y queda comformada por la estructura que se muestra en la figura 3-19.

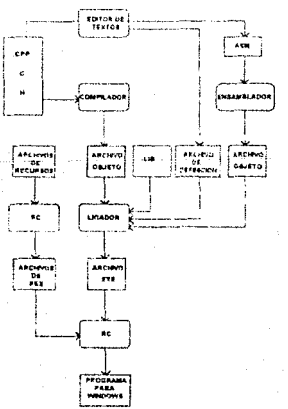


Figura 3-19.

Se debe observar que la estructura continua planeada para usar compiladores y utilerías de línea. El compilador a usar puede ser el Micorsoft C/C++ versión 7.0, MSC/C++, o Borland C++ , BCC, en sus versiones 2.0, 3.0 y 3.1. Se ha agregado en la figura 3-8 una parte para combinar módulos realizados en lenguaje ensamblador, para obtener archivos .OBJ generados de tales módulos se puede usar el Microsoft Assembler, MASM, o el Turbo Assembler de Borland, TASM. Para ligar los programas es mucho más conveniente usar el Turbo Link, TLINK , que el Microsoft LINK (MSLINK), pues el TLINK ofrece muchas más opciones y es más rápido que el de Microsoft. El compilador de recursos a usar se encuentra en función del ligador que se uso anteriormente.

Para crear los archivos de recursos, el programador puede elegir entre los Microsoft Develoment Tools, MSDT; o Resource Tools, RT, y Resource Workshop, RW, distribuidos por Borland. Los MSDT son programas que trabajan por separado para crear exclusivamente algun tipo de recurso. EL RT y el RW son programas semejantes a los del MSDT, pero integran en un solo módulo las diferentes herramientas para crear los recursos. MSDT, RT y RW generan por igual archivos .RC con la mismas caracteristicas, igualmente corren en ambiente Windows.

Los archivos .LIB son bibliotecas secundarias propias del lenguaje y otras desarrolladas por las compañías, tales como las anteriormente cometadas a la bibliotecas WINDOWS.H y OWL.H. En si, el estilo de trabajo propuesto se encamina a usar los compiladores de línea; no es muy conveniente usar medioambientes integrados, a menos que se encuentren en diseñados para Windows como es el caso de Borland C++ 3.1 y QuickC para Windows.

Así, como una forma de auxiliar al programador, algunas compañías han diseñado cierto tipo de editores en Windows capaces de realizar llamadas a los compiladores y ligadores de línea (aunque estos trabajen dentro de DOS), figura 3-20.

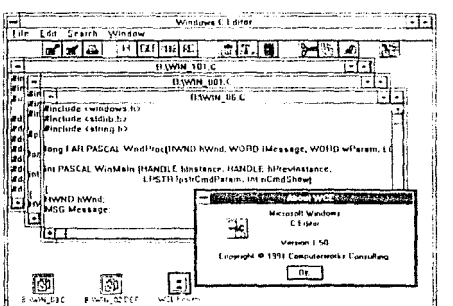


Figura 3-20.

3.5.1. Trabajando con Microsoft C/C++

Se podría afirmar que trabajar con Microsoft C/C++ no tiene ningún privilegio en especial. El compilador y el ligador en línea poseen una gran variedad de comandos para ser utilizados cuando se realiza el proceso de volver ejecutable un programa. En la figura 3-21 se muestra un formato genérico para poder trabajar con el compilador Microsoft C/C++.

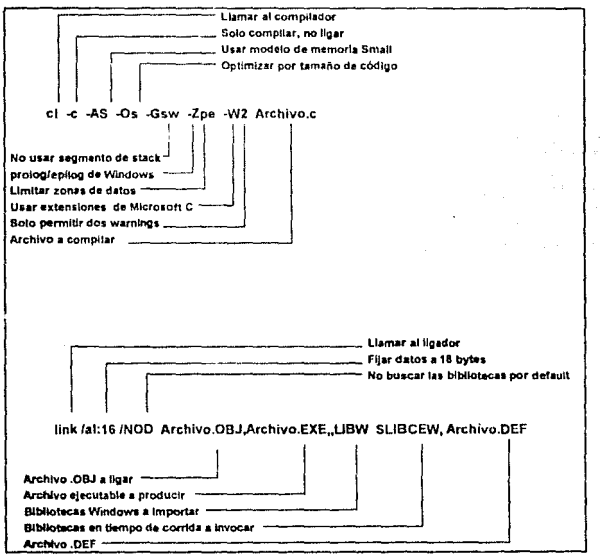


Figura 3-21:Formato genérico para C/C++.

Una parte especial a cuidar, no solamente para este compilador sino para cualquier otro es verificar la direcciones de los directorios donde se buscaran las diferentes bibliotecas y establecer un directorio especial para los programas a crear, tabla 3-2.

Tabla 3-2: Algunas opciones del compilador de C/C++ de Microsoft.

Opción	Descripción
-AC	Selecciona el modelo de memoria compacto. Con esta opción se usan las bibliotecas LIBW.LIB,CLIBCAW.LIB y CLIBCEW.LIB.
-AH	Selecciona el modelo de memoria huge. Con esta opción se usan las bibliotecas LIBW.LIB, HLIBCAW.LIB y HLIBCEW.LIB.
-AL	Selecciona el modelo de memoria large. Con esta opción se usan las bibliotecas LIBW.LIB, LLIBCAW.LIB y LLIBCEW.LIB.
-AM	Selecciona el modelo de memoria medium. Con esta opción se usan las bibliotecas LIBW.LIB, MLIBCAW.LIB y MLIBCEW.LIB.
-AS	Selecciona el modelo de memoria small. Con esta opción se usan las bibliotecas LIBW.LIB, SLIBCAW.LIB y SLIBCEW.LIB.
-c	Solo compilar. Compilador cl, posee en su estructura un ligador, pero no puede generar programas ejecutables para Windows.
-FPa	Indica al compilador generar código para el manejo de intrucciones de punto flotante. Para ello se llama a las bibliotecas CLIBFA.LIB, SLIBFA.LIB, MLIBFA.LIB o LLIBFA.LIB, dependiendo del modelo de memoria elegido.
-Fo	Indica al compilador la ruta a seguir para colocar el archivo .OBJ.
-Gs	Indica al compilador que no genere código que use el segmento de pila (stack).

3.5.2. Trabajando con Borland C++

Usar el paquete Borland C++ puede parecer mucho más agradable que el de Microsoft. Borland ofrece al programador la opción de trabajar con un compilador y ligador de linea o en un ambiente integrado. Actualmente la versión 3.1 de Borland C++ viene

dividida en programas de ambiente integrado para correr tanto en MS-DOS como en Windows. En la versión para MS-DOS, el ambiente integrado se encuentra en dos formas: BC, para modo real, y BCX, para modo Protegido. En cualquiera de las dos formas, el ambiente integrado se presenta con menús y ventanas en modo texto, clásicos del sofware de Borland.

Ahora bien, si optamos por usar el formato de compiladores y ligador en línea (lo cual es lo más recomendable dado el poco espacio que se ocupa en disco) se tendrá que usar un formato genérico como el que muestra en la figura 3-22

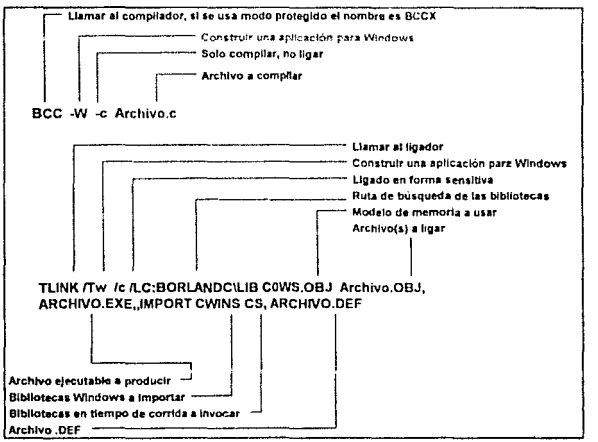


Figura 3-22:Formato genérico para Borland C++.

Sin embargo, las opciones del compilador y el ligador de línea de Borland son muy variadas la tabla 3-3 menciona algunas de las más requeridas por los progamadores.

Tabla 3-3: Algunas opciones del compilador de Borland C++.

Opción	Descripción
-1	Genera código para 80186
-1-	Genera código para 8088/8086
-2	Genera código para 80286 compatible

	para modo protegido.
-B	Compila y llama partes en ensamblador.
-C	Comentarios anidados.
-c	Solo compila, no hace el ligado automático.
-Ff	Crear variables lejanas (far) automáticamente.
-Ff=tamaño	Crear variables lejanas (far) automáticamente, pero fijando un tamaño.
-f	Emular punto flotante (por default).
-f-	No hacer emulación de punto flotante.
-f87	Generar código para 8087.
-f287	Generar código para 80287.
-G	Optimiza para la velocidad.
-G-	Optimiza la generación de código.
-gn	Detener compilación a los n warnings.
-mc	Usar modelo de memoria compact
-mh	Usar modelo de memoria huge
-ml	Usar modelo de memoria large
-mm	Usar modelo de memoria medium
-ms	Usar modelo de memoria small (por default).
W	Crear un .OBJ con todas las funciones exportables.
WD	Crear un .OBJ con todas las funciones para ser ligadas a un DLL.
WDE	Crear un .OBJ con todas las funciones para ser ligadas a un DLL y que sean exportables.

3.5.3. Usando Make

Make es una herramienta originaria de Unix que se usa para procesar secuencias de instrucciones de compiladores y ligadores en línea. Borland y Microsoft proporcionan dicha herramienta para ayudar a los programadores de Unix a establecerce en DOS. Lo que hace Make parece ser muy simple, recibe un programa, semejante a los .BAT de MS-DOS, y comienza a ejecutar las diferentes instrucciones, que nos son más que comados del sistema operativo y llamadas de programas.

Pero, a diferencia de un programa .BAT, al ejecutar Make un archivo diseñado para compilar y ligar un programa; Make hace una serie de detecciones específicas para determinar cuáles son las partes son los archivos o versiones más nuevas para cambiarlas por nuevas. De esta forma Make, se asegura de hacer solo las partes que han cambiado recientemente y que no poseen archivos de salida (OBJ y EXE)

Dentro de los archivos dirigidos a Make pueden usarse un grupo de directivas, y tablas, para crear una forma más eficiente sobre la manera de compilar y ligar un programa.

Tabla 3-4:Directivas de Make.

DIRECTIVA	FUNCION
.autodepend	Pasa a verificar un dependencia.
!elif	Condicional de ejecusión
!else	Condicional de ejecusión
!endif	Condicional de ejecusión
!error	Ha ucurrido un error detener la ejecusión de Make.
!if	Condicional de ejecución
.ignore	Obliga a Make a ignorar un valor generado.
!include	Especifica a un archivo a incluir en el proceso.
.noautodepend	No verificar una dependencia.
.noignore	No hacer una la ignoración (.ignore).
.nosilent	Pasar a pantalla la ejecusión de los comandos.
.noswap	No hacer intercambio de partes a memoria

.path.*ext*	**Dar a make una ruta para buscar un archivo con extensión especifica (.*ext*).**
.silent	**No pasar a pantalla la ejecusión de los comandos.**
.swap	Hacer intercambio de partes a memoria
!undef	**Quitar condicones epecificadas por una macro-**

4

Programación Básica

En este capítulo se plantearán las bases para realizar programas que trabajen en Windows usando el lenguaje C. Al conjunto de funciones que Microsoft agregó a C para crear programas bajo Windows se le conoce como API (Aplications Programming Interfacing); y son un total de 600 funciones. Por lo cual, tratar de explicar todas las funciones de la API escapa muy notablemente al objetivo del presente trabajo, por lo cual nos concretaremos a resumir las principales funciones y su alcances dentro de un programa.

4.1. Mensajes y Manejadores

Como se vió en el capítulo anterior, un programa en Windows se comunica con el despachador de USER usando mensajes. Los diferentes mensajes que controla un programa estan definidos por medio de constantes contenidas en el archivo WINDOWS.H. La tabla 4-1 muestra algunos mensajes usados en un programa típico, igualmente se presentan su valor correspondiente, en hexadecimal y su significado.

Tabla 4-1.

MENSAJE	VALOR EN HEXADECIMAL	SIGNIFICADO O USO
WM_NULL	0000H	Mensaje para indicar un nulo.
WM_DESTROY	0002H	Mensaje para destruir la ventana principal.
WM_SIZE	0005H	Mensaje para indicar cambio en tamaño de la ventana.

WM_ACTIVATE	0006H	Mensaje para incicar que esta activada una ventana hija.
WM_PAINT	000FH	Mensaje para indicar quo se debe efectuar una actividad de pintar (trazar gráficos).
WM_CLOSE	0010H	Mensaje para indicar que se pretende cerrar una ventana.
WM_KEYFIRST	0100H	Mensaje para indicar que se ha presionado una tecla por primera vez
WM_KEYDOWN	0100H	Mensaje para preguntar si el teclado esta activado.
WM_COMMAND	0111H	Elegir una opción de un menú.
WM_TIMER	0113H	Mensaje para generar ciclos
WM_HSCROLL	0114H	Mensaje de la barra horizontal.
WM_VSCROLL	0115H	Mensaje de la barra vertical.
WM_INITMENUPOPUP	0117H	Mensaje para preguntar si un menú esta deplegado.
WM_MOUSEMOVE	0200H	Mensaje para preguntar si el mouse se ha movido.
WM_LBUTTONDOWN	0201H	Mensaje para indicar que el botón izquierdo ha sido presionado.
WM_LBUTTONUP	0202H	Mensaje para indicar que el botón izquierdo ha dejado de ser presionado.

Hay alrededor de 200 mensajes diferentes dentro de WINDOWS H y son divido en varias categorías según su aplicaciones. La tabla 4-2 muestra los diferentes tipos y el prefijo con el que son nombrados los mensajes.

Tabla 4-2.

TIPO	PREFIJO
Mensajes dirigidos a un botón	BM_
Mensajes de un botón	BN_
Mensajes dirigidos a una caja combo	CB_

Mensajes de una caja combo	**CBN_**
Mensajes de una caja de diálogo	**DM_**
Mensajes de un controlador de edición	**EM_**
Mensajes para un controlador de edición	**EN_**
Mensajes de una lista de elementos	**LB_**
Mensajes para una lista de elementos	**LBN_**
Mensajes dirigidos a una ventana	**WM_**

Los manejadores son una especie de apuntadores que direccionan los objetos contenidos en un programa. Los manejadores son vistos como tipos de datos de 16 y 32 bits y se encuentran definidos dentro de WINDOWS.H. Los siete manejadores más comunes en un programa son HANDLE, HWND, HDC, LPSTR, MSG, LONG Y WORD. Otros manejadores se irán definiendo conforme se avance en este capítulo.

4.2. Estableciendo la Ventana Principal

Dentro de un programa para windows escrito en C, la parte "ventana principal" es referenciada como la función WinMain, siendo establecida de la forma

```
int PASCAL WinMain( HANDLE hInstancia, HANDLE hPreInstancia, LPSTR LpCmdLine, int NCmdShow)
```

Los parámetros hInstancia y hPreInstancia corresponden al estado de los objetos en el momento de correrse. Puesto que Windows permite el multiprocesamiento usa los valores hInstancia y hPreInstance para asignar prioridades de corrida y asignación. Al correr más de una vez un mismo programa, Windows sólo carga una vez el código, pero asigna diferentes partes de la memoria para los datos de las correspondientes corridas. A hInstance se le asiga un número de uso y un lugar para el sistema, mientras que hPreInstance es usada para indicar que existen otras versiones del programa corriendo. Cuando solo existe una versión del programa en ejecución hPreInstance recibe el valor de cero. LpCmdLine es un parámetro por medio del cual se reciben órdenes para alterar el estado de la ventana. NCmdShow indica si la ventana esta abierta o en estado de icono. la palabra reservada PASCAL es usada convencionalmente para permitir el paso de los parámetros a la función usando el formato del lenguaje Pascal.

La definición del cuerpo de WinMain se inicia con el establecimiento de tres variables locales esenciales para la creación de la ventana principal

```
{
MSG msg;
HWND hWnd;
WNDCLASS ClaseVentana;
```

La variable msg es del tipo MSG, que es una estructura compleja usada para contener los mensajes de programa. La estructura MSG es la siguiente

```
{
 HWND  Hwnd;
 WORD  Message;
 WORD  Wparam;
 LONG  Lparam;
 DWORD Time;
 POINT  Pt;
} MSG;
```

hWnd es el manejador asignado para la ventana principal. La variable ClaseVentana es del tipo WNDCLASS, que corresponde a una clase que define las características deseadas de la ventana. La clase WNDCLASS se estrutura de la siguiente forma:

```
{
WORD          style;
LONG          (FAR PASCAL *lpfnWndProc)();
int           cbClsExtra;
int           cbWndExtra;
HANDLE        hInstancia;
HINCON        hIncon;
HCURSOR       hCursor;
HBRUSH        hbrbackground;
LPSTR         LpszMenuName;
LPSTR         LpszClassName;
} WNDCLASS;
```

HICON, HCURSOR y HBRUSH son manejadores correspondientes a los objetos icono, cursor y fondo de la pantalla, respectivamente, que se manejan dentro de la ventana. Estos son apuntadores de 16 bits. la siguiente estructura dentro del cuerpo de WinMain ejemplifica como se establecen los lineamientos de creación de la ventana

```
if (!hPreInstancia)
    {

    ClaseVentana.style=CS_HREDRAW | CS_VREDRAW;
    ClaseVentana.lpfnWndProc=WndProc;
    ClaseVentana.cbClsExtra=0;
    ClaseVentana.cbWndExtra=0;
    ClaseVentana.hInstance=hInstancia;
    ClaseVentana.hIcon=LoadIcon(hInstancia,"tubular");
    ClaseVentana.hCursor=LoadCursor(NULL,IDC_ARROW);
    ClaseVentana.hbrBackground=GetStockObject(WHITE_BRUSH);
    ClaseVentana.lpszMenuName=NULL;
    ClaseVentana.lpszClassName="WIN_UNO";

    if (!RegisterClass(&ClaseVentana))
      exit(FALSE);
```

}

En esta parte se definen las características de presentación de la ventana principal. Lo primero que se hace es preguntar si WinMain tenía una presentación anterior (una instancia anterior). Si es la primera ves que se crea la ventana hPrevInstancia tendrá un valor 0 y entrará en el bloque del if. Dentro del bloque se procederá a la asignación de la variable del tipo WNDCLASS de todos aquellos valores necesarios para su estructura. Para confirmar que se ha completado la asignacion de las caracteristicas de la variable ClaseVentana, se usa otro bloque if y la función RegisterClass, la cual proporciona una respuesta booleana. La asignación errónea e incompleta puede causar que la ventana principal nuca sea creada y por lo tanto que el programa falle al ser ejecutado

Establecidas las caracteristicas, se procede a darle al manejador las instrucciones en las que deberá aparecer la ventana, tales como el nombre de la ventana, el mensaje en la barra de título, posición, tamaño y menús. El código es el siguiente:

```
hWnd=CreateWindow("WIN_UNO",
                  "Primera Ventana Fundamental",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstancia,
                  NULL);
```

La función CreateWindows contiene los siguientes parámetros y tipo de salida:

```
HWND CreateWindow(LPSTR lpClassName,
                  LPSTR lpWindowName,
                  DWORD dwStyle,
                  int X,
                  int Y,
                  int nWidth,
                  int nHeight,
                  HWND hWndParent,
                  HMENU hMenu,
                  HANDLE hInstance,
                  LPSTR lpParam)
```

La tabla 4-3 resume las cualidades de los diferentes parámetros de CreateWindow

Tabla 4-3.

PARAMETRO	FUNCION
lpClassName	Apuntador a una cadena de caracteres que contiene el nombre de clase que se le ha dado a la ventana principal.
lpWindowName	Apuntador a una cadena de caracteres

	que contiene el nombre de la ventana principal.
dwStyle	**Por medio de este argumento se determina el estilo de presentación de la ventana: marco simple y/o usando barras de control. Los valores estan predefinidos dentro de WINDOWS.H, tales como WS_CHILD y WS_HSCROLL. Para efectuar las diversas combinaciones de presentacion se usa el operador |.**
X	**Valor entero que fija la posición horizontal de la esquina de la ventana al ser dezplegada en la pantalla. Puede usarse el valor CW_USEDEFAULT para que Windows "decida" donde colocar la ventana.**
Y	**Valor entero que fija la posicion vertical de la esquina de la ventana al ser dezplegada en la pantalla. Puede usarse el valor CW_USEDEFAULT para que Windows "decida" donde colocar la ventana.**
nWidth	**Valor entero que fija el tamaño del ancho de la ventana al ser dezplegada en la pantalla. Puede usarse el valor CW_USEDEFAULT para que Windows "decida" el valor adecuado.**
nHeight	**Valor entero que fija el tamaño de la altura de la ventana al ser dezplegada en la pantalla. Puede usarse el valor CW_USEDEFAULT para que Windows "decida" el valor adecuado.**
hWndParent	**Apuntador que la ventana usa para direccionar la ventana madre; si se trata de la misma ventana madre el valor debe ser NULL.**
hMenu	**Apuntador del menú de ventana. Si no se maneja menú el valor debe ser NULL.**
hInstance	**Apuntador que recibe el valor de la ventana (instancia).**
lpParam	**Apuntador de caracteristicas generales**

de la ventana. Por lo general se maneja con el valor de NULL

Hasta este momento se ha definido la ventana principal, sin embargo, no se ha dezplegado en pantalla. Para asegurarse de la creación correcta de la ventana principal, el programador puede agregar la siguiente condición:

```
if (!hWnd) return(FALSE);
```

Así, se verifica que la variable hWnd este apuntado a la información adecuada, de no ser el caso, simplemente el programa se detiene. De estar todo bien, se procede a dezplegar la ventana principal por medio de la función:

```
ShowWindow(hWnd,nCmdShow);
```

A continuación se establece un mecanismo de comunicación entre el programa, o más bien, la ventana principal y el USER. Este mecanismo esta formado por la siguiente estructura y fuciones:

```
while(GetMessage (&Message,NULL,0,0))
  {
    TranslateMessage(&Message);
    DispatchMessage(&Message);
  }

return Message.wParam;
}
```

La estructura While crea un ciclo controlado por GetMessage, función encargada de detectar y enrutar los mensajes del exterior al interior del programa; la variable participante de este lazo es Message. La sintáxis paramétrica de GetMessage es la siguiente:

```
BOOL GetMessage(LPMSG lpMsg, HWND hWnd, WORD MIN, WORD MAX)
```

El tipo LPMSG corresponde a un apuntador encargado de vaciar toda la información necesaria del mensaje; en este caso a la variable Message. El siguiente parámetro corresponde al apuntador de la ventana a la que se enviará el mensaje; puesto que sólo existe una ventana de control, el valor que recibe es el de NULL. MIN y MAX corresponden a parámetros para filtrar cierto tipo de mensajes; recordando que los mensajes son en realidad valores binarios, MIN correspondería a la parte baja de los mensajes y MAX a la alta. Puesto que se desea pasar todos los tipos de mensajes a MIN y MAX se le da el valor de cero. El valor de regreso de la función GetMessage cuando se detecta el mensaje WM_QUIT es cero, lo que ocasiona la salida inmediata del ciclo, para cualquier otro mensaje el valor es diferente de cero.

Dentro del ciclo, la función TranslateMessage se encarga de hacer una conversión del mensaje "vía teclado" a un mensaje en forma de cadena de caracteres, y

DispatchMessage lo envía hacia los procesos de aplicación. La figura **4-1** resume lo expuesto en esta sección.

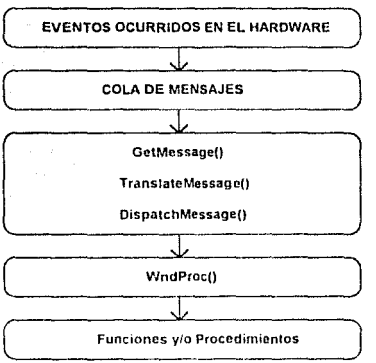


Figura 4-1.

4.3. Estableciendo los Procesos de Aplicación

Los procesos de aplicación parten de una función base, WndProc, cuyo trabajo es el de enviar los diferentes mensajes que recibe de la ventana principal a los procesos o funciones necesarios. La estructura de esta función es la siguiente:

```
long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
switch(iMessage)
    {
    case WM_DESTROY:
            PostQuitMessage(0);
            break;
    default:
            return DefWindowProc(hWnd,iMessage,wParam,lParam);
    }
        return(0L);
}
```

Por medio de la variable **iMessage** y de **switch-case**, WndProc enruta los diferentes casos correspondientes al mensaje recibido. Cada área de caso (case) puede describir un proceso, o bien llamar a un proceso externo.

4.4. Construyendo el Archivo de Definición

El archivo de definición es una especie de tarjeta de presentación del programa hacia el sistema de hardware, pues por medio de este archivo se le asignan al programa ciertas características de manejo sobre la memoria y la información que ahí será colocada (código y variables). De igual forma, el archivo de definición indica cuando un programa es un ejecutable normal ó una DLL. Un archivo típico de definición se presenta a continuación:

```
NAME            WIN_UNO
DESCRIPTION     ' Ventana Fundamental'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
```

La columna situada al lado izquierdo representa los condicionamientos de la presentación del programa, a continuación se explica su significado.

NAME define el módulo donde se ubica WinMain y WndProc.

DESCRIPTION define un letrero de identificación.

EXETYPE es una indicación para el ligador, establece el tipo de programa a crear (**WINDOWS**)

STUB (no mostrado en el ejemplo) es una indicación para que el programa, una vez siendo ejecutable, al ser invocado desde DOS pueda a si mismo llamar a Windows y ejecutarse automáticamente. Para esto Microsoft usa el programa **WINSTUB.EXE.**

CODE establece la forma en que se introducirá y manejará la parte del código en la memoria (segmento de código). Los posibles valores para este caso son **PRELOAD, FIXED, MOVEABLE y DISCARABLE.**

DATA establece la forma en que se manejarán los datos en su segemento. Los posibles valores pueden ser **PRELOAD, MOVEABLE y MULTIPLE**

HEAPSIZE establece el tamaño del heap de las variables de programa

STACKSIZE establece el tamaño de la pila (stack) del programa

EXPORTS (no mostrado en el ejemplo) establece si el programa se ligara hacia otros programas individuales

4.5. Contruyendo el Archivo de Recursos

Como se menciono en el cápitulo anterior, para construir el archivo de recursos es extremadamente recomendable usar algún programa diseñado para tales tareas. En el presente trabajo nos concentraremos en el uso del Whitewater Resourse Toolkit (WRT)

versión 3.01; aunque es conveniente señalar que muchas características de trabajo del WRT se encuentran en otros programas de tal índole.

4.5.1. WRT

El WRT esta incluido como parte del Borland C++ 2.0 y es ejecutable desde Windows, donde se presenta por medio del icono que semeja a una caja de herramientas, figura 4-2 (a). Activado, el WRT se presenta en una ventana cuyo tamaño no puede ser cambiado figura 4-3 (b).

Figura 4-2.

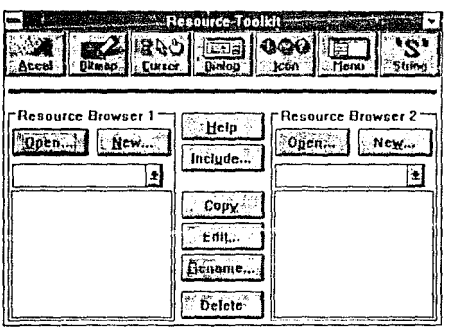


Figura 4-3.

En la parte superior de la ventana, se ubican siete botones que describen la creación de algún recurso en especial. Ocupando la principal área de la ventana encontramos dos zonas separadas por una serie de seis botones colocados en forma vertical. Cada área tiene su nombre en la parte superior, Resourse Browser 1 y Resourse Browser 2, y poseen una pareja de botones, Open y New. La razón por la que existen las áreas anteriores, y no una, es para permitir al usuario cargar un archivo de recursos, Resourse Browser 1, y copiarlo a otro, Resourse Browser 2, para el momento en que se le hagan las modificaciones adecuadas. En la práctica es muy común que un solo archivo de recursos derive otros agregándosele nuevas cararacteristicas.

En WRT se iniciará un nuevo archivo partiendo del área de Resource Browser 1, para ello se puede elegir el botón **Open**, para cargar un archivo anteriormente construido, o **New**, para crearlos.En ambos casos se abrirá una ventana similar a la de la figura 4-4, cuya variación principal será su título. Cuando se carga un archivo (File Open) el programador debe proceder a elegir, por medio del cursor algunos de los archivos o con Alt-F, algunos de los archivos situados en la caja con el título de **Files**. Se puede "navegar" hacia otros directorios por medio de la caja **Directories**, o bien, escribir el nombre del archivo deseado en la caja de diálogo **Filename**; dado el nombre, se procederá a buscar el archivo en el directorio concurrente. La razón por la que se puede recurrir a este método es debido a que el WRT restringe la entrada a los archivos que no puede manejar, por ello en el la caja titulada **File Type** las diferentes extensiones que maneja el programa, y solamente las que este remarcadas podran ser cargadas.

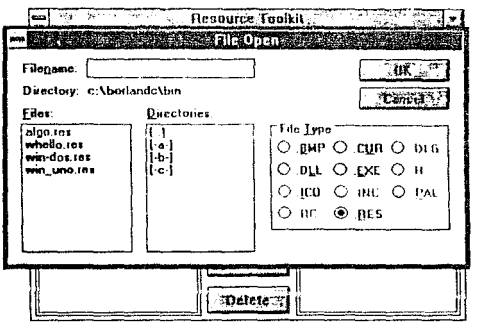


Figura 4-4.

La parte de File Type debe ser observada con mucho cuidado, pues WRT nos permite crear tanto archivos .RC, como archivos .RES. Igualmente nos permite separar los diversos recursos contenidos en archivo .EXE como .DLL (biblioteca de liga dinámica).

El funcionamiento del caso **New** es bastante similar al anterior, con la tornda diferencia de que al escribir un nombre en la caja de diálogo, se intepretará como un archivo a crear. Al cargar o crear un archivo .RES, ocurrirá un cambio al regresar a Resource Browser 1. Los botones **Open** y **New** habrán sido sustituidos por uno llamado **Close** y en la parte superior se mostrará el nombre del archivo con el que se piensa trabajar y su ruta de acceso. En las cajas situadas abajo de **Close**, cuando se trata de un archivo creado con anterioridad, deberán aparecer los diferentes elementos que forman ese archivo de recursos. En la figura 4-5 , se ejemplifica con el archivo de recursos win_uno res, el cual contiene la descripción de un solo recurso, **Icon**, es decir, un ícono. En la siguiente caja aparece la palabra **TUBULAR**, la cual se refiere al archivo donde se encuentra almacenado el icono de nombre TUBULAR.

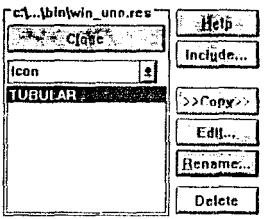


Figura 4-5.

Si hubiese otros elementos en el archivo, usando el ratón se accesarían facilmente. Accesado un elemento, es decir, que estuviese remarcado; se puede usar el botón de editar (**Edit...**) para entrar en alguno de los módulos de recursos. Tales módulos se encuentran en la parte superior de la ventana del WRT como botones del medioambiente, figura 4-6

Figura 4-6.

Por medio de los módulos de recursos se confecciona un archivo .RES, si se esta creando un nuevo archivo, los módulos son activados como simples botones. Los módulos se describen a continuación

Accel por medio de este módulo se puede crear una interface de control de procesos del programa. La interface se crea por medio de secuencias selecionadas de teclas.

Bitmap permite crear mapas de bits.

Cursor con este módulo el usuario puede crear sus propios cursores de ratón para ser usados en sus programas.

Dialog es un interesante módulo que permite la creación de cajas de diálogos. La mejor parte de todo, es que el programador genera su caja de diálogo directamente, eligiendo tamaño y tipo de botones de control.

Icon permite crear iconos.

Menu es un módulo para generar la estructura de los diferentes menús a usar en su programa.

String por medio de este módulo se pueden generar cadenas de caracteres especiales para ser usadas en mensajes de error del programa (un ejemplo de esto, puede ser cuando ocurre una división entre cero dentro del programa).

4.5.2. Creando un Icono

Para la creación e identificación de los futuros programas a diseñarse en el presente capítulo se requerirá hacer un archivo de recursos que contenga la descripción de un ícono. Usando WRT se crea el archivo WIN_UNO, mecionado para el caso de la figura 4-7. Usando la sección Icon, se entra a una ventana de diseño exclusiva de íconos. Semejante a un programa de dibujo, Icon nos presenta una paleta de colores, una sección de dibujo del ícono y una zona del tamaño real del ícono. Además, se cuenta con dos menús, uno formado por letreros en presentación horizontal, y otro descrito por iconos que presentan diferentes opciones para dibujar el ícono.

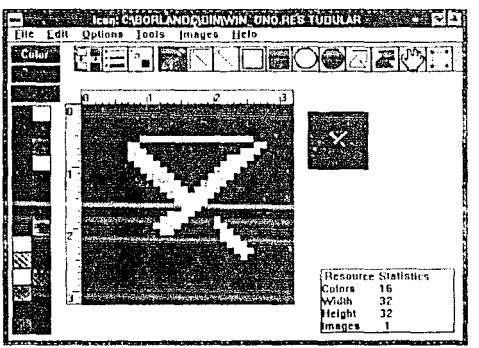


Figura 4-7.

El menú iconográfico permite crear un ícono fácilmente por medio del ratón. La figura 4-8 muestra los diferentes íconos del menú y sus funciones.

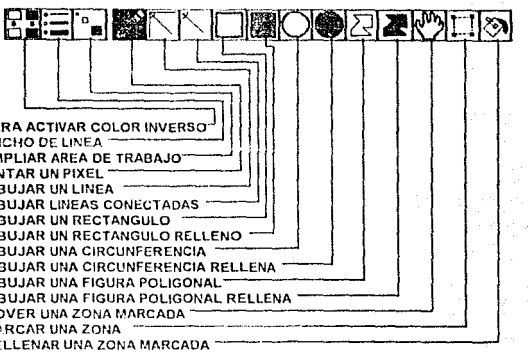


Figura 4-8.

4.6. La Notación Húngara

En los fragmentos de listados anteriores se puede observar una curiosa forma de escribir los nombres de variables, constantes y tipos dentro de la estructura de C; a esta forma se le conoce como notación húngara. Desarrollada por Charles Simonyi a principios de 1970, la notación húngara es un estilo de escribir variables con elementos estandarizados dentro de un programa sin importar el tipo de lenguaje que se use. Diversos equipos de programación dentro de Xerox, Apple, 3Com y, por supuesto, Microsoft, la han incorporado en sus estilos internos de trabajo. La influencia de la notación húngara se encuentra presente en la biblioteca Windows.h, y por lo tanto, el no entender este estilo puede acarrear dificultades en la compresión de los futuros listados de programas usados en el presente trabajo y en los de la referencia bibliográfica

4.6.1. Las Reglas Húngaras

Las reglas para seguir la notación húngara son simples, pero complejas para los programadores novatos. La primera regla establece que toda variable debe escribirse con una o más letras minúsculas, que representen las correspondientes letras iniciales del tipo al que pertenece la variable. Así, variables tales como **lpClassName**, **dwStyle**, **hWndParent**, **hMenu** y **hInstancia**, derivan el inicio de su nombre de los tipos de datos **LPSTR**, **DWORD**, **HWND**, **HMENU** y **HANDLE**, respectivamente. La siguiente parte de la escritura del nombre de una variable, es la de escribir nombres claves con referencia al uso u origen de la variable. Se pueden realizar contracciones cuando son nombres referidos a dos o más palabras, como en el caso de **hWndParent**, que proviene de **Windows Parent**. Obsevese que para este ejemplo, para denotar el uso de palabras se escribieron letras

mayúsculas. El número máximo de caracteres usados en el nombre de la variable debe ser fijado por el programador.

La segunda regla establece que cuando se crean en algún lenguaje nuevos tipos de datos, sus nombres sean escritos en mayúsculas, como ejemplo, **LPSTR, DWORD, HWND, HMENU y HANDLE**. La tercera regla se refiere a la escritura de procedimientos y funciones, la cual no dista mucho de la forma en que se escriben las variables, pues en esencia los nombres son creados por contracciones de palabras claves referidas con mayúsculas en su inicio, pero siguen una estructura de escritura:

(Tipo de salida del procedimiento) (Accion a realizar) (Parámetros de efectos)

Por supuesto, que si se sigue tal estructura con mucha exigencia puede escribirse un nombre demasiado largo y complejo; por ello se recomienda usar formatos simples y fáciles de recordar. Como ejemplo tenemos.

WndProc()
PostQuitMessage()
DefWindowProc()

Pese a lo que se pudiera pensar, la notación húngara no establece el uso del caracter _ ; la mayoría de los programadores seguidores de la notación, sólo lo este caracter usan para referenciar constantes. Algunos grupos de programadores tienden a desechar el empleo de letras claves para establecer variables del tipo entero o real; otros tienden a asignar nombres comunes en el sentido matemático, así las variables que se refieren a coordenadas espaciales se le asignan los nombres de **X, Y o Z**. A través del tiempo diferentes grupos de programadores de la notación húngara han establecido ciertos prefijos comunes para usarse en la construcción de nombres de variables; algunos ejemplos generales se muestran en la tabla 4-4.

Tabla 4-4

p	Para definir apuntadores.
h	Para definir manejadores (apuntadores especiales).
mp o map	Para definir arreglos o tablas de indexación de datos.
i	Variables indexadas.
c	Contadores.
d	Variable de diferencia.
f	Variable de tipo booleana.
ch	Caracter.
sz	Cadena variable con valor nulo en su terminal.

4.6.2. Cuantificadores

Los cuantificadores son aquellas variables que se usan para llevar la cuenta o control de elementos dentro de un programa. Entre los usuarios de la notación húngara se han establecido ciertos tipos de nombres comunes, basados en el idioma inglés, para usarse en cualquier tipo de programa sin importar el lenguaje, tabla 4-5

Tabla 4-5

NOMBRE A LA VARIABLE	SIGNIFICADO
Temp (o T)	**variable de tipo temporal dentro de un proceso.**
Sav	**variable para savalr un valor.**
Prev	**variable para guardar un valor que sea necesario después.**
Cur	**variable usada como indice de acceso en arreglos.**
Next	**siguiente elemento en un arreglo.**
Dest	**variable destino para una operación.**
Src	**variable fuente de una operación.**
Nil	**variable para indicar vacio en un arreglo**
Min	**variable para indicar un valor mínimo en un proceso o arreglo.**
Max	**variable para indicar un valor máximo en un proceso o arreglo.**

4.7. Formalizando una esquema de trabajo

Por lo anterior podemos derivar un esquema básico para comenzar a programar en Windows usando lenguaje C, listado 4-1.

Listado 4-1.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#pragma argsused

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstancia, HANDLE hPrevInstancia,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
```

```
HWND hWnd;
MSG Mensaje;

if (!hPrevInstancia)
        {
        WNDCLASS WndClase;
        wClaseVentana.cbClsExtra=0;
        wClaseVentana.cbWndExtra=0;
        wClaseVentana.hbrBackground=GetStockObject(WHITE_BRUSH);
        wClaseVentana.hInstance=hInstancia;
        wClaseVentana.hCursor=LoadCursor(NULL,IDC_ARROW);
        wClaseVentana.hIcon=LoadIcon(hInstancia,"tubular");
        wClaseVentana.lpfnWndProc=WndProc;
        wClaseVentana.lpszClassName="WIN_UNO";
        wClaseVentana.lpszMenuName=NULL;
        wClaseVentana.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&wClaseVentana))
          exit(FALSE);
        }
hWnd=CreateWindow("WIN_01",
                  "Primera Ventana Fundamental",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstancia,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Mensaje,NULL,0,0))
{
TranslateMessage(&Mensaje);
DispatchMessage(&Mensaje);
}
return Mensaje.wParam;}

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam)
{
switch(iMensaje)
        {
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return    DefWindowProc(hWnd,iMensaje,wParam,lParam);
        }
return(0L);}
```

Genera la ventana presentada en la figura 4-9.

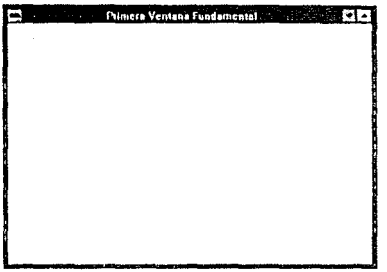


Figura 4-9.

Al presionar el menú control, este deberá desplegarse con las diferentes opciones típicas de una ventana normal, figura 4-10.

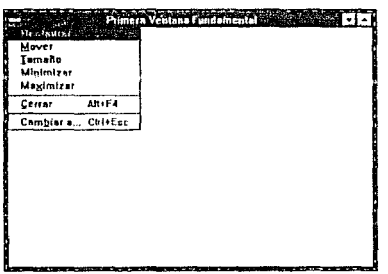


Figura 4-10.

El icono asignado a esta ventana es TUBULAR.ICO, figura 4-11, y fue creado por medio del WRT.

Figura 4-11.

El archivo RC, WIN_01.RC, contiene la información

tubular ICON tubular.ico

Si no se desea asignar un ícono al programa, se puede alterar el valor en la sección donde se le asignan las características a la variable WndClass.

```
wClaseVentana.hIcon=LoadIcon(hInstancia,"tubular");
```

En lugar del nombre "tubular", se le asigna el valor "End"; el resultado será un ícono "transparente". O bien, También puede usarse un ícono propio del sistema, para ello se escribe:

```
wClaseVentana.hIcon=LoadIcon(hInstancia,IDI_APPLICATION);
```

Lo cual significara que se use un ícono propio del sistema, como puede ser el de la figura 4-12

Figura 4-12

La siguiente parte del programa es el archivo .DEF que se muestra a continuación en listado 4-2.

Listado 4-2.

```
NAME            WIN_01
DESCRIPTION 'Ventana Fundamental'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
```

El nombre que se escriba en registro NAME deberá concidir con el que se asigne en la función CreateWindow, ver siguiente listado. De no concidir, aunque la compilación y el ligado sea exitoso, no ocurrira nada al intentar correr el programa.

4.8. Usando el GDI

Antes de proceder a usar el área de trabajo de una ventana es necesario mencionar las características que ofrece el manejador de interface gráfica (GDI) tanto para escribir textos como gráficas.

4.8.1. Las tarjetas de video

A diferencia de trabajar con cualquier tarjeta de video; CGA/RGB (640x350), HERCULES (720x348) o VGA (800x350, 400, 480), Windows proporciona cierto sentido estándar en el despliegue de los pixeles. Windows 3.0 podía trabajar con tarjetas de baja resolución como CGA o HERCULES; donde esta última, a pesar de tener una buena resolución solamente se presentaba en forma monocromática. La tarjeta recomendada para usar Windows 3.0, para una buena presentación, era la VGA.

La actual versión 3.1 a relegado al VGA como la resolución básica para un despliegue del medioambiente Windows, mientras tarjetas como SVGA (1024x768) o creadas especialmente para el medio ambiente (S-3, 1280x1024), permiten el uso de señales de video interactivas con Windows.

4.8.2. Los modos de mapeo del GDI

Los modos de mapeo son aquellas formas en que se despliegan los pixeles en el área de trabajo. El GDI presenta ocho modos de mapeo para dezplegar gráficos (no olvidar que los caracteres de alguna forma también son considerados como gráficos), tabla 4-6

Tabla 4-6.

MM_TEXT	**MODO TEXTO TIPICO**
MM_ ISOTROPIC	**MODO GRAFICO ISOTROPICO**
MM_ANISOTROPIC	**MODO GRAFICO ANISOTROPICO**
MM_HIENGLISH	**MODO DE ALTA RESOLUSION EN SISTEMA INGLES.**
MM_LOENGLISH	**MODO DE BAJA RESOLUCION EN SISTEMA INGLES**
MM_HIMETRIC	**MODO DE ALTA RESOLUCION EN SISTEMA INTERNACIONAL.**
MM_LOMETRIC	**MODO DE BAJA RESOLUCION EN SISTEMA INTERNACIONAL**
MM_TWIPS	**MODO DE ALTA RESOLUCION EN SITEMA DE PIXELES**

El modo MM_TEXT podemos considerarlo como el "clásico", pues el despliegue de los gráficos se basa en los pixeles propios de la tarjeta graficadora. A su vez, los modos MM_ISOTROPIC Y MM_ANISOTROPIC, se basan directamente en MM_TEXT; pero con variaciones en referencia al tamaño de la ventana. Para explicar las propiedades de estos modos usaremos el programa GDI Test desarrollado por FutureSoft, figura 4-13.

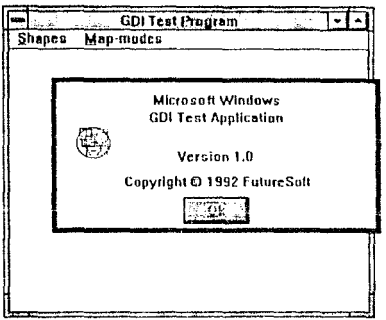


Figura 4-13.

El GDI Test es un programa que trabaja con los modos MM_TEXT, MM_ISOTROPIC y MM_ANISOTROPIC. Para ello permite desplegar cuatro dibujos: un cuadrado, una elipse, un rectángulo con esquinas redondeadas y un arco, figura 4-14.

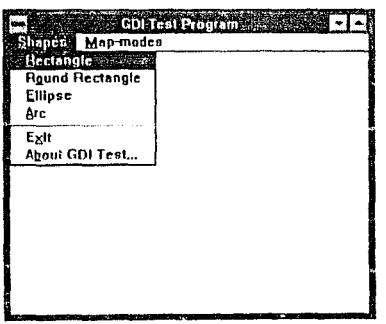


Figura 4-14.

Ahora bien, supongamos que estamos en el modo de mapeo MM_TEXT, figura 4-15, y desplegamos las cuatro formas del menú **Shapes**.

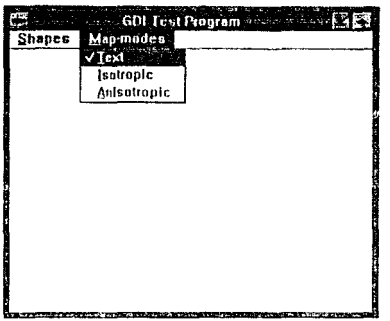


Figura 4-15.

Si variamos el tamaño de la ventana, las figuras permanecerán iguales y solamente se mostrarán aquellas que queden dentro de la nueva área de trabajo, figura. 4-16

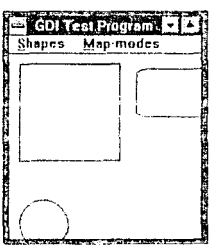


Figura 4-16.

Ahora bien, seleccionemos la opción MM_ISOTROPIC , conservando los cuatro dibujos anteriores. Existirá un pequeño ajuste en el tamaño de los objetos, pero no desaparecerán, figura 4-17.

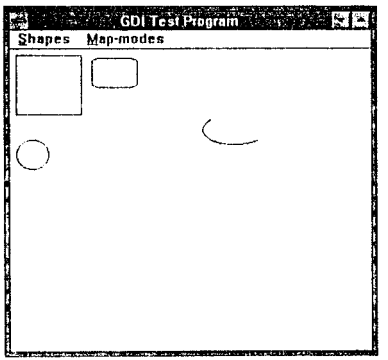


Figura 4-17.

Ahora bien, si se redujerá nuevamente la ventana, ocurrirá que los dibujos se escalan sin deformarse a las nuevas caracteristicas de tamaño, figura 4-18

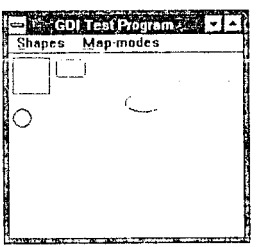


Figura 4-18.

Entoces, en MM_ISOTROPIC, se escalan automáticamente los dibujos en función del tamaño de la ventana sin que se deformen o se pierda su simetría. Pasemos ahora al modo de mapeo MM_ANISOTROPIC, figura 4-19.

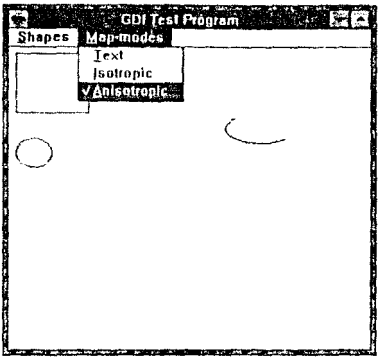


Figura 4-19.

En este modo no se respeta la forma de los dibujos al variar el tamaño de la ventana, es decir, se pierde el aspecto del radio de dibujo, figuras 4-20 y 4-21.

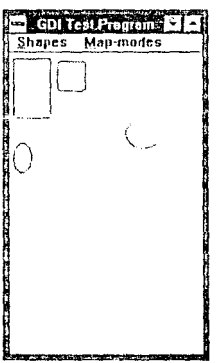


Figura 4-20.

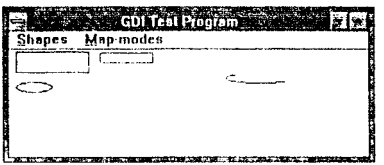


Figura 4-21.

Tanto MM_ISOTROPIC y MM_ANISOTROPIC construyen su dibujos en forma de pixel, como MM_TEXT, el resto de los modos de mapeo crean pixeles en función de aspectos especificos especiales de la tarjeta de video y el monitor. Asi MM_HIENGLISH debe de crear pixeles de 0.001 pulgadas; MM_LOENGLISH de 0.01 pulgadas; MM_HIMETRIC de 0.01 milimetros; MM_LOMETRIC de 0.1 milimetros y MM_TWIPS 1/1440 de pulgada. Por lo general, el modo de mapeo más usado es MM_TEXT, seguido de MM_ISOTROPIC y al final MM_ANISOTROPIC. El resto de los modos no es conveniente emplearlos si no se tiene el hardware adecuado.

4.8.3. Primer Dibujo en GDI

Para indicar en un programa que se desea proceder a dibujar sobre una ventana se usa el mensaje **WM_PAINT**, el cual es colocado dentro de la estructura switch-case perteneciente a la funcion de procesos de aplicación. Sin embargo nos basta con establecer la linea del mensaje. El área de trabajo, es en si una área de memoria y también debe ser direccionada por manejador.

El manejador es del tipo HDC, y para establecer una liga directa entre el área de trabajo y lo que se pretende "pintar" se usa una variable del tipo **PAINTSTRUCT**; la estructura de tal tipo es la siguiente:

```
typedef struct tagPAINTSTRUCT {
        HDC hdc;
        BOOL fErase;
        RECT rcPaint;
        BOOL fRestore;
        BOOL fIncUpdate;
        BYTE rgbReserve[16];
} PAINTSTRUCT;
```

La parte hdc se encarga de apuntar a la zona especifica de memoria de la pantalla para activar un pixel , fErase es usado para indicar si un pixel será activado o no, rcPaint es una estructura interna que contiene los valores de las esquina superior-izquierda e inferior-derecha del área de trabajo; fRestore, fIncUpdate y rgbReserve[16] son usados por el GDI para efectuar el encendido. Dentro de la terminología de Microsoft se lo llama Dispositivo de Contexto (Divice Context) a la zona de trabajo donde se realizan todas las operaciones de gráficas.

A continuación se muestra una sección de la función de procesos preparada para el caso WM_PAINT:

```
long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
PAINTSTRUCT PintaEstr;

switch(iMessage)
        {
        case WM_PAINT:
                        hDC=BeginPaint(hWnd,&PintaEstr);
                        . . .
                        . . .
                        . . .
                        EndPaint(hWnd,&PtStr);
                        return 0;
```

Se observa en primer término, la definición de las variables **hDC** y **PintaEstr**, para establecer contacto con el dispositivo de contexto (DC); el uso de las funciones **BeginPaint** y **EndPaint** indica el principio y el fin del bloque destinado a intrucciones que se comuniquen hacia el DC.

La función **SetMapMode** establece el modo de mapeo con el que se desplegaran los dibujos en la ventana; se puede omitir esta instrucción dado que por default el modo de mapeo es **MM_TEXT**. Es muy importante el papel que juega la variable hDC, pues toda función de dibujo es el unico medio para comunicarse con el DC. Después que se a pasado **BeginPaint**, esta función le direcciona a **hDC** las caracteristicas del DC tales como el modo de mapeo (mencionado anteriormente), coordenadas y colores de despliegue, la tabla muestra las diferentes caracteristicas del DC al ser direccionadas por primera vez por el hDC.

Tabla 4-7.

Color de Fondo
Modo de Fondo
Color de Relleno
Posición del origen
Tipo de letra
Color a usar (pen)
Color del texto
Modo de mapeo
Alineación del Texto
Espaceo entre caractores
Paleta de Colores

Entre las cosas que se deben de tomar en cuenta para desplegar un dibujo por computadora esta el como esta distribuido el sistema de coordenadas. Windows conserva el sentido clásico de iniciar desde el ángulo superior izquierdo, figura 4-22; sin embargo, es posible cambiar esta situación para comodidad del programador.

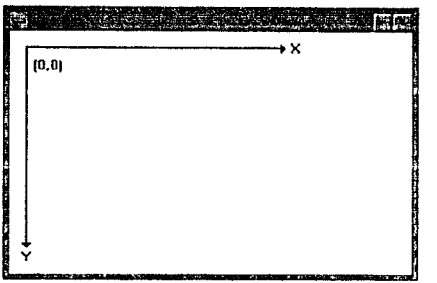


Figura 4-22.

Para ello es necesario conocer la situación en la que se encuentra la ventana. Para esto se introduce una nueva estructura dentro del sistema: **RECT**. Esta estructura es bastante sencilla en su definición como se muestra a continuación:

```
type struct tagRECT {
        int left;
        int top;
        int right;
        int bottom;
} RECT;
```

Por medio de RECT se crean variables para el trazo y control de rectángulos. Ahora bien, analizemos el siguiente trozo de programa que podría estar contenido entre **BeginPaint** y **EndPaint** del listado anterior.

```
RECT rect;

GetClientRect(hDC, &rect);
SetViewportOrg(hDC,0,-rec.bottom);
```

Por medio **GetClientRect**, se le entregan las coordenadas de las esquinas superior-derecha e izquierda-inferior a la variable **rect**; la función **SetViewportOrg** se encarga de ajustar un nuevo origen al DC; al seguir estas intrucciones obtendríamos un efecto como el que se muestra en la figura 4-23.

Figura 4-23.

Un efecto más fructífero de usar las instrucciones anteriores sería el siguiente

```
RECT rect;

GetClientRect(hDC, &rect);
SetViewportOrg(hDC,rect.right/2,rect.bottom/2);
```

Donde obtenemos una situación más apegada a los efectos matemáticos, figura 4-24.

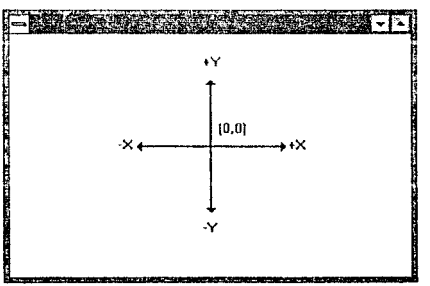


Figura 4-24.

Usando parte del programa del listado 4-1, podemos obtener una ventana con la siguiente salida, figura 4-25.

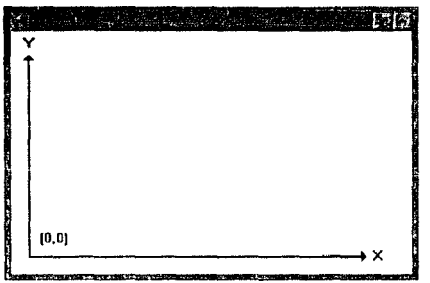


Figura 4-23.

Un efecto más fructífero de usar las instrucciones anteriores sería el siguiente

```
RECT rect;

GetClientRect(hDC, &rect);
SetViewportOrg(hDC,rect.right/2,rect.bottom/2);
```

Donde obtenemos una situación más apegada a los efectos matemáticos, figura 4-24.

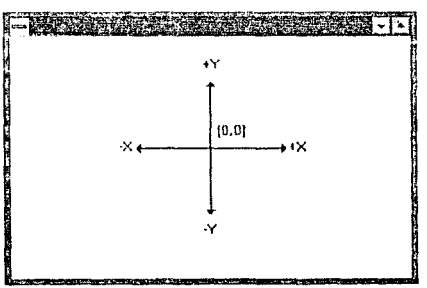


Figura 4-24.

Usando parte del programa del listado 4-1, podemos obtener una ventana con la siguiente salida, figura 4-25

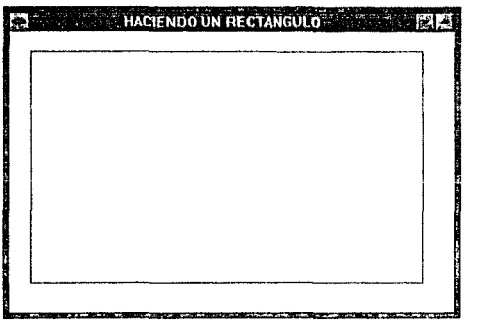


Figura 4-25.

El listado 4-3 muestra el programa fuente que da origen a la ventana de la figura 4-25.

Listado 4-3.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#pragma argsused

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstancia, HANDLE hPrevInstancia,
                    LPSTR lpszCmdParam, int nCmdShow)

{
HWND hWnd;
MSG Mensaje;

if (!hPrevInstancia)
        {
        WNDCLASS wClaseVentana;
        wClaseVentana.cbClsExtra=0;
        wClaseVentana.cbWndExtra=0;
        wClaseVentana.hbrBackground=GetStockObject(WHITE_BRUSH);
        wClaseVentana.hInstance=hInstancia;
        wClaseVentana.hCursor=LoadCursor(NULL,IDC_ARROW);
        wClaseVentana.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        wClaseVentana.lpfnWndProc=WndProc;
        wClaseVentana.lpszClassName="WIN_REC";
        wClaseVentana.lpszMenuName=NULL;
        wClaseVentana.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&wClaseVentana))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_REC",
                  "HACIENDO UN RECTANGULO",
                  WS_OVERLAPPEDWINDOW,
```

```
                    CW_USEDEFAULT,
                    0,
                    CW_USEDEFAULT,
                    0,
                    NULL,
                    NULL,
                    hInstancia,
                    NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Mensaje,NULL,0,0))
{
TranslateMessage(&Mensaje);
DispatchMessage(&Mensaje);
}
return Mensaje.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam)
{
HDC hDC;
PAINTSTRUCT PtStr;
RECT rect;
switch(iMensaje)
        {
        case WM_PAINT:

                        hDC=BeginPaint(hWnd,&PtStr);
                        SetMapMode(hDC,MM_ANISOTROPIC);
                        GetClientRect(hWnd,&rect);
                        Rectangle(hDC,20,20,rect.right-30,rect.bottom-30);
                        EndPaint(hWnd,&PtStr);
                        return 0;
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return    DefWindowProc(hWnd,iMensaje,wParam,lParam);
        }
                return(0L);
}
```

El archivo .DEF se muestra en el listado 4-4.

Listado 4-4.

```
NAME            WIN_REC
DESCRIPTION     'HACIENDO UN RECTANGULO'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
```

Es importante señalar una caracteristica de los elemetos usados en la zona de WM_PAINT. Cuando en un programa se desea usar una variable de estructura PAINTSTRUCT, implica que se pretende "repintar" los elemetos cada vez que la ventana sufra una modificación de tamaño.

Cuando la acción de "repintar" requiera una supervisión más compleja y controlada, el programador debe sustituir las funciones BeginPaint() y EndPaint() por las funciones GetDC() y ReleaseDC(), respectivamente; usándose la siguiente forma:

```
HDC hDC
switch(iMessage)
        {
        case WM_PAINT:
                        hDC=GetDC(hWnd);
                        ...
                        ...
                        ...
                        ReleaseDC(hWnd,hDC);
                        return 0;
```

4.8.4. Pixeles y Colores

Como se tratará más adelante, las figuras como rectángulos, círculos y elipses son fáciles de dibujar en una ventana, sin embargo, cuando se trata de "pintar" un pixel pueden ocurrir ciertas complicaciones. Para poder hacer dicha acción se procede a usar la función **SetPixel** cuyos argumentos son una variable del tipo **HDC**, manejador del dispositivo de contexto, dos variales enteras, la **x** y la **y** de las coordenadas, y una variable del tipo **COLORREF**. El tipo **COLORREF** es una estructura interna para controlar el color con el que desea deplegar uno o varios pixeles. Para obtener el valor del tipo COLORREF se usa la función **RGB**, la cual permite controlar tonalidades combinadas de rojo, verde y azul.

Como argumentos, RGB admite solo tres valores enteros de 0 a 255, así obtenemos 16,777,216 de posibles colores, pero en la mayoría de los casos, dependiendo de la tarjeta, monitor y circustancias físicas, para el usuario muchos de los colores de tales combinaciones parecerán iguales.

Unas posibles instrucciones para usar combinaciones en nuestro esquema podrían ser las siguientes:

```
SetPixel(hDC, 100, 100, RGB(255,0,0);      /* Un pixel rojo   */
SetPixel(hDC, 100, 101, RGB(0,255,0);      /* Un pixel verde  */
SetPixel(hDC, 100, 102, RGB(0,0,255);      /* Un pixel azul   */
SetPixel(hDC, 100, 103, RGB(255,255,255);  /* Un pixel negro  */
SetPixel(hDC, 100, 104, RGB(0,0,0);        /* Un pixel blanco*/
```

Como una forma de ejemplificar lo anterior, podemos crear el efecto que aparece en la figura 4-26.

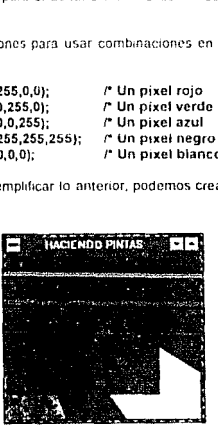


Figura 4-26.

La figura anterior se obtiene usando los siguientes los listados 4-5 y 4-6.

Listado 4-5.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#pragma argsused

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstancia, HANDLE hPrevInstancia,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Mensaje;

if (!hPrevInstancia)
        {
        WNDCLASS wClaseVentana;
        wClaseVentana.cbClsExtra=0;
        wClaseVentana.cbWndExtra=0;
        wClaseVentana.hbrBackground=GetStockObject(WHITE_BRUSH);
        wClaseVentana.hInstance=hInstancia;
        wClaseVentana.hCursor=LoadCursor(NULL,IDC_ARROW);
        wClaseVentana.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        wClaseVentana.lpfnWndProc=WndProc;
        wClaseVentana.lpszClassName="WIN_PTN";
        wClaseVentana.lpszMenuName=NULL;
        wClaseVentana.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&wClaseVentana))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_PTN",
                  "HACIENDO PINTAS",
                  WS_OVERLAPPEDWINDOW,
                  0,
                  0,
                  256,
                  256,
                  NULL,
                  NULL,
                  hInstancia,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Mensaje,NULL,0,0))
{
TranslateMessage(&Mensaje);
DispatchMessage(&Mensaje);
}
return Mensaje.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam)
{
HDC hDC;
PAINTSTRUCT PtStr;
unsigned x,y;
switch(iMensaje)
        {
        case WM_PAINT:

                        hDC=BeginPaint(hWnd,&PtStr);
```

```
                    for ( x=0; x<256; x++)
                    for ( y=0; y<256; y++)
                    SetPixel(hDC, x, y, RGB(x,y,abs(x-y)));
                    EndPaint(hWnd,&PtStr);
                    return 0;
        case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
        default:
                    return    DefWindowProc(hWnd,iMensaje,wParam,lParam);
        }
            return(0L);
}
```

Listado 4-5.

```
NAME            WIN_PTN
DESCRIPTION     'HACIENDO PINTAS'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
```

4.8.5. Plumas

Ahora bien, cuando se trata del trazo de líneas en una ventana, la situación es más variada que en el caso de los pixeles. Para iniciar el trazo de un simple recta se usan dos funciones típicas de las bibliotecas de gráfico en otros paquetes: **MoveTo y LineTo**. **MoveTo** sirve para enviar el cursor de dibujo a alguna coordenada en especial, **LineTo** para trazar una línea desde la posión del cursor de dibujo, hasta una nueva coordenada que recibe como argumento. La estructura básica de ambas funciones se muestra a continuación

```
DWORD  MoveTo (HDC hDC, int X, int Y);
BOOL   LineTo  (HDC hDC, int X, int Y);
```

El valor que regresa la función MoveTo corresponde a la *posición anterior* donde se encontraba el cursor de dibujo; mientras que el valor booleano que regresa LineTo corresponde a la verificación de si se pudo o no hacer la línea

En esta parte se introduce el concepto de "cursor de dibujo" para indicar un cursor imaginario que dibuja las figuras. En lenguajes tales como Logo el concepto de "cursor de dibujo" pasa a ser conocido como pluma (pen) y es usado alegoricamente en todas aquellas funciones para dibujar figuras. En la biblioteca WINDOWS H existen una serie de funciones que basan su funcionamiento en el de una pluma. Estas funciones se encargan de dibujar líneas con distintos tipos de colores y espesores; en forma semejante a las plumas que usa un dibujante en la realidad

En si cuando se inicia un dibujo, existe una pluma predefinida de color negro y espesura sencilla (de un pixel). Un usuario no solo puede alterar las características de la pluma, sino también crear diferentes tipos como si fueran variables. Para crear una pluma se usa la función **CreatePen**, la descripción de sus parámetros se muestra a continuación

```
HPEN CreatePen(int Estilo, int Espesor, COLORREF Color)
```

HPEN corresponde a un tipo de apuntador especial; ese el tipo del que se crean las variables. Para el **Estilo** se pueden usar valores predefinidos dentro de la biblioteca WINDOWS.H; estos valores se muestran en la tabla.

Tabla 4-8.

PS_SOLID	———
PS_DASH	- - - - - -
PS_DOT	.....
PS_DASHDOT	- . - . - .
PS_DASHDOTDOT	- . . - . . - . .
PS_NULL	
PS_INSIDEFRAME	———

Los valores usados para el espesor corresponden a números enteros de 0 a 16 (recomendado como posible máximo), cuando escribe el espesor con valor 0, en realidad la pluma traza con espesor de un pixel.

Como se mencionó anteriormente, es posible definir varias variables del **HPEN**, para indicarle al DC que se pretende usar alguna se utiliza la función SelectObject; la estructura de se muestra a continuación:

```
HPEN SelectObject(HDC hDC, HPEN Pluma)
```

Cuando se le asigna al DC una nueva pluma, la función **SelectObject** entrega como resultado la pluma anterior.

Cuando se desea deshechar una pluma, se invoca a la función **DeleteObject**, su estructura se muestra a continuación:

```
BOOL DeleteObject(HPEN Pluma)
```

SelectObject y **DeleteObject** son usadas para seleccionar y borrar otros tipos de variables usadas dentro del DC, su funcionamiento es igual al de crear variables dinámicas, es decir que cuando se usa **SelectObject** se esta creando en memoria la variable y con **DeleteObject** se borra.

Existe otra forma de crear variables del tipo **HPEN** usando la función **CreatePenIndirect**, cuya estructura es la siguiente:

```
HPEN CreatePenIndirect(LPLOGPEN LogPluma)
```

Con esta función se usa un tipo especial, la cual se muestra a continuación:

```
typedef struct tagLOGPEN {
  WORD lopnStyle;
  POINT lopnWidth;
  DWORD lopnColor;
} LOGPEN;
```

La estructura POINT se define de la siguiente manera:

```
typedef struct tagPOINT {
   int x;
   int y;
} POINT;
```

La forma de usar **CreatePenIndirect** es definiedo las variables de la siguiente forma:

```
LOGPEN lpPlumaRoja;
lpPlumaRoja.lopnStyle=PS_SOLID;
lpPlumaRoja.lopnWidth.x=5;
lpPlumaRoja.lopnColor=RGB(255,0,0);

HPEN hPlumaRoja;

HPlumaRoja=CreatePenIndirect(&lpPlumaRoja);
```

O bien, de la siguiente forma

```
LOGPEN lpPlumaRoja = { PS_SOLID, 5, 5, RGB(255,0,0)};

HPEN hPlumaRoja;

HPlumaRoja=CreatePenIndirect(&lpPlumaRoja);
```

Se debe notar que la parte **lopnWidth.y** no tiene un uso relevante. Ahora bien, WINDOWS.H tiene definidas tres plumas: **BLACK_PEN, WHITE_PEN y NULL_PEN**. La BLACK_PEN es la pluma con la que siempre inicia toda ventana. Para poder instalar, o reinstalar cualquiera de esta plumas se debe usar la función **GetStockObject**, de la siguiente forma

```
GetStockObject(BLACK_PEN)
```

Al igual que SelectObject, GetStockObject también regresa el valor de la pluma anterior

En la figura 4-27 se muestra un rectangulo parecido al de la figura 4-25, pero dibujado con una "pluma" más gruesa y de otro color

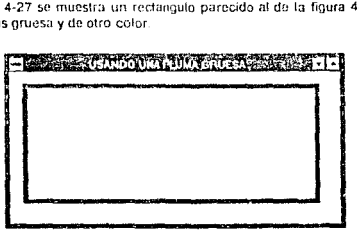


Figura 4-27.

Los listados 4-7 y 4-8 muestran le programa y archivo .DEF generadores de la figura 4-27.

Listado 4-7.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#pragma argsused

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstancia, HANDLE hPrevInstancia,
                    LPSTR lpszCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Mensaje;

if (!hPrevInstancia)
        {
        WNDCLASS wClaseVentana;
        wClaseVentana.cbClsExtra=0;
        wClaseVentana.cbWndExtra=0;
        wClaseVentana.hbrBackground=GetStockObject(WHITE_BRUSH);
        wClaseVentana.hInstance=hInstancia;
        wClaseVentana.hCursor=LoadCursor(NULL,IDC_ARROW);
        wClaseVentana.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        wClaseVentana.lpfnWndProc=WndProc;
        wClaseVentana.lpszClassName="WIN_REC";
        wClaseVentana.lpszMenuName=NULL;
        wClaseVentana.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&wClaseVentana))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_REC",
                  "USANDO UNA PLUMA GRUESA",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstancia,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Mensaje,NULL,0,0))
{
TranslateMessage(&Mensaje);
DispatchMessage(&Mensaje);
}
return Mensaje.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam)
{
HDC hDC;
PAINTSTRUCT PtStr;
RECT rect;
HPEN hPen;
switch(iMensaje)
        {
        case WM_PAINT:

                        hDC=BeginPaint(hWnd,&PtStr);
```

```
                    SetMapMode(hDC,MM_ANISOTROPIC);
                    hPen=CreatePen(PS_SOLID,8,RGB(0,0,200));
                    SelectObject(PtStr.hdc,hPen);
                    GetClientRect(hWnd,&rect);
                    Rectangle(hDC,20,20,rect.right-30,rect.bottom-30);
                    EndPaint(hWnd,&PtStr);
                    return 0;
        case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
        default:
                    return    DefWindowProc(hWnd,iMensaje,wParam,lParam);
        }
            return(0L);
}
```

Figura 4-8.

```
NAME            WIN_REC
DESCRIPTION     'USANDO UNA PLUMA GRUESA'
EXETYPE             WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
```

4.8.6. Figuras

A parte de rectángulos, en una ventana pueden dibujarse elipsoides, arcos, rebanas de circunferencias (pie), rectángulos con esquinas redondeadas y polígonos. Las funciones para crear dichas figuras se muestran a continuación

Ellipse(HDC hDC, int X1, int Y1, int X2, Int Y2)

Chord(HDC hDC, int X1, int Y1, int X2, int Y2, int iniX, int iniY, int finX, int finY)

Pie((HDC hDC, int X1, int Y1, int X2, int Y2, int iniX, int iniY, int finX, int finY)

Arc(HDC hDC, int X1, int Y1, int X2, int Y2, int iniX, int iniY, int finX, int finY)

RoundRect(HDC hDC, int X1, int Y1, int X2, int Y2, int ElipseA, int ElipseL)

Polygon(HDC hDC, LPPOINT Puntos, int NumPuntos)

Las funciones **Ellipse**, **Chord**, **Pie** y **Arc** contruyen sus figuras dentro de un rectángulo, invisible para el usuario. Los primeros cuatro valores enteros que reciben como argumentos corresponden a las coordenadas de las esquinas superior-izquierda e inferior-derecha de dicho rectangulo. Los últimos cuatro valores enteros, en el caso de **Chord**, **Pie** y **Arc**, corresponden a coordenadas del mismo rectángulo que permiten la abertura; ver figuras 4-28 y 4-29

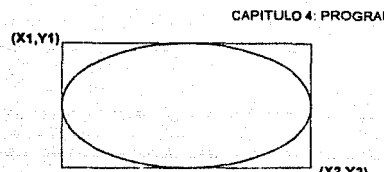


Figura 4-28.

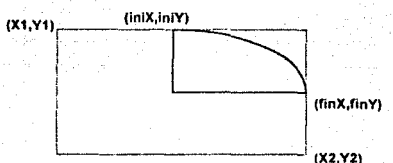


Figura 4-29.

Para el caso de la función RoundRect los últimos enteros corresponden a los diámetros de una elipse invisible con la que se forman las partes curvas de las esquinas de la figura 4-30.

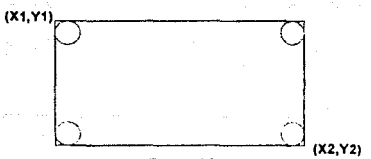


Figura 4-30.

Para el caso de la función **Polygon** aparece un tipo, **LPPOINT**, el cual resulta ser un apuntador de la estructura **POINT**. Esto es porque Polygon, para poder realizar una figura, debe recibir un conjunto de pares puntos. Así, la mejor manera de escribir un polígono es escribiendo sus puntos dentro de un arreglo del tipo POINT, el siguiente fragmento de listado nos ejemplifica lo anterior.

```
POINT p[3] = {150,20, 250,125, 25,125};
Polygon(hDC, p, 3);
```

El último valor corresponde al número de parejas de puntos que se desean unir.

4.8.7. Brochas

El concepto de brocha (brush) en las bibliotecas gráficas se refiere a aquellas funciones usadas para rellenar figuras. Al igual que las plumas, existe una brocha predeefinida dentro del DC y un manejador para tal efecto dentro Windows.h, el **HBRUSH**.

Los tipos de brochas predefinidos son siete y se muestran en la tabla 4-9.

Tabla 4-9.

BLACK_BRUSH	**Negro**
DKGRAY_BRUSH	**Gris Obscuro**
GRAY_BRUSH	**Gris**
HOLLOW_BRUSH	**Sin color**
LTGRAY_BRUSH	**Gris brillante**
NULL_BRUSH	**Sin color**
WHITE_BRUSH	**Blanco**

Por default la brocha de una ventana es WHITE_BRUSH. Para hacer uso de las otras brochas se usan las funciones **GetStockObject** y **SelectObject** de la siguiente forma:

```
HBRUSH hbBrocha;
hbBrocha=GetStockObject(GRAY_BRUSH);
SelectObject(hDC, hbBrocha);
Rectangle(hDC,x1,x2,x3,x4);
```

Así, una figura, un rectángulo, será rellenado de color gris. Ahora bien, el programador puede crear sus propias brochas por medio de cinco funciones: **CreateSolidBrush**, **CreateHatchBrush**, **CreatePatternBrush**, **CreateBrushIndirect** y **CreateDIBPPatternBrush**.

Por medio de la función CreateSolidBrush se puede crear la "brocha" que uno desea, el formato es muy similar al que usa con los pinceles:

```
HBRUSH hbBrochaRoja=CreateSolidBrush(RGB(255, 0, 0);
```

Igualmente para la asignación del DC se usa **SelectObject**.

También se pueden crear brochas que rellenen de determinadas formas de sombreado. Para ello usamos **CreateHatchBrush**, cuya estructura es la siguiente:

```
CreateHatchBrush(int TipoSom, COLORREF Color);
```

Los tipos de sombreados dentro del sistema se enlistan en la tabla.

Tabla 4-10.

NOMBRE	FORMA
HS_BDIAGONAL	**////////////**
HS_CROSS	**########**
HS_DIAGCROSS	**XXXXXXX**

HS_FDIAGONAL	\\\\\\\\\\\\\\\\
HS_HORINZONTAL	========
HS_VERTICAL	|||||||||||||||||

Otra forma de crear variables "brocha" es por medio de la función CreateBrushIndirect; su estructura es la siguiente:

HBRUSH CreateBrushIndirect(LPLOGBRUSH LogBrush)

Donde el tipo LPLOGBRUSH tiene la siguiente estructura:

```
typedef struct tagLOGBRUSH {
WORD   lbStyle;
DWORD lbColor;
int        lbHatch;
} LOGBRUSH;
```

Al igual que con las plumas, también podemos dar características por medio de una estructura, en este caso LPLOGBRUSH. Un posible formato se enlista a continuación:

```
LOGBRUSH lbBrocha;
lbBrocha.lbStyle=BS_SOLID;
lbBrocha.lbColor=RGB(200,0,0);
HBRUSH= CreateBrushIndirect(&lbBrocha);
```

El valor que recibe la parte lbStyle, es un valor predefinido semejante al de los tipos de brochas para uso exclusivo de la estructura LOGBRUSH, en la tabla 4-11 se enlistan tales valores.

Tabla 4-11.

BS_DIPATTERN	La brocha puede se define a partir de un DIB (dispositivo de bitmap).
BS_HATCHED	La brocha puede ser definda a partir combinaciones de los valores de la tabal 4-10.
BS_HOLLOW	La brocha se vuelve "invisible".
BS_NULL	Se anula el tipo de brucha que se use en el momento.
BS_PATTERN	La brocha puede se define a partir de un de bitmap.
BS_SOLID	La brocha se define por medio de un color sólido.

En la figura 4-31 se muestra un relleno un rectángulo generado por una brocha diseñada por el programador.

Figura 4-31.

Los listado 4-9 y 4-10 muestran el programa y el archivo de .DEF respectivamente.

Listado 4-9.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#pragma argsused

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstancia, HANDLE hPrevInstancia,
                    LPSTR lpszCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Mensaje;

if (!hPrevInstancia)
        {
        WNDCLASS wClaseVentana;
        wClaseVentana.cbClsExtra=0;
        wClaseVentana.cbWndExtra=0;
        wClaseVentana.hbrBackground=GetStockObject(WHITE_BRUSH);
        wClaseVentana.hInstance=hInstancia;
        wClaseVentana.hCursor=LoadCursor(NULL,IDC_ARROW);
        wClaseVentana.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        wClaseVentana.lpfnWndProc=WndProc;
        wClaseVentana.lpszClassName="WIN_RYN";
        wClaseVentana.lpszMenuName=NULL;
        wClaseVentana.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&wClaseVentana))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_RYN",
                  "HACIENDO RELLENOS",
                  WS_OVERLAPPEDWINDOW,
                  0,
                  0,
                  300,
                  300,
                  NULL,
```

```
                    NULL,
                    hInstancia,
                    NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Mensaje,NULL,0,0))
{
TranslateMessage(&Mensaje);
DispatchMessage(&Mensaje);
}
return Mensaje.w.Param;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMensaje, WORD wParam, LONG lParam)
{
HDC hDC;
PAINTSTRUCT PtStr;
RECT    rRect;
HBRUSH  hBrush;

switch(iMensaje)
        {
        case WM_PAINT:

                        hDC=BeginPaint(hWnd,&PtStr);
                        hBrush=CreateHatchBrush(HS_DIAGCROSS,RGB(0,0,255));
                        SetRect(&rRect,10,10,250,250);
                        FillRect(PtStr.hdc,&rRect,hBrush);
                        EndPaint(hWnd,&PtStr);
                        DeleteObject(hBrush);
                        return 0;
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return  DefWindowProc(hWnd,iMensaje,wParam,lParam);
        }
                return(0L);
}
```

Listado 4-10.

```
NAME            WIN_NYN
DESCRIPTION     'HACIENDO RELLENOS'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
```

Por supuesto, en la figura 4-31 se ha rellenado una zona de forma rectángular, pero ¿dónde esta el perimetro del susodicho rectángulo? Pues simplemente no se ordenó que se "pintara". Para que se ejecute tal caso basta agregar las siguientes instruciones.

```
SelectObject(PtStr.hdc,hBrush);
Rectangle(PtStr.hdc,10,10,250,250);*/
```

Asi la zona de WM_PAINT queda de la siguiente forma:

```
case WM_PAINT:

                hDC=BeginPaint(hWnd,&PtStr);
                hBrush=CreateHatchBrush(HS_DIAGCROSS,RGB(0,0,255));
                SelectObject(PtStr.hdc,hBrush);
```

```
Rectangle(PtStr.hdc,10,10,250,250);*/
SetRect(&rRect,10,10,250,250);
FillRect(PtStr.hdc,&rRect,hBrush);
EndPaint(hWnd,&PtStr);
DeleteObject(hBrush);
return 0;
```

Esto origina la ventana de la figura 4-32

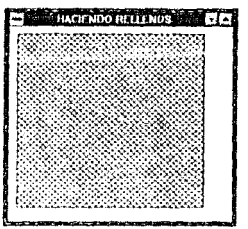


Figura 4-32.

4.8.8. Texto en el GDI

Para poder escribir un texto dentro de una ventana, el GDI debe ser preparado. Para ello se usa la sección del mensaje WM_CREATE para posteriormente pasar a WM_PAINT, donde existirán las principales instrucciones del texto

Las cinco funciones fundamentales para escribir texto se muestran en la tabla 4-12

Tabla 4-12.

FUNCION	LO QUE HACE
DrawText	Escribe un texto dentro de un rectángulo.
ExtTextOut	Escribe una cadena de caracteres usando el tipo de letra especificada (inscrito dentro de un rectángulo).
GrayString	Escribe una cadena en color gris.
TabbedTextOut	Escribe una cadena siguiendo un espaciemiento determinado (tab).
TextOut	Escribe una cadena de caracteres usando el tipo de letra especificada.

También existen las funciones **wsprintf** y **wsvprintf** que son versiones para Windows de la legendaria printf y vsprint, respectivamente, que se usan dentro de C

A su vez, estas funciones dependen indirectamente de otro grupo de funciones que son las que se escriben dentro de la zona del WM_CREATE. Dentro de este grupo tenemos la función GetTextMetrics, cuya estructura es la siguiente:

GetTextMetrics(HDC hDC,lpMetrics LPMETRICS)

Por medio de esta función es posible establecer características de tamaño de las letras que conformen el texto dentro de una ventana. El tipo lpMetrics corresponde a un apuntador de la estructura **TEXTMETRIC** cuya definición se muestra a continuación:

```
typedef struct tagTEXTMETRIC {
        short int tmHeight;
        short int tmAscent;
        short int tmDescent;
        short int tmInternalLeading;
        short int tmExternalLeading;
        short int tmAveCharWidth;
        short int tmMaxCharWidth;
        short int tmWeight;
        Byte tmItalic;
        Byte tmUnderlined;
        Byte tmStruckOut;
        Byte tmFirstChar;
        Byte tmLastChar;
        Byte tmDefaultChar;
        Byte tmBreakChar;
        Byte tmPitchAndFamily;
        Byte tmCharSet;
        short int tmOverhang;
        short int tmDigitizedAspectX;
        short int tmDigitizedAspectY;
} TEXTMETRIC;
```

Cuando se dibuja una letra, esta se crea dentro de un rectángulo dividio en tres zonas , figura 4-33

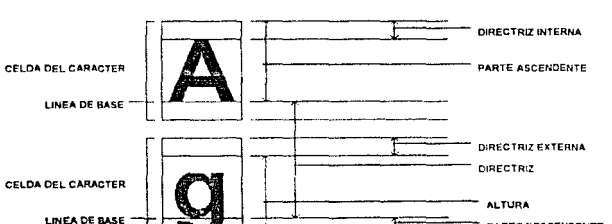


Figura 4-33.

Asi dentro de la estructura TEXTMETRIC, los elementos que la forman controlan los siguientes aspectos que se muestran en la tabla 4-13.

Tabla 4-13.

tmHeight	**Corresponde a la altura del caracter, unidades según el modo de mapeo.**
tmAscent	**Corresponde a la parte ascendente del caracter (ver figura 4-33)**
tmDescent	**Corresponde a la parte descendente del caracter (ver figura 4-33)**
tmInternalLeading	**Corresponde a distancia entre la parte alta del caracter**
tmExternalLeading	**Corresponde al espacio entre un reglón y otro.**
tmAveCharWidth	**Ancho promedio de los caracteres.**
tmMaxCharWidth	**Ancho máximo de los caracteres.**
tmWeight	**Espesor del tipo de letra.**
tmItalic	**Presentacion en forma itálica del caracter, un valor diferente de cero indica que si.**
tmUnderlined	**Presentación en forma subrayada del caracter, un valor diferente de cero indica que si.**
tmStruckOut	**Presentación en de la silueta del caracter, un valor diferente de cero indica que si.**
tmFirstChar	**Primer caracter define el tipo de letra.**
tmLastChar	**Ultimo caracter define el tipo de letra.**
tmDefaultChar	**Un caracter no usa el tipo de letra**
tmBreakChar	**Un caracter es usado para pasar a la siguiente linea de escritura.**
tmPitchAndFamily	**Byte de selección de caracteristica de la familia de letras.**
tmCharSet	**Adición de un caracter para el tipo de letra usado.**
tmOverhang	**Ancho extra a usar en caracteres especiales.**
tmDigitizedAspectX	**Aspecto de la horizontalidad para el depliegue de los caracteres de acuerdo al dispositivo de video.**
tmDigitizedAspectY	**Aspecto de la verticalidad para el depliegue de los caracteres de acuerdo al dispositivo de video.**

Ahora bien, si se desea manejar diferentes tipos de letras, el usuario debe proceder a usar tipos, funciones y variables semejantes a las usadas para el caso de las plumas y brochas.

Así, para poder desplegar un texto usando algun tipo de letra se definen variables que representen "las plumas para escribir" con el tipo de letras. El manejador que se usa para estas definiciones HFONT, de igual forma se vuelve a usar para las funciones GetStockObject() y SelectObject(). El valor que recibe GetStockObject corresponde a valores constantes de tipos de letras contenidos en Windows.h en la tabla 4-14 muestra tales tipos de letras.

Tabla 4-14

ANSI_FIXED_FONT
ANSI_VAR_FONT
DEVICE_DEFAULT_FONT
OEM_FIXED_FONT
SYSTEM_FONT
SYSTEM_FIXED_FONT

Pero, también pueden ser usados tipos de letras externos, como lo usa Windows en algunas de su aplicaciones. Cuando se toman tipos de letras externos debemos ser explícitos en lo que se refiere a su tamaño y estilo de despliegue (subrayado, negritas, forma itálica). Dentro del programa que pretende usar tipos externos se considera que los tipos se encuentran clasificados en familias, y son los nombres de tales familias por medio de los cuales se hacen referencias en la estructura de programa al tipo a cargar. La tabla 4-15 nos muestra el nombre común de tipos de letras típicos en Windows 3.1, la familia con la que se le reconoce y un ejemplo de su depliegue.

Tabla 4-15.

Courier	FF_MODERN	Courier
Helv	FF_SWISS	Helv
Modern	FF_MODERN	Modern
Roman	FF_ROMAN	Roman
Script	FF_SCRIPT	*Script*
Symbol	FF_DECORATIVE	Σψμβολ
System	FF_SWISS	**System**
System	FF_DONTCARE	**System**
Terminal	FF_MODERN	Terminal
Tms Rmn	FF_ROMAN	Tms Rmn

La funciones para indicar el uso de tipos de letras externos son **CreateFont** y **CreateFontIndirect**. La estructura de **CreateFont** es la siguiente.

HFONT CreateFont(int Height, int width,int Escapement, int Orientation, int Weight, BYTE Underline, BYTE StrikeOut, BYTE Outputprecision, BYTE ClipPrecision, BYTE Quality, BYTE PitchAndFamily, LPSTR FontName);

Todos los argumentos que maneja CreateFont son necesarios para otorgársele a la variable de tipo **HFONT** las características de despliegue. Al contrario, **CreateFontIndirect** es mucho más simple:

```
HFONT CreateFontIndirect(LOGFONT FAR * lpLogFont)
```

Donde la estructura de **LOGFONT** es la siguiente:

```
typedef struct tagLOGFONT {
        short int lfHeight;
        short int lfWidth;
        short int lfEscapement;
        short int lfOrientation;
        short int lfWeight;
        BYTE lfItalic;
        BYTE lfUnderline;
        BYTE lfStrikeOut;
        BYTE lfCharSet;
        BYTE lfOutPrecision;
        BYTE lfClipPrecision;
        BYTE lfPitchAndFamily;
        BYTE lfFaceName[LF_FACESIZE];
} LOGFONT;
```

Y los significados de sus partes se muestran a continuación:

lfHeight altura promedio del tipo de letra en unidades del modo de mapeo.

lfWidth anchura promedio del tipo de letra en unidades del modo de mapeo.

lfEscapement ángulo entre la horizontal de la pantalla y la línea escape del caracter.

lfOrientation ángulo entre la horizontal de la pantalla y la línea base del caracter.

lfWeight el espesor de tipo de letra con que será dezplegado.

lfItalic indicador para desplegar el tipo de letra en forma itálica (con valores diferentes de cero).

lfUnderline indicador para desplegar el tipo de letra en forma subrayada (con valores diferentes de cero).

lfStrikeOut indicador para desplegar el tipo de letra con una línea central (con valores diferentes de cero).

lfCharSet conjunto de caractres que podría manejar el tipo de letra.

lfOutPrecision presición (resolución) con la que podría parecer el tipo de letra.

lfClipPrecision tipo presición en la apariencia física de los caracteres.

lfPitchAndFamily calidad en la presentación del tipo de letra.

lfFaceName caracter nulo.

Un programa que ejemplique un despliegue básico se muestra a continuación en la figura 4-34

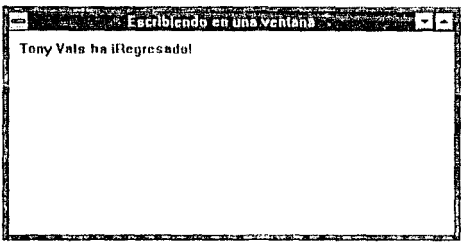


Figura 4-34

El listado 4-11 crea el programa de la figura 4-34

Listado 4-11.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(hInstance,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_LET";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_LET",
                  "Escribiendo en una ventana",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
```

```
                    CW_USEDEFAULT,
                    0,
                    NULL,
                    NULL,
                    hInstance,
                    NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
PAINTSTRUCT PtStr;

switch(iMessage)
        {
        case WM_PAINT:
                        hDC=BeginPaint(hWnd,&PtStr);
                        TextOut(hDC,10,10,"Tony Vals ha ¡Regresado!",24);
                        ReleaseDC(hWnd,hDC);
             break;
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return   DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

Practicamente, ninguna de las funciones relacionadas con texto pueden llamar a algún tipo de letra en especial, excepto por la función **TextOut()** que es usada para desplegar el letrero. La sintáxis de esta función es la siguiente:

```
TextOut( HDC hDC,   /* MANEJADOR*/
 int  x,            /* COORDENADA X PARA DESPLEGAR*/
 int  y,            /* CORDENADA Y PARA DESPLEGAR*/
 LPSTR cadena,      /* LETRERO A DESPLEGAR*/
 int  cont          /* TAMAÑO DEL LETRERO A DEZPLEGAR*/
);
```

Cuando no se selecciona ningun tipo de letra en especial, Windows otorga al texto desplegado el tipo System.

El programa del listado 4-12 crea un tipo de letra especial, nuevamente se usa la función TextOut, la ventana generada se muestra en la figura 4-35.

Listado 4-12.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
```

```
{
HWND hWnd;
MSG Message;

If (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(hInstance,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_FON";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_FON",
                  "Creando un nuevo tipo de letra",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
HFONT hFont;
PAINTSTRUCT PtStr;

switch(iMessage)
        {
        case WM_PAINT:
                        hDC=BeginPaint(hWnd,&PtStr);
                        hFont=CreateFont(24,16,0,0,400,0,0,0,
                        OEM_CHARSET, OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                        DEFAULT_PITCH|FF_SCRIPT,"script");
                        SelectObject(hDC,hFont);
                        TextOut(hDC,10,10,"Tony Vais ha ¡Regresado!",25);
                        ReleaseDC(hWnd,hDC);
                        DeleteObject(hFont);
            break;
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return    DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);}
```

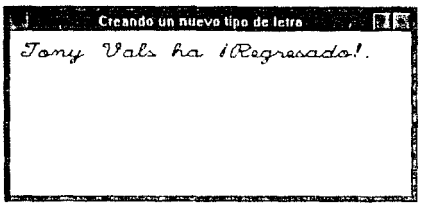


Figura 4-35.

4.9. Barras Deslizables de Control

Las barras deslizables de Control son las partes más interesantes dentro de la programación, no solo dentro de Windows, si no en cualquier medio ambiente que manejo ventanas. Los mensajes que se encargan de los botones de control son **WM_VSCROLL** (para el movimiento vertical) y **WM_HSCROLL** (para el movimiento horizontal). Un tercer mensaje es considerado, **WM_SIZE**, el cual se encarga de los cambios ocurridos sobre el tamaño de la ventana principal.

Ahora bien, podemos tener seis formas primarias de realizar el movimiento de una barra de control vertical:

Subir una línea
Bajar una línea
Subir una página
Bajar una página
Subir hasta la parte más superior
Bajar hasta la parte más inferior

Las cuatro primeras están en función de las líneas que se encuentren desplegadas en el área de trabajo, sin considerar el modo de mapeo. Los restantes movimientos se pueden realizar por medio de funciones especiales que conozcan el total de líneas. Por supuesto que por el momento estamos considerando que solo se esta desplegando texto en la ventana; así que podemos establecer un séptimo movimiento de la barra, uno que se base en las unidades del modo de mapeo: movimiento por unidades de pantalla.

Todos estos movimientos son manejados como mensajes contenidos en una estructura switch-case de la parte de **WM_VSCROLL**. La tabla 4-16 muestra los mensajes correspondientes a los movimientos antes mencionados.

Tabla 4-16.

SB_LINEUP	**Subir una línea**
SB_LINEDOWN	**Bajar una línea**
SB_PAGEUP	**Subir una página**
SB_PAGEDOWN	**Bajar una página**
SB_TOP	**Ir hasta la parte más superior**

SB_BOTTOM	**Ir hasta la parte más inferior**
SB_THUMBPOSITION	**Movimiento por unidades dadas**

Para el caso de WM_HSCROLL se consideran los mismos mensajes, pero tomando en cuenta las consideraciones mostradas en la tabla 4-17.

Tabla 4-17.

SB_LINEUP	Mover un caracter a la izquierda
SB_LINEDOWN	Mover un caracter a la derecha
SB_PAGEUP	Mover varios caracteres a la izquierda
SB_PAGEDOWN	Mover varios caracteres a la derecha
SB_TOP	Ir hasta la parte final derecha
SB_BOTTOM	Ir hasta la parte final izquierda
SB_THUMBPOSITION	Movimiento por unidades dadas

En la figura 4-36, se puede observar las partes que afectan los mensajes en una ventana.

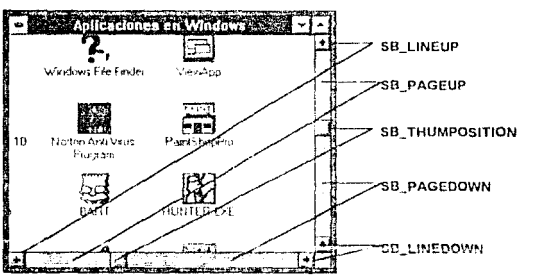


Figura 4-36.

Las funciones comunes para las barras de control se muestran en la tabla 4-18.

Tabla 4-18.

FUNCION	LO QUE HACE
EnableScrollBar	"Enciende" o "apaga" el control de la barra deslizable.
GetScrollPos	Regresa la pocisión del botón de la barra.
GetScrollRange	Regresa el rango mínimo y máximo de la barra de control
ScrollDC	Controla una zona del dispositivo de contexto.
ScrollWindows	Controla una zona del área de trabajo.
SetScrollPos	Establece una pocisión del botón de la barra de control.
SetScrollRange	Establece los rangos de la barra de control.
ShowScrollBar	Despliega la barra de control.

El uso de las barras de control implica controlar las caracteristicas directas de la ventana principal, sin embargo puede ser necesario crear ventanas hijas donde se ejecute la barra (ventana de edición); por ello es muy común usar la función CreateWindow() dentro de WndProc para tales fines.

4.10. Periféricos

4.10.1. El Teclado

Los mensajes provenientes del teclado pasan por medio de los parámetros lParam y wParam. Por medio del lParam pasan los códigos de las teclas que podemos considerar puras, es decir letras, números y caracteres especiales (no incluyen símbolos extranjeros), a este tipo de teclas se dice que pertenecen al sistema. Por medio de wParam se detectan los códigos de las teclas virtuales, esto es, aquellas teclas usadas para propositos especiales. En la tabla 4-19 se muestran los diferentes mensajes a usar según se trate de teclas del sistema o virtuales.

Tabla 4-19.

WM_KEYDOWN	Se ha presionado una de las teclas virtuales.
WM_CHAR	Se presionado una de las teclas alfanuméricas (código ASCII)
WM_KEYUP	Se ha dejado de presionar una de las teclas vituales.
WM_SYSKEYDOWN	Se ha presionado una de las teclas virtuales.
WM_SYSKEYUP	Se ha dejado de presionar una de las teclas vituales.
WM_SYSCHAR	Se presionado una de las teclas alfanuméricas (código ASCII) junto con la tecla ALT.
WM_SISKEYUP	Se ha dejado de presionar una de las teclas alfanuméricas (código ASCII) junto con la tecla ALT.

Los mensajes son colocados como parte de la estructura switch-case del WinProc, a su vez puede ser construida otra estructura para distribuir los casos según la tecla deseada, para esto, existen una serie de constantes definidas dentro de Windows.h según la tecla, tabla 4-20.

Tabla 4-20.

VK_ACCEPT	0x1E	Kanji (caracter japones).
VK_ADD	0x6B	Tecla de suma (+).
VK_BACK	0x08	Tecla de regreso.
VK_CANCEL	0x03	Tecla Cancel.

VK_CAPITAL	0x14	Tecla Shift.
K_CLEAR	0x0C	Tecla Clear (Numpad 5).
K_CONTROL	0x11	Tecla Ctrl.
K_CONVERT	0x1C	Kanji (caracter japones).
K_DECIMAL	0x6E	Punto decimal.
K_DELETE	0x2E	Tecla Delete.
K_DIVIDE	0x6F	Tecla de división (/).
K_DOWN	0x28	Flecha abajo.
K_END	0x23	Tecla End.
K_ESCAPE	0x1B	Tecla Esc.
K_EXECUTE	0x2B	Tecla Execute (si la hay).
K_F1	0x70	Tecla de función
K_F2	0x71	"
K_F3	0x72	"
K_F4	0x73	"
K_F5	0x74	"
K_F6	0x75	"
K_F7	0x76	"
K_F8	0x77	"
K_F9	0x78	"
K_F10	0x79	"
K_F11	0x7A	Teclado extendido.
K_F12	0x7B	"
K_F13	0x7C	Teclado especial.
K_F14	0x7D	"
K_F15	0x7E	"
K_F16	0x7F	"

K_HIRAGANA	0x18	Kanji (caracter japones).
K_HOME	0x24	Tecla Home.
K_INSERT	0x2D	Tecla Insert.
K_KANA	0x15	Kanji (caracter japones).
K_KANJI	0x19	Kanji (caracter japones).
K_LBUTTON	0x01	Botón izquierdo del ratón.
K_LEFT	0x25	Flecha izquierda.
K_MBUTTON	0x04	Botón central del ratón.
K_MENU	0x12	Tecla de menú.
K_MODECHANGE	0x1F	Kanji (caracter japones).
K_MULTILY	0x6A	Tecla de multiplicación (*).
K_NEXT	0x22	Tecla Next.
K_NONCONVERT	0x1D	Kanji (caracter japones).
K_NUMLOCK	0x90	Tecla Num Lock.
K_NUMPAD0	0x60	Tecla del Numpad.
K_NUMPAD1	0x61	"
K_NUMPAD2	0x62	"
K_NUMPAD3	0x63	"
K_NUMPAD4	0x64	"
K_NUMPAD5	0x65	"
K_NUMPAD6	0x66	"
K_NUMOAD7	0x67	"
K_NUMPAD8	0x68	"
K_NUMPAD9	0x69	"
K_PAUSE	0x13	Tecla pause.
K_PRINT	0x2A	Tecla print.
K_PRIOR	0x21	Tecla Page up.

K_RBUTTON	0x02	Botón derecho del ratón.
K_RETURN	0x0D	Tecla return.
K_RIGHT	0x27	Flecha derecha.
K_ROMAJI	0x16	Kanji (caracter japones).
K_SELECT	0x29	Tecla selectora.
K_SEPARATOR	0x6C	Tecla separadora.
K_SHIFT	0x10	Tecla shift.
K_SNAPSHOP	0x2C	Imprimir pantalla.
K_SPACE	0x20	Barra espaciadora.
K_SUBTRACT	0x6D	Tecla menos (-).
K_TAB	0x09	Tecla tab.
K_UP	0x26	Flecha arriba.
K_ZENKAKU	0x17	Kanji (caracter japones).

El programa mostrado en el listado 4-13 es un editor primitivo escrito por James McCord (ver bibliografía), en donde se ejemplifica el uso de mensajes provenientes del teclado. Una ventana de salida se muestra en la figura 4-37.

Listado 4-13.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>


long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

/*void NEAR PASCAL CaretPos(HWND hWnd, int nArrayPos, char *cCharBuf, int *xCaret,
                          int *yCaret, int nCharWidth);*/

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
          LPSTR lpstrCmdParam, int nCmdShow)

{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
    {
    WNDCLASS WndClass;
    WndClass.cbClsExtra=0;
    WndClass.cbWndExtra=0;
    WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
    WndClass.hInstance=hInstance;
    WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
    WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
```

```
    WndClass.lpszClassName="WIN_TEC";
    WndClass.lpszMenuName=NULL;
    WndClass.style=CS_HREDRAW | CS_VREDRAW;

    if (!RegisterClass(&WndClass))
      exit(FALSE);

    }
hWnd=CreateWindow("WIN_TEC",
          "EL TECLADO",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,
          0,
          CW_USEDEFAULT,
          0,
          NULL,
          NULL,
          hInstance,
          NULL);



ShowWindow(hWnd,nCmdShow);
UpdateWindow(hWnd);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}


void NEAR PASCAL CaretPos(HWND hWnd, int nArrayPos, char *cCharBuf, int *xCaret,
                          int *yCaret, int nCharWidth)
{
DWORD dWord;
HDC hDC;

hDC=GetDC(hWnd);
dWord=GetTextExtent(hDC,cCharBuf,nArrayPos);
ReleaseDC(hWnd,hDC);
*xCaret=( LOWORD(dWord)+nCharWidth);
SetCaretPos(*xCaret,*yCaret);
}


long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
#define BuffSize 400
static unsigned char cCharBuf[BuffSize];
static unsigned linea=1;
static int nNumChar=0;
static int nArrayPos=0;
static int nLineHeight;
static int nCharWidth;
static int xCaret, yCaret;
int x;
HDC hDC;
TEXTMETRIC tm;
PAINTSTRUCT PtStr;


switch(iMessage)
    {
    case WM_CHAR:
                {
                if (wParam==VK_BACK)
```

```
                {
                 if (nArrayPos==0)
                        MessageBox(hWnd,"No puedo espaciar",NULL,MB_OK);
                 else
                        {
                        nArrayPos=nArrayPos-1;
                        CaretPos(hWnd,nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                        for (x=nArrayPos;x<nNumChar;x=x+1)
                        cCharBuf[x]=cCharBuf[x+1];
                        InvalidateRect(hWnd,NULL,TRUE);
                        }
                 break;
                 }
                if (wParam<=VK_BACK)
                 {
                        MessageBox(hWnd,"Trate otra tecla",NULL,MB_OK);
                        break;
                 }
                 if (nNumChar>=400)
                 {
                        MessageBox(hWnd,"Buffer lleno",NULL,MB_OK);
                        break;
                 }
                 for(x=nNumChar;x>nArrayPos;x=x-1)
                        cCharBuf[x]=cCharBuf[x-1];
                 cCharBuf[nArrayPos]=(unsigned char)wParam;
                 nArrayPos=nArrayPos+1;
                 nNumChar=nNumChar+1;
                 CaretPos(hWnd,nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                 InvalidateRect(hWnd,NULL,TRUE);
                 }
                 break;
case WM_CREATE:
                 {
                  hDC=GetDC(hWnd);
                  GetTextMetrics(hDC,&tm);
                  nLnHeight=tm.tmHeight+tm.tmExternalLeading;
                  nCharWidth=tm.tmAveCharWidth;
                  yCaret=nLnHeight;
                  ReleaseDC(hWnd,hDC);
                 }
                 break;
case WM_SETFOCUS:
                  CreateCaret(hWnd,0,0,nLnHeight);
                  CaretPos(hWnd, nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                  ShowCaret(hWnd);
                  break;
case WM_KILLFOCUS:
                  DestroyCaret();
                  break;
case WM_KEYDOWN:
                  {
                        switch(wParam)
                        {
                        case VK_END:
                                nArrayPos=nNumChar;
                                CaretPos(hWnd,nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                                break;
                        case VK_HOME:
                                nArrayPos=0;
                                CaretPos(hWnd,nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                                break;
                        case VK_DELETE:
                                if(nArrayPos==nNumChar)
                                MessageBox(hWnd,"Fin del Buffer",NULL,MB_OK);
                                else
                                {
                                for(x=nArrayPos;x<nNumChar;x=x+1)
                                cCharBuf[x]=cCharBuf[x+1];
```

```
                    nNumChar=nNumChar-1;
                    InvalidateRect(hWnd,NULL,TRUE);
                    }
                    break;
                case VK_LEFT:
                    if(nArrayPos>0)
                    {
                    nArrayPos=nArrayPos-1;
                    CaretPos(hWnd,nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                    }

                    else
                    MessageBox(hWnd,"No se puede mover a la izquierda",NULL,MB_OK);
                    break;
                case VK_RIGHT:

                    if(nArrayPos<nNumChar)
                    {
                    nArrayPos=nArrayPos+1;
                    CaretPos(hWnd,nArrayPos,cCharBuf,&xCaret,&yCaret,nCharWidth);
                    }

                    else
                    MessageBox(hWnd,"en el fin del buffer",NULL,MB_OK);
                    break;
                case VK_RETURN:
                    linea++;
            break;

                }
            break;
    case WM_PAINT:
                {
                    hDC=BeginPaint(hWnd,&PtStr);
                    TextOut(hDC,nCharWidth,nLnHeight*linea,cCharBuf,nNumChar);
                    EndPaint(hWnd,&PtStr);
                }
               break;

    case WM_DESTROY:

        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd,iMessage,wParam,lParam);
    }
    return(0L);
}
```

Figura 4-37.

4.10.2. El Ratón

El control del ratón puede estudiarse de dos formas: por control de la forma cursor por zona y por detección de un mensaje generado por uno de los botones del ratón. Las diferentes formas del ratón dentro del sistema se detallan en la tabla 4-19

Tabla 4-19.

NOMBRE DEL CURSOR	FORMA QUE ADOPTA
IDC_ARROW	Cursor en forma de Flecha (normal).
IDC_CROSS	Cursor en forma de cruz
IDC_IBEAM	Cursor en forma de de raya de texto.
IDC_ICON	Cursor vacio.
IDC_SIZE	Cursor en forma de un pequeño cuadro.
IDC_SIZENESW	Cursor apuntado hacia el noroeste y suroeste.
IDC_SIZENS	Cursor apuntado hacia el norte y sur.
IDC_SIZEWE	Cursor apuntado hacia el este y este.
IDC_UPARROW	Cursor de flecha arriba.
IDC_WAIT	Cursor en forma de reloj de arena.

Para crear nuevas formas se usa un tipo especial de recurso, el cual será detallado en el siguiente capitulo. El programador puede alterar la forma del cursor del ratón definiendo el manejador adecuado, HCURSOR, y usado la función LoadCursor. Para tales casos, el programador debe saber en que lugar se encuentra el cursor del ratón para hacer los cambios, para ello, Windows permite la comunicación continua entre el programa y periférico y usa una serie de mensajes especiales para detallar la ubicación, tabla 4-20.

Tabla 4-20.

MENSAJE	LUGAR
HTBOTTOM	Borde horizontal inferior de la ventana.
HTBOTTOMLEFT	Esquina izquierda inferior del borde de la ventana.
HTBOTTOMRIGHT	Esquina derecha inferior del borde de la ventana.
HTCAPTION	Area de escritura.
HTCLIENT	Area del Cliente.
HTERROR	Zona fuera de la ventana.

HTGROWBOX	**Botón de tamaño de la ventana.**
HTHSCROLL	**Botón deslizable horizontal.**
HTLEFT	**Borde izquierdo de la ventana.**
HTMENU	**Area del menú de la ventana.**
HTNOWHERE	**Zona entre el borde y la parte exterior de la ventana.**
HTREDUCE	**Botón de minimización.**
HTRIGHT	**Borde derecho de la ventana.**
HTSIZE	**Redimensionando la ventana.**
HTSYSMENU	**Botón de menú control.**
HTTOP	**Borde horizontal superior de la ventana.**
HTTOPLEFT	**Esquina izquierda superior.de una ventana.**
HTTOPRIGHT	**Esquina derecha superior.de una ventana.**
HTTRANSPARENT	**Zona entre una ventana y otra.**
HTVSCRORLL	**Botón deslizable vertical.**
HTZOMM	**Botón de maximización.**

Para los mensajes ubicados dentro del switch-case de WinProc; se usan, segun el caso, lo que muestra la tabla 4-21.

Tabla 4-21.

MENSAJE	REPRESENTA
WM_LBUTTONDOWN	El botón izquierdo del ratón ha sido presionado en el área de trabajo.
WM_LBUTTONUP	El botón izquierdo del ratón ha dejado de serpresionado en el área de trabajo.
WM_LBUTTONDBLCLK	El botón izquierdo del ratón ha sido presionado dos veces en el área de trabajo.
WM_BUTTONDOWN	El botón central del ratón ha sido presionado en el área de trabajo.
WM_BUTTONUP	El botón central del ratón ha dejado de ser presionado en el área de trabajo.

WM_BUTTONDBLCLK	**El botón central del ratón ha sido presionado dos veces en el área de trabajo.**
WM_MOUSEMOVE	**El ratón se ha movido en el área de trabajo.**
WM_RBUTTONDOWN	**El botón derecho del ratón ha sido presionado en el área de trabajo.**
WM_RBUTTONUP	**El botón derecho del ratón ha dejado de ser presionado en el área de trabajo.**
WM_RBUTTONDBLCLK	**El botón derecho del ratón ha sido presionado dos veces en el área de trabajo.**

Para el caso de carecer del ratón puede usarse el teclado usando teclas preprogramadas que actuen como lo botones del ratón. Los mensajes a usar en tal situación cambian, tabla 4 -22.

Tabla 4-22.

MENSAJE	REPRESENTA
MK_CONTROL	La tecla de control (Ctrl) esta siendo presionada.
MK_LBUTTON	Tecla programada para aparentar ser botón izquierdo del ratón
MK_MBUTTON	Tecla programada para aparentar ser el botón medio del ratón
MK_RBUTTON	Tecla Programada para aparentar ser el botón derecho del ratón.
MK_SHIFT	La tecla de control (Ctrl) esta siendo presionada.

El listado 4-14 es paracrear una ventana dividida, invisiblemente, en seis zonas; cuando el cursor del raton pasa de una zona a otra cambia de forma según sea el caso. La formas de cursor que se usan son las especificadas en la tabla 4-19.

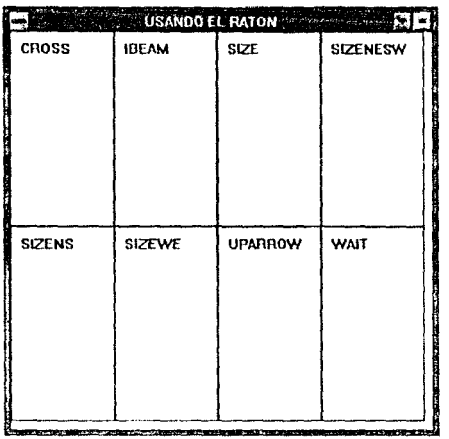


Figura 4-38.

Listado 4-38.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_RAT";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
```

```
        exit(FALSE);

    }
hWnd=CreateWindow("WIN_RAT",
                  "USANDO EL RATON",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  420,
                  440,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}


long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
PAINTSTRUCT ps;
/*HDC hDC;*/
WORD x,y;
HCURSOR hCursor;

switch(iMessage)
        {
        case WM_PAINT:
                        BeginPaint(hWnd,&ps);
                        Rectangle(ps.hdc,0,0,100,200);
                        Rectangle(ps.hdc,100,0,200,200);
                        Rectangle(ps.hdc,200,0,300,200);
                        Rectangle(ps.hdc,300,0,400,200);

                        Rectangle(ps.hdc,0,200,100,400);
                        Rectangle(ps.hdc,100,200,200,400);
                        Rectangle(ps.hdc,200,200,300,400);
                        Rectangle(ps.hdc,300,200,400,400);

                        TextOut(ps.hdc,10,10,"CROSS    ",10);
                        TextOut(ps.hdc,110,10,"IBEAM    ",10);
                        TextOut(ps.hdc,210,10,"SIZE     ",10);
                        TextOut(ps.hdc,310,10,"SIZENESW ",10);
                        TextOut(ps.hdc,10,210,"SIZENS   ",10);
                        TextOut(ps.hdc,110,210,"SIZEWE   ",10);
                        TextOut(ps.hdc,210,210,"UPARROW  ",10);
                        TextOut(ps.hdc,310,210,"WAIT     ",10);
                        TextOut(ps.hdc,410,210,"CROSS    ",10);
                        EndPaint(hWnd,&ps);
                break;
        case WM_MOUSEMOVE:
                        x=LOWORD(lParam);
                        y=HIWORD(lParam);


                        if (x>=0 && x<=400 && y>=0 && y<=500)
                          {
                          if (x>=0 && x<=100 && y>=0 && y<=200)
                            {
                            hCursor=LoadCursor(NULL,IDC_CROSS);
                            SetCursor(hCursor);
                            }
```

```
                        if (x>=100 && x<=200 && y>=0 && y<=200)
                         {
                         hCursor=LoadCursor(NULL,IDC_IBEAM);
                         SetCursor(hCursor);
                         }

                        if (x>=200 && x<=300 && y>=0 && y<=200)
                         {
                         hCursor=LoadCursor(NULL,IDC_SIZE);
                         SetCursor(hCursor);
                         }

                        if (x>=300 && x<=400 && y>=0 && y<=200)
                         {
                         hCursor=LoadCursor(NULL,IDC_SIZENESW);
                         SetCursor(hCursor);
                         }

                        if (x>=0 && x<=100 && y>=200 && y<=400)
                         {
                         hCursor=LoadCursor(NULL,IDC_SIZENS);
                         SetCursor(hCursor);
                         }

                        if (x>=100 && x<=200 && y>=200 && y<=400)
                         {
                         hCursor=LoadCursor(NULL,IDC_SIZENWSE);
                         SetCursor(hCursor);
                         }

                        if (x>=200 && x<=300 && y>=200 && y<=400)
                         {
                         hCursor=LoadCursor(NULL,IDC_SIZEWE);
                         SetCursor(hCursor);
                         }

                        if (x>=300 && x<=400 && y>=200 && y<=400)
                         {
                         hCursor=LoadCursor(NULL,IDC_UPARROW);
                         SetCursor(hCursor);
                         }

                        if (x>=400 && x<=500 && y>=0 && y<=200)
                         {
                         hCursor=LoadCursor(NULL,IDC_WAIT);
                         SetCursor(hCursor);
                         }
                        if (x>=400 && x<=500 && y>=200 && y<=400)
                         {
                         hCursor=LoadCursor(NULL,IDC_CROSS);
                         SetCursor(hCursor);
                         }
                         }
            return(0);

        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return    DefWindowProc(hWnd,iMessage,wParam,lParam);

            return(0L);
}
}
```

5

Usando Recursos

En este capítulo se continuara con el uso del WRT como parte creadora de los recursos a usar en un programa bajo Windows. Debe recordarse que el uso del WRT es sólo como parte ejemplificadora, y aunque este editor de recursos es considerado como uno de los mejores siempre podrán aparecer otros muy superiores.

5.1. Aceleradores

Como se comentó en el capítulo anterior, los aceleradores se refieren a ciertas combinacionaciones de teclas que permiten activar determinados procedimientos. Los aceleradores son manejados por grupo, y estos grupos son definidos en los archivos .RC. Por lo tanto para poder manejar uno o más grupos de aceleradores se necesita usar un programa creador y editor de recursos.

Para crear un grupo de aceleradores procedemos a teclear el botón mostrado en la figura 5-1.

Figura 5-1.

Al "oprimir" el botón aparecerá una ventana como la que se muestra en la figura 5-2.

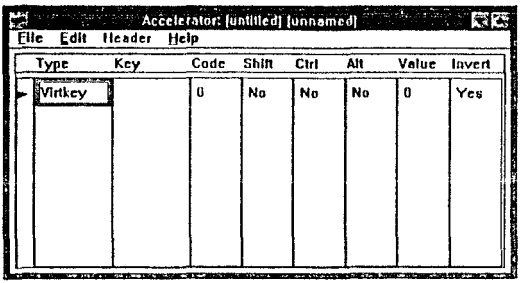


Figura 5-2.

Sobre las siete columnas se describen, de manera horizontal, las teclas o combinaciones de teclas y los valores con lo que deberán ser indentificados dentro de un programa. A continuación de explica cada columna.

Type define la forma en que se manejará el acelerador. Hay dos valores para esta columna **Virtkey** y **ASCII**. Así, por ejemplo, si pretendieramos crear un acelerador formado por las teclas Ctrl S, y si el valor fuera Virtkey se tendría S; mientras que para el caso de ASCII se tendría ^S.

Key muestra el caracter "acelerador". Para el ejemplo anterior se procede a escribir (presionar) solamente el caracter (tecla) S para Virtkey; y en caso de ASCII se procede a escribir (presionar) los caracteres (teclas) Ctrl y S para obtener ^S.

Code, en esta columna aparecerá un valor número correspondiente a la secuencia de teclas, y será por medio de este valor que dentro del programa se identifique la secuencia especificada.

Shift, en esta columna se especifica si la secuencia incluirá la tecla Shift. Solamnte se cuenta con dos valores Yes y No.

Ctrl, en esta columna se especifica si la secuencia incluirá la tecla Ctrl. Solamente se cuenta con dos valores Yes y No.

Value, en esta columna el usuario otorga un valor a la secuencia, y será por medio de este valor que dentro de programa se identifique la secuencia especificada.

Invert por medio de esta columna se otorga al área de menú el comportamiento de invertir sus colores al activar un acelerador.

Para ejemplificar el uso de los aceleradores, supongamos que pretendemos escribir un programa y que al presionar ciertas teclas se active una ventana de aviso, indicando que se ha activado un acelerador. La edición del recurso en cuestión, se mustra en la figura 5-3 la forma en que se escribe la secuencia de teclas para activar la ventana.

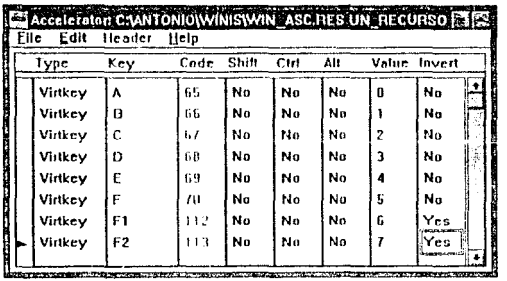


Figura 5-3.

Se puede observar en la figura 5-3 que las teclas elegidas como aceleradores son A, B, C, D, E, F, F1 y F2, siendo estas dos últimas del tipo inversor. Al salvar esta parte se genera el archivo .RC con el siguiente contenido:

```
UN_RECURSO ACCELERATORS
BEGIN
  "A", 0, VIRTKEY, NOINVERT
  "B", 1, VIRTKEY, NOINVERT
  "C", 2, VIRTKEY, NOINVERT
  "D", 3, VIRTKEY, NOINVERT
  "E", 4, VIRTKEY, NOINVERT
  "F", 5, VIRTKEY, NOINVERT
  VK_F1, 6, VIRTKEY
  VK_F2, 7, VIRTKEY
END
```

En la primera línea se puede leer UN_RECURSO, siendo esta palabra el nombre que se le a otorgado a este grupo de aceleradores. El listado 5-1 es muy semejante al listado 4-1 (ver capitulo 4), pero con algunas variaciones.

Listado 5-1.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;
HANDLE hAccel;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
```

```
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(hInstance,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_ASC";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_ASC",
                  "USANDO ACELERADORES",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);


hAccel=LoadAccelerators(hInstance,"UN_RECURSO");

while(GetMessage (&Message,NULL,0,0))
{
if (!TranslateAccelerator(hWnd,hAccel,&Message))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

char letrero[40];

switch(iMessage)
        {

        case WM_COMMAND:
                        wsprintf(letrero,"Acelerador ID=%d",wParam);
                        MessageBox(hWnd,letrero,"Acelerador",MB_OK);
                        return 0;
            break;

        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return   DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

En WinMain se ha agregado una variable con el nombre de hAccel, que es un manejador de tipo HANDLE. Por medio de hAccel se conducira el recurso acelerador dentro del programa. Posteriomente a la definición de la ventana principal aparece una nueva funcion

LoadAccelerators(), con la que se cargará al recurso acelerador específico dentro de la instancia del programa, esto se muestra en la línea descrita de la siguiente forma:

```
hAccel=LoadAccelerators(hInstance,"UN_RECURSO");
```

La *última* estructura por lo respecta a los aceleradores dentro de WinMain es una condicional para transferir, con prioridad, la señal de las secuencias de teclas marcadas en el recurso, esta es:

```
if (!TranslateAccelerator(hWnd,hAccel,&Message))
```

Para este ejemplo, se pretende que al presionar las teclas aceleradoras se despliegue una pequeña ventana indicando el valor de las teclas. Usando el mensaje WM_COMMAND se coloca la siguiente secuencia de código:

```
case WM_COMMAND:
        wsprintf(letrero,"Acelerador ID=%d",wParam);
        MessageBox(hWnd,letrero,"Aceleradar",MB_OK);
        return 0;
break;
```

La función MessageBox() es una interesante rutina con la que se puede crear una ventana con un botón de control. La sintaxis de MessageBox() se muestra continuación:

```
int MessageBox( HWND hWnd,
        LPSTR Texto,        /*Apuntador a un texto que se escribirá dentro de la
                            ventana generada por la función*/
        LPSTR Caption,      /*Apuntador a un texto que corresponderá a la
                            barra de título de la ventana generada*/
        WORD wTipoBoton)    /*Variable para generar iconos y botones funcionales
                            dentro de la ventana generada*/
```

La ventana generada por *MessageBox* entra en la categoría de caja de diálogo, por ello el argumento wTipoBoton puede generar iconos y botones. La tabla 5-1 muestra varios de los posibles valores que puede recibir este argumento, con sus respectivas funciones.

Tabla 5-1.

VALOR	FUNCION
MB_ABORTRETRYIGNORE	Generará tres botones con las inscripciones "Abort", "Retry" e "Ignore".
MB_DEFBUTTON1	Al generar alguno de los botones, el primero se podrá activar por medio de enter. El botón deberá aparecer marcado.
BM_DEFBUTTON2	Al generar alguno de los botones, el segundo se podrá activar por medio de enter. El botón deberá aparecer marcado.
BM_DEFBUTTON3	Al generar alguno de los botones, el tercero se podrá activar por medio de enter. El botón deberá

	aparecer marcado.
MB_ICONASTERISK	Aparecerá un ícono simbolizando la señal de información (una i minúscula).
MB_ICONEXCLAMATION	Aparecerá un ícono simbolizando la señal de exclamación (!).
MB_ICONHAND	Aparecerá un ícono simbolizando la señal de alto (una mano mostrando su palma).
MB_ICONINFORMATION	Aparecerá un ícono simbolizando la señal de infomación (una i minúscula).
MB_ICONQUESTION	Aparecerá un ícono simbolizando la señal de interrogación (?).
MB_ICONSTOP	Aparecerá un ícono simbolizando la señal de transito "ALTO".
MB_OK	Generará un botón con la inscripción "OK".
MB_OKCANCEL	Generará un botón con la inscripción "CANCEL".
MB_YESNO	Generará dos botones con las inscripción "YES" y "NO".
MB_YESNOCANCEL	Generará tres botones con las inscipciones "Abort", "Retry" y "Cancel".

Es posible combinar estos valores usando el operador |

MessageBox() regresa un valor en función del tipo botón que se haya "presionado", en la tabla 5-2 se muestran los nombres con los que se identifican los diferentes valores.

Tabla 5-2.

VALOR	REPRESENTA
IDABORT	Se ha presionado el botón "ABORT".
IDCANCEL	Se ha presionado el botón "CANCEL".
IDNO	Se ha presionado el botón "NO".
IDOK	Se ha presionado el botón "OK".
IDRETRY	Se ha presionado el botón "RETRY".
IDYES	Se ha presionado el botón "YES".

El resultado final del programa se puede apreciar en la figura 5-4

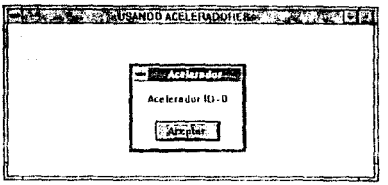


Figura 5-4.

5.2. Menús

Uno de los recursos más socorridos dentro de Windows es el de tipo menú, el cual es creado por WRT por medio del botón de la figura 5-5

Figura 5-5.

La ventana editora de menús puede ser interpretada como dos zonas; el Test Menu, donde se muestra la forma en la que se presentarán los menús y la de Menú, donde se crearán los diferentes niveles de los menús, figura 5-6.

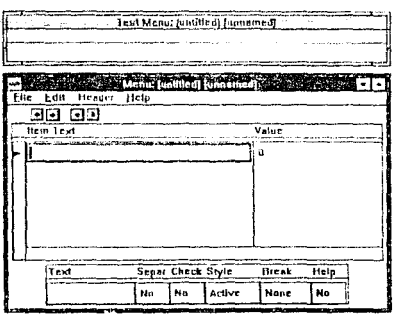


Figura 5-6.

En la parte **Menú**, el rectángulo del centro se encuentra dividido en dos zonas: **Item Text y Value**. En **Item Text** se escribirán los diferentes letreros que corresponderán al menú a desplegar, al igual que los posibles submenús que podrían desplegar cada una de las opciones. Una parte común entre los editores de menús es usar el símbolo & para indicar las letras con las que se pueden activar las opciones del menú desde el teclado. La parte **Value** sirve para darle un valor de indetificación a las diferentes opciones del texto, WRT detecta los valores iguales en los mismos niveles de jerarquía.

El rectángulo inferior es usado para especificar la forma de despliegue del letrero de una opción elegida. En la figura 5-7 se muestra un menú generado y editado por esta parte del WRT.

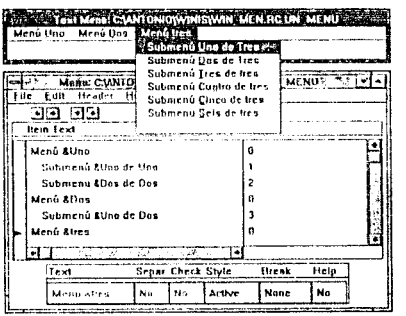


Figura 5-7.

El archivo .RC generado contiene la siguiente información:

```
UN_MENU  MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
 POPUP "Menú &Uno"
 BEGIN
  MenuItem  "Submenú &Uno de Uno", 1
  MenuItem  "Submenu &Dos de Dos", 2
 END
 POPUP "Menú &Dos"
 BEGIN
  MenuItem  "Submenú &Uno de Dos", 3
 END
 POPUP "Menú &tres"
 BEGIN
  MenuItem  "Submenú &Uno de Tres", 4
  MenuItem  "Submenú &Dos de Tres", 5
  MenuItem  "Submenú &Tres de tres", 6
  MenuItem  "Submenú Cu&atro de tres", 7
  MenuItem  "Submenú &Cinco de tres", 8
  MenuItem  "Submenu &Seis de tres", 9
  MenuItem  "", 0
 END
```

```
END
```

La primera palabra, UN_MENU, es el nombre con el que se identifica este menú dentro del archivo. Esto quiere decir que es posible tener varios recursos de menú para un mismo programa.

El listado 5-2 muestra como introducir el uso del recurso menú dentro de un programa

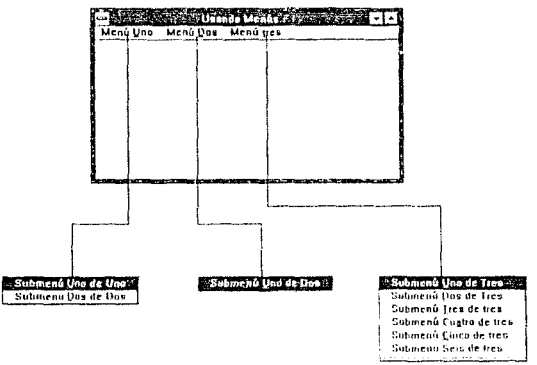
Listado 5-2.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(hInstance,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_MENU";
        WndClass.lpszMenuName="UN_MENU";
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_MENU",
                  "Usando Menus",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

HMENU hMenu;

switch(iMessage)
```

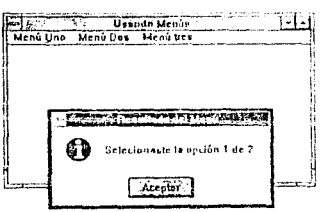
```
{
case WM_COMMAND:
                    hMenu=GetMenu(hWnd);
                    switch(wParam)
                    {
                    case 1:MessageBox(hWnd,"Selecionaste la opción 1 de 1 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;
                    case 2:MessageBox(hWnd,"Selecionaste la opción 2 de 1 ",
                                    "Resultado del Menú",
                                        MB_ICONASTERISK|MB_OK);

                                break;
                    case 3:MessageBox(hWnd,"Selecionaste la opción 1 de 2 ",
                                        "Resultado del Menu",
                                            MB_ICONASTERISK|MB_OK);
            break;
                    case 4:MessageBox(hWnd,"Selecionaste la opción 1 de 3 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;

                    case 5:MessageBox(hWnd,"Selecionaste la opción 2 de 3 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;

                    case 6:MessageBox(hWnd,"Selecionaste la opción 3 de 3 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;

                    case 7:MessageBox(hWnd,"Selecionaste la opción 4 de 3 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;
                    case 8:MessageBox(hWnd,"Selecionaste la opción 5 de 3 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;
                    case 9:MessageBox(hWnd,"Selecionaste la opción 6 de 3 ",
                                        "Resultado del Menú",
                                            MB_ICONASTERISK|MB_OK);

                                break;
                    }
        break;
    case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
    default:
                    return    DefWindowProc(hWnd,iMessage,wParam,lParam);
    }
            return(0L);
}
```

Analizando el listado anterior se tiene que en la definición de la ventana principal, en la parte WndClass.lpszMenuName, recibe el valor "UN_MENU"; es decir el nombre del recurso menú que se le asignará. Posteriormente, dentro de la función WndProc, se ha agregado un manejador de menú, HMENU, el cual servirá de enlace para recibir los diferentes mensajes de

activación de las opciones del menú. Para ello se parte de la zona correspondiente a **WM_COMMAND**, una opción switch-case dentro de esta zona será escrita para encaminar los diferentes mensajes del menú a sus correspondientes partes del programa. Por medio de la función GetMenu() se establece el correspondiente recurso de menú hacia el manejador de WndProc. Debe observarse que los mensajes pasan por medio de la variable wParam, y cada opción case usa el número con el que fueron asignadas cada una de las opciones del menu al ser creadas. La figura 5-8 muestra las diferentes opciones generadas por el menú "UN_MENU" ya dentro del programa. La figura 5-9 muestra un ejemplo de la activación de una de esas llamadas.



Listado 5-8.



Listado 5-9.

5.3. Cajas de Diálogo

Quizás el recurso más complejo y en el cual se fundamenta la interface entre un programa y un usuario sea el de las cajas de diálogo. Y, por supuesto, su editor no podía ser menos complejo, el cual se activa desde el WRT con el botón de la figura 5-10.

Figura 5-10.

Recordemos que una caja de diálogo es usa para pedir instrucciones, durante la corrida de un programa, sobre la forma de continuar el proceso que se ejecuta. Botones y zonas para escribir mensajes proporcionan al programa la información que su estructura requiere para determinar un comportamiento especial. Una caja de se diálogo clásica es, por ejemplo la que aparece para cargar un programa en los editores de palabras, figura 5-11.

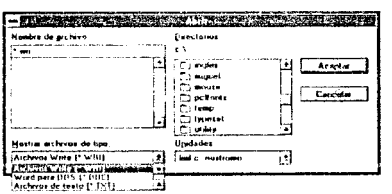


Figura 5-11.

Además de botones y zonas de textos, otros aditamentos pueden aparecer en la caja de diálogo tales como barras de control e iconos que representen botones. El editor de cajas de diálogo de WRT, figura 5-12, es muy complejo de usar.

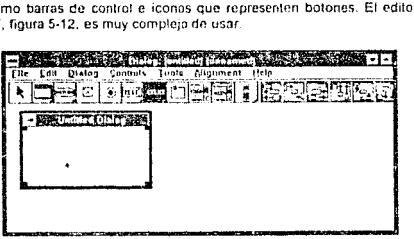


Figura 5-12.

La mayor parte de los procesos para crear una caja de diálogo se pueden elegir desde el menú de paleta, que de alguna forma resume las capacidades del editor, figura 5-13

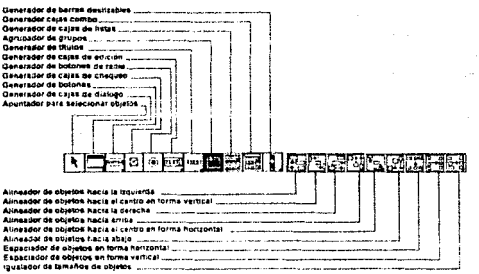


Figura 5-13.

Si observa con cuidado la figura 5-13, aquellos iconos que tienen un pequeño relieve en la esquina inferior derecha, son de opciones multiple. Para proceder a ejemplificar un el uso de recurso con caja diálogo es necesario crear un programa en cierta forma complejo. Así, el programa que se usará tendrá un menú con dos opciones, uno para invocar la caja de diálogo y otro para cancelar la ejecución del programa, figura 5-14.

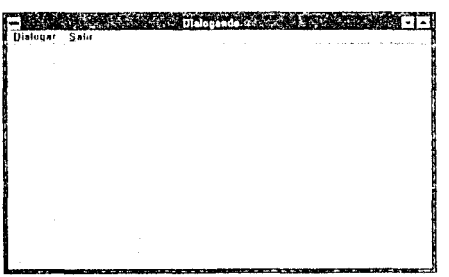


Figura 5-14.

El archivo .RC para el menú es el siguiente

```
UN_MENU  MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  Menuitem  "&Dialogar", 1
  Menuitem  "&Salir", 2
END
```

Para este ejemplo supongamos, que lo único que se haga con la caja de diálogo sea desplegar un botón para desactivar la misma caja. La edición de la caja de diálogo es entonces rápida, figura 5-15

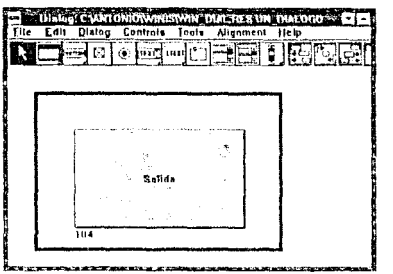


Figura 5-15.

El número 104 que aparece a un lado del botón es el de indentificación del objeto sobre la caja. El letrero "Salida" fue escrito por medio de la opción Atributes del menú Control. Desafortunadamente el WRT no genera archivo RC de las cajas de diálogo; pero para este ejemplo es sencillo ejemplificarlo de la siguiente manera

```
UN_DIALOGO DIALOG LOADONCALL MOVEABLE DISCARDABLE 10,18,139,75
STYLE WS_DLGFRAME | WS_POPUP | WS_CAPTION
FONT 10, "Helv"
BEGIN
        CONTROL "SALIDA", 104, "button",
        BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 45, 60, 36, 12
END
```

Al igual que en otros recursos, la primera palabra del archivo pertenece al nombre del recurso, para este caso el nombre es "UN_DIALOGO". Las palabras posteriores indican el tipo de recurso, DIALOG, y sus caracteristicas de aparición en la pantalla, incluyendo la posición de la caja de diálogo y el tipo de letra con la que serán desplegados los letreros.

La zona de elementos, limitada por BEGIN-END, describe cada uno de los posibles elementos dentro de la caja de diálogo. Para este caso sólo se describe un botón.

El programa del ejemplo se ha complicado en varios aspectos, listado 5-3.

Listado 5-3.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#define IDM_DIALOGAR 1
#define IDM_SALIR   2

int ghInstance;

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

ghInstance=hInstance;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(hInstance,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_DIAL";
        WndClass.lpszMenuName="UN_MENU";
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_DIAL",
                  "Dialogando",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}


BOOL FAR PASCAL ProceDialogo(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

switch(iMessage)
        {
        case WM_COMMAND:
                switch(wParam)
```

```
        {
                        case 104 :
                                EndDialog(hWnd,NULL);
                                return TRUE;
                                break;
                        }
    }
return FALSE;
}


long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

static FARPROC LlamarProcDial;

switch(iMessage)
        {
        case WM_COMMAND:
                        switch(wParam)
                        {
                         case IDM_DIALOGAR:
                                LlamarProcDial=MakeProcInstance(ProceDiálogo,ghInstance);
                                DialogBox(ghInstance,"UN_DIÁLOGO",hWnd,LlamarProcDial);
                                FreeProcInstance(LlamarProcDial);
                        break;

                         case IDM_SALIR:
                                DestroyWindow(hWnd);
                                break;
                        }
            break;
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return    DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

Iniciando con el analisis del listado se puede observar que se han definido dos constantes:

```
#define IDM_DIALOGAR 1
#define IDM_SALIR   2
```

Las cuales serán usadas para las opciones del menú. Una variable entera, **ghInstance** , ha sido definida como global, la importancia de esta variable radica en transferir el valor de instacia de la ventana principal a la ventana de diálogo, como se puede apreciar en las primera linea de WinMain

Una nueva función ha sido agregada:

```
BOOL FAR PASCAL ProceDiálogo(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
```

Esta función corresponde exclusivamente a la ventana de diálogo a invocar (por cada ventana de diálogo a usar se debe tener su respectiva función). Se puede observar que los argumentos son los mismos que los de WinProc; sin embargo su respuesta es del tipo booleano.

Dentro de la función de diálogo pueden definirse variables y estructuras necesarias para la ventana; la estructura generativa de switch-case es común:

```
switch(iMessage)
        {
        case WM_COMMAND:
                switch(wParam)
        {
                        case 104 :
                                EndDialog(hWnd,NULL);
                                return TRUE;
                                break;
                        }
    }
```

Se observa que el paso de variables hacia una caja de diálogo es primero por iMessage y después, por medio del mensaje WM_COMMAND se selecciona la opción invocada desde wParam. Desde el switch-case interno de WM_COMMAND se procede a la distribución de los mensajes de los diferentes objetos de la caja de diálogo. Puesto que para el botón de nuestro ejemplo, etiquetado con 104, solo debe finalizar el despliegue de la caja al ser "presionado" se tienen escencialmente dos líneas

```
                        case 104 :
                                EndDialog(hWnd,NULL);
                                return TRUE;
```

La función EndDialog() finaliza cualquier tipo caja de diálogo al ser invocada; para indicar el fin de la caja de diálogo, su función debe regresar el valor de TRUE (cierto). Para WinProc corresponde la parte más compleja. Al activar una caja de diálogo es como si se activara una ventana totalmente nueva; que cubre una parte de la pantalla y que debe guardarse para recuperarse posteriormente. Por todo esto, se hace la definición de un apuntador exclusivo de la caja de diálogo

```
static FARPROC LlamarProcDial;
```

Cuando se llama a una caja de diálogo se le debe asignar la memoria necesaria para poder guardar la zona sobre la que va aparecer; por ello se debe asignar la memoria para la caja, MakeProcInstance(), activar la caja con su respectivo recurso y zona de memoria, DialogBox() y (posteriormete) liberar la zona ocupada por la caja de diálogo, FreeProcInstance.

```
        case WM_COMMAND:
                        switch(wParam)
                        {
                         case IDM_DIALOGAR:
                                LlamarProcDial=MakeProcInstance(ProceDialogo,ghInstance);
                                DialogBox(ghInstance,"UN_DIÁLOGO",hWnd,LlamarProcDial);
                                FreeProcInstance(LlamarProcDial);
                        break;
```

El resultado se puede apreciar en la figura 5-16.

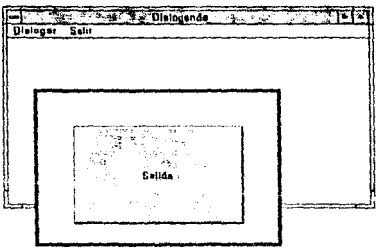


Figura 5-16.

5.4. Recursos Bitmap

Para un medio ambiente gráfico como lo es Windows hay una gran variedad de usos para los mapas de bits (bitmaps). Este tipo especial de recursos puede ser creado por medio del botón mostrado en la figura 5-17, perteneciente al WRT.

Figura 5-17.

El editor es semejante en presentación y en funcionalidad al de Iconos, figura 5-18.

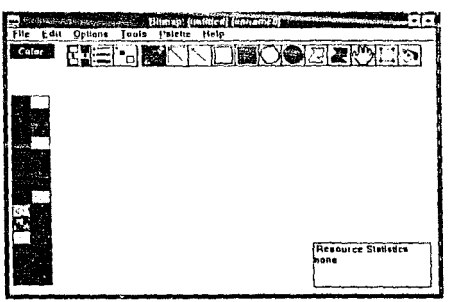


Figura 5-18.

La variación más notoria entre el editor de iconos y el de bitmaps, es el hecho de poder hacer arreglos de pixeles de un tamaño variable (el límite esta en función de la resolución de la tarjeta de video), para ejemplificar el uso de recursos del tipo bitmaps de procederá a realizar un programa que despliegue varios de ellos en el área de trabajo. Los bitmaps que se crearan representaran cuatro banderas individuales, figura 5-19, de los países México, Canada, Francia y Japón cuyos tamaños individuales serán 72x72 pixeles.

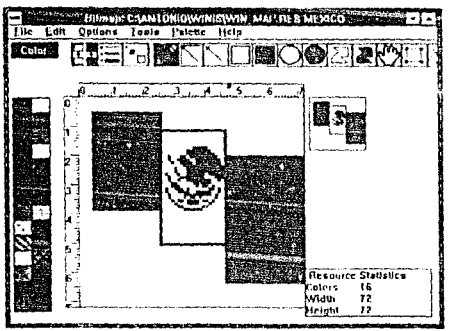


Figura 5-19.

El archivo .RC resultante es el siguiente:

```
MEXICO  BITMAP  MEXICO.BMP
CANADA BITMAP  CANADA.BMP
FRANCIA BITMAP  FRANCIA.BMP
JAPON   BITMAP  JAPON.BMP
```

Para este grupo de bitmaps no se utiliza el nombre del grupo, WIN_MAP, por que son llamados indistintamente desde el programa principal por medio de su nombre. Al final de cada línea del archivo .RC se observa la extensión .BMP, lo que significa que por cada bitmap creado se guarda su equivalente en archivo .BMP.

El archivo fuente del ejemplo se muestra en listado 5-4.

Listado 5-4.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

static HANDLE hMexico;
static HANDLE hCanada;
static HANDLE hFrancia;
static HANDLE hJapon;
```

```
long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
    {
    WNDCLASS WndClass;
    WndClass.cbClsExtra=0;
    WndClass.cbWndExtra=0;
    WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
    WndClass.hInstance=hInstance;
    WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
    WndClass.hIcon=LoadIcon(hInstance,"tubular");
        WndClass.lpfnWndProc=WndProc;
    WndClass.lpszClassName="WIN_20";
    WndClass.lpszMenuName=NULL;
    WndClass.style=CS_HREDRAW | CS_VREDRAW;

    if (!RegisterClass(&WndClass))
      exit(FALSE);

    }
hWnd=CreateWindow("WIN_20",
          "USANDO MAPAS DE BITS (BANDERAS)",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,
          0,
          CW_USEDEFAULT,
          0,
          NULL,
          NULL,
          hInstance,
          NULL);


hMexico=LoadBitmap(hInstance,"MEXICO");
hCanada=LoadBitmap(hInstance,"CANADA");
hFrancia=LoadBitmap(hInstance,"FRANCIA");
hJapon=LoadBitmap(hInstance,"JAPON");

ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC, hMemDC;
short x,y;
PAINTSTRUCT PtStr;

switch(iMessage)
    {
    case WM_PAINT:
            hDC=BeginPaint(hWnd,&PtStr);
            hMemDC=CreateCompatibleDC(hDC);
            for (y=30; y<300; y=y+80)
                {
                SelectObject(hMemDC,hMexico);
                BitBlt(hDC,30,y,72,72,hMemDC,0,0,SRCCOPY);
```

```
                SelectObject(hMemDC,hCanada);
                BitBlt(hDC,130,y,72,72,hMemDC,0,0,SRCCOPY);
                SelectObject(hMemDC,hFrancia);
                BitBlt(hDC,230,y,72,72,hMemDC,0,0,SRCCOPY);
                SelectObject(hMemDC,hJapon);
                BitBlt(hDC,330,y,72,72,hMemDC,0,0,SRCCOPY);
                }
            DeleteDC(hMemDC);
            EndPaint(hWnd,&PtStr);
            return 0;

    case WM_DESTROY:
            DeleteObject(hMexico);
            DeleteObject(hCanada);
            DeleteObject(hFrancia);
            DeleteObject(hJapon);
            PostQuitMessage(0);
            break;
    default:
            return DefWindowProc(hWnd,iMessage,wParam,lParam);
    }
        return(0L);
}
```

Dentro del programa, se han colocado cuatro manejadores, HANDLE, para cada uno de los bitmaps. Posteriomente, dentro de WinMain, por medio de la función LoadBitmap() se asignan los bitmaps a sus correspondientes manejadores.

```
hMexico=LoadBitmap(hInstance,"MEXICO");
hCanada=LoadBitmap(hInstance,"CANADA");
hFrancia=LoadBitmap(hInstance,"FRANCIA");
hJapon=LoadBitmap(hInstance,"JAPON");
```

Se debe observar la definición de un segundo manejador, hMemDC, del tipo HDC; el cual recibe un valor por medio de CreateCompatible(hDC), en la zona de WM_PAINT. Por medio de la función CreateCompatible(), hMemDC se copian las características hDC, simulando una "segunda pantalla" del dispositivo de contexto. Esto es usado para lograr una dezpliegue casi instantáneo de los bitmaps, por ello, todo el manejo relacionado de los bitmaps hacia el dispositivo de contexto tendrá como intermediario a hMemDC.

De igual forma que las plumas y brochas, para manipular un bitmap se hace uso de SelectObject():

```
        for (y=30; y<300; y=y+80)
            {
            SelectObject(hMemDC,hMexico);
            BitBlt(hDC,30,y,72,72,hMemDC,0,0,SRCCOPY);
            SelectObject(hMemDC,hCanada);
            BitBlt(hDC,130,y,72,72,hMemDC,0,0,SRCCOPY);
            SelectObject(hMemDC,hFrancia);
            BitBlt(hDC,230,y,72,72,hMemDC,0,0,SRCCOPY);
            SelectObject(hMemDC,hJapon);
            BitBlt(hDC,330,y,72,72,hMemDC,0,0,SRCCOPY);
            }
```

La función BitBlt() es la encargada de desplegar un bitmap dentro del área de trabajo; su sintáxis es la siguiente:

```
BitBlt(HDC hDC,         /*Dispositivo de Contexto que recibirá el bitmap*/
    int x,              /*Coordenada x de despliegue*/
    int y,              /*Coordenada y de despliegue*/
    int nAncho,         /*Ancho del bitmap*/
    int nLargo,         /*Largo del bitmap*/
    HDC shDC,           /*Dipositivo de Contexto auxiliar*/
```

```
int sx,          /*Coordenada x del bitmap desde donde se desplegará*/
int sy,          /*Coordenada y de bitmap desde donde se desplegará*/
DWORD dw)        /*Forma de efecto de despliegue del bitmap*/
```

El argumento que se refiere a la forma de despliegue se maneja por medio de los valores que se muestran en la tabla 5-3

Tabla 5.3.

VALOR	REPRESENTA
BLACKNESS	Oscurece el bitmap.
DSTINVERT	Invierte la zona de despliegue.
MERGECOPY	Combina el bitmap con su zona de destino usando la función booleana AND.
MERGEPAINT	Combina el bitmap con su zona de destino usando la función booleana OR.
NOTSRCCOPY	Invierte el bitmap al ser colocado en la zona de destino.
NOTSRCERASE	Invierte el resultado de hacer la operación OR entre el el bitmap y su zona de destino.
PATCOPY	Copia un patron de relleno a la zona de destino.
SRCOPY	Copia directamente un bitmap a una zona DC.
WHITENESS	Pinta de blanco la zona de destino de un bitmap.

El resultado del programa se muestra a continuación en la figura 5-20

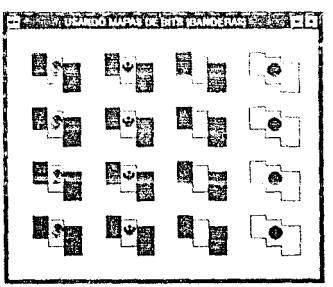


Figura 5-20.

5.5. Cursor para el Ratón

Pueden también crearse cursores para el ratón por medio de recursos. En WRT el botón que se muestra en la figura 5-21 llama a un editor para tal propósito

Figura 5-21.

Y nuevamente el editor es semejante al de iconos y bitmaps, figura 5-22

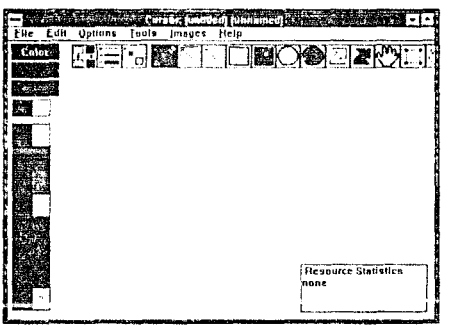


Figura 5-22.

En este caso, la variación entre los otros editores, es el hecho que nuestro dibujo sólo puede ser dibujado en negro o en blanco, el resto de los colores a usar corresponde al fondo.

Para el ejemplo de esta sección se editará un cursor en forma de nave espacial, figura 5-23.

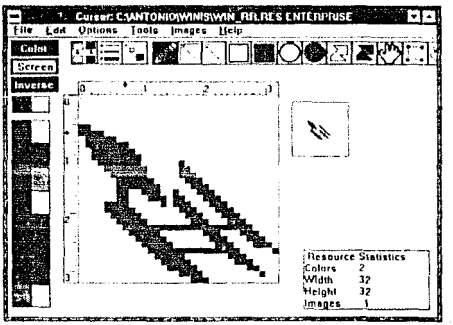


Figura 5-23.

El archivo .RC resultante es el siguiente:

```
ENTERPRISE      CURSOR ENTERPRI.CUR
```

El nombre que recibe el recurso es "ENTERPRISE", y al igual que los iconos y bitmaps también se guarda su imágen dentro de un archivo con extensión .CUR

El archivo fuente del programa se muestra en el listado 5-5

Listado 5-5.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>


long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
          LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
    {
    WNDCLASS WndClass;
    WndClass.cbClsExtra=0;
    WndClass.cbWndExtra=0;
    WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
    WndClass.hInstance=hInstance;
    WndClass.hCursor=LoadCursor(hInstance,"ENTERPRISE");
    WndClass.hIcon=LoadIcon(hInstance,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
    WndClass.lpszClassName="WIN_RR";
    WndClass.lpszMenuName=NULL;
```

```
    WndClass.style=CS_HREDRAW | CS_VREDRAW;

    if (!RegisterClass(&WndClass))
      exit(FALSE);

    }
hWnd=CreateWindow("WIN_RR",
                  "VENTANA PARA UN RECURSO DEL CURSOR DE RATON",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,
          0,
          CW_USEDEFAULT,
          0,
          NULL,
          NULL,
          hInstance,
          NULL);


ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{


switch(iMessage)
    {


    case WM_DESTROY:

            PostQuitMessage(0);
            break;
    default:
            return DefWindowProc(hWnd,iMessage,wParam,lParam);
    }
      return(0L);
}
```

Niguna variación interesante aparece en el listado 5-5, salvo por la inserción del nombre del recursos cursor en la caracterización correspondiente de la ventana principal:

```
WndClass.hCursor=LoadCursor(hInstance,"ENTERPRISE");
```

El resultado del programa aparece en la figura 5-23.

Figura 5-24.

6

Programación Avanzada

6.1. El manejo de la Memoria

Independientemente de las ventajas de los diferentes modelos de memoria con los que puede ser diseñado un programa para Windows, puede hacer uso de un grupo de funciones especialmente diseñadas para interactuar con la zona del segmento del heap y más alla de los otros segmentos reglamentarios del programa.

Como se trato en el capitulo dos, a la zona del heap se le dió el nombre de ***segmento de datos no inicializados*** y por conveniencia la idealizamos como si se encontrara en la parte alta del esquema de un programa en memoria, figura 6.1.

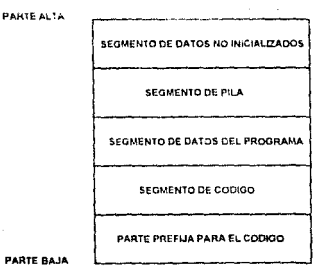


Figura 6.1.

Para Windows, el heap dentro de este esquema es llamado **heap local**, y que es usado para establecer variables dinámica y, si hay suficiente espacio, bibliotecas dinámicas. Cuando el programador cree que su programa deberá hacer uso de más memoria que la existe en el heap local, puede accesar otra zona de memoria totalmente fuera del área que usa el programa; esta área recibe el nombre de **heap global** y es administrada directamente por Windows.

El heap global puede estar dentro o fuera de la llamada barrera de lo 640 Kbyte, y más allá de un mega de memoria. Esta es una gran ventaja para el programador, pues técnicamente resulta transparente si Windows se encuentra trabajando en el modo protegido. Las funciones que puede utilizar un programador para accesar los heap's local y global son muy semejantes a las funciones alloc() y malloc() de C, aunque existe una característica muy importante ha tomar en cuenta: las funciones dedicadas al heap local regresan direcciones de 16 bits, mientras que las del heap global regresan direcciones de 32 bits. Las tablas 6.1 y 6.2 muestran las diferentes funciones para los heap's local y global respectivamente.

Tabla 6.1

FUNCION	USO
LocalAlloc	Direcciona memoria del heap local.
LocalCompac	Reorganiza el heap local.
LocalDiscard	Descarta una área sin bloquear.
LocalFlags	Da información de un objeto en memoria (dato).
LocalFree	Libera una área del heap local.
LocalHandle	Le entrega al manejador de un objeto la dirección de una área del heap local.
LocalInit	Inicializa el heap local.
LocalLock	Bloquea el acceso a una área específica del heap local.
LocalReAlloc	Cambia el tamaño de la zona de memoria de un dato.
LocalShrink	Reorganiza y reduce el tamaño del heap local (lo más posible), si la función tiene éxito, esta permanecerá; en caso contrario los datos pasan al heap global.
LocalSize	Da el tamaño del heap local.
LocalUnlock	Desbloquea el acceso a una área del heap local.

Tabla 6.2

FUNCION	USO
GlobalAlloc	Direcciona memoria del heap global.
GlobalCompac	Reorganiza el heap global.
GlobalDiscard	Descarta un área sin bloquear.
GlobalFlags	Da información de un objeto en memoria (dato).

GlobalFree	**Libera una área del heap local.**
GlobalHandle	**Le entrega al manejador de un objeto la dirección de una área del heap global.**
GlobalLock	**Bloquea el acceso a una área especifica del heap global.**
GlobalReAlloc	**Cambia el tamaño de la zona de memoria de un dato.**
GlobalSize	**Da el tamaño de una zona del heap global.**
GlobalUnlock	**Desbloquea el acceso a una área del heap global.**
GetFreeSpace	**Da el tamaño de la memoria global.**
GlobalDosAlloc	**Direcciona la parte libre de la memoria usada por el DOS.**
GlobalDosFree	**Libera la parte de la memoria direccionada por GlobalDosAlloc.**
GlobalFix	**Causa que una área del heap global permanezca en forma lineal con el resto de la memoria.**
GlobalUnFix	**Acción contraria de GlobalFix.**
GlobalLRUNewest	**Mueve información hacia una nueva zona del heap global.**
GlobalLRUOldest	**Mueve información hacia la zona más usada del heap global.**
GlobalNotify	**Avisa que ha sido ubicada una biblioteca**

Cuando en las funiones del heap global se refieren a "área", se debe entender que en realidad son segmentos. Por otra parte en las funciones, tanto del heap local como del global, que aparecen los prefijos Discable, Fix, Lock y RLU, se refieren a un cierto tipo de acciones que ocurren en los heap para mantenerlos libres y de un buen tamaño. Por ejemplo, supongamos que se tiene en cualquiera de los heap tres tipos de información distinta, figura 6 2

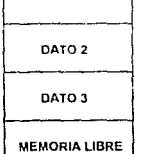


Figura 6.2.

Por alguna causa en especial, el DATO 2 es eliminado, quedando la situación de la figura 6.3

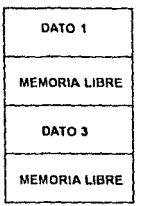


Figura 6.3.

Al aplicar, según el caso, alguna de la fuciones "Fix", la situación cambia a lo que muestra la figura 6.4.

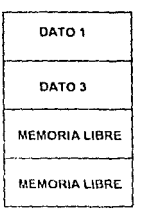


Figura 6.4.

Las situaciones mostradas en las figuras anteriores pueden ocurrir aunque el programador no haya puesto las funciones; esto ocurre debido a la forma en que fueron especificados los segmentos de datos y de código en el archivo de definición del programa.

6.2. Usando el Clipboard

Usar el clipboard es muy semejante a usar alguno de los heap; sin embargo, presenta ventajas muy notables y más avanzadas. El espacio del clipboard siempre esta presente y apesar de ser parte del heap global jamás es afectado por alguna de las acciones que ahí ocurran. Las funciones SetClipboardData() y GetClipboardData son usadas para enviar y obtener, respectivamente, información para el clipboard. Otras funciones, OpenClipboard (abrir), EmptyClipboard (vaciar) y CloseClipboard (cerrar), son usadas durante el envio y recepción de información.

El clipboard es capaz de contener una gran variedad de datos, sin embargo, los formatos de información a guardar son limitados, aunque cubren la mayor parte de los que se usan en la programación y paquetes de Microsoft. En la tabla 6-3 se establecen el nombre de las constantes que pueden ser usadas para indicarle al Clipboard el tipo de información que contendra

Tabla 6.3

VALOR CF_BITMAP	FUNCION MAPA DE BITS
CF_DIB	MAPA DE BITS PROPORCIONADO POR UNA ESTRUCTURA DE DATOS
CF_DIF	FORMATO DE IMAGENES USADO POR LA COMPAÑIA SOFTWARE ARTS.
CF_DISPBITMAP	MAPA DE BIT DE UN FORMATO ESPECIAL
CF_DSPMETAFILEPICT	METARCHIVO
CF_DISPTEXT	INFORMACION TEXTUAL
CF_OEMTEXT	INFORMACION TEXTUAL CON CARACTERES OEM.
CF_OWNERDISPLAY	INFORMACION PROPORCIONADO EN PARTES POR EL MISMO CLIPBOARD.
CF_PALETE	ESTRUTURA DE UNA PALETA DE COLORES.
CF_SYLK	FORMATO SYLK (SYMBOLIC LINK) DE MICROSOFT.
CF_TEXT	INFORMACION TEXTUAL, CADA LINEA TERMINA CON UNA PAREJA DE CTRL-LF
CF_TIFF	IMAGENES TIFF.

El programa clipboard (o visor del portapapeles, en la versión en español) que es proporcionado por Microsoft dentro del ambiente Windows, es semejante a una ventana que permite al usuario "ver" lo contenido en el clipboard, esta es su única función.

Es posible crear un programa que sustituya las funciones del clipboard del sistema. En la figura 6-5 se muestra una ventana que "imita" al clipboard.

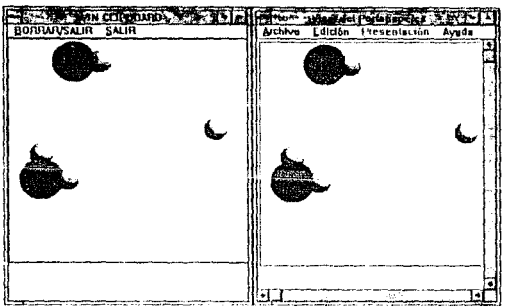


Figura 6-5.

En la parte derecha de la figura 6-5 aparece la ventana del clipboard (visor de portapapeles) que muestra una imágen transferida a la zona de memoria. En el lado izquierdo de la figura, encontramos la ventana del programa WIN_CLIPBOARD, diseñado para funcionar de manera semejante al clipboard. Sin embargo, WIN_CLIPBOARD posee la caracteristica de inhibir el funcionamiento del programa clipboard para demostrar su eficaz funcionamiento. En la figura 6-6 se muestra un nuevo estado de ambos programas.

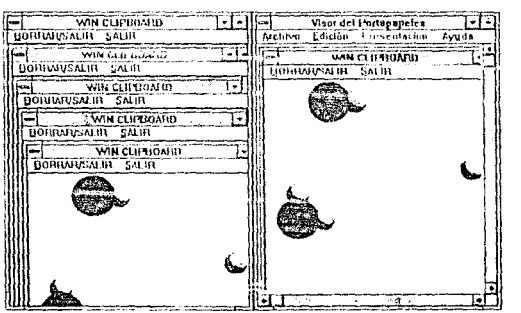


Figura 6-6.

Es bien sabido por los usuarios de Windows que se puede copiar la imágen de toda la pantalla a la zona clipboard usando la tecla "PrnScr" (ImpPant). Cuando se hace esto, teniendo

la ventana del clipboard abierta, se puede observar el paso directo de la imágen a la zona de trabajo. Si se presiona varias veces la tecla, durante la corrida del programa ejemplo, el efecto en la zona de trabajo será el de imágenes anidadas. Así, en la figura 6-6, aparecen en la ventana de WIN_CLIPBOARD, varias pantallas anidadas, mientras que en la ventana del clipboard su zona de trabajo esta igual que en la figura 5-5. Una parte importante de WIN_CLIPBOARD es regresar el "comando" de visualización al clipboard cuando termina su ejecución; de no ser así el usuario no podra hacer uso del clipboard, por lo menos durante la presente sesión.

El listado 6-1 muestra el programa WIN_CLIPBOARD, el cual se basó en un ejemplo sobre el tema presentado por James L. Conger en su libro "Windows API Bible", ver bibliografia.

Listado 6-1.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#define IDM_HACER 1
#define IDM_SALIR 2

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_CL";
        WndClass.lpszMenuName="WIN_MENU";
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_CL",
                  "WIN CLIPBOARD",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
```

```
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

PAINTSTRUCT         ps;
HDC                 hMemDC;
HBITMAP             hBitmap;
BITMAP              bm;
RECT                rClientRect;
HANDLE              hMem;
LPSTR               lpMem;
static HWND         hNextViewer;

switch(iMessage)
{
        case WM_CREATE:
                                hNextViewer=SetClipboardViewer(hWnd);
                                break;
        case WM_PAINT:
                                BeginPaint(hWnd,&ps);
                                GetClientRect(hWnd, &rClientRect);
                                OpenClipboard(hWnd);
                                if (hMem=GetClipboardData(CF_TEXT))
                                {
                                        lpMem=GlobalLock(hMem);
                                        DrawText(ps.hdc,lpMem,-1,&rClientRect,DT_LEFT);
                                        GlobalUnlock(hMem);
                                }
                                else if (hBitmap=GetClipboardData(CF_BITMAP))
                                {
                                        hMemDC=CreateCompatibleDC(ps.hdc);
                                        SelectObject(hMemDC,hBitmap);
                                        GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bm);
                                        BitBlt(ps.hdc,0,0,bm.bmWidth,
                                        bm.bmHeight,hMemDC,0,0,SRCCOPY);
                                        DeleteDC(hMemDC);
                                }
                                CloseClipboard();
                                EndPaint(hWnd,&ps);
                                break;
        case WM_DRAWCLIPBOARD:
                                if (hNextViewer)
                                SendMessage(hNextViewer,WM_DRAWCLIPBOARD,
                                wParam,lParam);
                                InvalidateRect(hWnd,NULL,TRUE);
                                break;
        case WM_CHANGECBCHAIN:
                                if (wParam==hNextViewer)
                                        hNextViewer=LOWORD(lParam);
                                else if (hNextViewer)
                                        SendMessage(hNextViewer,WM_CHANGECBCHAIN,
                                                wParam,lParam);
                                break;
        case WM_COMMAND:
                                switch(wParam)
                                {
                                case IDM_HACER:
                                        OpenClipboard(hWnd);
                                        EmptyClipboard();
                                        CloseClipboard();
                                        InvalidateRect(hWnd,NULL,TRUE);
                                        DestroyWindow(hWnd);
                                        break;
                                case IDM_SALIR:
                                        DestroyWindow(hWnd);
                                        break;
                                }
                                break;
```

```
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return  DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

El archivo de recursos se muestra continuación:

```
WIN_MENU        MENU
BEGIN
        MENUITEM        "&BORRAR/SALIR",1
        MENUITEM        "&SALIR",       2
END
```

Dentro del programa, se han agregado dos cosntantes, IDM_HACER e IDM_SALIR, para facilitar el uso de los valores de identificación del menú. Este mismo, el menú WIN_MEN es asignado a la ventana principal en la línea de procedimiento WndMain:

```
WndClass.lpszMenuName="WIN_MENU";
```

En WndProc, en la zona de WM_CREATE se utiliza la función SetClipboardViewer() para ligar al programa con el clipboard. Esta liga solo es en aspecto de "ver" (lectura). Obsérvese que el valor de regreso es a un manejador con formato "estático". El ligamiento causa un mensaje hacia el programa para "preguntar el cómo se deberá llevar a cabo este encadenamiento", es decir, especificar si el programa clipboard continuará siendo usado para ver el contenido. La respuesta a este mensaje es otorgado por la zona WM_CHANGECBCHAIN:

```
case WM_CHANGECBCHAIN:
                        if (wParam==hNextViewer)
                                hNextViewer=LOWORD(lParam);
                        else if (hNextViewer)
                                SendMessage(hNextViewer,WM_CHANGECBCHAIN,
                                        wParam,lParam);
                        break;
```

Dentro de la zona WM_PAINT, responsable directa de mostrar el contenido del clipboard, se establecen primero los límites de la zona de trabajo.

```
                        GetClientRect(hWnd, &rClientRect);
```

A continuación se abre y se lee el contenido del clipboard, el cual será analizado para determinar su forma de despliegue (texto o imágen)

```
                        OpenClipboard(hWnd);
                        if (hMem=GetClipboardData(CF_TEXT))
                        {
                                lpMem=GlobalLock(hMem);
                                DrawText(ps.hdc,lpMem,-1,&rClientRect,DT_LEFT);
                                GlobalUnlock(hMem);
                        }
                        else if (hBitmap=GetClipboardData(CF_BITMAP))
                        {
                                hMemDC=CreateCompatibleDC(ps.hdc);
                                SelectObject(hMemDC,hBitmap);
                                GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bm);
                                BitBlt(ps.hdc,0,0,bm.bmWidth,
                                bm.bmHeight,hMemDC,0,0,SRCCOPY);
                                DeleteDC(hMemDC));
```

Después del despliegue, se procede a cerrar la llamada al clipboard con la siguiente secuencia de intrucciones:

```
CloseClipboard();
EndPaint(hWnd,&ps);
break;
```

6.3. Aspectos del MS-DOS en Windows

Cuando Windows despliega su ventana de diálogo para guardar y salvar archivos, o invoca procesos relacionados, debe hacer uso de la parte encargada de administrar los archivos dentro del MS-DOS.

En compiladores de C tales como Turbo C, existen una gran cantidad de funciones para el manejo de archivos, directorios y llamadas especiales al BIOS, para poder lograr efectos como el de la figura 6-7.

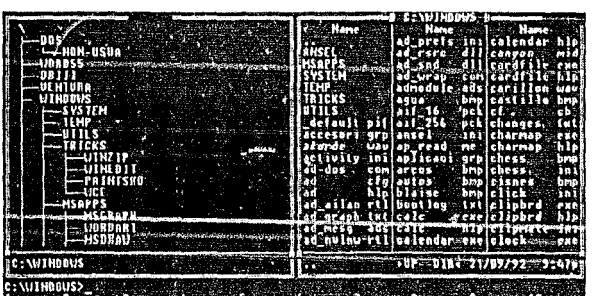


Figura 6-7.

Las funciones son variadas de un compilador a otro, y cuando se trata de programar sobre algun sistema UNIX, muchos aspectos cambian en lo que respecta al BIOS. Tomando en cuenta todo esto, Microsoft dotó a la biblioteca Windows.h de una nueva serie de funciones de entrada y salida de archivos. Dentro de estas nuevas fuciones se consideron la forma de trabajo de las cajas de diálogo para la carga y descarga de los archivos controlada al antojo del usuario en Windows. Las funciones y sus significados se enlistan en la tabla 6-4.

Tabla 6-4.

FUNCION	HACE
DlgDirList	Crea una lista de archivos dentro de una caja de diálogo.
DlgDirListComboBox	Crea una lista de archivos controlada por una caja combo, dentro de una caja de diálogo

DlgDirSelect	Entrega el archivo que se seleccione por medio de una lista de archivos.
DlgDirSelectComboBox	Entrega el archivo que se seleccione por medio de una caja combo.
GetDOSEnvironment	Obtiene una cadena de caracteres sobre la dirección del DOS.
GetDriveType	Obtiene el tipo de unidad de disco que se esta direccionando.
GetEnviroment	Regresa una cadena de caracteres sobre la dirección general de la unidad de disco.
GetPrivateProfileInt	Regresa el número de aplicaciones dentro de un archivo .INI.
GetPrivateProfileString	Regresa una cadena de caracteres de alguna aplicación contenida dentro de un archivo .INI.
GetSystemDirectory	Regresa la ruta de ubicación del subdirectorio SYS de Windows.
GetTempDrive	Determina cual de las unidades de discos contiene (o contendrá) los archivos temporales.
GetTempFileName	Crea un directorio para colocar los archivos temporales.
GetWindowsDirectory	Regresa la ruta de ubicación del subdirectorio Windows.
_lclose	Cierra un achivo.
_lcreat	Crea un nuevo archivo.
_llseek	Permite moverse dentro de un archivo.
_lopen	Abre un archivo para escribir o leer información.
_lread	Lee información de archivo.
_lwrite	Escribe información.
OpenFile	Crea, abre o borra archivos.
SetEnviroment	Cambia la configuración de un puerto.
SetErrorMode	Establece códigos de errores en el manejo de archivos.
SetHandleCount	Cambia el número de archivo que puede abrir una aplicación.
WritePrivateProfileString	Escribe una cadena de caracteres dentro de un archivo .INI.

WriteProfileString	**Escribe una cadena de caracteres dentro del un archivo WIN.INI.**

Las funciones relacionadas con las cajas de diálogo usan los valores de la tabla 6-5 para abrir, cerrar y destruir archivos.

Tabla 6-5.

VALOR	SIGNIFICA
0x0000	Obtener archivos de lectura y escritura
0x0001	Obtener archivos de escritura solamente
0x0002	Obtener archivos de lectura solamente.
0x0004	Obtener archivos aocultos.
0x0010	Obtener subdirectorios
0x0020	Obtener archivos en el directorio.
0x2000	Dirección a travez de la bandera LB_DIR, de barra de deslizable.
0x4000	Obtener unidades de disco.
0x8000	Uso del bit de exclusividad para recobrar atributos de un archivo.

Condiderando que algunos compiladores permiten el uso de estructuras especiales para manejar archivos, Microsoft también creó una estructura interna para el manejo de caractensticas de archivos

```
typedef struct tagOFSTRUCT
{
BYTE cByte;              /* Longitud de la estructura*/
BYTE fFixedDisk;         /*Tipo de unidad de disco, 0 si es disco duro*/
WORD nErrCode;           /*Código error al MS-DOS*/
BYTE reserved[4];        /*Fecha y tiempo de un archivo*/
BYTE szPathName[128];    /*Ruta de ubicación y nombre de l archivo*/
} OFSTRUCT
```

El manejo de esta estructura puede llevarse por medio de apuntadores normales, cercano y lejano

```
Typedef OFSTRUCT        *POFSTRUCT;
Typedef OFSTRUCT NEAR   *NPOFSTRUCT;
Typedef OFSTRUCT FAR    *LPOFSTRUCT;
```

En una primera forma de aproximación a las funciones que manejan archivos se analizara el programa WIN_DOS, cuyos diferentes estados se muestran en las figura 6-8, 6-9 y 6-10.

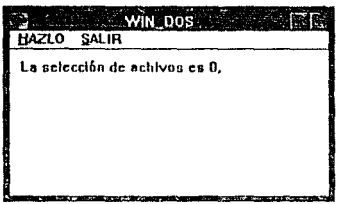


Figura 6-8.

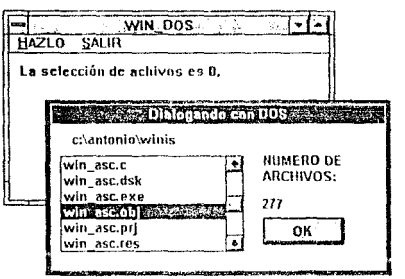


Figura 6-9.

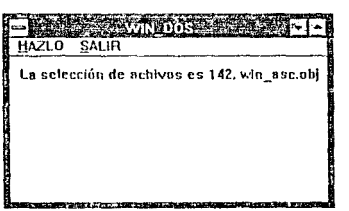


Figura 6-10.

Win_DOS permite invocar una caja de diálogo por medio de la opción de menú HAZLO. Al invocar la caja diálogo, esta presenta una lista de archivos del directorio concurrente en disco

duro durante la ejecución del programa. La búsqueda del archivo deseado puede llevarse a cabo por medio de una barra deslizable asociada a la caja de la lista; por otra parte, la misma caja nos da el número de archivos del directorio y un botón para indicar que se ha seleccionado el archivo deseado, figura 6-9.

Al presionar el botón, la caja de diálogo desaparece, y en el área de trabajo de la ventana principal, se escribe la leyenda **"La selección de archivo es 142, win_asc.obj"**, figura 6-10. Esto significa que fue seleccionado el archivo 142 de la lista, cuyo nombre es win_asc.obj. La lista es desplegada en forma alfabética, y de esa forma el proceso que crea la lista le asigna una valor. El listado 6-2 muestra el programa WIN_DOS.

Listado 6-2.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#define IDM_HACER 1
#define IDM_SALIR 2

#define DLL_LISTBOX 104
#define DLL_NUMFILES 103
#define DLL_DIRSTRING 102
#define DLL_OK 101

int nSelection=0;
char cSelection[128];
int ghInstance;
long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{

HWND hWnd;
MSG Message;

ghInstance=hInstance;

if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_DOS";
        WndClass.lpszMenuName="WIN_MENU";
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_DOS",
                  "WIN_DOS",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
```

```
                    hInstance,
                    NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

BOOL FAR PASCAL ProceDialogo(HWND hDlg, WORD iMessage, WORD wParam, LONG lParam)
{
int nFiles;

switch(iMessage)
        {
        case WM_INITDIALOG:
                DlgDirList(hDlg,"*.*",DLI_LISTBOX,DLI_DIRSTRING,0);
                nFiles=SendDlgItemMessage(hDlg,DLI_LISTBOX,LB_GETCOUNT,0,0L);
                SetDlgItemInt(hDlg,DLI_NUMFILES, nFiles,TRUE);
        return TRUE;
        case WM_COMMAND:
                switch(wParam)
                        {
                        case DLI_OK:
                                EndDialog(hDlg,0);
                                return TRUE;
                        case DLI_LISTBOX:
                                if (HIWORD (lParam)==LBN_SELCHANGE)
                                {
                                nSelection=SendDlgItemMessage(hDlg,
                                DLI_LISTBOX,LB_GETCURSEL,0,0L);
                                DlgDirSelect(hDlg,(LPSTR)cSelection,
                                DLI_LISTBOX);
                                }
                        }
                        return TRUE;
                        case WM_DESTROY:
                                EndDialog(hDlg,NULL);
                return TRUE;
                                break;
                        }


return FALSE;
}


long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

HDC hDC;
FARPROC         lpfnDlgProc;
PAINTSTRUCT ps;
char cBuf[128];

/*HMENU hMenu;*/
switch(iMessage)
        {
        case WM_PAINT:
                BeginPaint(hWnd,&ps);
                TextOut(ps.hdc,10,10,cBuf,wsprintf(cBuf,
                "La selección de achivos es %d, %s%",
                nSelection,(LPSTR) cSelection));
        EndPaint(hWnd,&ps);
        case WM_COMMAND:
                /*hMenu=GetMenu(hWnd);*/
```

```
                switch(wParam)
                        {
                        case IDM_HACER:
                                        lpfnDlgProc=MakeProcInstance(ProceDialogo,ghInstance);
                                        DialogBox(ghInstance,"DIALOGDOS",hWnd,lpfnDlgProc);
                                        FreeProcInstance(lpfnDlgProc);
                    InvalidateRect(hWnd,NULL,TRUE);
                                        break;
                        case IDM_SALIR:
                        DestroyWindow(hWnd);
                                        break;

                        }
                break;<
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return  DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

Los recursos que forman parte del archivo .RES se muestran en el listado 6-3.

Listado 6-3.

```
WIN_MENU MENU
BEGIN
        MENUITEM        "&HAZLO",       1
        MENUITEM        "&SALIR",       2
END

DIALOGDOS DIALOG LOADONCALL MOVEABLE DISCARDABLE 20,36,162,75
CAPTION "Dialogando con DOS"
FONT 10, "Arial"
STYLE WS_BORDER|WS_CAPTION|WS_DLGFRAME|DS_MODALFRAME|WS_POPUP
BEGIN
        CONTROL "OK",101,"button",BS_DEFPUSHBUTTON|WS_TABSTOP|
        WS_CHILD,102,48,40,14
        CONTROL "","104,"listbox",
        LBS_STANDARD | LBS_HASSTRINGS | WS_BORDER | WS_VSCROLL| WS_CHILD,
        5,17,87,49
        CONTROL "NUMERO DE ARCHIVOS:",-1,"static",SS_LEFT | WS_CHILD,
        102,16,54,10
        CONTROL "",102,"static",SS_LEFT | WS_CHILD,
        10,4,141,12
        CONTROL "",103,"static",SS_LEFT | WS_CHILD,
        102,37,41,10
END
```

Dentro del listado 6-2 encontramos la adición de una serie de constantes para facilitar su uso y colocación en las partes para encaminar los mensajes del menú y de la caja de diálogo:

```
#define IDM_HACER 1
#define IDM_SALIR 2
#define DLI_LISTBOX 104
#define DLI_NUMFILES 103
#define DLI_DIRSTRING 102
#define DLI_OK 101
```

A continuación, se definen tres variables globales:

```
int nSeleccion=0;
char cSeleccion[128];
```

```
int ghInstance;
```

nSelection guardará el número del archivo que se seleccione en la caja de diálogo; cSelection conservará el nombre del archivo seleccionado; ghInstance, desde un inicio, copiará el valor de la variable hInstance, para procurar que se conserve el direccionamiento adecuado entre la ventana principal y la amalgama Windows/DOS.

Así, dentro de WinMain se llevan a cabo las siguientes asignaciones:

```
ghInstance=hInstance;

WndClass.lpszClassName="WIN_DOS";
WndClass.lpszMenuName="WIN_MENU";
```

Estas dos últimas, para ligar los recursos al programa. Para WndProc, la estructura se encamina a controlar una caja de diálogo:

```
switch(iMessage)
        {
        case WM_PAINT:
                BeginPaint(hWnd,&ps);
                TextOut(ps.hdc,10,10,cBuf,wsprintf(cBuf,
                "La selección de archivos es %d, %s%",
                nSelection,(LPSTR) cSelection));
                EndPaint(hWnd,&ps);
        case WM_COMMAND:
                switch(wParam)
                        {
                        case IDM_HACER:
                                        lpfnDlgProc=MakeProcInstance(ProceDialogo,ghInstance);
                                        DialogBox(ghInstance,"DIALOGDOS",hWnd,lpfnDlgProc);
                                        FreeProcInstance(lpfnDlgProc);
                                        InvalidateRect(hWnd,NULL,TRUE);
                                        break;
                        case IDM_SALIR:
                                        DestroyWindow(hWnd);
                                        break;
                        }
        break;
```

Una parte interesante, es la manera en que se lleva a cabo el despliegue de la leyenda "La selección de archivos ...", se ha usado la función TextOut(), pero combinada con la función wsprintf(), que similar a la del C normal, pero que al regresar la cadena del formato indicado, es desplegada por la función que la contiene por medio de la variable cBuf. Dentro de wsprintf() se encuentran las variables nSelection y cSelection, esta última como un apuntador.

Por lo que se puede apreciar, la responsabilidad de los archivos recae directamente dentro del procedimiento de la caja de diálogo.

```
BOOL FAR PASCAL ProceDialogo(HWND hDlg, WORD iMessage, WORD wParam, LONG lParam)
{
int nFiles;

switch(iMessage)
        {
        case WM_INITDIALOG:
                DlgDirList(hDlg,"*.*",DLI_LISTBOX,DLI_DIRSTRING,0);
                nFiles=SendDlgItemMessage(hDlg,DLI_LISTBOX,LB_GETCOUNT,0,0L);
                SetDlgItemInt(hDlg,DLI_NUMFILES, nFiles,TRUE);
        return TRUE;
        case WM_COMMAND:
                switch(wParam)
                        {
```

```
        case DLI_OK:
                EndDialog(hDlg,0);
                return TRUE;
        case DLI_LISTBOX:
                if (HIWORD (lParam)==LBN_SELCHANGE)
                {
                nSelection=SendDlgItemMessage(hDlg,
                DLI_LISTBOX,LB_GETCURSEL,0,0L);
                DlgDirSelect(hDlg,(LPSTR)cSelection,DLI_LISTBOX);
                }
        }
        return TRUE;
        case WM_DESTROY:
                EndDialog(hDlg,NULL);
                return TRUE;
                break;
        }

return FALSE;
}
```

Por medio de DlgDirList(hDlg,"*.*",DLI_LISTBOX,DLI_DIRSTRING,0), se establecen las condiciones de la lista de archivos a desplegar. La función SendDlgItemMessage() es la reponsable directa de "contestar" cuantos archivos de han listado; el valor es asignado a nFiles. La función SetDlgItemInt() despliega en lugar fijado por el recurso el número de archivos listados.

Dentro de la zona WM_COMMAND, la opción:

```
        case DLI_LISTBOX:
                if (HIWORD (lParam)==LBN_SELCHANGE)
                {
                nSelection=SendDlgItemMessage(hDlg,
                DLI_LISTBOX,LB_GETCURSEL,0,0L);
                DlgDirSelect(hDlg,(LPSTR)cSelection,DLI_LISTBOX);
                }
```

Establece lo que se debe hacer cuando se selecciona un archivo.

6.4. Interface de Documentos Múltiples

Un programa que usa una interface de documentos múltiples (MDI) es aquel que permite manejar varias ventanas de aplicaciones contenidas dentro de otra que funciona como supervisora. Un ejemplo de un programa con MDI, de la gran variedad que existe en Windows, es el Administrador de archivos, figura 6-11.

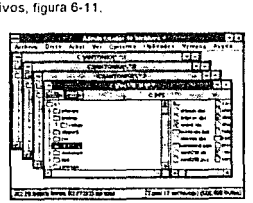


Figura 6-11.

Las ventanas dentro de la ventana supervisora reciben el nombre de ventanas hijas. Las ventanas hijas pueden ser reducidas a íconos o aumetadas hasta ocupar todo el área de trabajo de la ventana supervisora. Desde la ventana supervisora el usuario puede acomodar las ventanas, aplicar algún proceso o destruirlas todas. Aunque parezca que todas la ventanas estan activas, sólo una es suceptible de recibir los procesos del ménu de la ventana supervisora.

Escribir un programa para Windows que use MDI no es tan complejo como pudiera parecer. En la figura 6-12 se muestra un esquema básico del control de ventanas hijas.

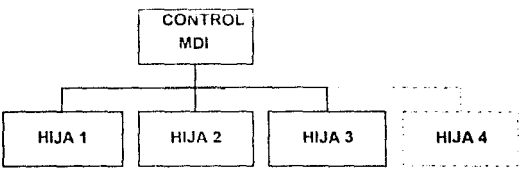


Figura 6-12.

La parte control MDI supervisa las entradas y salidas de los mensajes de las diferentes ventanas, igualmente el control es único y puede crear y destruir las ventanas hijas. Dentro de la literatura sobre el tema, es común llamar a la ventana supervisora como "frame" (marco), por conveniencia sólo adoptaremos el término "frame" para variables relacionadas con la ventana supervisora. El mismo control MDI posee un grupo de mensajes especiales para comunicarse hacia el despachador. La tabla 6-6 muestra los mensajes hacia el control MDI.

Tabla 6-6.

MENSAJE	SIGNIFICADO
WM_MDIACTIVATE	Activa una ventana hija.
WM_MDICASCADE	Arreglar las ventanas hijas en forma de cascada.
WM_MDIDESTROY	Destruir la ventana hija activa.
WM_MDIGETACTIVE	Obtiene el manejador de la ventana hija activa .
WM_MDIICONARRANGE	Arreglar los íconos de las ventanas hijas.
WM_MDIMAXIMIZE	Maximizar en tamaño la ventana hija activa.
WM_MDINEXT	Activar la siguiente ventana, según el orden de creación.
WM_MDIRESTORE	Reducir una ventana a su icono.
WM_MDISETMENU	Alterar el menú de la ventana supervisora.
WM_MDITILE	Arreglar las ventanas hijas en forma de mosaicos.

Escribir un programa que maneje MDI no es realmente difícil, pero puede resultar un poco confuso por la gran cantidad de manejadores que se usan para una tarea de este estilo. Para empezar, se deben de declarar tanto las características de la ventana supervisora, como la característica (generales) de la ventanas hijas; esto se hace dentro del procedimiento WndMain con la variable de tipo WNDCLASS:

```
WNDCLASS WndClass;

if (!hPrevInstance)
        {

        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=FrameWndProc;
        WndClass.lpszClassName="WIN_MDI_FRAME";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=sizeof(LOCALHANDLE);
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=DocWndProc;
        WndClass.lpszClassName="WIN_MDI_DOC";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
```

Se puede observar en el fragmento de listado anterior, que la variable WndClass se caracteriza dos veces, resaltando diferencias en las partes lpfnWndProc y lpszClasName. Cuando el compilador se "topa" con esta redundancia, "entiende" que el programador trata de establecer un MDI; así, la primera caracterización corresponde a la ventana supervisora, la siguiente caracterización establece la caracterización de todas las ventanas hijas. En la parte lpfnWndProc se establece el programa de procesos (WndProc) respectivo de cada ventana, al igual que nombre de clase con la parte lpszClasName. En la segunda caracterización se puede observar:

```
WndClass.cbWndExtra=sizeof(LOCALHANDLE);
```

Lo cual significa que los datos almacenados en cada ventana pasarán a una zona reservada del heap local, para cada ventana los datos serán exclusivos de ella. Algunas veces, se podría tener la necesidad de programar ventanas hijas con características diferentes; un ejemplo concreto de esto es el paquete Works de Microsoft, en el cual se permite manejar ventanas hijas del tipo edición de textos como también lo puede hacer con hojas de cálculo. Para este caso se caracterizaran tantas veces la variable de tipo WNDCLASS como tipos de ventanas hijas se pretenda tener.

Como se decía anteriormente, uno de los problemas al programar y manejar MDI es la cantidad de manejadores; pues se recomienda crear tantos manejadores para la ventana supervisora como para el grupo ventanas hijas. Así, el manejador de tipo HWND que por lo general se maneja uno solo (para la ventana de procesos) de forma global, tendrá un compañero correspondiente a la área de trabajo donde aparece el MDI, como ejemplo podríamos tener:

```
HWND ghWndFrame,ghWndClient; /*manejadores globales*/
```

La canalización de los mensajes puede variar, tan sólo se agrega una función especial para separar los mensajes de la ventana supervisora y las ventana hijas; en la parte correspondiente a la atención de los mensajes WndMain tenemos ahora lo siguiente:

```
while(GetMessage (&Message,NULL,0,0))
{
        if (!TranslateMDISysAccel(ghWndClient,&Message))
        {
                TranslateMessage(&Message);
                DispatchMessage(&Message);
        }
}
```

La función TranslateMDISysAccel() es la encargada de la separación, obsérvese que uno de sus argumentos es el manejador del área de trabajo (HWND ghWndClient).

La creación de la ventana supervisora es por medio de la función CreateWindow, como normalmente se hace; la creación de las ventanas hijas se lleva a cabo en la función que se definió en la caracterización de la ventana supervisora, para el ejemplo que estamos manejando el nombre de la función es FrameWndProc. Dentro de la función se definirán dos manejadores locales, cuyos valores esten ligados a los manejadores globales ghWndFrame y ghWndClient (ejemplo). Un tercer manejador del mismo tipo será definido como local, el cual nos permitirá activar las ventanas hijas dentro del MDI. A este manejador lo llamaremos hWndChlid. Dos variables más serán necesarias para la creación de las ventana hijas usando las estructuras CLIENTCREATESTRUCT y MDICREAESTRUCT. La primera estructura permite ligar el nombre de la ventana hija con el menú, mientras la segunda estructura permite definir las características generales de la ventanas hijas. Ambas estructuras son extremadamente complejas en su relación con otras funciones, por lo cual es recomendable revisar el manual del SDK para una mejor comprensión.

La preparación del área de trabajo para el MDI se realiza en la parte del mensaje WM_CREATE usando la función CreateWindow, como ejemplo tenemos:

```
ghWndClient=CreateWindow("MDICLIENT","NULL",
                        WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
                        0,0,600,400,hWnd,NULL,ghInstance,(LPSTR) &mdi);
```

El manejador ghInstance, corresponde a una variable de tipo global que posee el valor de manejador hInstance definido en WndMain. El argumento mdi es la variable definida por medio de estructura MDICREAESTRUCT. Posteriomente se establecerá una "vía" de comunicación entre la zona de trabajo y el despachador; esto se hace usando una función SendMessage.

La definición de las ventanas hijas se hace dentro de la zona del mensaje WM_COMMAND, dentro de una zona switch-case de selección de valores de wParam. Esto se debe a que cada ventana hija debe ser creada por medio de un comando del menú de la ventana supervisora o por secuencia de teclas. Las ventanas hijas son creadas a través de una

estructura general y enrutadas por medio de la función SendMessage; a continuación se muestra un ejemplo:

```
case WM_COMMAND:

switch(wParam)
        {
        case 10:
                mdi.szClass="WIN_MDI_DOC";
                mdi.szTitle="Documento";
                mdi.hOwner=ghInstance;
                mdi.x=CW_USERDEFAULT;
                mdi.y=CW_USERDEFAULT;
                mdi.cx=CW_USERDEFAULT;
                mdi.cy=CW_USERDEFAULT;
                mdi.style=0;
                mdi.lParam=NULL;

                hWndChild=SendMessage(hWndClient,WM_MDICREATE,0,
                (LONG)(LPMDICREATESTRUCT)&mdi);
                return 0;
```

El listado y figura ejemplifican un programa que usa de MDI, dicho programa se basó en un ejemplo presentado en el libro "Windows API Bible" de James L Conger (ver bibliografía).

Listado 6-3.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#define IDM_MDILIST         1
#define IDM_SALIR           2
#define IDM_NUEVORECT       3
#define IDM_NUEVOELIP       4
#define IDM_CASCADA         5
#define IDM_ARREGLO         6
#define IDM_CIERRAR         7
#define IDM_TOPE            8
#define IDM_MOSAICO         9
#define IDM_ROJO            10
#define IDM_VERDE           11
#define IDM_AZUL            12
#define IDM_SIGUIENTE       13
#define VENTANA_HIJA        100

int ghInstance;
HWND ghWndFrame,ghWndClient;


long FAR PASCAL FrameWndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);
long FAR PASCAL ElipProc(HWND hChild, WORD iMessage, WORD wParam, LONG lParam);
long FAR PASCAL RectProc(HWND hChild, WORD iMessage, WORD wParam, LONG lParam);
void SetFrameMenu (void);
void SetChildMenu (void);
BOOL FAR PASCAL EnumChildDestroy(HWND hWndChild,DWORD lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HANDLE hAccel;
HWND hWnd;
```

```
MSG Message;
WNDCLASS WndClass;
HMENU hMenu;

ghinstance=hinstance;
hMenu=LoadMenu(hinstance,"FRAMEMENU");

if (!hPrevInstance)
        {
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=FrameWndProc;
        WndClass.lpszClassName="WIN_MDI";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);
        }

ghWndFrame=CreateWindow("WIN_MDI",
                "WINDOWS MDI",
                WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                600,
                400,
                NULL,
                LoadMenu(ghInstance,"FRAMEMENU"),
                hInstance,
                NULL);

ShowWindow(ghWndFrame,nCmdShow);
DrawMenuBar(ghWndFrame);
UpdateWindow(ghWndFrame);
hAccelTLoadAccelerators(hInstance,"ACCEL");

while(GetMessage (&Message,NULL,0,0))
{
        if (!TranslateMDISysAccel(ghWndClient,&Message) &&
           !TranslateAccelerator(ghWndFrame,hAccel,&Message))
        {
                TranslateMessage(&Message);
                DispatchMessage(&Message);
        }
}
return Message.wParam;
}


long FAR PASCAL FrameWndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HWND                    hChild;
CLIENTCREATESTRUCT      ccs;
MDICREATESTRUCT         mcs;
WNDCLASS                WndClass;
HMENU                   hMenu;
FARPROC                 lpEnumFunc;

switch(iMessage)
        {
    case WM_CREATE:
                        ccs.idFirstChild=VENTANA_HIJA;
                        hMenu= LoadMenu(ghInstance,"FRAMEMENU");
```

```
                ccs.hWindowMenu=GetSubMenu(hMenu,0);
                ghWndCliente=CreateWindow("MDICLIENT","NULL",
                WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
                0,0,600,400,hWnd,NULL,ghInstance,(LPSTR) &ccs);
                SendMessage(hWnd,WM_MDISETMENU,0,
                        MAKELONG(ccs.hWindowMenu,hMenu));

                WndClass.style=CS_HREDRAW | CS_VREDRAW;
                WndClass.cbClsExtra=0;
                WndClass.cbWndExtra=sizeof(GLOBALHANDLE);
                WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
                WndClass.hInstance=ghInstance;
                WndClass.hCursor=LoadCursor(NULL,IDC_CROSS);
                WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
                WndClass.lpszMenuName=NULL;

                WndClass.lpfnWndProc=ElipProc;
                WndClass.lpszClassName="Elipse";
                RegisterClass(&WndClass);

                WndClass.lpfnWndProc=RectProc;
                WndClass.lpszClassName="Rectangulo";
                RegisterClass(&WndClass);
                return(0);
        case WM_COMMAND:
                switch(wParam)
                        {
                        case IDM_NUEVORECT:
                                mcs.szClass="Rectangulo";
                                mcs.szTitle="Rectangulo";
                                mcs.hOwner=ghInstance;
                                mcs.x=CW_USEDEFAULT;
                                mcs.y=CW_USEDEFAULT;
                                mcs.cx=CW_USEDEFAULT;
                                mcs.cy=CW_USEDEFAULT;
                                mcs.style=0;
                                mcs.lParam=NULL;
                                SendMessage(ghWndCliente,WM_MDICREATE,0,
                                        (LONG) (LPMDICREATESTRUCT) &mcs);
                                break;
                        case IDM_NUEVOELIP:
                                mcs.szClass="Elipse";
                                mcs.szTitle="Elipse";
                                mcs.hOwner=ghInstance;
                                mcs.x=CW_USEDEFAULT;
                                mcs.y=CW_USEDEFAULT;
                                mcs.cx=CW_USEDEFAULT;
                                mcs.cy=CW_USEDEFAULT;
                                mcs.style=0;
                                mcs.lParam=NULL;
                                SendMessage(ghWndCliente,WM_MDICREATE,0,
                                        (LONG) (LPMDICREATESTRUCT) &mcs);
                                break;
                        case IDM_CIERRAR:
                                hChild=LOWORD(SendMessage(ghWndCliente,
                                                WM_MDIGETACTIVE,0,0L));
                                SendMessage(ghWndCliente,WM_MDIDESTROY,hChild,0L);
                                break;
                        case IDM_ARREGLO:
                                SendMessage(ghWndCliente,WM_MDIICONARRANGE,0,0L);
                                break;
                        case IDM_CASCADA:
                                SendMessage(ghWndCliente,WM_MDICASCADE,0,0L);
                                break;
                        case IDM_MOSAICO:
                                SendMessage(ghWndCliente,WM_MDITILE,0,0L);
                                break;
                        case IDM_SIGUIENTE:
                                SendMessage(ghWndCliente,WM_MDINEXT,0,0L);
```

```
                        break;
                case IDM_SALIR:
                        lpEnumFunc=MakeProcInstance(EnumChildDestroy,
                                        ghInstance);
                        EnumChildWindows(ghWndClient,lpEnumFunc,0L);
                        FreeProcInstance(lpEnumFunc);
                        DestroyWindow(hWnd);
                        break;
                default:
                        hChild=LOWORD(SendMessage(ghWndClient,
                                WM_MDIGETACTIVE,0,0L));
                        if (IsWindow(hChild))
                          SendMessage(hChild,
                                WM_COMMAND,wParam,lParam);
                        break;
                }
                break;
        case WM_DESTROY:
                        hMenu=FindResource(ghInstance,"FRAMEMENU",RT_MENU);
                        hMenu=LoadResource(ghInstance,hMenu);
                        if (hMenu)
                                while(FreeResource(hMenu));
                        hMenu=FindResource(ghInstance,"CHILDMENU",RT_MENU);
                        hMenu=LoadResource(ghInstance,hMenu);
                        if (hMenu)
                                while(FreeResource(hMenu));
                        PostQuitMessage(0);
                        break;
        default:
                        return  DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
return(0L);
}


long FAR PASCAL ElipProc(HWND hChild, WORD iMessage, WORD wParam, LONG lParam)
{

RECT            rClient;
PAINTSTRUCT     ps;
GLOBALHANDLE    hMem;
LPSTR           lpMem;
HBRUSH          hBrush;


switch(iMessage)
        {
        case WM_CREATE:
                hMem=GlobalAlloc(GHND,sizeof(DWORD));
                SetWindowWord(hChild,0,hMem);
                break;
        case WM_MDIACTIVATE:
                if (wParam)
                        SetChildMenu();
                else
                        SetFrameMenu();
                DrawMenuBar(ghWndFrame);
                return(0);
        case WM_PAINT:
                GetClientRect(hChild,&rClient);
                hMem=GetWindowWord(hChild,0);
                lpMem=GlobalLock(hMem);
                BeginPaint(hChild,&ps);
                hBrush=CreateSolidBrush(RGB(*(lpMem+1),*(lpMem+2) ,*(lpMem+3)));
                GlobalUnlock(hMem);
                SelectObject(ps.hdc,hBrush);
                Ellipse(ps.hdc,10,10,rClient.right-10,rClient.bottom-10);
                DeleteObject(hBrush);
                EndPaint(hChild,&ps);
```

```
                return(0);
        case WM_COMMAND:
                switch(wParam)
                        {
                        case IDM_ROJO:
                        case IDM_AZUL:
                        case IDM_VERDE:

                                hMem=GetWindowWord(hChild,0);
                                lpMem=GlobalLock(hMem);
                                *(lpMem+1)=(wParam==IDM_ROJO ? 255:0);
                                *(lpMem+2)=(wParam==IDM_AZUL ? 255:0);
                                *(lpMem+3)=(wParam==IDM_VERDE ? 255:0);
                                GlobalUnlock(hMem);
                                InvalidateRect(hChild,NULL,TRUE);

                        case IDM_TOPE:
                SetFrameMenu();
                                break;

                        }
                return(0);

        case WM_DESTROY:
                hMem=GetWindowWord(hChild,0);
                GlobalFree(hMem);
                return(0);

        }
return  DefMDIChildProc(hChild,iMessage,wParam,lParam);
}

long FAR PASCAL RectProc(HWND hChild, WORD iMessage, WORD wParam, LONG lParam)
{

RECT            rClient;
PAINTSTRUCT     ps;
GLOBALHANDLE    hMem;
LPSTR           lpMem;
HBRUSH          hBrush;


switch(iMessage)
        {
        case WM_CREATE:
                hMem=GlobalAlloc(GHND,sizeof(DWORD));
                SetWindowWord(hChild,0,hMem);
                break;
        case WM_MDIACTIVATE:
                if (wParam)
                        SetChildMenu();
                else
                        SetFrameMenu();
                DrawMenuBar(ghWndFrame);
                return(0);
        case WM_PAINT:
                GetClientRect(hChild,&rClient);
                hMem=GetWindowWord(hChild,0);
                lpMem=GlobalLock(hMem);
                BeginPaint(hChild,&ps);
        hBrush=CreateSolidBrush(RGB(*(lpMem+1),*(lpMem+2) ,*(lpMem+3)));
                GlobalUnlock(hMem);
                SelectObject(ps.hdc,hBrush);
                Rectangle(ps.hdc,10,10,rClient.right-10,rClient.bottom-10);
                DeleteObject(hBrush);
                EndPaint(hChild,&ps);
                return(0);

        case WM_COMMAND:
```

```
                switch(wParam)
                        {
                        case IDM_ROJO:
                        case IDM_AZUL:
                        case IDM_VERDE:

                                hMem=GetWindowWord(hChild,0);
                                lpMem=GlobalLock(hMem);
                                *(lpMem+1)=(wParam==IDM_ROJO ? 255:0);
                                *(lpMem+2)=(wParam==IDM_AZUL ? 255:0);
                                *(lpMem+3)=(wParam==IDM_VERDE ? 255:0);
                                GlobalUnlock(hMem);
                                InvalidateRect(hChild,NULL,TRUE);

                        case 8:
                SetFrameMenu();
                                break;

                        }
                return(0);
        break;
        case WM_DESTROY:
                hMem=GetWindowWord(hChild,0);
                GlobalFree(hMem);
                return(0);

        }
return  DefMDIChildProc(hChild,lMessage,wParam,lParam);
}


void SetFrameMenu (void)
{
static HMENU hMenu=NULL;
HMENU   hSubMenu;
int     nMenuItems;

if (!hMenu)
        hMenu=LoadMenu(ghInstance,"FRAMEMENU");
nMenuItems=GetMenuItemCount(hMenu);
hSubMenu=GetSubMenu(hMenu,nMenuItems-1);
SendMessage(ghWndClient,WM_MDISETMENU,0,MAKELONG(hMenu,hSubMenu));
DrawMenuBar(ghWndFrame);
}

void SetChildMenu (void)
{
static HMENU hMenu=NULL;
HMENU   hSubMenu;
int     nMenuItems;

if (!hMenu)
        hMenu=LoadMenu(ghInstance,"CHILDMENU");
nMenuItems=GetMenuItemCount(hMenu);
hSubMenu=GetSubMenu(hMenu,nMenuItems-1);
SendMessage(ghWndClient,WM_MDISETMENU,0,MAKELONG(hMenu,hSubMenu));
DrawMenuBar(ghWndFrame);
}

BOOL FAR PASCAL EnumChildDestroy(HWND hWndChild,DWORD lParam)
{
SendMessage(ghWndClient,WM_MDIDESTROY,hWndChild,0L);
return(TRUE);
}
```

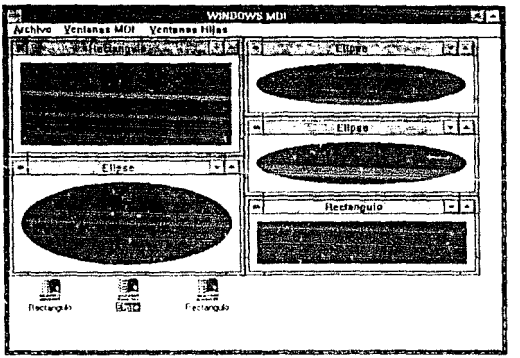

WINDOWS MDI
Archivo Ventanas MDI Ventanas Hijas
Elipse
Elipse
Elipse
Rectangulo
Rectangulo
Elipse
Rectangulo

Figura 6-12.

7

Ejemplos Prácticos

7.1. El Clásico Programa "Hola Mundo".

Un programa que se ha vuelto clásico para los programadores que se inician en el lenguaje C, es el de desplegar la cadena de caracteres "Hola Mundo". Para hacerlo, un programador de lenguaje C, puede llevarse una secuencia básica de una docena de palabras. Sin embargo para realizar esto en Windows, se debe seguir la relación WndMain-WndProc.

7.1.1. Primera Aproximación

Usando la función TextOut() se puede realizar el despliegue de la cadena. El listado 7.1 muestra un programa que genera la salida de la figura 7.1

Listado 7.1.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;



if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
```

```
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_H1";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_H1",
                  "Un Clásico Programa",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
switch(iMessage)
        {
        case WM_PAINT:
                        hDC=GetDC(hWnd);
                        TextOut(hDC,10,10,"Hola, Mundo",11);
                        ReleaseDC(hWnd,hDC);
             break;
        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return    DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

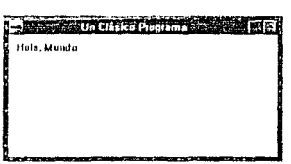


Figura 7.1

Debe considerarse que se ha procedido a usar un manejador del DC (hDC); lo cual significará que solo se pintará una vez. Como contraparte pudo haberse usado una estructura PAINTSTRUCT, entonces la zona del mensaje WN_PAINT se escribiría de la siguiente forma:

```
PAINTSTRUCT ps;
.
.
.
case WM_PAINT:
                BeginPaint(hWnd,&ps);
                TextOut(hDC,10,10,"Hola, Mundo",11);
                EndPaint(hWnd,&ps)
                break;
```

La diferencia en este caso es que se continuará efectuando la el proceso de dibujo en intervalos de tiempo, pero la salida seguirá siendo la misma.

7.1.1. Segunda Aproximación

No conformes con una simple salida de caracteres, podemos realizar un segundo programa que maneje un tipo de letra que haga el despliegue de la cadena más vistoso. Para ello procederemos a usar un manejador de tipos (HFONT) y la función CreateFont(), cuyos argumentos nos permiten configurar a nuestras necesidades un tipo de letra. el listado 7.2 obtiene un programa con la salida de la figura 7.2

Listado 7.2.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;



if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_H1";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_H1",
                  "Un Clásico Programa",
```

```
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT,
                    0,
                    CW_USEDEFAULT,
                    0,
                    NULL,
                    NULL,
                    hInstance,
                    NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
HFONT hTipo
switch(iMessage)
        {
        case WM_PAINT:
                    hDC=GetDC(hWnd);
                    hTipo=Create(20,20,0,0,400,0,0,0,
                                OEM_CHARSET,OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                                DEFAULT_PITCH|FF_SCRIP,"script");
                    TextOut(hDC,10,10,"Hola, Mundo",11);
                    ReleaseDC(hWnd,hDC);
                    DeleteObject(hTipo);
            break;
        case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
        default:
                    return    DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
                return(0L);
}
```

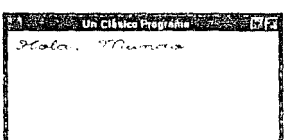


Figura 7.2.

7.1.2. Variaciones Sobre el Mismo Tema

Para un proceso tan simple como el hecho de dezplegar una cadena, se puede construir una especie de muestrario de tipos y estilos de presentación de letras. Para ello, estableceremos que la forma en que se hagan los cambios será por medio de un menú semejante al que usan los editores de texto tipicos de Windows. Nuestro menú deberá mostrar los tipos de letras disponibles para cambiar la cadena de caracteres, las diferentes formas de presentación

(negrita, itálica o subrayada). También, pondríamos el tamaño de los caracteres. El menú del programa podría quedar estructurado como se muestra en la figura 7.3.

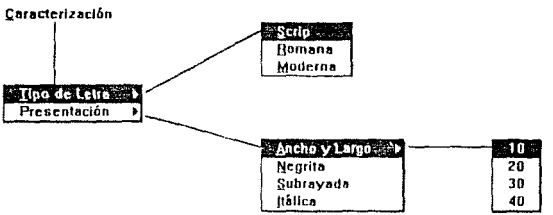


Figura 7-3.

El menú de la figura 7-3 se activa por medio de la palabra "caracterización", y permite llegar hasta tres niveles de menús verticales. El archivo .RC quedaría de la siguiente forma.

```
UN_MENU  MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
 POPUP "&Caracterización"
 BEGIN
  POPUP "&Tipo de Letra"
  BEGIN
   MenuItem  "&Scrip", 11
   MenuItem  "&Romana", 12
   MenuItem  "&Moderna", 13
  END
  POPUP "Presentación"
  BEGIN
   POPUP "&Ancho y Largo"
   BEGIN
    MenuItem  "10", 10
    MenuItem  "20", 20
    MenuItem  "30", 30
    MenuItem  "40", 40
   END
   MenuItem  "&Negrita", 22
   MenuItem  "&Subrayada", 23
   MenuItem  "&Itálica", 24
  END
 END
END
```

Como se observa en listado anterior, cada una de las partes del menú (MenuItem) posee un número de control; se debe recordar que el usar menús o secuencias de teclas dentro de un programa implica el uso de la zona WM_COMMAND, la cual transfiere el número de control de las partes del menú hacia una estructura switch-case por medio de wParam. Esta parte tendría un esqueleto inicial de la siguiente forma:

```
case WM_COMMAND:
        switch(wParam)
        {
        case 11:
                /*SELECCION PARA TIPO DE LETRA SCRIP*/
                break;
```

```
                case 12:
                        /*SELECCION PARA TIPO DE LETRA ROMAN*/
                        break;

                case 13:
                        /*SELECCION PARA TIPO DE LETRA MODERN*/
                        break;
                case 10:
                        /*SELECCION PARA TAMAÑO DE LETRA DE 10 PIXELES*/
                        break;

                case 20:
                        /*SELECCION PARA TAMAÑO DE LETRA DE 20 PIXELES*/
                        break;
                case 30:
                        /*SELECCION PARA TAMAÑO DE LETRA DE 30 PIXELES*/
                        break;

                case 40:
                        /*SELECCION PARA TAMAÑO DE LETRA DE 40 PIXELES*/
                        break;

                case 22:
                        /*SELECCION DE FORMA NEGRITA*/
                        break;
                case 23:
                        /*SELECCION DE FORMA SUBRAYADO*/
                        break;
                case 24:
                        /*SELECCION DE FORMA ITALICA*/
                        break;
                }
        break;
```

Por medio de los "case" se alterará algún valor de los parámetros usados dentro de la función CreateFont. Posteriomente, dentro de la misma zona de WM_COMMAND, se procederá a "pintar" el cambio. Para ello se usará la misma secuencia de instrucciones que en WM_PAINT, pero usando las funciones GetDC() y RealeaseDC() por ser interactivas. El siguiente fragmento muestra dicha secuencia:

```
hDC=GetDC(hWnd);
hFont=CreateFont(iAncho,iLargo,0,0,iNegrita,
        iItalica,iSubraya,0,
        OEM_CHARSET,OUT_DEFAULT_PRECIS,
        CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
        bTipoLetra,cTipoLetra);
SelectObject(hDC,hFont);
TextOut(hDC,10,10,"Hola, Mundo",11);
ReleaseDC(hWnd,hDC);
DeleteObject(hFont);
```

Obsérvese que en el fragmento anterior de listado se han colocado los nombres de las variables para configurar el tipo de letra en CreateFont. Estos valores pueden ser declarados como locales dentro de WinProc.

El programa en general es mostrado en el listado 7.3; un posible resultado se muestra en la figura 7.3.

Listado 7.3.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;



if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_H4";
        WndClass.lpszMenuName="UN_MENU";
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);

        }
hWnd=CreateWindow("WIN_H4",
                  "Un Clásico Programa",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  0,
                  CW_USEDEFAULT,
                  0,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
HFONT hFont;
PAINTSTRUCT ps;

BYTE SCRIP=DEFAULT_PITCH|FF_SCRIPT;
char Scrip[]="script";

BYTE ROMAN=DEFAULT_PITCH|FF_SWISS;
char Roman[]="roman";
```

```
BYTE MODERN=DEFAULT_PITCH|FF_MODERN;
char Modern[]="modern";

BYTE bTipoLetra=SCRIP;
char *cTipoLetra=Scrip;

int iAncho=10;
int iLargo=10;
int iNegrita=400;
int iItalica=0;
int iSubraya=0;

switch(iMessage)
        {
        case WM_PAINT:
                        BeginPaint(hWnd,&ps);
                        hFont=CreateFont(iAncho,iLargo,0,0,iNegrita,
                                        iItalica,iSubraya,0,
                                        OEM_CHARSET,OUT_DEFAULT_PRECIS,
                                        CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                                        bTipoLetra,cTipoLetra);
                SelectObject(ps.hdc,hFont);
                        TextOut(ps.hdc,10,10,"Hola, Mundo",11);

                        EndPaint(hWnd,&ps);
                        DeleteObject(hFont);
                        break;
        case WM_COMMAND:
                        switch(wParam)
                                {
                                case 11:
                                        bTipoLetra=SCRIP;
                                        cTipoLetra=Scrip;
                                        break;
                                case 12:
                                        bTipoLetra=ROMAN;
                                        cTipoLetra=Roman;
                                        break;

                            case 13:
                                        bTipoLetra=MODERN;
                                        cTipoLetra=Modern;
                                        break;
                                case 10:
                                    iAncho=10;
                                    iLargo=10;
                                        break;

                                case 20:
                                    iAncho=20;
                                    iLargo=20;
                                        break;
                                case 30:
                                    iAncho=30;
                                    iLargo=30;
                                        break;

                                case 40:
                                    iAncho=40;
                                    iLargo=40;
                                        break;

                                case 22:
                    if (iNegrita==400) iNegrita=700;
                                else iNegrita=400;
                                        break;
                                case 23:
                                        if (iSubraya==0) iSubraya=1;
                        else iSubraya=0;
```

```
                        break;
                case 24:
                        if (italica==0) italica=1;
                                else italica=0;
                        break;
                }
                hDC=GetDC(hWnd);
                hFont=CreateFont(tAncho,iLargo,0,0,iNegrita,
                        italica,iSubraya,0,
                        OEM_CHARSET,OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                        bTipoLetra,cTipoLetra);
        SelectObject(hDC,hFont);
                TextOut(hDC,10,10,"Hola, Mundo",11);

                ReleaseDC(hWnd,hDC);
                DeleteObject(hFont);

    break;
case WM_DESTROY:
        PostQuitMessage(0);
        break;
default:
        return   DefWindowProc(hWnd,iMessage,wParam,lParam);
}
        return(0L);
}
```

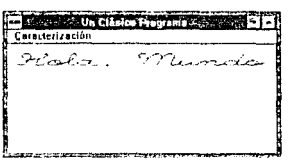


Figura 7.3.

7.2. Un Programa Educativo

Los programas educativos son aquellos que permiten la enseñanza a diferentes niveles. En la actualidad los programas educativos se apoyan en multimedia para lograr un mejor ambiente y relación entre el usuario y el programa.

7.2.1. Temas y Limitaciones

Para esta parte se pretende crear un programa que muestre una serie de imágenes con una reseña. Los programas educativos y de tendencia a multimedia diseñados en DOS, son creados para desplegar imágenes contenidas en disco. Este método es el más común, pero se topa con dos inconvenientes:

* Capacidad de imágenes
* Tiempo de despliegue de una imágen.

Una imágen altamente compleja, que sobrepase los 256 colores típicos y un tamaño de 600 x 400, puede ocupar hasta 250 Kbytes; si se encuentra como Bitmap. Por supuesto, se pueden elegir otros formatos de almacenamiento de imágenes, como PCX (Paint Brush) o GEM (Ventura); sin embargo, el tiempo de despliegue puede crecer mientras más compactada este una imágen.

Por otra parte, un factor que influencia demasiado el diseño del programa es el tema del mismo. Un programa enfocado a nivel pre-escolar puede requerir imágenes sencillas, donde el tiempo de despliegue podría compensarse con música, pero no así un programa enfocado a un nivel universitario, como podría ser un tema médico, donde el despliegue de las imágenes debe ser rápido y mostrarse con una muy alta definición.

Gracias al desarrollo de los discos compactos y de nuevas tarjetas de video, se han podido resolver los problemas de capacidad y tiempo de despliegue. Un disco compacto puede almacenar 600 Mbytes.

7.2.2. Programa "Orígenes de las Computadoras Digitales"

El tema para el programa educativo que se uso como ejemplo en esta sección es sobre los orígenes de las computadoras digitales, basándose en la información propocionada en el libro "The Origins of Digital Computer" (véase bibliografía).

Dado que es un programa de ejemplo se pretende que solamente muestre diez imágenes, que contengan un breve resumen. Las imágenes son creadas antes que el programa de despliegue. El proceso que se uso para cada una, fue el siguiente:

- Se seleccionaron los principales temas a tratar.
- De cada uno de los temas se buscaron dibujos ó fotografías significativas.
- Por medio de un scanner se capturaron a formato de bitmap los dibujos y las fotografías.
- Con el programa Logitech Ansel, se corrigió el tamaño y definición de las imágenes.
- Por medio del programa Paintbrush se le agregó a cada imágen su respectivo texto.
- Desde Paintbrush cada imagen se salvó con una clave y con formato de bitmap.

Cada imágen se creó con tamaño de 600x320 pixeles. Así, en la figura 7.4 se muestra una de las imágenes creadas, donde se ha marcado cada una de sus partes, con su respectivo proceso de creación.

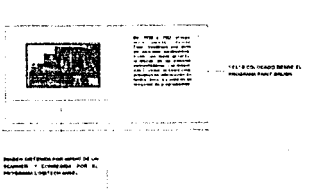


Figura 7-4.

Para evitar el problema de llamar las imágenes desde disco y esperar el despliegue, se opto por colocar todas las imágenes como un recurso de bitmaps. De esta forma las imágenes serán parte del programa y serán cargadas a RAM, haciendo más versátil su despliegue. Por supuesto, que deberá tomarse en cuenta que el programa ejecutable alcanzará un tamaño superior a 1 Mbyte, pues las imágenes en su estado nativo tienen un tamaño de 96118 bytes. Así, se debe establecer que el programa solo podrá ejecutarse en un sistema que posea 2 MBytes de RAM como mínimo.

7.2.3. Detalles de Control y Presentación

Ahora bien, como es de suponerse el programa debe poseer una forma de controlar la continuación de imágen a imágen, al igual que un control de retroceso, y una opción para finalizar el programa. Por lo regular, para este tipo de controles pensamos en menús y botones. Usar el formato de menú no es muy conveniente, pues éste estaría en la parte superior de la ventana y no es muy conveniente. Lo correcto sería usar botones, tres para las acciones descritas anteriormente. Se establece que el programa estará siempre controlado por el ratón, lo cual es muy común en los programas comerciales de este tipo.

La ventana definitiva que se estableció se muestra en la figura 7.5.

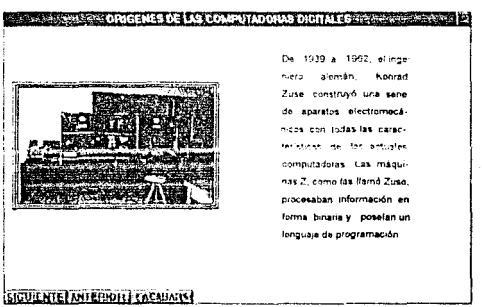


Figura 7.5.

7.2.4. Programación

La parte más confusa para la realización de este programa puede ser el hecho de crear los botones de la ventana. En realidad no hay ningún misterio, los botones se crean con la misma función con la que se caracteriza la ventana principal: CreateWindow().

Por medio de la función CreateWindow() se pueden crear elementos típicos de una ventana de diálogo (botones, barras deslizables, etc.) los cuales son manejados desde la función WndProc. Para ello, el primer argumento de la función debe recibir una contante que defina a

uno de los elementos; para nuestro caso se trata de "BOTTON". A continuación se muestran las líneas que generan los tres botones:

```
hButton1=CreateWindow("BUTTON","SIGUIENTE",
WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
1,360,80,20,hWnd,201,ghInstance,NULL);
ShowWindow(hButton1,SW_SHOW);

hButton2=CreateWindow("BUTTON","ANTERIOR",
WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
82,360,80,20,hWnd,202,ghInstance,NULL);
ShowWindow(hButton2,SW_SHOW);

hButton3=CreateWindow("BUTTON","ACABAR",
WS_CHILD WS_VISIBLE BS_PUSHBUTTON,
163,360,80,20,hWnd,200,ghInstance,NULL);
ShowWindow(hButton3,SW_SHOW);
```

Se puede observar, que el segundo argumento que recibe la función es el letrero de identificación del botón, parámetros restantes establecerán la presentación, lugar de aparición, tamaño, liga con el manejadador de ventana principal (hWnd), número de control e instancia del momento (ghInstance)

Con el número de control se ligarán los botones con sus correspondientes procesos, para ello se usa, al igual que con un menú o teclado, una zona de WM_COMMAND con su respectiva estructura switch-case en función de la variable wParam. En el de listado anterior, también se puede observar que cada CreateWindow() regresa un valor a tres variables (hButton1, hButton2 y hButton3). Estas variables son del tipo HWND y son manejadores para establecer su aparición por medio de ShowWindow().

A continuación, en el listado 7.4, se muestra el programa completo.

Listado 7.4.

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

int ghInstance;
long FAR PASCAL _export WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;

ghInstance=hInstance;
if (!hPrevInstance)
        {
        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName="WIN_PRO";
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;

        if (!RegisterClass(&WndClass))
          exit(FALSE);
```

```
        }
hWnd=CreateWindow("WIN_PRO",
                  "ORIGENES DE LAS COMPUTADORAS DIGITALES",
                  WS_MAXIMIZE|WS_MINIMIZEBOX,
                  10,
                  10,
                  600/*CW_USEDEFAULT*/,
                  400/*CW_USEDEFAULT*/,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}

long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{

static HBITMAP hBitmap[13];
static HWND   hButton1, hButton2, hButton3;
HDC      hDC,hMemDC;
static int cont=0;

switch(iMessage)
        {
        case WM_CREATE:
                  hButton1=CreateWindow("BUTTON","SIGUIENTE",
                  WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                  1,360,80,20,hWnd,201,ghInstance,NULL);
                  ShowWindow(hButton1,SW_SHOW);

                  hButton2=CreateWindow("BUTTON","ANTERIOR",
                  WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                  82,360,80,20,hWnd,202,ghInstance,NULL);
                  ShowWindow(hButton2,SW_SHOW);

                  hButton3=CreateWindow("BUTTON","ACABAR",
                  WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                  163,360,80,20,hWnd,200,ghInstance,NULL);
                  ShowWindow(hButton3,SW_SHOW);

                  hBitmap[0]=LoadBitmap(ghInstance,"C0");
                  hBitmap[1]=LoadBitmap(ghInstance,"C1");
                  hBitmap[2]=LoadBitmap(ghInstance,"C2");
                  hBitmap[3]=LoadBitmap(ghInstance,"C3");
                  hBitmap[4]=LoadBitmap(ghInstance,"C4");
                  hBitmap[5]=LoadBitmap(ghInstance,"C5");
                  hBitmap[6]=LoadBitmap(ghInstance,"C6");
                  hBitmap[7]=LoadBitmap(ghInstance,"C7");
                  hBitmap[8]=LoadBitmap(ghInstance,"C8");
                  hBitmap[9]=LoadBitmap(ghInstance,"C9");
                  hBitmap[10]=LoadBitmap(ghInstance,"C10");
                  hBitmap[11]=LoadBitmap(ghInstance,"C11");
                  hBitmap[12]=LoadBitmap(ghInstance,"C12");
                  break;

        case WM_COMMAND:
                        switch(wParam)
            {
                              case 200:
                                  DestroyWindow(hWnd);
                                  break;
```

```
                    case 201:
                        hDC=GetDC(hWnd);
                        hMemDC=CreateCompatibleDC(hDC);
                        SelectObject(hMemDC,hBitmap[cont]);
                        BitBit(hDC,10,10,600,460,hMemDC,0,0,SRCCOPY);
                        DeleteDC(hMemDC);
                        ReleaseDC(hWnd,hDC);
                        cont++;
            if (cont>12) cont=0;
                        break;

                    case 202:
                        cont--;
                        if (cont<0) cont=12;

                        hDC=GetDC(hWnd);
                        hMemDC=CreateCompatibleDC(hDC);
                        SelectObject(hMemDC,hBitmap[cont]);
                        BitBit(hDC,10,10,600,460,hMemDC,0,0,SRCCOPY);
                        DeleteDC(hMemDC);
                        ReleaseDC(hWnd,hDC);
                        break;
        }
            break;
    case WM_DESTROY:
                PostQuitMessage(0);
                break;
    default:
                return  DefWindowProc(hWnd,iMessage,wParam,lParam);
    }
            return(0L);
}
```

7.3. Programa para Desplegar Cuerpos Tridimensionales

Sin duda uno de los usos más extraordinarios que se le ha dado a la computadora es el poder desplegar cuerpos tridimensionales, lo cual a derivado dos nuevas ramas de estudio dentro de la computación: el Diseño Asistido por Computadora y Manufactura Asistida por Computadora.

En esta sección se mostrará un programa con la capaidad de desplegar objetos tridimensionales dentro de una ventana. El programa se basa en el artículo "Three-Dimensional Modeling Under Windows 3.1" , publicado en la revista "The C User Journal" (vease bibliografía).

7.3.1. La Descripción Matemática

Se han desarrollado varios algoritmos para poder desplegar una serie de puntos ubicados en el espacio tridimensional a un espacio de dos dimensiones. El algoritmo del ojo (o de cámara, como lo concen algunes autores) es el más usado. Para el algoritmo de ojo se establece un observador viendo la proyección de un objeto tridimensional a través de una ventana, la cual representa en si el monitor de una computadora, figura 7-6

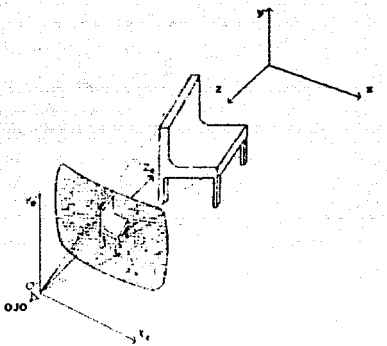


Figura 7-6.

El ojo también se encuentra en el espacio, con sus coordenadas (Xe,Ye,Ze), donde Ze es el resultado de la sustración D - z, siendo D la distancia con respecto al origen de los ejes coordenados respectivos del objeto hacia el ojo, y z es la distancia del objeto respecto a lo ejes.

Entonces, si deseamos pasar las coordenadas del ojo, a coordenadas con respecto a los ejes del objeto podriamos establecer que

$$(Xe,Ye,Ze) = (x,y,D-z)$$

Pasando a una forma vectorial más conveniente, se establece la siguiente igualdad:

$$(Xe,Ye,Ze) = (x,y,z,1) \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & D & 1 \end{vmatrix}$$

$$=(x,y,D-z,1)$$

La matriz, a la cual disignaremos como T, es el resultado de establcer el ojo en distintas posiciones con respecto al plano zy. Ahora bien, suponiendoe que nos colocamos en un punto del espacio ilustrado en la figura 7-6., viendo hacia el plano zy, de los ejes del objeto; podriamos establecer las relaciones de la figura 7-7.

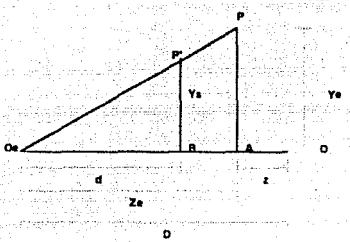


Figura 7-7.

Así se tiene que P' corresponde al punto reflejado sobre la pantalla a P, que es un punto del objeto. Tomando en cuenta los triangulos AOeP Y BOeP', se establecen las siguientes relaciones:

$$\frac{Ys}{d} = \frac{Ye}{Ze} \quad \text{implica} \quad Ys = d\,\frac{Ye}{Ze} = d\,\frac{Y}{D - z}$$

$$\frac{Xs}{d} = \frac{Xe}{Ze} \quad \text{implica} \quad Xs = d\,\frac{Xe}{Ze} = d\,\frac{X}{D - z}$$

Y, si hacemos que D tienda hacia el infinito obtendremos:

$$Xs = x \qquad y \qquad Ys = y$$

A este tipo de relaciones se le llama proyección ortogonal. Pero si, cambiamos la situación a un ojo que se mueva en todo el espacio (considerando movimientos esféricos) tendremos situaciones como la mostrada en en la figura 7-8

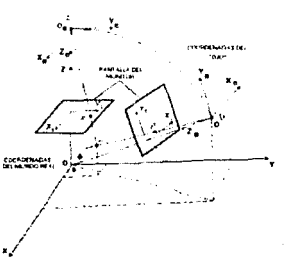


Figura 7-8.

La matriz T cambia radicamente sus componentes y establecemos lo siguiente:

$$(Xe,Ye,Ze) = (x,y,z,1) \begin{vmatrix} -\text{sen}\ \theta & -\cos\theta\cos\phi & -\cos\theta\,\text{sen}\ \phi & 0 \\ \cos\ \theta & -\text{sen}\,\theta\cos\phi & -\cos\theta\,\text{sen}\ \phi & 0 \\ 0 & \text{sin}\ \phi & -\cos\ \phi & 0 \\ 0 & 0 & D & 1 \end{vmatrix}$$

Obteniedo que:

$$Xe = -x\ \text{sen}\ \theta\ +\ y\cos\theta$$
$$Ye = -x\cos\theta\cos\phi\ -\ y\,\text{sen}\,\theta\cos\phi + z\,\text{sen}\,\phi$$
$$Ze = -x\cos\theta\cos\phi\ -\ y\,\text{sen}\,\theta\cos\phi + z\,\text{sen}\,\phi + D.$$

Las relaciones obtenidas de la figura 7-7 permanecen igual:

$$Ys = d\,\frac{Ye}{Ze}$$

$$Xs = d\,\frac{Xe}{Ze}$$

7.3.2. Estructuras de Datos a Considerar para un Programa

Primero que nada, se debe establecer la estructura de datos a usar para guardar los datos del modelo del "mundo real". Para los programadores en C la solución inmediata es definir una estructura de tres elementos del tipo double para permiter valores de números reales. La estructura podria ser de la siguiente forma

```
typedef struct tagPUNTO3D
{
double x;
double y;
double z;
} PUNTO3D;
```

Para guardar todos los puntos de un objeto, podemos recurrir a usar una estructura dinámica (lista doblemente ligada); sin embargo, puesto que esto es un ejemplo básico podemos recurrir a un arreglo con una cantidad adecuada de elementos.

Establecido lo anterior, debemos recapacitar en que solamente hemos hablado de los puntos que conforman un objeto. Para formar un objeto debemos recurrir a unir los puntos con líneas; es decir, nuestro programa debe recibir los puntos del objeto del "mundo real" y la relación de que puntos se conectan entre si para forma una linea o superfice del objeto

Un método común para realizar la unión de puntos, fácil y rápida es la de construir una estructura que le indique al programa cuantos y cuales puntos se van a unir una vez pasadas sus coordenadas al "mundo de la computadora". Por referencia geométrica, a estos puntos se les denomina vértices.

El proceso de unir los vértices puede ser semejante a esos juegos de unir los puntos para formar una figura. Así, primero se nos muestra una secuencia de puntos numerados, los

cuales unimos según la precedencia numerica. Basandonos en esto podemos escribir en pseudocódigo la forma en que el programa formará una figura:

1. Dame los puntos que desea unir.
2. Dame la secuencia de unión.
3. Hago la unión de los puntos en pantalla.
4 Si hay más secuencias regreso al paso 1.

La secuencia de los puntos es guardada como una lista de índices del arreglo de los puntos en un arreglo temporal para dicha lista también puede usarse otro arreglo que posea índices de control de secuencias. Como ejemplo de lo anterior tenemos lo siguiente:

Puntos para formar un cubo (formato en mundo real)	Número del Punto (vértice)
{1,1-1}	0
{1,1,1}	1
{1,-1,1}	2
{1,-1,-1}	3
{-1,1,-1}	4
{-1,-1,-1}	5
{-1,-1,1}	6
{-1,1,1}	7

Mapa de puntos para formar las caras (superficies) del Cubo

0,1,2,3,0
0,4,7,1,0
4,5,6,7,4
3,2,6,5,3
1,7,6,2,1
0,3,5,4,0

Forma se seguir la secuencia en el mapa de puntos

0,4, (del mapa inicie desde 0 y una los siguientes 4 puntos)
4,4, (del mapa inicie desde 4 y una los siguientes 4 puntos)
8,4, (del mapa inicie desde 8 y una los siguientes 4 puntos)
12,4, (del mapa inicie desde 12 y una los siguientes 4 puntos)
16,4, (del mapa inicie desde 16 y una los siguientes 4 puntos)
20,4, (del mapa inicie desde 20 y una los siguientes 4 puntos)

Por lo tanto considerando lo anterior, podemos establecer las siguientes estructuras de datos:

```
typedef struct tagSUPERFICIE
{
int mapIndex;
int noOfVertices;
} SUPERFICIE;

typedef struct tagOBJETO
{
```

```
VERTICES vertice[NUM_DE_VERTICES];
int map[NUM_DE_SUPERFICIE_VERTICES];
SUPERFICIE info[NUM_DE_SUPERFICIES];
} OBJETO;
```

Con la estructura tagSuperficie, se guarda la información para seguir la secuencia de creación de una superficie. En la estructura tagObjecto se guardan tanto los valores del objeto en "mundo real", el mapa de vértices y la información de la secuencia. NUM_DE_VERTICES, NUM_DE_SUPERFICIE_VERTICES, NUM_DE_SUPERFICIES, corresponden a las contantes que limitan el número de puntos en el "mundo real", el número de secuencias a seguir y número de puntos a usar para crear una superficie, respectivamente.

Así, para el ejemplo del cubo, podríamos tener una estrutura de la siguiente forma:

```
OBJETO cubo
{
{
{1,1,-1},
{1,1,1},
{1,-1,1},
{1,-1,-1},
{-1,1,-1},
{-1,-1,-1},
{-1,-1,1},
{-1,1,1}
},

{
0,1,2,3,0
0,4,7,1,0
4,5,6,7,4
3,2,6,5,3
1,7,6,2,1
0,3,5,4,0
},

{
{0,4},
{4,4},
{8,4},
{2,4},
{16,4},
{20,4}}
} };
```

Los valores de las contantes de los arreglos serían los siguientes:

NUM_DE_VERTICES	= 8
NUM_DE_SUPERFICIE_VERTICES	= 30
NUM_DE_SUPERFICIES	= 6

7.3.4. Consideraciones para el Despliegue en Pantalla

Cuando se pasan los puntos del "mundo real" al "mundo de la computadora", o sea, a la pantalla del monitor debemos establecer un mecanismo para que estos puedan ser gráficados de acuerdo al número de pixeles de la tarjeta de video. Trabajando en Windows poco puede ser la preocupación de incompatibilidad de resolución, así que usaremos el concepto de coordenadas homegéneas. Con coordenadas homogéneas solo se considera el tamaño de la pantalla de despliegue, así retomando las ecuaciones para la generación de los puntos tridimensionales a bidimensionales, tenemos que:

$$Ys = \frac{d\ Yc}{S\ Ze}$$

$$Xs = \frac{d\ Xc}{S\ Ze}$$

Donde la S representa la longitud media del monitor de despliegue del objeto, un monitor cuadrado. Por supuesto que no existen monitores cuya altura y ancho tengan el mismo valor, pero para el medio ambiente Windows es totalmente válido este tipo de concepto cuando se fija el modo MM_ISOTROPIC (escalamiento automático en función del tamaño de la ventana). Por conveniencia podemos fijar los valores de d y S en 40 y 10, respectivamente. Dentro del programa esta relación será manejada como una constante y recibirá el nombre de VISION.

Otros valores a considerar, son las coordenadas del ojo. Como se mencionó en relación a la figura 7-8 , las coordenadas estan dadas en forma esférica, así que tenemos una distancia y dos ángulos. La ventaja de manejar coordenadas esféricas es el hecho de poder facilitar el movimieto del ojo en el espacio y permitir diferentes perpectivas del objeto. Las variables esféricas del ojo, con sus respectivos valores iniciales, son las siguientes:

distancia = 75
thetaDegrados = 90
phiDegrados = 60

Las tres variables son del tipo double. Para el caso de las variables que manejan los ángulo, se debe recordar que se manejan las operaciones trigonométricas, por lo cual el programa cuenta con funciones de conversión de radianes a grados y viceversa.

Ahora bien, como una forma de aprovechar mejor las variables esféricas el programa se diseño para usar las barras de deslizables horizontal y vertical para controlar los movimientos del ojo. De tal forma, que la barra horizontal controla los incrementos y decrementos correpondientes a la variable thetaDegrados y la barra vertical los de la variable phiDegrados. Usando la función SetScrollRange() se establecen los maximos valores a manejar las barras, esto se muestra a continuación.

```
SetScrollRange(hWnd,SB_HORZ,MIN_DEGRADOS,MAX_DEGRADOS,TRUE);
SetScrollRange(hWnd,SB_VERT,MIN_DEGRADOS,MAX_DEGRADOS,TRUE);
```

Donde MIN_DEGRADOS y MAX_DEGRADOS son constantes con los valores de 0 y 359, respectivamente. Y para establecer la posición las barras con los valores iniciales de thetaDegrados y phiDegrados, se usa la función SetScrollPos() de la siguiente forma

```
SetScrollPos(hWnd,SB_HORZ,(int)thetaDegrados,TRUE);
SetScrollPos(hWnd,SB_VERT,(int)phiDegrados,TRUE);
```

La estructura de las zonas de WM_HSCROLL y WM_VSCROLL quedan definidas de la siguiente forma:

```
case WM_HSCROLL:
        if (wParam==SB_THUMBTRACK)
                break;
        hPos=GetScrollPos(hWnd,SB_HORZ);

        switch(wParam)
        {
                case SB_TOP:
                        hPos=MIN_DEGRADOS;
                        break;
                case SB_BOTTOM:
                        hPos=MAX_DEGRADOS;
                        break;
                case SB_LINEUP:
                        hPos-=ROTAR_DEGRADOS;
                        break;
                case SB_LINEDOWN:
                        hPos+=ROTAR_DEGRADOS;
                        break;
                case SB_THUMBPOSITION:
                        hPos=LOWORD(lParam);
                        break;
        }
        if (hPos<MIN_DEGRADOS)
                hPos=MAX_DEGRADOS;
        if (hPos>MAX_DEGRADOS)
                hPos=MIN_DEGRADOS;

        SetScrollPos(hWnd,SB_HORZ,hPos,TRUE);
        thetaDegrados=(double)hPos;
        InvalidateRect(hWnd,NULL,TRUE);
        break;

case WM_VSCROLL:
        if (wParam==SB_THUMBTRACK)
                break;
        vPos=GetScrollPos(hWnd,SB_VERT);

        switch(wParam)
        {
                case SB_TOP:
                        vPos=MIN_DEGRADOS;
                        break;
                case SB_BOTTOM:
                        vPos=MAX_DEGRADOS;
                        break;
                case SB_LINEUP:
                        vPos-=ROTAR_DEGRADOS;
                        break;
                case SB_LINEDOWN:
                        vPos+=ROTAR_DEGRADOS;
                        break;
                case SB_THUMBPOSITION:
                        vPos=LOWORD(lParam);
                        break;
        }
        if (vPos<MIN_DEGRADOS)
                vPos=MAX_DEGRADOS;
        if (vPos>MAX_DEGRADOS)
                vPos=MIN_DEGRADOS;

        SetScrollPos(hWnd,SB_VERT,vPos,TRUE);
        phiDegrados=(double)vPos;
        InvalidateRect(hWnd,NULL,TRUE);
```

```
break;
```

Las variables hPos, VPos son usadas para transmitir los principales cambios que ocurran en sus respectivas zonas. ROTAR_DEGRADOS es una constante usada para establecer un tamaño de giro en particular; es decir, nuestro objeto al ser "rotado" por cualquiera de las barras lo hará en cinco grados.

7.3.5. El Programa

El listado 7.5, muestra todo el programa para el despliegar objetos tridimensionales; cuya información en la variable cuerpo permite dibujar un avión. La figura 7-8 muestra la ventana de salida del programa con el avión desplegado

Listado 7.5.

```
#include <windows.h>
#include <stdlib.h>
#include <math.h>

#define CENTRO                  0
#define PI                      3.141593
#define RADIANES(a)             (a*(PI/180.0))
#define DEGRADOS(a)             (a*(180.0/PI))
#define NUM_DE_VERTICES         66
#define NUM_DE_SUPERFICIES              45
#define NUM_DE_SUPERFICIE_VERTICES      220
#define VISION          (40/10)
#define MIN_DEGRADOS            0
#define MAX_DEGRADOS            359
#define ROTAR_DEGRADOS                  5

typedef struct tagPUNTO3D
{
double x;
double y;
double z;
} PUNTO3D;

typedef struct tagVERTICES
{
PUNTO3D mundo;
PUNTO3D ojo;
PUNTO3D pantalla;
} VERTICES;


typedef struct tagSUPERFICIE
{
int mapIndex;
int noOfVertices;
} SUPERFICIE;

typedef struct tagOBJETO
{
VERTICES vertice[NUM_DE_VERTICES];
int map[NUM_DE_SUPERFICIE_VERTICES];
SUPERFICIE info[NUM_DE_SUPERFICIES];
} OBJETO;
```

```
OBJETO cuerpo
{
{

{0.00,0.00,0.00},
{0.00,-17.0,9.0},{0.0,-13.0,9.0},
{0.0,-14.5,3.0},{0.0,-8.0,3.0},
{0.0,-14.5,2.0},{0.0,-8.0,2.0},
{-0.5,-16.0,1.0},{-1.0,-15.0,0.0},
{-0.5,-18.0,-0.5},{0.0,-18.0,-1.0},
{0.5,-16.0,-0.5},{1.0,-18.0,0.0},
{0.50,-16.0,1.0},
{-1.0,-13.0,2.0},{-2.0,-13.0,0.0},
{-1.5,-13.0,-1.5},{0.0,-13.0,-2.0},
{1.5,-13.0,-1.5},{2.0,-13.0,0.0},
{1.0,-13.0,2.0},{-1.0,7.0,2.0},
{-2.0,7.0,0.0},{-1.5,7.0,-1.5},
{0.0,7.0,-2.0},{1.50,7.0,-1.5},
{2.0,7.0,0.0},{1.0,7.0,2.0},
{2.0,-8.0,0.0},{11.0,-8.0,0.0},
{11.0,-5.0,0.0},{5.0,0.0,0.0},
{2.0,7.0,0.0},{-2.0,-8.0,0.0},
{-11.0,-8.0,0.0},{-11.0,-5.0,0.0},
{-5.0,0.0,0.0} {-2.0,7.0,0.0},
{0.5,2.0,2.0},{-0.5,2.0,2.0},
{1.0,5.0,2.0},{0.5,5.0,3.5},
{-0.5,5.0,3.5},{-1.0,5.0,2.0},
{1.0,8.0,2.0},{0.5,8.0,3.5},
{-0.5,8.0,3.5},{-1.0,8.0,2.0},
{0.5,11.0,2.0},{-0.5,11.0,2.0},
{0.0,7.0,-1.0},{-1.0,11.0,2.0},
{-2.0,11.0,0.0},{0.0,11.0,-1.0},
{2.0,11.0,0.0},{1.0,11.0,2.0},
{0.0,11.0,-1.0},{0.0,17.0,0.0},
{-2.0,-16.0,0.0},{-8.0,-16.0,0.0},
{-8.0,-14.0,0.0},{-2.0,-10.0,0.0},
{2.0,-16.0,0.0},{8.0,-16.0,0.0},
{8.0,-14.0,0.0},{2.0,-10.0,0.0}
},

{
58,61,60,59,58,62,63,64,65,62,58,59,60,61,58,
62,65,64,63,62,
1,3,4,2,1,3,5,6,4,3,1,2,4,3,1,
3,4,6,5,3,
14,15,8,7,14,15,16,9,8,15,16,17,10,9,16,
17,18,11,10,17,18,19,12,11,18,19,20,13,12,19,
20,14,7,13,20,
14,21,22,15,14,15,22,23,16,15,16,23,24,17,16,
17,24,25,18,17,18,25,26,19,18,19,26,27,20,19,
20,27,21,14,20,
28,29,30,31,32,28,33,37,36,35,34,33,28,32,31,
30,29,28,33,34,35,36,37,33,
21,51,52,22,21,22,52,53,50,53,54,26,50,
26,54,55,27,26,27,55,51,21,27,
55,54,57,55,54,56,57,54,56,52,57,56,52,51,57,52,
51,55,57,51,
41,42,39,38,41,45,46,42,41,45,48,49,46,45,48,
40,41,38,40,42,43,39,42,44,48,41,40,44,
48,47,43,42,46,48,45,44,48,49,47,48,49
},

{

{0,5},{5,5},
{10,5},{15,5},
{20,5},{25,5},
{30,5},{35,5},
{40,5},{45,5},
```

```
{60,8},{58,6},
{60,6},{58,6},
{70,8},
{78,8},{80,6},
{88,8},{90,6},
{98,8},{100,8},
{105,6},
{110,6},{116,6},
{122,6},{128,6},
{134,6},{139,6},
{144,6},{149,6},
{154,5},
{159,4},{162,4},
{167,4},{171,4},
{175,4},
{179,5},{184,6},
{189,5},{194,4},
{198,4},{202,6},
{207,6},{212,4},
{216,4}
}
};


double distancia = 75, thetaDegrados=90,phiDegrados=60;


long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam);
void  DrawObject(HWND hWnd, HDC hDC, OBJETO *object);



int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpstrCmdParam, int nCmdShow)
{
HWND hWnd;
MSG Message;


        WNDCLASS WndClass;
        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=GetStockObject(WHITE_BRUSH);
        WndClass.hInstance=hInstance;
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName=(LPSTR) "3D";
        WndClass.lpszMenuName=(LPSTR) NULL;
        WndClass.style=(WORD)NULL;

        if (!RegisterClass(&WndClass))
          exit(FALSE);


hWnd=CreateWindow("3D",
                  "CUERPOS 3D",
                  WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
```

```
if (hWnd)
        return(FALSE);

SetScrollRange(hWnd,SB_HORZ,MIN_DEGRADOS,MAX_DEGRADOS,TRUE);
SetScrollRange(hWnd,SB_VERT,MIN_DEGRADOS,MAX_DEGRADOS,TRUE);
SetScrollPos(hWnd,SB_HORZ,(int)thetaDegrados,TRUE);
SetScrollPos(hWnd,SB_VERT,(int)phiDegrados,TRUE);



ShowWindow(hWnd,nCmdShow);
while(GetMessage (&Message,NULL,0,0))
{
TranslateMessage(&Message);
DispatchMessage(&Message);
}
return Message.wParam;
}


long FAR PASCAL WndProc(HWND hWnd, WORD iMessage, WORD wParam, LONG lParam)
{
HDC hDC;
PSSTRUCT ps;
HMENU hMenu;
int vPos, hPos;

switch(iMessage)
        {
        case WM_PS:
                        hDC=BeginPs(hWnd,&ps);
                        DrawObject(hWnd,hDC,&cuerpo);
                        ReleaseDC(hWnd,hDC);
                        EndPs(hWnd,&ps);
                        return 0;

        case WM_HSCROLL:
                        if (wParam==SB_THUMBTRACK)
                                break;
                        hPos=GetScrollPos(hWnd,SB_HORZ);

                        switch(wParam)
                        {
                                case SB_TOP:
                                        hPos=MIN_DEGRADOS;
                                        break;
                                case SB_BOTTOM:
                                        hPos=MAX_DEGRADOS;
                                        break;
                                case SB_LINEUP:
                                        hPos-=ROTAR_DEGRADOS;
                                        break;
                                case SB_LINEDOWN:
                                        hPos+=ROTAR_DEGRADOS;
                                        break;
                                case SB_THUMBPOSITION:
                                        hPos=LOWORD(lParam);
                                        break;
                        }
                        if (hPos<MIN_DEGRADOS)
                                hPos=MAX_DEGRADOS;
                        if (hPos>MAX_DEGRADOS)
                                hPos=MIN_DEGRADOS;
```

```
                        SetScrollPos(hWnd,SB_HORZ,hPos,TRUE);
                        thetaDegrados=(double)hPos;
                        InvalidateRect(hWnd,NULL,TRUE);
                        break;

        case WM_VSCROLL:
                        if (wParam==SB_THUMBTRACK)
                                break;
                        vPos=GetScrollPos(hWnd,SB_VERT);

                        switch(wParam)
                        {
                                case SB_TOP:
                                        vPos=MIN_DEGRADOS;
                                        break;
                                case SB_BOTTOM:
                                        vPos=MAX_DEGRADOS;
                                        break;
                                case SB_LINEUP:
                                        vPos-=ROTAR_DEGRADOS;
                                        break;
                                case SB_LINEDOWN:
                                        vPos+=ROTAR_DEGRADOS;
                                        break;
                                case SB_THUMBPOSITION:
                                        vPos=LOWORD(lParam);
                                        break;
                        }
                        if (vPos<MIN_DEGRADOS)
                                vPos=MAX_DEGRADOS;
                        if (vPos>MAX_DEGRADOS)
                                vPos=MIN_DEGRADOS;

                        SetScrollPos(hWnd,SB_VERT,vPos,TRUE);
                        phiDegrados=(double)vPos;
                        InvalidateRect(hWnd,NULL,TRUE);
                        break;

        case WM_SIZE:
                        InvalidateRect(hWnd,NULL,TRUE);
                        break;

        case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        default:
                        return  DefWindowProc(hWnd,iMessage,wParam,lParam);
        }
    return(0L);
}


void DrawObject(HWND hWnd, HDC hDC, OBJETO *object)
{
double sinTheta, cosTheta, sinPhi, cosPhi;
double s1,s2,s3;
PUNTO3D *v1,*v2,*v3;
POINT centro;
RECT  rect;
POINT puntos[10];
HBITMAP hBitmap, hOldBitmap;
HRGN hRgn;
HDC hMemDC;
int superficie, vertice, mapIndex, verticeIndex, loop;
```

```
GetClientRect(hWnd,&rect);
centro.x=(rect.right/2);
centro.y=(rect.bottom/2);
hRgn=CreateRectRgn(rect.left, rect.top,rect.right,rect.bottom);
hMemDC=CreateCompatibleDC(hDC);
hBitmap=CreateCompatibleBitmap(hDC,rect.right,rect.bottom);
hOldBitmap=SelectObject(hMemDC,hBitmap);

cosTheta= cos( RADIANES(thetaDegrados));
sinTheta= sin( RADIANES(thetaDegrados));
cosPhi= cos( RADIANES(phiDegrados));
sinPhi= sin( RADIANES(phiDegrados));

for (loop=0; loop<NUM_DE_VERTICES; loop++)
{
        object->vertice[loop].ojo.x=
                                        (-object->vertice[loop].mundo.x*sinTheta)+
                                        (object->vertice[loop].mundo.y*cosTheta);
        object->vertice[loop].ojo.y=
                                        (-object->vertice[loop].mundo.x*cosTheta*cosPhi)-
                                        (object->vertice[loop].mundo.y*sinTheta*cosPhi)+
                                        (object->vertice[loop].mundo.z*sinPhi);
        object->vertice[loop].ojo.z=
                                        (-object->vertice[loop].mundo.x*sinPhi*cosTheta)-
                                        (object->vertice[loop].mundo.y*sinTheta*sinPhi)+
                                        (object->vertice[loop].mundo.z*cosPhi)+distancia;

        object->vertice[loop].pantalla.x=(int)
                (VISION*(object->vertice[loop].ojo.x/object->vertice[loop].ojo.z)*centro.y+centro.x);

        object->vertice[loop].pantalla.y=(int)
                (VISION*(object->vertice[loop].ojo.y/object->vertice[loop].ojo.z)*centro.y+centro.y);
}
FillRgn(hMemDC,hRgn,GetStockObject(WHITE_BRUSH));
SelectObject(hMemDC,GetStockObject(BLACK_PEN));


for (superficie=0; superficie<NUM_DE_SUPERFICIES; superficie++)
{
        mapIndex=object->info[superficie].mapIndex;


                for (vertice=0, vertice<object->info[superficie].noOfVertices; vertice++,mapIndex++)
                {
                        verticeIndex=object->map[mapIndex];
                        puntos[vertice].x=object->vertice[verticeIndex].pantalla.x;
                        puntos[vertice].y=object->vertice[verticeIndex].pantalla.y;
                }
                        Polyline(hMemDC, &puntos[0],vertice);

}

        BitBlt(hDC,0,0,rect.right,rect.bottom,hMemDC,0,0,SRCCOPY);
        SelectObject(hMemDC,hOldBitmap);
        DeleteObject(hBitmap);
        DeleteObject(hRgn);
        DeleteDC(hMemDC);
}
```

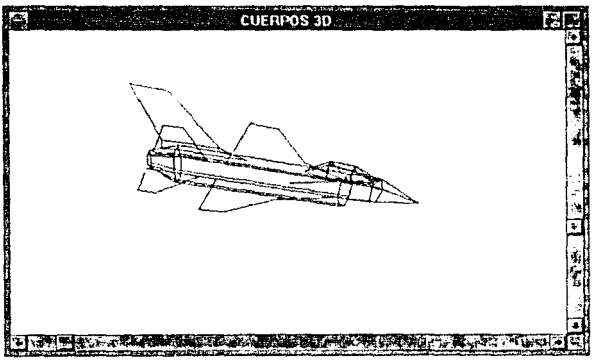


Figura 7-9.

Conclusiones

El medioambiente Microsoft Windows es una interface gráfica que permite correr en multiprocesamiento, establece el control común entre periféricos tales como tarjetas de videos e impresoras. Windows, escencialmente, es controlado por ratón, lo que permite que hasta el más inesperto de los usuarios pueda trabajar comodamente sin aprender los complejos comandos del sistema operativo. Windows permite el uso de la memoria extendida y expandida en forma transparente al usuario, por lo que presenta un mejor aprovechamiento de la RAM de una computadora en comparación con los programas diseñado para DOS, los cuales deben de ser diseñados para trabajar en modo estandar o modo protegido para usar la memoria.

La metodología para realizar programas bajo ambiente Windows usando las especificaciones de Microsoft es fácil, siempre y cuando se tenga un buen conocimiento del lenguaje C y se usen las herramientas especialmente diseñadas para tal situación. Las herramientas más sofisticadas son los editores de recursos que pueden contar con una gran variedad de elementos para la creación de un recurso y que proporcionan al programador una visión de la presentación del programa. Sin embargo, el uso de los editores de recursos causa el desconocimiento pleno de la codificación del archivo de recursos, pues la gran mayoría de los editores soló crean un archivo de trabajo ya compilado (archivo .RES).

Por otra parte los editores de recursos son una arma de dos filos. Con un editor de recursos, por ejemplo, podemos alterar directamente los recursos de un programa ya ejecutado, así se puede pasar a "español" un programa en "inglés", pero de no haber un cuidado especial en la alteración de los recursos, el programa podría sufrir daños irreparables.

Ahora bien, la estructuración de los procesos WinMain y WinProc marca el uso de objetos orientados, pero manejados de una forma más simple y menos compleja que la que se encuentra dentro de C++. La enrutación de los mensajes de los objetos manejados (menús, barras desilizables, botones, cajas de diálogo, etc.) esta marcada por el uso de la estructura switch-case; en la cual se establecen las secciones respectivas para cada tipo de mensaje. Por supuesto, una de las labores del programador es conocer el nombre de referencia y su uso de cada uno de los 200 mensajes que especifica Microsoft para Windows.h; siendo esta biblioteca esencial para poder programar en el medioambiente.

Sin embargo, Windows presenta limitaciones al relacionarse el hardware que debe poseer la computadora donde corre. La actual versión, Windows 3.1, soló permite su uso considerando a las tarjetas VGA como básicas; es decir, solo tarjetas de video con una resolusión de 640 x 480, como mínimo. Windows 3.1 puede correrse en una computadora con microprocesador 80286, pero no puede usar la capacidad plenamente su capacidad de interactuar con el DOS. Para esto es necesario poseer computadoras con 80386 ó 80486. Windows 3.1 marca como un mínimo 1 Mbyte de RAM, para un uso decoroso; pues no garantiza su eficacia para correr en multiprocesamiento. Windows trabaja excelentemente con mas de 2 Mbytes de RAM.

La liga entre Windows y DOS, aunque parece muy delgada, en realidad es una cadena. Cuando un programa en Windows interfiere con ciertas interrupciones, puede causar su desconexión del medioambiente, acarreando al resto de los programas.

Sin embargo, el impacto de Windows en la comunidad de usuarios de computadoras personales a sido enorme. Windows ha permitido el desarrollo de nuevos periféricos, tales como tarjetas de sonido y video; permitiendo el desarrollo de multimedia, que antaño se encontraba restrigido a mainframes y a las computadoras Apple. El impacto sobre la mercadotecnia del hardware en nuestro pais también ha sido muy interesantes. La mayor parte de las computadoras personales se venden con Windows, lo que implica un hardware mínimo que posea un tarjeta VGA y un microprocesador 80386.

El software diseñado para Windows casi ha sido igualado en cantidad y en uso al que existe en DOS, y de proseguir la tendencia de uso de Windows y de sus futuras versiones (como lo es NT) lograria definitivamente derrocar al DOS y quedarse como la plataforma de trabajo única de las computadoras personales basadas en los microprocesadores Intel.

Otras metodologías han sido desarrolladas por compañías competidoras de Microsoft para auxiliar a los programadores. Borland ha propuesto una plataforma basada totalmente en objetos, cuya complejidad radica en recordar la relación que guardan los diferentes objetos con la ventana principal. Microsoft también ha propuesto varias "fórmulas" para usar objetos dentro de su metodología, pero no con mucha aceptación.

Microsoft también ha creado dos compiladores que explotan la llamada programación visual, es decir, crear primero la manera de presentar un programa y después la lógica. El primero, Visual Basic, tuvo una buena aceptación a nivel mundial y gracias a este éxito una gran variedad de programas shareware. El segundo lenguaje, de reciente aparición, se llama Visual C++, un curioso híbrido que combina la metodología original con el modo visual. Sin embargo, los compiladores "visuales" de Microsoft carecen de elementos para realizar programas complejos que combinen interfaces multiples. Por ello, caemos nuevamente en el uso de la metodología inicial.

El siguiente gran paso de Microsoft es Windows NT, lanzado al mercado el 31 de agosto de 1993. Windows NT es un sistema operativo con la misma interface gráfica que su antecesor Windows 3.1. La meta de Microsoft es alcanzar el mercado de las estaciones de trabajo y de redes. Microsoft unió fuerzas con Intel para la creación de Pentium (80586) para establecer una una línea de estaciones de trabajo compatibles con las computadoras personales. Por otra parte, no soló comercializará Windows NT para computadoras basadas en 80x86, sino también en otras que usen microprocesadores como el Alpha, de la compañía DEC, o el 68040 ,de Motorola. Así, Microsoft tratará de captar más usuarios para Windows. Pero, por si fuera poco todo lo anterior, Microsoft ha garantizado la portabilidad de software entre los sistemas NT; lo cual significaría la estandarización de su metodología de trabajo. Esto significaría el

establecimiento, por primera vez en la historia del software, de una base real y única para aprender a programar.

Pero, aún hay más dentro de los planes de Microsoft; pues esta preparando una versión de Windows cuyo nombre sería Windows Avanced Server (WAS). Esta versión sería destinada hacia las grandes redes de información y grandes sistemas de cómputo Microsoft ha asegurado que la misma capacidad de portabilidad de el éxito de WAS depende de la aceptación que tenga Windows NT.

Los sistemas competidores de Windows NT y WAS son principalmente OS/2, NetWare, Unix y, curiosamente, el MS-DOS. Ninguno de ellos presenta en la actualidad la portabilidad plena que ofrece Windows NT, pero su evolución ha estado muy marcada por la misma desarrollo de Windows.

Bibliografía

Adams, Lee. ***High-Performance C Graphics Programming for Windows.*** Windcrest/McGraw-Hill , 1992. U.S.A.

Aumiaux, Michel. ***Microprocesadores 16 Bits.*** Masson Editores, 1987. México.

Barkakati, Naba. ***The Waite Group's Essential Guide to Turbo C.*** Howard W. Sams and Company, 1989. U.S.A.

Barnes, John. ***DOS 6.0.*** Personal Computer World, 1993, no. 3, pp 296-302. U.K.

Barlow, David R. y et al. ***Interactive Programming Environments.*** McGraw-Hill Book Company, 1986. U.S.A.

Booch, Grady. ***Object Oriented Design with Applications.*** The Benjamin/Cummings Publishing, 1991. U.S.A.

Bell, Doug y et al. ***Software Engineering: A Programming Approach.*** Prentice-Hall International, 1987. U.S.A.

Bertran, Michel. ***Fast Bezier Curves in Windows.*** PC Techniques, 1992, vol. 2, no. 6, pp 25-30. U.S.A.

Borland International. ***Borland C++ 2.0 - Getting Started. Borland International,*** 1991. U.S.A.

Borland International. ***Borland C++ 2.0 - Whitewater Resourse Toolkit.*** 1991. U.S.A.

Borland International. ***Paradox para Windows -Guia de referencia Objectpal.*** 1993. U.S.A.

Randell, Brian (Editor). ***The Origins of Digital Computer.*** Springer-Verlag, 1970. U.S.A.

Charete, Robert N. ***Software Engineering Enviroments: Concepts and Technology.*** McGraw-Hill Book Company, 1987. U.S.A.

Conger, James L. ***The Waite Group's Windows API Bible: The Definite Programmer's Reference.*** Waite Group Press, 1992. U.S.A.

Devoney, Chris, ***Dos 6.0.*** PC Computing, 1993, no. 5, pp 129-159. U.S.A.

Donovan, John, ***Operating-System Trend*** Byte, 1992, no. 10, vol. 17, pp 159-166. U.S.A.

Duncan, Ray. ***Advanced MS-DOS Programming.*** Microsoft Press, 1986. U.S.A.

Duncan, Ray. *The MS-DOS Encyclopedia.* Microsoft Press, 1988. U.S.A.

Eggebrecht, Lewis. ***Interfacing the IBM Personal Computer.*** Howard W. Sams and Company, 1990. U.S.A.

Entsminger, Gary. ***Turbo Pascal for Windows Bible.*** Howard W. Sams and Company, 1992. U.S.A.

Ezzel, Ben. ***Windows Graphics*** Programming. Ziff-Davis Press, 1992. U.S.A.

Farrel, Tim y Runnoe Connally. ***Programming in Windows 3.1.*** Que Corporation, 1992. U.S.A.

Graham, Ian. ***Object Oriented Methods.*** Adisson-Wesley Publishing, 1991. U.S.A.

Hall, Douglas V. ***Microprocessors and Interfacing: Programming and Hardware.*** McGraw-Hill International, 1986. U.S.A.

Heiny, Loren. ***Windows Graphics Programming with Borland C++.*** John Wiley & Sons, 1992. U.S.A.

Hekmatpour, Sharam. *C++: Guia para Programadores en C.* Prentice-Hall Hispanoamericana, 1992. Mexico.

Hyman, Michel. ***Windows Dynamics Data Exchange.*** PC Techniques, 1992, vol. 2, no. 6, pp 35-40. U.S.A.

Holzner, Steven y The Peter Norton Computing Group. ***Advanced Assembly Language.*** Brady Publishing, 1991. U.S.A.

Jones, Gregory W. ***Software Engineering.*** John Wiley & Sons, 1990. U.S.A.

Kernighan, Brian W. y P. J. Plauger. ***The Elements of Programming Style.*** McGraw-Hill International, 1978. U.S.A.

Khoshafian, Setrag y et al. ***Object Orientation: Concepts, Languages, Databases and User Interfaces.*** John Wiley & Sons, 1990. U.S.A.

LeBlond Group. ***Windows 3 Power Tools.*** Bantam Books, 1991. U.S.A.

Lecarme Olivier y Mireille Pellissier Gart. ***Software Portability.*** McGraw-Hill International, 1987. U.S.A.

Levy, Steven. ***Hackers: Heroes of the Computer Revolution.*** Dell Publishing, 1984. U.S.A.

McCord, James W. ***Developing Windows Applications with Borland C++ 3.*** Howard W. Sams and Company, 1992. U.S.A.

Meyer, William. ***Los Creadores de Imagen.*** Planeta, 1987. México.

Microsoft Windows, ***Manual del Usuario.*** 1992. U.S.A.

Minasi, Mark. ***The OS/2 Alternative,*** Special Issue Byte, 1992, vol. 17, no. 11, pp 55-60. U.S.A.

Mischel, Jim. ***A Guide to Windows Help.*** PC Techniques, 1993, vol. 3, no. 6, pp 24-31. U.S.A.

Nicholson, Matt. ***Learnig Visual Basic.*** PC Plus, 1993, no. 3, pp 299-302. U.K.

Nicholson, Matt y et al. ***6 of the Best?.*** PC Plus, 1993, no. 3, pp 218-225. U.K.

Nace, Barry. ***How OLE Work.*** Special Issue Byte, 1992, vol. 17, no. 11, pp 45-52. U.S.A.

Norton, Peter y Paul Yao. ***Borland C++ Programming for Windows.*** Bantam Books, 1992. U.S.A.

Norton, Peter y Richard Wilton. ***The New Peter Norton Programmer's Guide to the IBM PC & PS/2.*** Microsoft Press, 1988. U.S.A.

Olsen, Tomas W. ***Three-Dimensional Modeling Under Windows 3.1.*** The C Users Journal, 1993, vol. 11, no. 3, pp 53-58. U.S.A.

Palmer, Scott D. ***Programmer's Introduction to Turbo Pascal for Windows.*** Sibex Inc. 1992, U.S.A.

Park, Chan S. Interactive Microcomputer graphics. Addison-Wesley, 1985. U.S.A.

Pfaffenberger, Bryan. ***Que's Computer User's Dictionary.*** Que Corporation. 1990. U.S.A.

Pressman, Roger S. ***Software Engineering: A Practitioner's Approach.*** McGraw-Hill International, 1987. U.S.A.

Prosise, Jeff. ***DOS 5 Memory Management with Utilities.*** Ziff-Davis Press, 1992. U.S.A.

Radcliffe, Mark y Quaterdeck Office Systems. ***DesqView/X: A Tecnical Perpective for '90s.*** Byte, 1992, vol 17, no 12, suplemento. U.S.A.

Robinson, Phillip. ***Dr. Dobb's Toolbook of 80286/80386 Programming.*** M&T Publishing, 1988. U.S.A.

Townsend, Carl. ***Advanced MS-DOS Expert Techniques for Programmers.*** Howard W. Sams and Company, 1989. U.S.A.

Schildt, Herbert. ***Born to Code in C.*** McGraw-Hill,1989. U.S.A.

Simonyi, Charles y Martin Heller. ***The Hungarian Revolution.*** Byte, 1991, vol. 16, no. 8, pp 131-138. U.S.A.

Smith, Gina. ***Will the Pentium kill the 486.*** PC Computing, 1993, no. 5, pp 116-125. U.S.A.

Smith, Jan. ***Windows Accelerators.*** PC Computing, 1993, no. 5, pp 274-275. U.S.A.

Simpson, Henry. ***Design of User-Friendely Programs for Small Computers.*** McGraw-Hill International. 1985. U.S.A.

Spraycar, Marjory. ***PS/2: The Market Context.*** PC Tech Journal, 1987, vol. 5, no. 8, pp 156-164. U.S.A.

Swan, Tom. ***Tips for Your Toolbox.*** PC Techniques, 1993, vol. 3, no. 6, pp 17-23. U.S.A.

Tiley, Ed. ***Tricks of the Windows 3.1 Masters.*** Howard W. Sams and Company, 1992. U.S.A.

Tuker, Allen B. ***Programming Languages.*** McGraw-Hill International, 1986. U.S.A.