Escuela Superior Politecnica
Del Litoral

ESCUELA DE COMPUTACION

Sistema Gráfico de Proyección Astronómica HIPARCO

proyecto:

Previo a la Obtención del Título de
ANALISTA DE SISTEMAS

Presentado por:

Míguel López V.

Director:

Anlst. Jorge Lombeida

Manual de

USUARIO / SISTEMA

Guayaquil - Ecuador

1989

Agradecimiento

Como no quiero utilizar cinco páginas sólo para mencionar los nombres de todas las personas que, de una u otra forma, han colaborado en la realización de este trabajo, voy sólo a resaltar a quien considero el principal responsable de que esta obra sea realidad: mi profesor, Analista Jorge Lombeida.

A Isabel, Miguel, Jorge, Matilde, Zaida, Hugo, Pablo, Walter, Juan, Maritza, y todos aquellos a quienes les importó más mi vida que a mí mismo.

Declaración Expresa

La responsabilidad de los hechos, ideas y doctrinas expuestas en esta tesis me corresponden exclusivamente; y el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL.

Miguel López Vite.

Anlst. Jorge Lombeida.
Director de Tesis

Manual de Usuario

Tabla de Contenido

1. Introducción

Durante el siglo II A.C., durante la época del pleno florecimiento de la ciencia griega, en que los sucesores de Alejandro Magno gobernaban desde Persia hasta el Alto Egipto, vivió y trabajó en Alejandría un filósofo y astrónomo llamado Hiparco. Sus aportes a la ciencia astronómica fueron quizás los más importantes de toda su época, junto con los de Aristóteles y Eratóstenes.

Aparte del hecho de haber inventado la trigonometría, y de haber descubierto la precesión del eje terrestre, Hiparco anticipó que las estrellas nacen, se desplazan lentamente en el transcurso de los siglos, y al final perecen. Fue el primero en catalogar las posiciones y magnitudes de las estrellas, de lo cual creó el antecesor de todos los mapas astronómicos actuales. El *Almagesto*, escrito por Tolomeo en el siglo II D.C., y basado casi enteramente en las teorías de Hiparco, fué el libro básico de la astronomía durante más de mil años. Desgraciadamente, este libro sentó las bases de la errónea creencia de que la tierra se encontraba en el centro de la bóveda celeste, lo cual demuestra que la antiguedad de un concepto no demuestra que sea correcto.

El autor de este sistema siempre ha sentido gran admiración por los logros de los científicos jonios, y ha querido en esta oportunidad demostrar un pequeño homenaje al padre de la cartografía estelar. El Sistema Gráfico de Proyección Astronómica Hiparco es el resultado de un muy largo esfuerzo en producir una herramienta sencilla y elegante para el estudio de las estrellas y otros objetos celestes relacionados.

Igualmente importante, el ejercicio de programación que constituye su estructura interna puede servir de ejemplo a otros analistas que deseen conformar aplicaciones de mayor o menor envergadura. Los científicos de la escuela pitagórica sostenían fervorosamente que el pensamiento puro era lo único capaz de discernir la forma real del universo, y que las expresiones abstractas de la ciencia eran el único camino a la perfección.

Tal vez si Pitágoras, Aristarco e Hiparco hubieran podido programar, en el presente el lenguaje favorito de los científicos y técnicos no sería el inglés, sino el griego.

Requerimientos

Hiparco necesita como mínimo de lo siguiente para funcionar correctamente:

- Un computador IBM PC, XT, AT, PS/2 o 100% compatible con los mencionados. Dado que el software utiliza ex-

clusivamente modos gráficos, se recomienda que se utilice un procesador rápido (8 Mhz o más).

- Una unidad de diskette de 360Kb.

- 512 Kb de memoria RAM (la cantidad exacta puede variar dependiendo de la cantidad de datos a utilizarse y de la tarjeta de gráficos instalada).

- Una tarjeta o adaptador de gráficos CGA, EGA, MCGA, VGA, Hércules o alguna 100% compatible con las mencionadas.

- Sistema operativo DOS, versión 2.10 o superior.

2. Visión General

2.1. Orientación general sobre Hiparco

Hiparco es un Sistema Gráfico de Proyección Astronómica, cuyo propósito es doble:

- Como una herramienta para el astrónomo aficionado o profesional, funciona como un catálogo de estrellas y constelaciones, al cual se puede accesar en forma sencilla para realizar consultas textuales o gráficas.

- Como aplicación, Hiparco constituye una demostración de las capacidades del lenguaje Pascal para realizar programas altamente modulares, independientes del hardware utilizado, y que utilicen los modernos principios de programación orientada a objetos.

El propósito de este manual es el de describir en detalle el primer aspecto mencionado: su utilización como un banco de datos astronómicos. Detalles sobre el segundo aspecto del producto se pueden encontrar en el Manual de Sistema.

2.2. ¿Quienes pueden usar Hiparco?

EL sistema Hiparco está orientado hacia los siguientes usuarios:

- Astrónomos profesionales y aficionados.
- Estudiosos del tema de la proyección tridimensional esférica.
- Cualquier persona con interés en el aprendizaje de los conceptos básicos de astronomía óptica.

2.3. Instalación

La instalación del sistema consiste en la creación de lo que se conoce como una "copia de trabajo", un duplicado de los archivos de Hiparco que usted utilizará para trabajar. *Recuerde que siempre se debe trabajar con copias de cualquier programa, no con los archivos originales; esta precaución lo protegerá contra el caso de que la copia con la que usted trabaje se dañe por cualquier motivo.*

Este proceso es sumamente sencillo, ya tenga usted conocimientos del DOS o no *. Si va a trabajar con diskettes, procúrese un diskette nuevo, inicializado y del tipo necesario para su máquina (360K, 720K, etc.); etiquete el diskette adecuadamente, como "Hiparco, copia de trabajo" o algo similar. Si va a trabajar con un disco fijo (o disco duro), sólo

* Sin embargo, y sobre todo si va a trabajar con un disco duro, es recomendable que revise su manual de DOS si encuentra algún término mencionado que no esté muy claro. Conceptos erróneos suelen tener consecuencias desagradables.

debe asegurarse de que haya suficiente espacio libre para el sistema (alrededor de 130K).

Cuando este listo para la instalación inserte el diskette etiquetado "Hiparco" en la unidad de diskette A:, e ingrese el siguiente comando para el DOS:

A:INSTALL *x:*

Donde ***x:*** corresponde a la letra de la unidad donde se va a instalar el sistema (B:, C:, etc.); no olvide incluir los dos puntos después de la letra o la instalación no procederá correctamente.

El programa de instalación empezará a copiar los archivos necesarios de la unidad A: al directorio corriente de la unidad especificada. Si se desea que el sistema sea instalado en un directorio específico, distinto del corriente, incluya el nombre del directorio después de la letra de la unidad, separado de ésta por uno o más espacios; por ejemplo:

A:INSTALL C: \HIPARCO

El directorio será creado si no existía previamente; en caso de que ya exista, ignore el mensaje "**Unable to create directory**".

Ejemplos: Si tiene usted una computadora con dos unidades de diskette (A: y B:), coloque el diskette original de Hiparco en la unidad A: y un diskette vacío en la unidad B:, digite lo siguiente y presione la tecla [Enter]:

A:INSTALL B:

Si tiene usted una computadora con una unidad de diskette y una unidad de disco duro (A: y C: respectivamente) y desea instalar Hiparco en el subdirectorio \GRAPH\STARS, coloque el diskette original de Hiparco en la unidad A:, digite lo siguiente y presione la tecla [Enter]:

A:INSTALL C: \GRAPH\STARS

Se recomienda que, si se instala el sistema en un disco duro, se lo instale en un subdirectorio, a fin de no mezclar los archivos de Hiparco con los de otros programas.

ATENCION: Si usted tiene sólo una unidad de diskette no se podrá ejecutar el programa de instalación. En este caso siga las instrucciones dadas en el Apéndice 6.1 (Pag. 23) para crear una copia del diskette original.

3. Estructura General del Sistema

3.1. Como iniciar

Una vez que el sistema esté instalado (Véase 2.3., Pag. 3), para empezar a utilizarlo solo debe ejecutar los siguientes pasos:

- Si instaló el sistema en un diskette, inserte el diskette etiquetado "copia de trabajo" en la unidad corriente. Si instaló el sistema en un disco duro, haga que éste sea la unidad corriente escribiendo su letra, dos puntos y presionando [Enter]

- Si instaló el sistema en un directorio, haga que éste sea el directorio corriente escribiendo **CD**, luego el nombre del directorio y presionando [Enter]

- Inicie el sistema escribiendo **HIPARCO** y presionando [Enter]

Por ejemplo, si instaló el sistema en el disco duro C: y en el subdirectorio \HIPARCO, escriba lo siguiente (presione [Enter] al final de cada línea):

```
C:
CD \HIPARCO
HIPARCO
```

Después de unos momentos aparecerá un mensaje indicándole que los datos del sistema están siendo inicializados. Luego, verá la pantalla principal de Hiparco, que se puede apreciar en la figura 1.

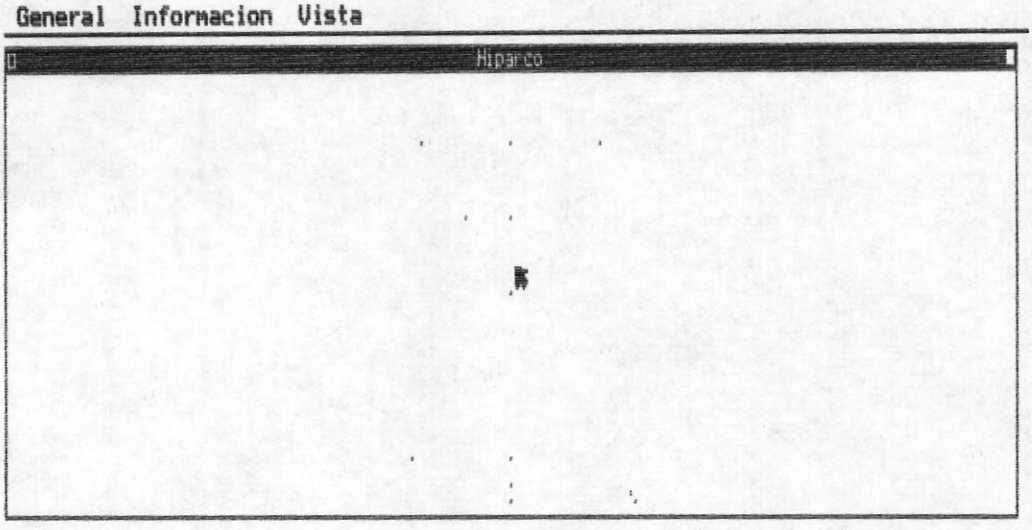


Figura 1. Pantalla inicial de Hiparco.

Como podemos notar, la pantalla se divide en dos áreas claramente diferenciadas:

- El área del menú, y

- El área de información, o de "ventanas"

Ambas áreas están divididas por una línea horizontal.

3.2. Menú general

El menú general consiste en la línea de texto situada en la primera fila de la pantalla. Las palabras coresponden a las categorías generales de actividades, bajo las cuales se agrupan las diferentes opciones del programa.

Usted puede activar el menú general y seleccionar la opción que desee con cualquiera de los siguientes métodos:

1) Presione la tecla [Esc], luego seleccione la opción utilizando las teclas de flechas (o de movimiento de cursor) situadas a la derecha del teclado, y presione [Enter] para ejecutar la opción; o

2) Presione la primera letra de la categoría, y luego la primera letra de la opción deseada.

Para mas detalles sobre el uso de las teclas, consulte el Capítulo 4 (Pag. 9). Para una descripción detallada de cada una de las opciones disponibles, consulte el Capítulo 5 (Pag. 12).

3.3. Ventanas

Todo el espacio por debajo del menú general está destinado a "Ventanas". Denominamos *Ventana* a un área rectangular de la pantalla, la cual se utiliza para intercambiar información entre el computador y usted.

Como ejemplo, en el momento de comenzar Hiparco, se "abre" una ventana para informarle que los datos están siendo inicializados, la misma que se "cierra" cuando la fase de inicialización ha terminado.

Asimismo, durante todo el tiempo en que Hiparco esté activo, la parte inferior de la pantalla estará ocupada por una ventana que mostrará las posiciones de las estrellas vistas desde el punto de observación fijado. Cuando otras ventanas aparecen "encima" de ésta, el color del nombre de la ventana cambia; usted siempre puede distinguir cuál es la ventana "activa" en un momento dado simplemente fijándose en los colores de los nombres de las ventanas.

Las funciones de las ventanas son de dos tipos:

1) Presentar mensajes; y

2) Solicitar información adicional para continuar una operación.

La mayoría de las ventanas presentan en la parte inferior opciones que le permiten ejecutar operaciones relacionadas con el contenido de la ventana; denominamos a estas opciones *botones* y se seleccionan de la misma forma que las opciones del menú general: ya sea posicionándose con las teclas de flechas o digitando la primera letra de su nombre (Véase el Capítulo 4, Pag. 9).

3.4. Selector

Al comenzar el programa, en la parte central de la pantalla, notará un símbolo similar a una flecha que apunta hacia arriba. Este es el *selector* que le permite indicar que punto del "cielo" le interesa. Usted mueve el selector con las teclas de flechas y presiona [Ins] para *marcar* el sitio sobre el que desee información.

Como puede apreciarse en la Figura 2, para señalar el sitio marcado se utiliza un símbolo similar a un óvalo; éste se denomina *selección*, o simplemente *marca*. El objeto que queda delimitado por la marca (o el más cercano a ella) será utilizado la siguiente vez que se ejecute una orden de ***Información***.

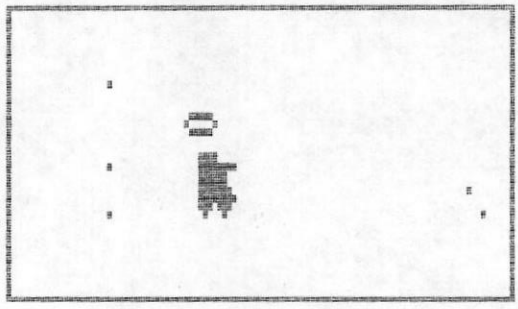


Figura 2. *Imagen ampliada del selector y de la marca de selección (en la parte superior.*

Presionar la tecla [Del] tiene el efecto opuesto a [Ins], es decir, oculta la marca. Aunque no siempre la marca es visible (sólo es visible si se presiona [Ins]) siempre existirá una posición seleccionada; por lo general ésta corresponde a la posición del selector.

4. Uso del Teclado

Se ha procurado que las teclas utilizadas por el programa sean lo más sencillas posible. Cada operación en el sistema suele tener al menos dos secuencias equivalentes de teclas para ser ejecutada; usted es libre de utilizar la forma que más le agrade en cualquier momento, o de combinar los distintos modos de ejecución.

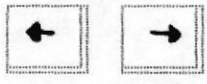
En los teclados de la computadoras tipo PC, las teclas se dividen en tres grupos: teclas funcionales, teclas alfabéticas, y teclas de movimiento de cursor.

- Las teclas funcionales se suelen encontrar formando dos columnas en el extremo izquierdo del teclado, o una fila en el extremo superior. Estas teclas no son utilizadas en la versión corriente de Hiparco.

- Las teclas alfabéticas se encuentran en la parte central del teclado, dispuestas en forma similar a las de una máquina de escribir.

- Las teclas de movimiento de cursor se encuentran en la parte derecha del teclado, formando lo que suele llamarse un *teclado numérico* (similar en forma al teclado de una calculadora de escritorio); éstas son las teclas más utilizadas en Hiparco, en parte debido a que su disposición hace que puedan ser manejadas rápidamente con una sola mano.

4.1. [flecha izquierda], [flecha derecha]



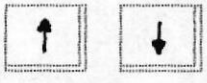
Se encuentran en el teclado numérico, en las posiciones marcadas por los números 4 y 6.

- Mueven el selector en la dirección de la flecha respectiva.

- En el menú general, permiten cambiar de submenú.

- Cuando hay varios botones en una ventana, permiten seleccionar un botón en la dirección de la flecha.

4.2. [flecha arriba], [flecha abajo]



Se encuentran en el teclado numérico, en las posiciones marcadas por los números 8 y 2.

- Mueven el selector en la dirección de la flecha respectiva.

- En el menú general, permiten cambiar de opción dentro de un submenú.

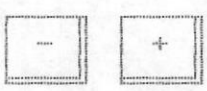
4.3. [Enter]

Se encuentra a la derecha de las teclas alfabéticas, entre éstas y el teclado numérico. En algunos teclados está etiquetada como [Return]. Dado que es una de las teclas más utilizadas, algunos teclados tienen una segunda tecla [Enter] a la derecha del teclado numérico.

- Ejecuta la opción ***Objeto*** del menú de ***Información*** (equivale a presionar [Ins] dos veces sobre un objeto).

- Si el menú está activo, o hay botones en la ventana, ejecuta la opción/botón seleccionada con las flechas.

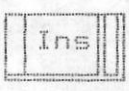
4.4. [-],[+]



Se encuentran a la derecha del teclado numérico, una encima de la otra. Tienen teclas equivalentes en la primera fila de las teclas alfabéticas.

- Permiten decrementar e incrementar, respectivamente, la velocidad del selector.

4.5. [Insert]



Se encuentra en la parte inferior derecha del teclado, justo debajo del teclado numérico, en la posición marcada por el número 0.

- Marca el área donde el selector se encuentra. Si hay una marca previa en otro sitio de la pantalla, ésta será removida. Si el selector se encuentra sobre una marca visible, se ejecuta la opción ***Objeto*** del menú de

Información (equivale a presionar [Enter] sobre un objeto).

- Si el menú está activo, o hay botones en la ventana, funcionará igual que [Enter].

4.6. [Delete]

Se encuentra en la parte inferior derecha del teclado, justo debajo del teclado numérico, en la posición marcada por el punto decimal.

- Oculta la marca de selección si ésta es visible. Tiene el efecto opuesto a [Ins]. La tecla no tiene efecto si la marca es invisible.

4.7. [Escape]

En la mayoría de los teclados se encuentra en la zona superior izquierda, aunque en algunos modelos se encuentra entre las teclas alfabéticas y el teclado numérico, formando la esquina superior izquierda de éste último.

- Activa y desactiva, alternativamente, el menú general (si está activo, lo desactiva y viceversa).

- Fuerza a una ventana activa a "cerrarse" (con la excepción de la ventana principal).

4.8. Letras

Constituyen la parte central de las teclas alfabéticas.

- Las teclas de letras pueden ser utilizadas en vez de las teclas de flechas y [Enter] para ejecutar opciones del menú o de botones. Para esto, digite la primera letra de la opción deseada (que se encuentra en mayúsculas); por ejemplo, para ejecutar la opción ***Posición*** del menú ***Vista***, presione [V] (para activar el submenú Vista) y luego [P] (para activar la opción Posición).

5. Menús y Opciones

Hiparco se controla por medio de un sistema de opciones, ordenadas bajo lo que denominamos un *menú general*. En el menú las opciones están relacionadas bajo categorías, los nombres de las cuales se encuentran listados en la primera línea de la pantalla. Para ejecutar una opción, usted selecciona primero la categoría, y luego la opción deseada.

Existen dos formas principales de seleccionar categorías y opciones:

- Presione la tecla [Esc], la cual selecciona la última categoría utilizada. Luego utilice las flechas izquierda y derecha para cambiar de categoría, si es necesario, y las flechas arriba y abajo para seleccionar una opción. Luego, presione [Enter] o [Ins] para ejecutar la opción seleccionada.

- Presione la primera letra de la categoría que desea seleccionar, y luego la primera letra de la opción que desea ejecutar. Este método suele ser más efectivo que el primero una vez que se ha aprendido la organización del menú.

Ambos métodos pueden ser mezclados si usted lo desea. Puede, por ejemplo, seleccionar una categoría presionanado su letra inicial, y luego seleccionar la opción utilizando las flechas.

Algunas opciones del submenú de Información sólo son válidas cuando hay un objeto seleccionado (Véase la explicación sobre la categoría **Información**, 5.2., Pag. 14).

5.1. General

Como se ve en la figura 3, ésta categoría contiene sólo dos entradas, aunque la segunda de ellas (***Fin***) es una de las más utilizadas.

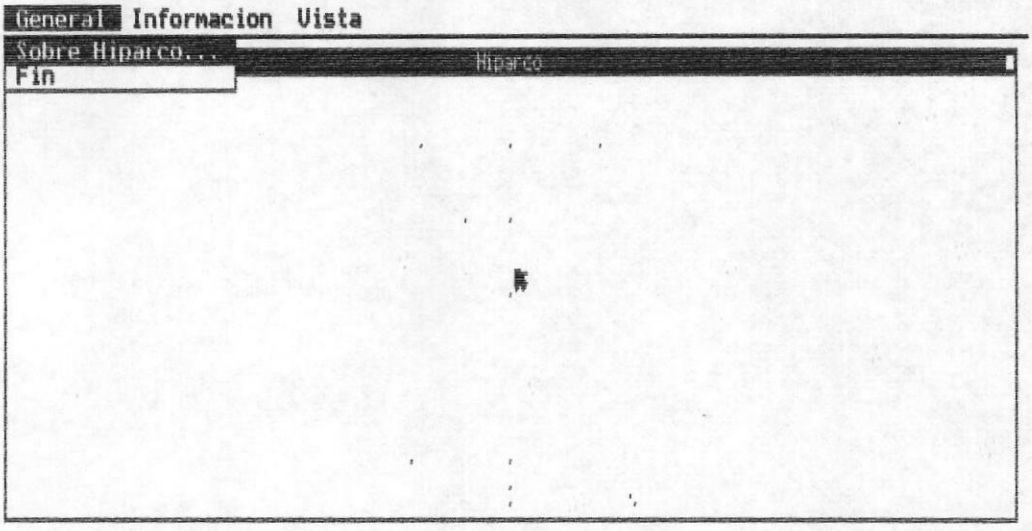


Figura 3. Submenú de la categoría General.

5.1.1. Sobre Hiparco

Esta opción presenta una ventana describiendo el nombre y propósito del sistema, junto con el nombre del autor del mismo (Véase la figura 4).

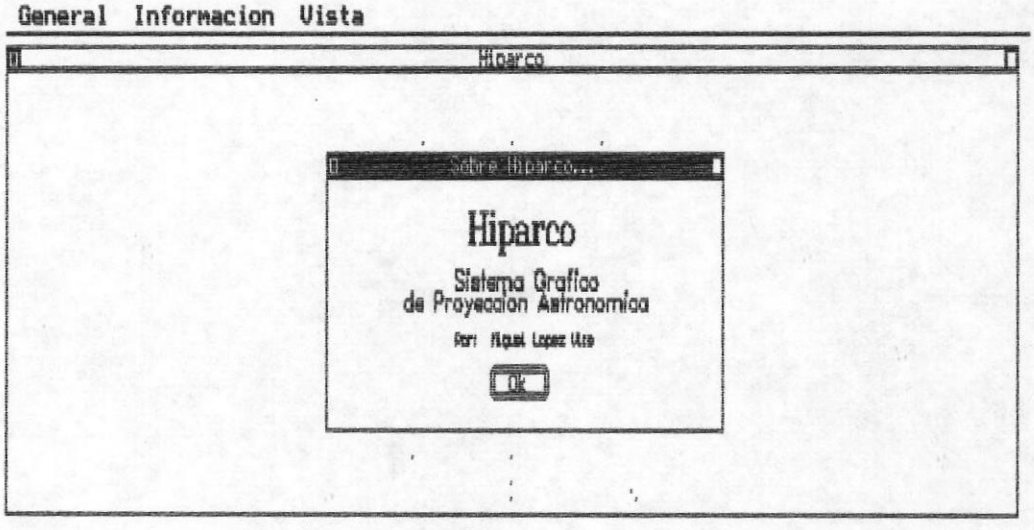


Figura 4. Ventana de descripción del sistema.

En la parte inferior hay un botón etiquetado "Ok". Presiónelo con la tecla [Enter] o [Ins] para retornar a la ventana principal.

5.1.2. Fin

Esta opción termina le ejecución del programa y retorna el control al DOS.

5.2. Informacion

Esta categoría engloba a las opciones que le permiten obtener información sobre alguno de los objetos (estrellas) o grupos (constelaciones) presentes en la pantalla. Podemos ver estas opciones en la figura 5:

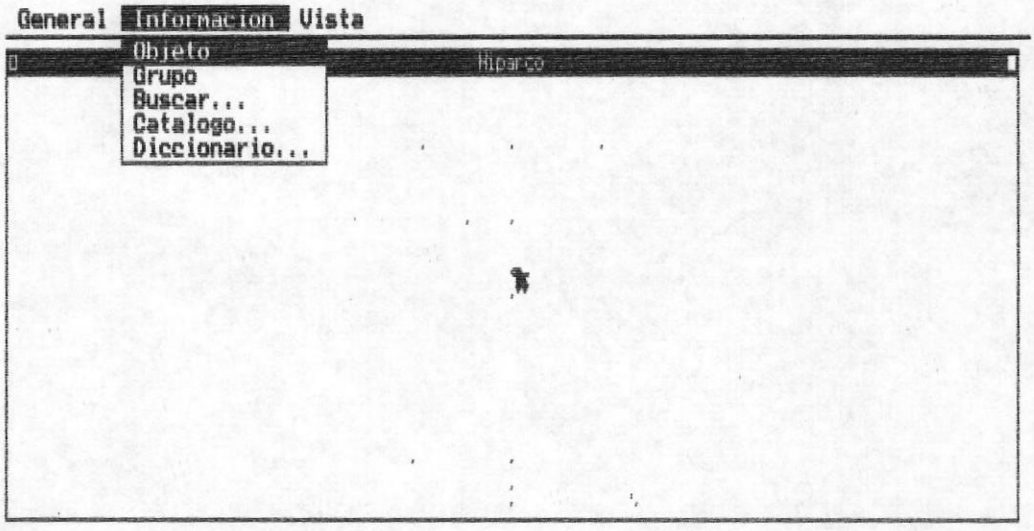


Figura 5. *Submenú de la categoría Información.*

Es probable que usted note, cuando active el menú de Información, que las dos primeras opciones se encuentran "cortadas" con una línea horizontal, y que no es posible seleccionarlas. Estas opciones (***Objeto*** y ***Grupo***) sólo son válidas cuando la marca de selección es visible (presione [Ins] para visualizar la marca). Una ampliación del aspecto del submenú se puede apreciar en la Figura 6; nótese que en dicha figura la opción seleccionada es ***Buscar***.

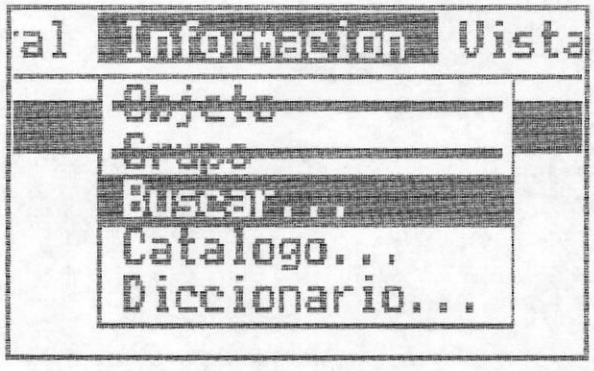


Figura 6. *Opciones inhabilitadas en el submenú de Información.*

5.2.1. Objeto

Esta opción presenta información astronómica sobre el objeto (estrella) seleccionado. Por ejemplo:

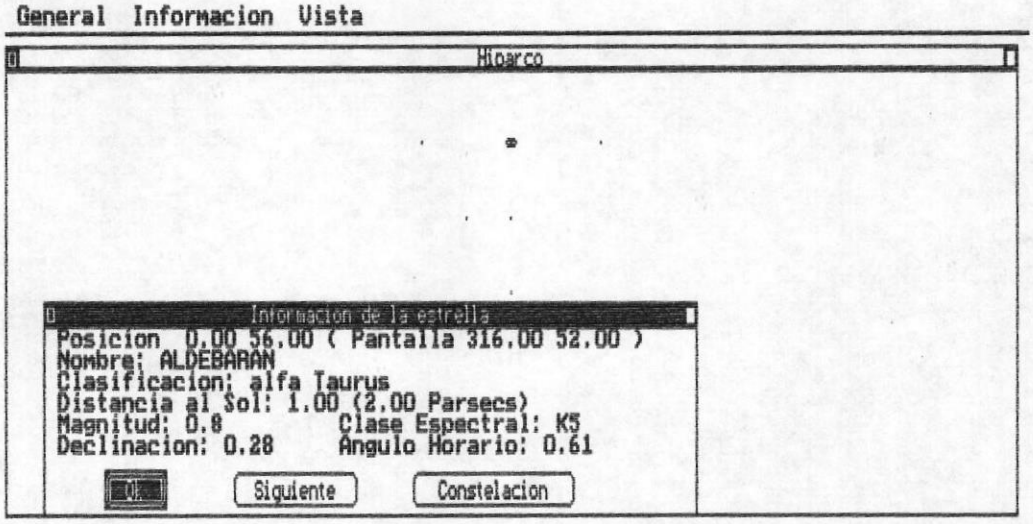


Figura 7. Información de un objeto astronómico.

La información presentada es la siguiente (para descripciones detalladas de algunos términos técnicos, véase el Apéndice 6.3, Pag. 25):

Posición: Presenta la posición (X,Y) del objeto con relación a las coordenadas de la bóveda celeste, cuyo origen queda en la intersección del meridiano 0 terrestre con el ecuador celeste. También presenta la posición relativa del objeto en la pantalla (tomando como origen la esquina superior izquierda).

Nombre: Identificación dada al objeto por la Asociación Astronómica Internacional.

Clasificación: Código formado por la categoría de brillantez del objeto, en letras griegas, y el nombre latino de la constelación a la que pertenece. Por ejemplo, la estrella más brillante de la constelación de Tauro se denomina *alfa Taurus* (véase la figura 7).

Distancia al Sol: Distancia aproximada del objeto al Sol, tanto en años luz como en parsecs.

Magnitud: Escala de brillantez del objeto.

Clase: Clasificación del objeto dentro del diagrama Hetzsprung-Russell.

Declinación: Angulo Norte del objeto con respecto al ecuador celeste.

Angulo Horario: Angulo Este del objeto con respecto al meridiano O celeste.

En la parte inferior de la ventana hay tres botones:

- Ok
- Siguiente
- Constelación

Las acciones que realizan son:

Ok: Finaliza la operación.

Siguiente: Ejecuta la operación para la siguiente estrella en secuencia. Para esto el programa busca la estrella más cercana a la mostrada (en términos de distancia espacial, no de distancia de pantalla) y presenta su información.

Constelación: Presenta la información del grupo o constelación a la cual pertenece el objeto (véase la descripción de la opción ***Grupo***, 5.2.2., Pag. 16).

Si la marca no se encuentra sobre un objeto válido, se mostrará el mensaje "No hay datos disponibles". En este caso, se puede utilizar el botón "Siguiente" para localizar la estrella deseada.

5.2.2. Grupo

Esta opción presenta información astronómica sobre el grupo (constelación) a la que pertenezca el objeto seleccionado, o en su defecto el objeto más cercano a la marca de selección. Como ejemplo, en la figura 8 tenemos la información del grupo de Orión, al cual pertenece la estrella marcada en la parte inferior de la pantalla.

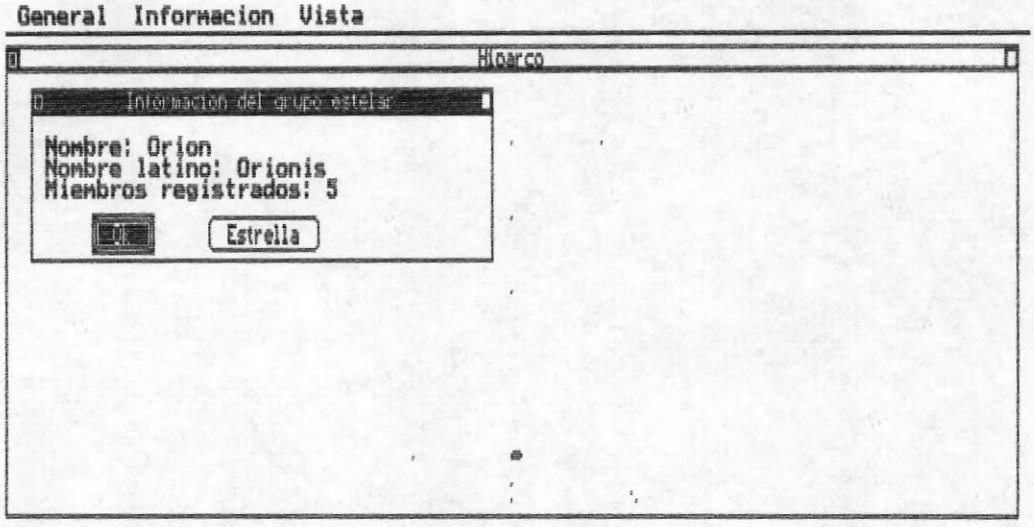


Figura 8. *Información de una agrupación de objetos.*

La información presentada es:

Nombre: Identificación dada a la constelación por la Asociación Astronómica Internacional.

Nombre latino: Nombre original (en latín) de la constelación.

Miembros registrados: Número total de objetos pertenecientes a este grupo. Este dato es dependiente de los objetos registrados en el archivo de Hiparco, no del número real de estrellas en la constelación.

En la parte inferior aparecen los siguientes botones:

- Ok
- Estrella

Estos tienen los siguientes efectos:

Ok: Finaliza la operación.

Estrella: Presenta la información de la estrella más cercana a la posición de la marca (véase la descripción de la opción ***Objeto***, 5.2.1, Pag. 15).

5.2.3. Buscar

Esta opción le permite buscar una estrella por medio de su nombre. Cuando la ejecuta, aparece una ventana solicitándole el nombre de la estrella a buscar (véase la figura 9).

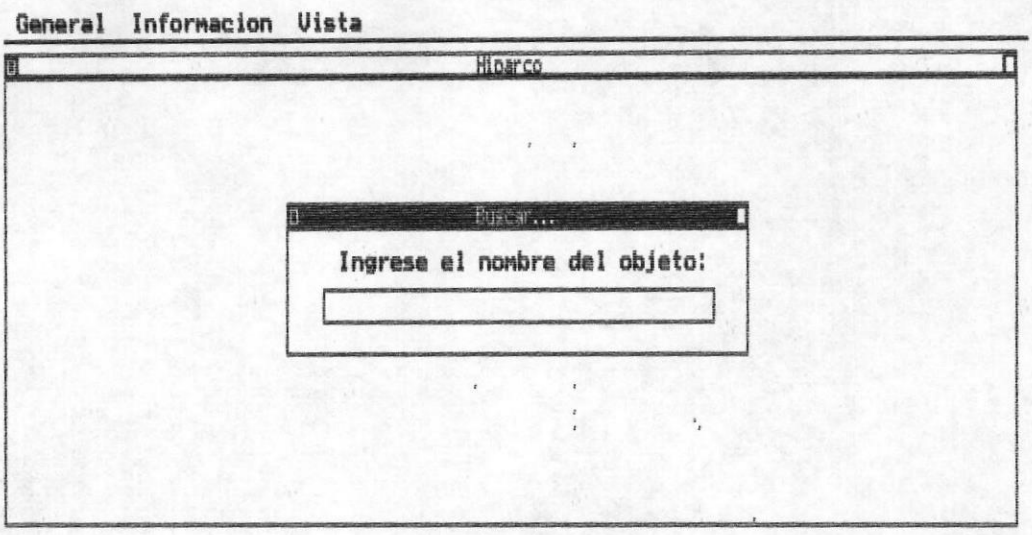


Figura 9. Campo para ingresar el nombre del objeto a buscar.

Si comete errores, puede corregirlos utilizando la tecla [BackSpace] (la tecla gris con una flecha izquierda, generalmente situada encima de [Enter]). Cuando haya terminado de escribir, presione [Enter].

Si el objeto no puede ser encontrado, aparecerá un mensaje informándoselo, junto con un botón etiquetado "Ok"; presione [Enter] o [Ins] para continuar. Si el objeto es encontrado, se presentarán sus datos respectivos tal y como si se hubiera ejecutado la opción ***Objeto*** (véase 5.2.1, Pag. 15).

5.2.4. Catalogo

Esta opción le presenta una ventana con un índice o catálogo de todos lo grupos presentes en el archivo de datos (véase la figura 10).

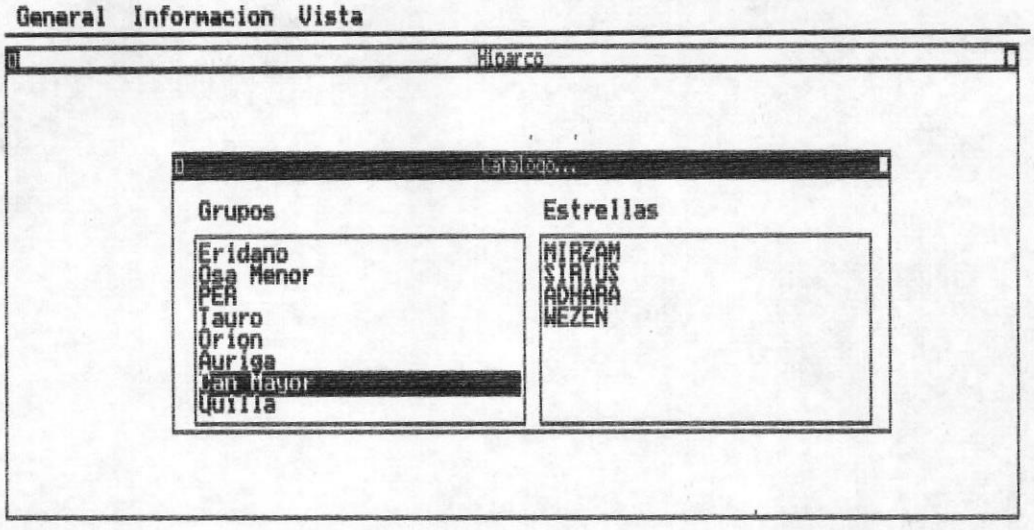


Figura 10. Catálogo de agrupaciones y objetos.

La ventana se encuentra dividida en dos partes: a la izquierda se encuentran los nombres de las constelaciones y a la derecha los nombre de las estrellas que componen la constelación seleccionada. Usted puede escoger la categoría (grupo u objeto) sobre la que desee información utilizando las flechas izquierda y derecha. seleccione el nombre deseado con las flechas arriba y abajo. Presione [Enter] o [Ins] para abrir una ventana que le mostrará la información pertinente (véase la figura 11). El formato de los datos presentados es similar al de la opción de ***Objeto*** (véase 5.2.1., Pag. 15).

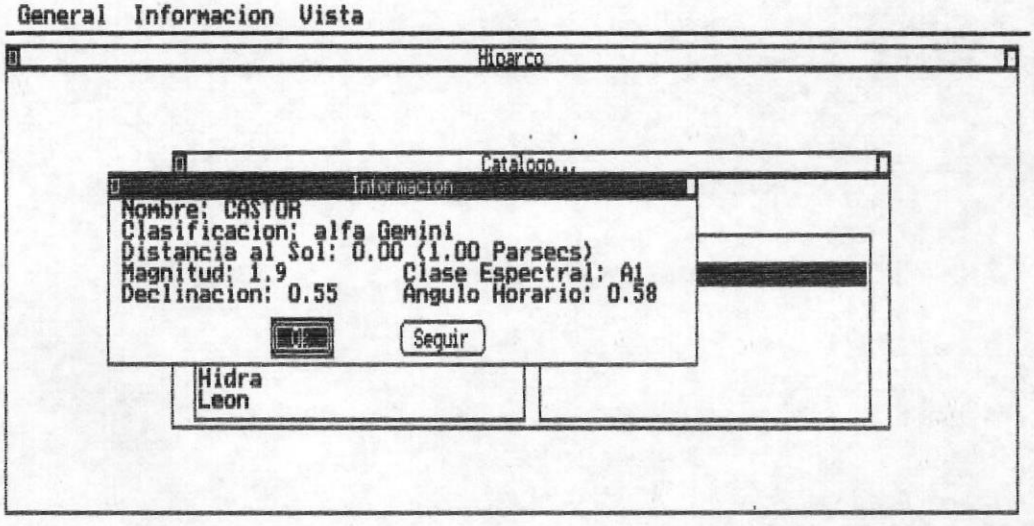


Figura 11. Información de un objeto, solicitada por medio del catálogo.

En la parte inferior de la ventana hay dos botones:

- Ok
- Seguir

Sus significados son:

Ok: Termina la operación.

Seguir: Cierra la ventana de información y le permite seguir consultando el catálogo

5.2.5. Diccionario

Esta opción sirve para que usted puede consultar una referencia rápida sobre algunos términos utilizados frecuentemente en el programa. Al seleccionarla se abrirá una ventana mostrando una lista alfabética de los términos en el diccionario, tal y como se puede apreciar en la figura 12.

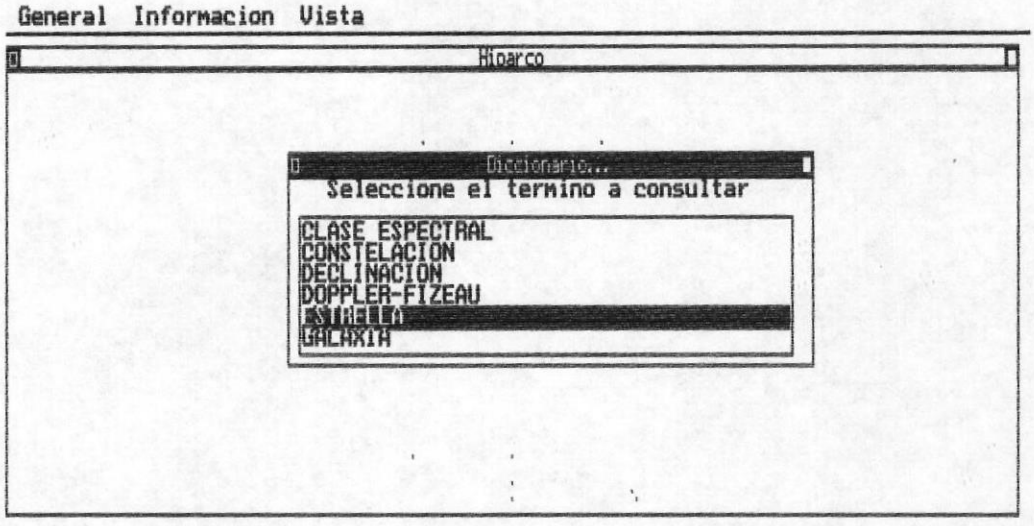


Figura 12. Diccionario de términos astronómicos.

Seleccione el término que desea consultar utilizando las flechas arriba y abajo, y luego presione [Enter] o [Ins]. Una amplia ventana se abrirá para mostrarle la definición del término.

En la parte inferior de dicha ventana hay cuatro botones:

- Ok
- Seguir
- Avanzar
- Retroceder

Sus acciones respectivas son:

Ok:	Termina la operación.
Seguir:	Cierra la ventana de explicación y le permite seguir consultando el diccionario.
Avanzar:	Muestra la explicación del siguiente término (en orden alfabético) del diccionario.
Retroceder:	Muestra la explicación del término anterior (en orden alfabético) del diccionario.

5.3. Vista

Bajo esta categoría se encuentran todas las operaciones que afectan el aspecto del "trozo" de cielo visible en la pantalla. Estas se pueden apreciar en la figura 13.

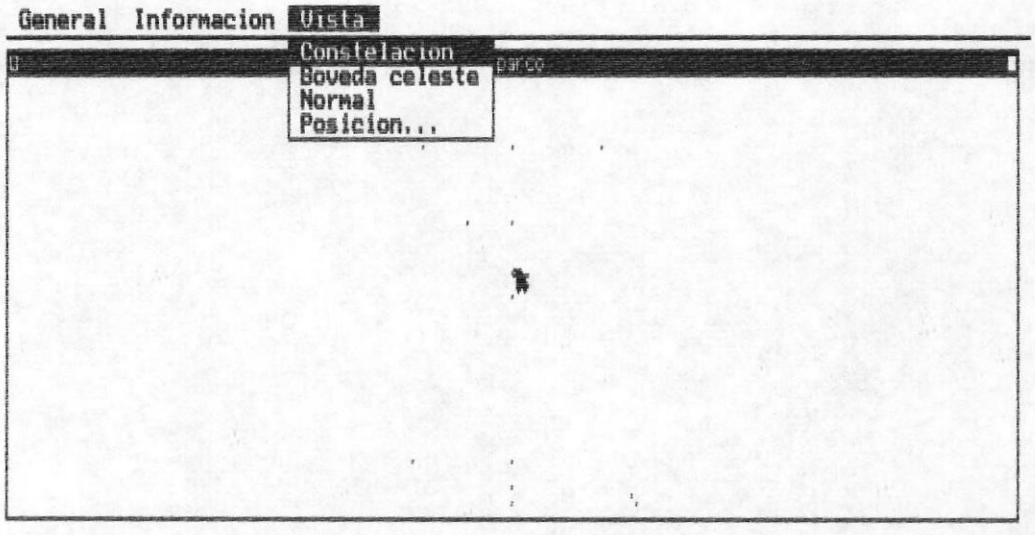


Figura 13. *Submenú de la categoría Vista.*

5.3.1. Constelacion

Esta opción dibuja en la pantalla el contorno de la constelación mas próxima a la marca de selección. El contorno, delimitado por una línea punteada, permanece activo hasta que se efectúe otra opción de ***Vista***.

5.3.2. Bóveda Celeste

Esta opción es similar a la anterior, con la diferencia de que se dibujan los contornos de todas las constelaciones vi-

sibles. Dichos contornos permanecen visibles hasta que se ejecute otra opción de vista (excepto contelación).

5.3.3. Normal

Esta opción simplemente vuelve a dibujar la ventana en su modo normal. Utilice esta opción para anular los efectos de las opciones Constelación y/o Bóveda Celeste.

5.3.4. Posición

Esta opción le permite "deslizar" la ventana principal por "sobre" la Bóveda Celeste. La ventana se encuentra inicialmente orientada de tal manera que el centro de la pantalla coincide aproximadamente con la coordenada 0,0 *. Para visualizar otra parte del "cielo" usted debe indicar un desplazamiento (en términos de X,Y) de la nueva posición con respecto a este origen (como ejemplo, véase la figura 14).

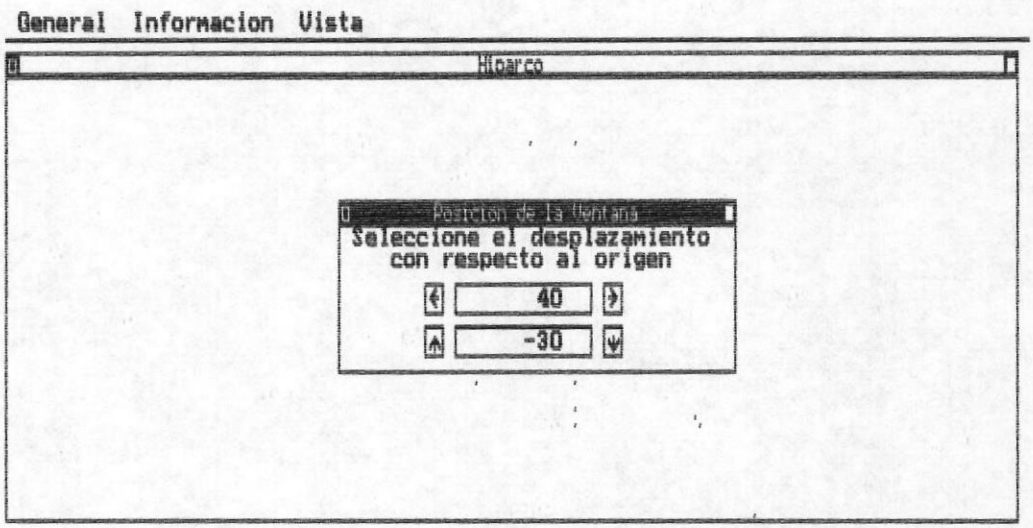


Figura 14. *Ventana para ajuste de posición.*

Cuando seleccione esta opción se abrirá una ventana mostrándole el desplazamiento actual de la ventana principal con respecto al origen. Utilice las flechas izquierda y derecha para cambiar el valor del desplazamiento X; utilice las flechas arriba y abajo para cambiar el valor del desplazamiento Y.

Los incrementos tanto de X como de Y son en múltiplos de 10.

* Es decir, el punto de intersección del meridiano cero terrestre con el Ecuador Celeste.

6. Apéndices

6.1. Creación de un duplicado del diskette original

El programa de instalación proporcionado con Hiparco no puede ejecutarse si sólo se dispone de una unidad de diskette (véase 2.3., Pag. 3). Si éste es su caso, puede seguir las siguientes instrucciones para crear una copia de trabajo:

- Prepare un diskette en blanco (vacío) donde colocar la copia. Procure también tener a la mano el diskette de Hiparco y un diskette con el programa DISKCOPY (suele estar en el diskette de DOS).

- Coloque el diskette con DISKCOPY en su unidad, digite **DISKCOPY** en el "prompt" y presione [Enter].

- Cuando se presente el mensaje "Insert SOURCE diskette in drive" inserte el diskette de Hiparco y presione [Enter].

- Cuando se presente el mensaje "Insert TARGET diskette in drive" inserte el diskette vacío y presione [Enter].

- Es posible que los dos pasos anteriores tengan que repetirse algunas veces hasta que la copia esté completa. Cuando el mensaje "Copy another diskette?" aparezca, el procesao habrá terminado. Presione [N] y [Enter] para regresar al "prompt".

Cuando haya creado la copia de trabajo, podrá ejecutar el sistema simplemente insertando dicha copia en la unidad de diskette, digitando **HIPARCO** y presionando [Enter].

Para mayor información sobre el uso del programa DISKCOPY, refiérase al manual del DOS.

6.2. Contenido del diskette

A continuación se presenta una lista de los archivos incluidos en el diskette original de Hiparco, junto con una descripción del propósito de cada uno. Se recomienda que se utilice el comando DIR para verificar que el contenido de su copia sea correcto.

Nombre	Descripción
HIPARCO .EXE	Programa principal de Hiparco.
STARDATA.HIP	Archivo con datos de las estrellas; debe residir en el mismo directorio que HIPARCO.EXE
DICTION .HIP	Archivo con datos del diccionario; debe residir en el mismo directorio que HIPARCO.EXE
????????.BGI	Archivos con datos de varios adaptadores gráficos; deben residir en el mismo directorio que HIPARCO.EXE
????????.CHR	Archivos con datos de los "fonts" o tipos de letras utilizados; deben residir en el mismo directorio que HIPARCO.EXE
INSTALL .BAT	Programa de instalación.
FUENTES .ARC	Programas fuentes de Hiparco, almacenados en formato empaquetado. Utilice ARCE.COM para extraer los programas.
MANUALES.ARC	Manuales de Hiparco, almacenados en formato empaquetado. Utilice ARCE.COM para extraer los documentos.
ARCE .COM	Programa para extraer la información empaquetada en FUENTES.ARC y MANUALES.ARC
ARCE .DOC	Documentación original de ARCE.COM (en inglés).
LEAME	Texto con cambios e informaciones de último minuto, no incluidos en los manuales. Se recomienda utilizar el comando TYPE para inspeccionar su contenido.

6.3. Glosario

6.3.1. Agujero negro

Objeto formado por el colapso gravitacional de una estrella de masa superior a 3 veces la masa del sol, aproximadamente. El colapso ocurre cuando la presión de la gravedad comprime a la estrella de tal forma que su radio llega a ser menor a cierto límite, llamado radio Schwarzschild, dependiente del tamaño de la estrella. Cuando esto ocurre la velocidad de compresión alcanza la velocidad de la luz. La distorsión espacio-temporal resultante causa que nada que se encuentre o caiga dentro de dicho radio pueda abandonar el área.

Existen muchas pruebas matemáticas de la existencia de los agujeros negros, pero no se ha descubierto hasta ahora ninguno.

6.3.2. Angulo horario (ascensión)

Angulo que forma una estrella con respecto al meridiano cero celeste, el que se define como el punto donde el movimiento aparente del Sol cruza el ecuador celeste en la primavera del hemisferio boreal. Se le asigna el valor positivo a los ángulos tomados en la dirección contraria a la rotación terrestre. Es equivalente a la longitud geográfica.

6.3.3. Año Luz

Unidad de medida de la distancia estelar. Equivale a la distancia que recorre la luz en el espacio, durante un año, a la velocidad de 300.000 kilómetros por segundo: unos 9.000 millones de kilómetros.

Un año luz equivale aproximadamente a 0.43 Parsecs.

6.3.4. Bóveda Celeste

Esfera imaginaria en la cual se proyectan las posiciones y los movimientos de las estrellas. Para realizar la representación se proyectan en el cielo visible los meridianos del globo terrestre para formar un sistema de coordenadas, sobre el cual se representan los astros en forma de puntos de mayor o menor tamaño segun su luminosidad. Los movimientos de la tierra y el sistema solar se representan como movimientos de la esfera completa.

La bóveda celeste se utiliza principalmente para contruir mapas astronómicos.

6.3.5. Clase Espectral

Sistema de clasificación de estrellas, con base a la cantidad de energía que emiten. De la más caliente a la mas fría,

las estrellas se agrupan en una de las clases siguientes: W, O, B, A, F ,G, K, M, R, N, S. A su vez, cada una de las clases se divide en diez subgrupos, numerados del 0 al 9. A cada clase corresponde aproximadamente un color característico, desde el azul oscuro de las W y O hasta el rojo intenso de las S.

El Sol es de la clase espectral G2, y su color más característico es el amarillo.

6.3.6. Constelación

Agrupación visual de estrellas. Los antiguos astrónomos y astrólogos griegos, babilonios y mayas vislumbraron imágenes en el cielo estrellado, las cuales asociaron con historias de su mitología y con escenas de su vida cotidiana. Los nombres con que bautizaron a dichas imagenes se han conservado hasta el día de hoy: Pegaso, Cisne, Acuario, etc. Modernamente, la Asociación Astronómica Internacional reconoce 88 constelaciones.

Para Catalogar las estrellas, se utiliza un sistema que asigna una letra griega a cada una de las estrellas principales, por orden de brillantez aparente (de mayor a menor). Así, por ejemplo, la estrella más brillante de la constelación del Centauro se denomina Alfa del Centauro o *Alfa Centauri*, si se utiliza la mas común denominación latinizada.

6.3.7. Declinación

Angulo que forma una estrella con respecto al ecuador celeste, es decir, con la proyección de la línea ecuatorial terrestre sobre la bóveda celeste. Se le asigna el valor positivo a los angulos entre el ecuador celeste (0 grados) y el polo norte celeste (90 grados), y negativo a los ángulos al sur del ecuador celeste. Es equivalente a la latitud geográfica.

6.3.8. Doppler-Fizeau

Efecto del movimiento de un astro sobre las ondas luminosas que este proyecta. Este efecto es comparable al de las ondas de sonido de un cuerpo en movimiento: dichas ondas se comprimen frente al objeto y se expanden tras el mismo. Asimismo la luz de una extrella sufre una compresión en la dirección de su movimiento lo que se traduce en un aumento de la radiación azul en su espectro. La luz también se expande en dirección contraria a su movimiento lo que causa un aumento de la radiación roja en su espectro.

El efecto doppler se usa para calcular la velocidad y dirección del movimiento de un astro.

6.3.9. Estrella

Concentración de gases comunes (hidrógeno en 97%, helio en 10%), formadas a partir de nubes de hidrógeno que se condensan en forma de esferas debido a su propia gravedad. La enorme presión de la materia hacia el centro de la esfera ocasiona que en éste se formen reacciones de fusión nuclear, combinándose los átomos de hidrógeno para formar helio, y liberándose energía en forma de radiaciones, las cuales, al llegar a la superficie de la esfera, se expanden en forma de ondas de radio, luz visible y rayos X.

Las estrellas tienen una vida promedio de 10.000 millones de años. La edad de una estrella se determina a partir de su color, siendo las más jovenes de color azul y las de mayor edad rojas, pasando por todos los colores del espectro. El Sol, siendo amarillo, tiene una edad aproximada de 5.000 millones de años.

6.3.10. Galaxia

Agrupación de estrellas, que giran en patrones circulares. Las galaxias tipicas tienen de 20.000 a 500.000 millones de estrellas; un ejemplo lo constituye la galaxia Via Láctea, hogar de Sol, compuesta por 100.000 millones de estrellas en forma de un disco de 50.000 años luz de radio. Esta galaxia pertenece al tipo espiral, ya que las estrellas que giran alrededor de su centro forman "brazos" espirales que irradian del centro del disco.

El Sol se encuentra a 2/3 de la distancia del centro de la galaxia al borde de la misma, justo en la frontera del brazo de Sagitario.

6.3.11. Magnitud

Unidad de medida del brillo de las estrellas. La escala utilizada asigna a las estrellas mas brillantes un valor numérico bajo y a las de escaso brillo un valor elevado. Asi, una estrella brillante como Aldebaran (Alfa Tauro) es de primera magnitud, mientras que estrellas que se encuentran en el límite de la visión directa son de magnitud sexta. La escala se ha dispuesto de tal modo que cada unidad de magnitud corresponda a una diferencia de brillo de 2.52 veces.

Las estrellas mas débiles captadas con telescopios ópticos oscilan entre las magnitudes 23 y 24. Hay también unas pocas estrellas cuyo brillo supera la primera magnitud, y para estas se utiliza una escala negativa; por ejemplo, la estrella más brillante (Sirius) es de magnitud -1.4.

6.3.12. Nebulosa

Nubes de materiales pesados, que se supone se forman hacia el final de la existencia de las estrellas. La mayoria de las nebulosas conocidas estan compuestas por compuestos complejos como hierro, magnesio, silicio, etc, los cuales se han formado en el interior de las estrellas por obra de los mismos procesos de fusión que generan el helio a partir del hidrógeno. Cuando la estrella ha superado cierta edad, ha perdido tanta masa en forma de rayos cósmicos que no poseee la gravedad suficiente para evitar que dichos compuestos escapen al espacio. Lentamente, los elementos flotan fuera de la estrella, formando una esfera de gases que puede llegar a tener algunos años luz de diámetro. Eventualmente, cuando la estrella madre desaparezca, la nebulosa continuará difuminándose lentamente, hasta que sus elementos formen el núcleo de nuevos planetoides.

6.3.13. Novas

Estrella que aumenta su luminosidad debido a causas desconocidas, aunque se piensa que se trata de explosiones debidas al choque de grandes masas con la estrella, o tal vez de 2 o mas estrellas (parte de sistemas múltiples). El aumento de brillo es variable en intensidad y especialmente en duración, pudiendo ser desde unas semanas a varios meses o años inclusive.

6.3.14. Parsec

Unidad de medida de la distancia estelar. Equivale a la distancia que tendria que estar una estrella para mostrar un paralaje de un segundo de arco (El *paralaje* es la mitad de la pequena diferencia en el ángulo de la posición de la estrella, a un intervalo de seis meses).

Un Parsec equivale aproximadamente a 2.3 años luz.

6.3.15. Planeta

Esfera de material pétreo, formado generalmente por los desechos de sustancias complejas (carbono, hierro, azufre, etc) emitidos por las estrellas. La gravedad agrupa los materiales pesados en el centro de los planetas, formando núcleos de material líquido; los materiales más livianos flotan sobre esta superficie formando cubiertas sólidas y frías, y los gases residuales suelen formar atmósferas de densidad variable.

El sistema planetario más estudiado es el de Sol, con 9 planetas conocidos. Se sabe que hay otros sistemas en estrellas cercanas (por lo menos tres planetas alderedor de Rigil Ken), ademas de un nuevo sistema formándose alrededor de Vega.

6.3.16. SuperNovas

Fenómeno producido al final de la vida de una estrella, generalmente de masa muy superior al promedio. En esta etapa, en el núcleo se han acumulado gran cantidad de elementos pesados, resultado de los numerosos procesos de fusión, los cuales acumulan gran energía; esta energía es equilibrada temporalmente por la gravedad de estos mismos elementos. Sin embargo, si la masa es superior a cierto límite, la presión resultante genera mas energía de lo que la gravedad puede soportar en un momento dado. Cuando esta condición se da, la atmósfera exterior estalla, despidiendo el 80% de la masa estelar en forma de gases que en algunos miles de años formarán una nebulosa. Los restos del nucleo generalmente forman una estrella neutrónica o, según se cree, un agujero negro.

Durante el breve período SuperNova, la estrella puede emitir tanta energía como una galaxia mediana.

Manual de Sistema

Tabla de Contenido

1. Introducción

Antes de presentar formalmente el Sistema Gráfico de Proyección Astronómica Hiparco, permítaseme brevemente, y a modo de introducción, exponer algunas de las ideas que influyeron en el estilo que se ha seguido en su desarrollo. En los últimos años ha habido numerosas corrientes de pensamiento filosófico que han intentado aumentar la productividad de los profesionales de la computación, fomentando prácticas de programación (léase: formas de conducta social) que permiten que los programas sean más duraderos y que abarquen más campos de a los que originalmente se destinaron.

Detrás del sistema que este manual describe hay uno de esos principios, y quizá el más interesante: encapsulación de datos. Este es un método de aislar las estructras de datos de todas las partes de una aplicación excepto de las rutinas que se supone deben accesarlas. Es decir, sólo los procedimientos y funciones que actúan en las estructuras de datos son visibles a la aplicación, no las estructuras en sí. Este ocultamiento de información reduce los conflictos entre la aplicación y las librerías o subsistemas que use, previene el acceso directo a los datos por rutinas no autorizadas, y le permite al programador cambiar la implementación sin cambiar el código fuente del programa de aplicación.

Diseñar rutinas encapsuladas involucra básicamente decidir cuáles procedimientos y funciones lógicamente tienen acceso a las estructuras; aún más, teóricamente esto no debe hacerse teniendo en consideración el lenguaje de programación a usarse. Por ejemplo, en el caso de un subsistema de ventanas, debe haber procedimientos para abrir ventanas, cerralas, moverlas, etc.

La clave en la implementación es la capacidad de ocultamiento de información. El lenguaje escogido debe ser capaz de limitar el alcance de algunos identificadores a un subconjunto de las rutinas de una aplicación. Estos identificadores deben ser locales en el sentido que sólo algunas rutinas pueden verlos, y globales en el sentido de que mantienen sus valores entre las invocaciones de los procedimientos/funciones. Finalmente, se debería codificar el conjunto estructuras/rutinas de tal manera que pueda ser compilado como una sola unidad, separadamente de los demás conjuntos del sistema; esto permitiría la máxima independencia de la unidad con respecto de la aplicación que la use.

El concepto de *Unit* (Unidad) recientemente introducido en Turbo Pascal 4.0 permite crear encapsulaciones, suministrando un eficiente método de ocultamiento, diseñado bajo el existente en Modula-2. Esto, sumado al hecho de que hay pocos ejemplos prácticos de su uso (en nuestro medio), llevó al autor a diseñar este sistema de una forma ligeramente distinta a la que se suele seguir en estos trabajos.

1. Introducción

Antes de presentar formalmente el Sistema Gráfico de Proyección Astronómica Hiparco, permítaseme brevemente, y a modo de introducción, exponer algunas de las ideas que influyeron en el estilo que se ha seguido en su desarrollo. En los últimos años ha habido numerosas corrientes de pensamiento filosófico que han intentado aumentar la productividad de los profesionales de la computación, fomentando prácticas de programación (léase: formas de conducta social) que permiten que los programas sean más duraderos y que abarquen más campos de a los que originalmente se destinaron.

Detrás del sistema que este manual describe hay uno de esos principios, y quizá el más interesante: encapsulación de datos. Este es un método de aislar las estructras de datos de todas las partes de una aplicación excepto de las rutinas que se supone deben accesarlas. Es decir, sólo los procedimientos y funciones que actúan en las estructuras de datos son visibles a la aplicación, no las estructuras en sí. Este ocultamiento de información reduce los conflictos entre la aplicación y las librerías o subsistemas que use, previene el acceso directo a los datos por rutinas no autorizadas, y le permite al programador cambiar la implementación sin cambiar el código fuente del programa de aplicación.

Diseñar rutinas encapsuladas involucra básicamente decidir cuáles procedimientos y funciones lógicamente tienen acceso a las estructuras; aún más, teóricamente esto no debe hacerse teniendo en consideración el lenguaje de programación a usarse. Por ejemplo, en el caso de un subsistema de ventanas, debe haber procedimientos para abrir ventanas, cerralas, moverlas, etc.

La clave en la implementación es la capacidad de ocultamiento de información. El lenguaje escogido debe ser capaz de limitar el alcance de algunos identificadores a un subconjunto de las rutinas de una aplicación. Estos identificadores deben ser locales en el sentido que sólo algunas rutinas pueden verlos, y globales en el sentido de que mantienen sus valores entre las invocaciones de los procedimientos/funciones. Finalmente, se debería codificar el conjunto estructuras/rutinas de tal manera que pueda ser compilado como una sola unidad, separadamente de los demás conjuntos del sistema; esto permitiría la máxima independencia de la unidad con respecto de la aplicación que la use.

El concepto de *Unit* (Unidad) recientemente introducido en Turbo Pascal 4.0 permite crear encapsulaciones, suministrando un eficiente método de ocultamiento, diseñado bajo el existente en Modula-2. Esto, sumado al hecho de que hay pocos ejemplos prácticos de su uso (en nuestro medio), llevó al autor a diseñar este sistema de una forma ligeramente distinta a la que se suele seguir en estos trabajos.

En vez de diseñar un sistema enfatizado a características externas, aquí se presenta un sistema que se concentra en presentar una estructura interna que pueda servir a otros analistas que deseen experimentar con esta técnica; al mismo tiempo, se presenta un conjunto amplio de rutinas de utilidad general para todo programador, incluyendo una librería de herramientas gráficas de cierta independencia de hardware que aumentarán su importancia conforme la actual tendencia hacia interfaces gráficas vaya asentándose localmente.

2. Estructura general de las rutinas

2.1. Principios básicos

Como se mencionó en la introducción, el principio en que se basa la programación de estas rutinas se ha extraído de los conceptos de la Programación Orientada a Objetos; dicho principio se conoce como encapsulación de datos. Las reglas básicas a seguir son las siguientes:

- El problema se define en base a una serie de una o mas entidades, cada una de las cuales representa los datos interrelacionados del problema.

- Cada entidad se representa en el programa con una estructura de datos (en memoria o disco), la cual deberá tener propiedades lo mas similares posible a su equivalente físico. De ser posible cada estructura será implementada en archivos o librerías separados.

- Para cada entidad se crea una serie de funciones y procedimientos que sirvan para que rutinas externas accesen a todas las caraterísticas de dicha entidad. En otras palabras cada estructura solo se puede comunicar con las otras a través de las funciones definidas al efecto.

- Se procurará que cada objeto (estructura más sus funciones de manejo) sea tan independiente de las otras como sea posible, de tal manera que un cambio interno en cualquiera de ellas no afecte en lo absoluto a las demás.

Por ejemplo, si se implementa un objeto que represente un directorio telefónico (tal vez por medio de un vector), se debería implementar una función que retorne el número telefónico de una persona dado su nombre, en vez de dejar simplemente que la aplicación accese a la estructura. Esto nos da la facilidad de cambiar la estructura sin que sea necesario modificar la aplicación.

2.2. Interface e Implementación

Utilizando las facilidades del lenguaje escogido (Turbo Pascal 4.0), Cada unidad que compone el paquete se ha estructurado en 2 partes: interface e implementación.

La sección de interface compone todas aquellas variables y funciones que pueden ser accesadas por otras unidades que conformen la aplicación.

La sección de implementación constituye de todos aquellos detalles de la implementación de la estructura que no necesitan ser conocidos por otras unidades, incluyendo los códi-

gos de los procedimientos/funciones declarados en la interface.

2.3. Inicializaciones/finalizaciones/errores

Gracias a la estructura de las unidades, cada elemento del programa se puede desarrollar en forma casi por completo independiente de las otras. En realidad, planificando cuidadosamente la distribución de las rutinas, es posible construir nuevas unidades utilizando enteramente elementos previamente definidos en otras. Cada unidad puede tener secciones de inicialización y finalización propias, las cuales se ejecutarán sin necesidad de la intervención del programa que las utiliza.

Por ejemplo, una de las unidades puede ser diseñadas de tal manera que grabe automaticamente la pantalla previa a la ejecución del programa y restaure dicha pantalla cuando el programa finaliza. Durante la ejecución del programa, este puede utilizar funciones de la unidad para extraer datos y modificar dicha pantalla.

Otra de las funciones de la finalización automática es la de reajustar las condiciones del sistema luego de un error fatal. Para errores no-fatales, se ha seguido un principio sencillo: cada unidad con funciones complejas cuenta con una variable booleana de interface denominada *StatusOk*, la cual se ajusta a FALSE cuando algún procedimiento registra errores (por ejemplo, parámetros incorrectos). Nótese que para las funciones no es necesario esta variable ya que éstas pueden retornar un código de error como resultado. Además, ya que hay varias unidades con una variable del mismo nombre, el módulo principal debe referirse a éstas con la notación Unit.Var; por ejemplo: *GrSelect.StatusOk* se refiere a la variable de status de la unidad GRSELECT.

3. Referencia de las rutinas

3.1. Funciones básicas (no gráficas)

3.1.1. ArcCos

Propósito

Dado un coseno determinado, obtener el ángulo correspondiente.

Declaración

```
Function ArcCos (X: Real): Real;
```

Parámetros

X: Coseno del ángulo deseado.

Retorna

El ángulo en radianes.

Operación

Normalmente sólo se dispone de la función standard *ArcTan* para calcular un ángulo cuando se conoce su tangente; cuando se conocen otras funciones, por lo general es necesario recurrir a fórmulas para calcular su tangente. Esta función puede servir de ayuda en esas ocasiones.

Observaciones

- Función inversa de *Cos*

Véase también

DegToRad (Pag. 9), *RadToDeg* (Pag. 15)

3.1.2. BiosDate

Propósito

Obtener la fecha de emisión del ROM-BIOS

Declaración

```
Var BiosDate: String[8];
```

Retorna

La fecha grabada en el BIOS.

Operación

Las versiones antiguas del ROM-BIOS de la IBM PC tenían algunas fallas, las cuales se han venido eliminando con la emisión de nuevas versiones. Para averiguar si su aplicación está teniendo problemas por fallas en el BIOS, puede utilizarse esta variable.

Observaciones

- Se recomienda que el valor de la variable no se altere.

- El formato de la fecha puede variar de manufacturador a manufacturador, aunque el esquema seguido generalmente es mm/dd/aa

Véase también

DosVersion (Pag. 10)

3.1.3. BitOn

Propósito

Verificar el estado de un bit específico en un byte dado.

Declaración

```
Function BitOn (Value,Bit: Byte) Boolean;
```

Parámetros

Value: El valor a probar

Bit: El número del bit correspondiente a verificar (contando de izquierda a derecha); debe estar en el rango 0 a 7.

Retorna

TRUE - El bit correspondiente en Value es 1

FALSE - El bit correspondiente en Value es 0

Operación

Algunas funciones del DOS se basan en el estado de uno o más bits dentro de un byte específico (por ejemplo, el byte de estado del teclado), para lo cual se puede usar esta función sin necesidad de utilizar máscaras u otros artificios de bajo nivel.

Véase también

SetBit (Pag. 16)

3.1.4. ByteToHex

Propósito

Convertir un número decimal a su equivalente hexadecimal

Declaración

```
Type HexType = String[2];

Function ByteToHex (Value: Byte): HexType;
```

Parámetros

Value: Valor a trasladar.

Retorna

Un string de dos posiciones, conteniendo el equivalente de Value en hexadecimal.

Operación

Por lo general esta función se utilizará para displayar el contenido hexadecimal de la memoria (útil en algunas situaciones de debugging).

Observaciones

- Para calcular el valor hexadecimal de una variable *X*, definida como integer o word, utilícese la fórmula:

```
ByteToHex(Hi(X)) + ByteToHex(Lo(X))
```

- Para obtener valores de otros tipos de variables, utilícese las facilidades de conversión de tipos:

```
ByteToHex(Byte(Variable))
```

Véase también

HexToByte (Pag. 14)

3.1.5. DegToRad

Propósito

Convertir grados a radianes

Declaración

```
Function DegToRad (Deg: Real): Real;
```

Parámetros

Deg: Angulo a convertir, en grados decimales.

Retorna

El equivalente del ángulo Deg en radianes

Operación

Las funciones trigonométricas de Turbo Pascal requieren sus parámetros en radianes, y suelen retornar resultados en radianes. Utilícense esta función y su inversa *RadToDeg* para hacer las conversiones necesarias.

Observaciones

- Nótese que el párametro es de tipo *Real*; por lo tanto, el programador deberá convertir aquellos ángulos que estén en el formato grados-minutos-segundos al formato decimal.

Véase también

RadToDeg (Pag. 15), *ArcCos* (Pag. 5)

3.1.6. DosVersion

Propósito

Obtener la versión del sistema operativo en que se está ejecutando la aplicación.

Declaración

```
Var DosVersion: String[5];
```

Retorna

Un string de 5 caracteres en el formato *mm.nn* donde *mm* corresponde al número de versión mayor y *nn* al número de versión menor.

Operación

Por lo general, se utiliza esta variable para verificar que la versión del DOS soporta algún comando especial que su aplicación pueda utilizar (por ejemplo, los comandos de acceso a archivos compartidos sólo están disponibles en versiones mayores a 3.00).

Observaciones

- Se recomienda que el valor de la variable no se altere.

Véase también

BiosDate (Pag. 6)

3.1.7. EscPressed

Propósito

Controlar que se haya presionado la tecla [Esc].

Declaración

```
Function EscPressed: Boolean;
```

Retorna

TRUE - El usuario ha presionado [Esc] antes de la llamada a la función

FALSE - El usuario no ha presionado ninguna tecla, o ha presionado una tecla distinta a [Esc]

Operación

Esta función puede usarse para abortar un determinado proceso (por ejemplo, un bucle) si se presiona la tecla [Esc]. La aplicación puede verificar periódicamente el valor de esta función como parte de su opración normal.

Observaciones

- A diferencia de la función standard *KeyPressed*, ésta limpia el buffer del teclado cada vez que se la invoca.

3.1.8. Equal

Propósito

Verificar que dos variables del mismo tipo y longitud sean iguales.

Declaración

```
Function Equal (Var V1,V2; Size: Word): Boolean;
```

Parámetros

V1,V2: Las dos variables a comparar; pueden ser de cualquier tipo, en tanto que ambas ocupen el mismo espacio en memoria.

Size: Este valor debe ser igual al tamaño, en bytes, que ocupan ambas variables en memoria (puede determinarse por medio de la función standard *SizeOf*).

Retorna

TRUE - Las variables son exactamente iguales

FALSE - Las dos variables difieren al menos en un byte

Operación

Esta función es especialmente útil para comparar tipos estructurados, en los cuales no es posible utilizar el operador de igualdad (=); utilícese como tercer parámetro la función *SizeOf* aplicada al tipo al que pertenecen.

Por ejemplo, para comparar dos variables RIn y ROut de tipo Registers, se hace la siguiente llamada:

```
Equal(R1,R2,SizeOf(Registers))
```

Par comparar dos arreglos A1 y A2 de 100 elementos de tipo integer (recuérdese que cada integer ocupa 2 bytes):

```
Equal(A1,A2,200)
```

Observaciones

- Si el parametro *Size* no corresponde al valor adecuado, el valor retornado es impredecible.

Véase También

SwapBytes (Pag. 17), *SwapVars* (Pag. 18)

3.1.9. HexToByte

Propósito

Convertir la representación string de un número hexadecimal a su equivalente numérico.

Declaración

```
Type HexType: String[2];

Function HexToByte (Value: HexType): Byte;
```

Parámetros

Value: un string de dos caracteres; cada caractér debe corresponder a un dígito hexadecimal (0-9,A-F)

Retorna

El valor número correspondiente al parámetro.

Operación

Un uso posible de esta función es interpretar códigos hexadecimales generados por ensambladores o debuggers.

Observaciones

- Si alguno de los dígitos del parámetro no está en el rango hexadecimal, se lo trata como si fuera cero.

- Si se pasa como parámetro un string de más de 2 caracteres, sólo se toman en cuenta los dos primeros.

Véase también

ByteToHex (Pag. 8)

3.1.10. RadToDeg

Propósito

Convertir radianes a grados

Declaración

```
Function RadToDeg (Rad: Real): Real;
```

Parámetros

Rad: Angulo a convertir, en radianes.

Retorna

El equivalente del ángulo Rad en grados decimales

Operación

Las funciones trigonométricas de Turbo Pascal requieren sus parámetros en radianes, y suelen retornar resultados en radianes. Utilícense esta función y su inversa *DegToRad* para hacer las conversiones necesarias.

Véase también

DegToRad (Pag. 9), *ArcCos* (Pag. 5)

3.1.11. SetBit

Propósito

Ajustar el valor de un bit específico de un byte.

Declaración

```
Function SetBit (Value, Bit, BitVal: Byte): Byte;
```

Parámetros

Value: El valor original del byte a ajustar

Bit: El número del bit correspondiente a ajustar (contando de izquierda a derecha); debe estar en el rango cero a siete (0..7).

BitVal: El nuevo valor del bit especificado; debe ser cero o uno.

Retorna

El parámetro *Value* con el bit correspondiente ajustado; los restantes bits permanecen sin cambio.

Operación

La función es útil para ajustar parámetros del DOS, sobre todo los que se encuentran en el área de datos del BIOS (segmento 40 hex). Por ejemplo, la siguiente instrucción activa el estado de CapsLock:

```
Mem [$40:$17] := SetBit (Mem [$40:$17],6,1);
```

Observaciones

- Si el parámetro *BitVal* no se encuentre en el rango 0,1, se lo toma como cero.

Véase también

BitOn (Pag. 7)

3.1.12. SwapBytes

Propósito

Intercambiar los valores de dos variables de hasta 255 bytes de tamaño.

Declaración

```
Procedure SwapBytes (Var Var1,Var2; Size: Byte);
```

Parámetros

Var1, *Var2*: Las dos variables a intercambiar; pueden ser de cualquier tipo, en tanto que ambas ocupen el mismo espacio en memoria.

Size: Este valor debe ser igual al tamaño, en bytes, que ocupan ambas variables en memoria (puede determinarse por medio de la función standard *SizeOf*).

Operación

Intercambiar dos variables es una operación común, sobre todo en rutinas de ordenación. Recuérdese que ambas variables no deben ocupar más de 255 bytes; esta condición se cumple para todos los tipos simple en la versión corriente de Turbo Pascal.

Observaciones

- Si el parámetro *Size* no corresponde al valor adecuado, el valor retornado es impredecible.

Véase También

SwapVars (Pag. 18)

3.1.13. SwapVars

Propósito

Intercambiar los valores de dos variables de hasta 64K de tamaño.

Declaración

```
Procedure SwapVars (Var Var1,Var2; Size: Byte);
```

Parámetros

Var1, *Var2*: Las dos variables a intercambiar; pueden ser de cualquier tipo, en tanto que ambas ocupen el mismo espacio en memoria.

Size: Este valor debe ser igual al tamaño, en bytes, que ocupan ambas variables en memoria (puede determinarse por medio de la función standard SizeOf).

Operación

Recuérdese que ambas variables no deben ocupar más de 64K; este procedimiento puede usarse con variables estructuradas (arreglos, registros, etc).

Observaciones

- Si el parámetro *Size* no corresponde al valor adecuado, el valor retornado es impredecible.

Véase También

SwapBytes (Pag. 17)

3.2. Manejo de Strings

3.2.1. After

Propósito

Extraer un substring de las últimas posiciones de un string dado.

Declaración

```
Function After (Source, Target: String): String;
```

Parámetros

Source: El string original a procesar.

Target: El substring de Source que servirá de base a la operación.

Retorna

El contenido de *Source* que se encuentre a la derecha de *Target* (sin incluir *Target*).

Operación

Utilícese está función y su contraparte *Before* para realizar análisis de strings sin necesidad de referirse a posiciones específicas dentro del mismo.

Por ejemplo, si *FileName* es un string que representa un nombre de archivo, la siguiente expresión retorna la extensión del mismo y la coloca en el string *Ext*:

```
Ext:= After (FileName,'.');
```

Observaciones

- Si el string *Target* no existe en *Source*, la función retorna un string nulo (longitud 0).

Véase También

Before (Pag. 20), *Separate* (Pag. 30)

3.2.2. Before

Propósito

Extraer un substring de las primeras posiciones de un string dado.

Declaración

```
Function Before (Source, Target: String): String;
```

Parámetros

Source: El string original a procesar.

Target: El substring de Source que servirá de base a la operación.

Retorna

El contenido de *Source* que se encuentre a la izquierda de *Target* (sin incluir *Target*).

Operación

Utilícese está función y su contraparte *After* para realizar análisis de strings sin necesidad de referirse a posiciones específicas dentro del mismo.

Por ejemplo, si *FileName* es un string que representa un nombre de archivo, la siguiente expresión retorna el nombre del mismo (sin extensión) y la coloca en el string *Name*:

```
Name:= Before (FileName,'.');
```

Observaciones

- Si el string *Target* no existe en *Source*, la función retorna el string *Source* sin modificaciones.

Véase También

After (Pag. 19), *Separate* (Pag. 30)

3.2.3. Center

Propósito

Centrar un string en un campo de espacios en blanco.

Declaración

```
Function Center (Source: String; N: Byte): String;
```

Parámetros

Source: El string original a procesar.

N: La longitud del campo a utilizar.

Retorna

Un string de *N* caracteres conteniendo al parámetro *Source* centrado entre espacios en blanco.

Operación

Esta función es especialmente útil en la conformación de pantallas y reportes.

Observaciones

- Si la longitud de *Source* es mayor a *N*, se retornan los primeros *N* caracteres de *Source*.

Véase También

CenterIn (Pag. 22)

3.2.4. CenterIn

Propósito

Centrar un string en un campo de caracteres específicos.

Declaración

```
Function CenterIn (Source: String; N: Byte; Ch: Char): String;
```

Parámetros

Source: El string original a procesar.

N: La longitud del campo a utilizar.

Ch: El caractér a utilizar para relleno.

Retorna

Un string de *N* caracteres conteniendo al parámetro Source centrado entre caracteres *Ch*.

Operación

Esta función es especialmente útil en la conformación de pantallas y reportes.

Observaciones

- Si la longitud de *Source* es mayor a *N*, se retornan los primeros *N* caracteres de *Source*.

Véase También

Center (Pag. 21)

3.2.5. DnCase

Propósito

Convertir un caractér alfabético a minúsculas.

Declaración

```
Function DnCase (Ch: Char): Char;
```

Parámetros

Ch: El caractér original a procesar.

Retorna

Si *Ch* es una letra mayúscula, retorna la letra minúscula equivalente; de lo contrario, retorna *Ch*

Operación

Turbo Pascal provee de una función standard, *UpCase*, para convertir letras minúsculas a mayúsculas, pero ninguna para la operación inversa. Si llega a presentarse dicha necesidad, puede utilizarse esta función.

Véase También

DownCase (Pag. 24), *UpperCase* (Pag. 34)

3.2.6. DownCase

Propósito

Convertir todas los caracteres alfabéticos de un string a minúsculas.

Declaración

```
Function DownCase (Source: String): String;
```

Parámetros

Source: El string original a procesar.

Retorna

El parámetro *Source* con las letras mayúsculas que contuviere convertidas a sus equivalentes minúsculas.

Operación

El uso más común de esta función es facilitar ciertas validaciones, convirtiendo strings de entrada a un formato uniforme.

Véase También

DnCase (Pag. 23), *UpperCase* (Pag. 34)

3.2.7. FirstWord

Propósito

Extraer la primera palabra de un string.

Declaración

```
Function FirstWord (Var Source: String): String;
```

Parámetros

Source: El string original a procesar.

Retorna

La primera secuencia de caracteres válidos existentes en *Source*.

Operación

Los caracteres considerados inválidos se almacenan en una variable con la siguiente declaración:

```
Var Invalid: Set of Char;
```

Cualquier caractér que no se encuentre en dicho conjunto se considera válido. La aplicación puede controlar los caracteres válidos, en consecuencia, modificando el conjunto *Invalid*.

Véase También

After (Pag. 19), *LTrim* (Pag. 26)

3.2.8. LTrim

Propósito

Eliminar los caracteres iniciales inválidos de un string.

Declaración

```
Function LTrim (Source: String): String;
```

Parámetros

Source: El string original a procesar.

Retorna

El parámetro *Source*, a partir de la primera ocurrencia de un caractér válido.

Operación

Esta función es especialmente útil para remover los espacios iniciales resultantes de la función *ValToStr*.

Observaciones

- Véase la discusión referente a la función *FirstWord*, sobre cómo determinar cúales son los caracteres inválidos.

Véase también

FirstWord (Pag. 25), *RTrim* (Pag. 29), *Trim* (Pag. 33)

3.2.9. ReplaceAll

Propósito

Buscar y reemplazar un caractér en un string.

Declaración

```
Function ReplaceAll (Source: String; Ch1,Ch2: Char): String;
```

Parámetros

Source: El string original a procesar.

Ch1: El caractér a buscar y reemplazar.

Ch2: El caracter que reemplazará a Ch1.

Retorna

El parámetro *Source* con todas las ocurrencias de *Ch1* reemplazadas por *Ch2*.

Operación

Esta función puede utilizarse como un filtro o máscara para eliminar caracteres no deseados de un string de entrada.

Véase también

Replicate (Pag. 28)

3.2.10. Replicate

Propósito

Llenar un string con ún sólo caractér.

Declaración

```
Function Replicate (N: Byte; Ch: Char): String;
```

Parámetros

N: El número de veces a repetir el caractér.

Ch: El caractér a repetir.

Retorna

Un string de *N* ocurrencias del caractér *Ch*.

Operación

Muchos lenguajes proporcionan una función equivalente a ésta, la cual es utilizada generalmente para formar pantallas.

Véase también

Spaces (Pag. 31)

3.2.11. RTrim

Propósito

Eliminar los caracteres finales inválidos de un string.

Declaración

```
Function RTrim (Source: String): String;
```

Parámetros

Source: El string original a procesar.

Retorna

El parámetro *Source*, desde el primer caractér hasta la última ocurrencia de un caractér válido.

Operación

Esta función es especialmente útil para remover los espacios finales resultantes de la lectura de líneas de archivos textos.

Observaciones

- Véase la discusión referente a la función *FirstWord*, sobre cómo determinar cúales son los caracteres inválidos.

Véase también

FirstWord (Pag. 25), *LTrim* (Pag. 26), *Trim* (Pag. 33)

3.2.12. Separate

Propósito

Separar un string en dos substrings.

Declaración

```
Procedure Separate (Source,Separator: String; Var LeftPart,RightPart: String);
```

Parámetros

Source: El string original a procesar.

Separator: Substring de Source a utilizar como punto de separación.

LeftPart,*RightPart*: Variables donde se colocarán las partes izquierda y derecha de Source, respectivamente.

Operación

Este procedimiento toma como punto de partida la primera ocurrencia de *Separator* dentro de *Source*. En *LeftPart* se coloca el contenido de *Source* hasta la posición inmediatamente anterior a *Separator*; en *RightPart* se coloca el contenido de *Source* desde la posición inmediatamente siguiente a *Separator*.

Observaciones

- Si *Separator* no es un substring de *Source*, en *LeftPart* se retorna *Source* sin modificación y en *RightPart* un string nulo (longitud 0).

Véase También

After (Pag. 19), *Before* (Pag. 20)

3.2.13. Spaces

Propósito

Llenar un string con espacios en blanco.

Declaración

```
Function Spaces (N: Byte): String;
```

Parámetros

N: El número de espacios a generar.

Retorna

Un string de *N* espacios en blanco.

Operación

Es utilizada generalmente para formar pantallas o para inicializar variables string.

Observaciones

- Esta función es prácticamente equivalente a la siguiente expresión:

```
Replicate (N, ' ')
```

Sin embargo, y en la mayoría de las circunstancias, el uso de *Spaces* es preferible debido a la mayor legibilidad del código resultante.

Véase también

Replicate (Pag. 28)

3.2.14. StrToVal

Propósito

Convertir un string a un número.

Declaración

```
Function StrToVal (S: String): Real;
```

Parámetros

S: Representanción string de una cantidad.

Retorna

El número representado en el parámetro *S*.

Operación

Turbo Pascal provee del procedimiento standard *Val* para convertir un string a número, pero en ocasiones es preferible disponer de una función equivalente, ya que es posible incluir directamente el resultado en expresiones.

Observaciones

- Si el parámetro *S* no representa un número válido, la función retorna cero.

- Nótese que, para máxima flexibilidad, el tipo numérico retornado es *Real*; para convertir números tipo integer o byte, será necesario utilizar funciones standard de redondeo (*Round*, *Trunc*, etc)

Véase También

ValToStr (Pag. 35)

3.2.15. Trim

Propósito

Eliminar los caracteres iniciales y finales inválidos de un string.

Declaración

```
Function Trim (Source: String): String;
```

Parámetros

Source: El string original a procesar.

Retorna

El parámetro *Source*, a partir de la primera ocurrencia hasta la última ocurrencia de caracteres válidos.

Operación

Esta función combina las características de *LTrim* y *RTrim*, siendo su ejecución más rápida que dos llamadas consecutivas a dichas funciones.

Observaciones

- Véase la discusión referente a la función *FirstWord*, sobre cómo determinar cúales son los caracteres inválidos.

Véase también

FirstWord (Pag. 25), *LTrim* (Pag. 26), *RTrim* (Pag. 29)

3.2.16. UpperCase

Propósito

Convertir todas los caracteres alfabéticos de un string a mayúsculas.

Declaración

```
Function UpperCase (Source: String): String;
```

Parámetros

Source: El string original a procesar.

Retorna

El parámetro *Source* con las letras minúsculas que contuviere convertidas a sus equivalentes mayúsculas.

Operación

Turbo Pascal provee de la función standard *UpCase* para convertir letras minúsculas a mayúsculas, pero sólo actúa sobre un caractér a la vez, siendo mucho más general el caso de tener que convertir un string completo.

El uso más común de esta función es facilitar ciertas validaciones, convirtiendo strings de entrada a un formato uniforme.

Véase También

DnCase (Pag. 23), *DownCase* (Pag. 24)

3.2.17. ValToStr

Propósito

Convertir un número a un string.

Declaración

```
Function ValToStr (N: Real; Width,Decimals: Byte): String;
```

Parámetros

N: El número a convertir

Width: La longitud, en caracteres, del string a retornar.

Decimals: El número de posiciones decimales a considerar.

Retorna

Un string de longitud *Width* conteniendo el número *N*. Si es necesario, el campo se rellena con espacios en blanco iniciales.

Operación

Turbo Pascal provee del procedimiento standard *Str* para convertir un número a string, pero en ocasiones es preferible disponer de una función equivalente, ya que es posible incluir directamente el resultado en expresiones.

Véase También

StrToVal (Pag. 32)

3.3. Acceso de archivos

3.3.1. DefaultDrive

Propósito

Obtener el número del disco corriente.

Declaración

```
Function DefaultDrive: Byte;
```

Retorna

El número del drive corriente:

1 - A:
2 - B:
3 - C: (...etc.)

Operación

El número de la unidad es necesario para ciertas operaciones de bajo nivel, como *DiskFree*. La alternativa normal es obtener el directorio corriente por medio de *GetDir* y extraer la letra del disco corriente del primer catactér del directorio, para luego calcular el número deseado. En la mayoria de los casos, es más sencillo utilizar esta función; además, a diferencia de *GetDir*, el uso de esta función no implica un acceso a la unidad de disco, por lo que es más rápido.

Observaciones

- Para obtener la letra de la unidad corriente, utilice la siguiente expresión:

```
Chr (DefaultDrive + n)
```

donde *n* es igual a 64 para letra mayúscula, o 96 para letra minúscula.

Véase también

DriveAtStart (Pag. 39)

3.3.2. DiskError

Propósito

Validar errores en el último acceso al disco.

Declaración

```
Function DiskError: Byte;
```

Retorna

El código de error BIOS de la última operación de disco, más uno; dichos códigos son:

0 - No error
1 - Dirección no encontrada
2 - Disco protegido contra escritura
3 - Sector no encontrado
4 - Sobrecarga del DMA
5 - Falla del CRC
6 - Falla del Controlador
7 - Falla de posicionamiento (seek)
8 - Falla de tiempo (time-out)

Operación

Llámese a la función inmediatamente después de una operación de acceso a disco fallida. Recuerde que el indicador de control de errores de I-O de Turbo Pascal debe ser apagado (directiva de compilador {$I-}) a fin de que la operación no produzca un error de ejecución. Asumiendo las variables *DataFile* y *BiosErr*, de tipo *File* y *Byte* respectivamente, una secuencia de uso típica sería:

```
{$I-}
Assign(DataFile,'DUMMY.DTA');
Reset(DataFile);
If IOResult <> 0
   Then Begin
        BiosErr:= DiskError;
        Case BiosErr of
         {.. rutinas de control de errores ..}
```

Observaciones

- A diferencia de *IOResult*, *DiskError* no reajusta su valor después de ser llamada, sino que se reajusta después de cada acceso al disco; por esto, puede llamarse varias veces después de cada operación sin que cambie su valor.

- Recuerde que no todas las operaciones de I-O en Turbo Pascal causan actividad física en el disco, siendo éstas las que activan el código de error BIOS. Un ejemplo lo constituye *Seek*.

Véase También

ExistFile (Pag. 40)

3.3.3. DriveAtStart

Propósito

Obtener el drive corriente al momento del arranque de la aplicación.

Declaración

```
Var DriveAtStart: Char;
```

Retorna

La letra del drive: A, B, C, etc. sin incluir los dos puntos.

Operación

Utilícese como una variable normal. Nótese que la variable no se actualiza si operaciones posteriores al arranque cambian el drive corriente.

Véase también

DefaultDrive (Pag. 36)

3.3.4. ExistFile

Propósito

Validar la existencia de un archivo.

Declaración

```
Function ExistFile (Name: String): Boolean;
```

Parámetros

Name: Corresponde al nombre de archivo a verificar; puede incluir un designador de drive, pero no caracteres "comodín" (* o ?).

Retorna

TRUE - el archivo existe y es accesible

FALSE - no es posible abrir el archivo

Operación

La funcion retornará FALSE si el nombre de archivo incluye caracteres invalidos según las reglas del DOS (por ejemplo, el signo de mayor que). Se recomienda que, ya que el uso de la función involucra un acceso al disco, sólo se utilize en el caso de que la aplicación haya validado previamente que las convenciones del DOS estén correctas.

Véase También

DiskError (Pag. 37)

3.4. Funciones básicas (gráficas)

3.4.1. BlinkArea

Propósito

Invertir momentáneamente el color de un área.

Declaración

```
Procedure BlinkArea (X1,Y1,X2,Y2: Integer);
```

Parámetros

X1,Y1: Coordenadas gráficas (columna, fila) de la esquina superior izquierda del área.

X2,Y2: Coordenadas gráficas (columna, fila) de la esquina inferior derecha del área.

Operación

Utilícese para simular el atributo texto intermitente, el cual normalmente no está disponible en modo gráfico.

Observaciones

- Las coordenadas son relativas al *ViewPort* corriente.

Véase También

ClearArea (Pag. 42), *InvertArea* (Pag. 46)

3.4.2. ClearArea

Propósito

Borrar un área.

Declaración

```
Procedure ClearArea (X1,Y1,X2,Y2: Integer);
```

Parámetros

X1,Y1: Coordenadas gráficas (columna, fila) de la esquina superior izquierda del área.

X2,Y2: Coordenadas gráficas (columna, fila) de la esquina inferior derecha del área.

Operación

El borrado se realiza cubriendo el área indicada con el color de fondo corriente (color 0).

Observaciones

- Las coordenadas son relativas al *ViewPort* corriente.

Véase También

BlinkArea (Pag. 41), *ClearView* (Pag. 43), *InvertArea* (Pag. 46)

3.4.3. ClearView

Propósito

Borrar un área de coordenadas absolutas.

Declaración

```
Procedure ClearView (X1,Y1,X2,Y2: Integer);
```

Parámetros

X1,Y1: Coordenadas gráficas (columna, fila) de la esquina superior izquierda del área.

X2,Y2: Coordenadas gráficas (columna, fila) de la esquina inferior derecha del área.

Operación

El borrado se realiza cubriendo el área indicada con el color de fondo corriente (color 0).

Observaciones

- Las coordenadas son absolutas relativas a la pantalla física.

Véase También

ClearArea (Pag. 42)

3.4.4. DrawArrow

Propósito

Dibujar una punta de flecha.

Declaración

```
Type MoveType = (MoveNull, MoveUp, MoveDown, MoveLeft, MoveRight);

Procedure DrawArrow (X,Y: Integer; Dir: MoveType);
```

Parámetros

X,Y: Coordenadas gráficas del punto central donde se dibujará la flecha.

Dir: Dirección en que apuntará la flecha.

Operación

El símblo individual más común en los dibujos, fuera de las líneas, son las flechas que suelen dibujarse al final de dichas líneas. Este procedimiento permite dibujar una flecha en cualquiera de las direcciones básicas (arriba, abajo, izquierda y derecha).

Observaciones

- El tamaño de la punta de flecha es el de un caractér en el tipo de letra corriente; por lo tanto, para modificar el tamaño de la flecha la aplicación sólo tiene que ajustar el tipo de letra que sea conveniente.

Véase También

PrintText (Pag. 49)

3.4.5. GetView

Propósito

Obtener las coordenadas de la ventana gráfica corriente.

Declaración

```
Procedure GetView (Var ViewPort: ViewPortType);
```

Parámetros

ViewPort: Variable donde se deben colocar las coordenadas.

Operación

A pesar de que las coordenadas gráficas son fácilmente accesibles a través de las variables *GrMinimun* y *GrMaximun*, hay situaciones en que es más practico tenerlas almacenadas en una sola variable *ViewPort*. En otras palabras, este procedimiento es un substituto aproximado del procedimiento standard *GetViewSettings*.

Véase También

ClearView (Pag. 43)

3.4.6. InvertArea

Propósito

Invertir el color de un área.

Declaración

```
Procedure InvertArea (X1,Y1,X2,Y2: Integer);
```

Parámetros

X1,*Y1*: Coordenadas gráficas (columna, fila) de la esquina superior izquierda del área.

X2,*Y2*: Coordenadas gráficas (columna, fila) de la esquina inferior derecha del área.

Operación

Utilícese para simular el atributo texto inverso, el cual normalmente no está disponible en modo gráfico.

Observaciones

- Las coordenadas son relativas al *ViewPort* corriente.

Véase También

BlinkArea (Pag. 41), *ClearArea* (Pag. 42)

3.4.7. PointsH

Propósito

Convertir coordenadas horizontales texto a gráficas

Declaración

```
Function PointsH (Coord: Byte): Integer;
```

Parámetros

Coord: Columna texto a convertir a columna gráfica.

Retorna

La coordenada gráfica donde empieza la columna *Coord*.

Operación

Utilícese, con el procedimiento standard *OutTextXY*, para imprimir texto en modo gráfico, conservando el sistema de coordenadas del modo texto.

Observaciones

- La función toma como base el tamaño del caractér en el tipo *DefaultFont*.

Véase También

PointsV (Pag. 48)

3.4.8. PointsV

Propósito

Convertir coordenadas verticales texto a gráficas

Declaración

```
Function PointsV (Coord: Byte): Integer;
```

Parámetros

Coord: Fila texto a convertir a fila gráfica.

Retorna

La coordenada gráfica donde empieza la fila *Coord*.

Operación

Utilícese, con el procedimiento standard *OutTextXY*, para imprimir texto en modo gráfico, conservando el sistema de coordenadas del modo texto.

Observaciones

- La función toma como base el tamaño del caractér en el tipo *DefaultFont*.

Véase También

PointsH (Pag. 47)

3.4.9. PrintText

Propósito

Imprimir un string en la pantalla, utilizando coordenadas texto.

Declaración

```
Procedure PrintText (Col,Row: Byte; TextToPrint: String);
```

Parámetros

Col,*Row*: Coordenada texto (columna, fila) a partir de la cual imprimir el texto.

TextToPrint: String a imprimir.

Operación

Utilícese en la misma forma en que se utilizaría la instrucción *Write* (posicionada con *GotoXY*) en el modo texto.

Observaciones

- Las coordenadas se consideran relativas a la ventana corriente.

Véase También

PointsH (Pag. 47), *PointsV* (Pag. 48), *SetFont* (Pag. 50)

3.4.10. SetFont

Propósito

Ajustar el tipo de letra corriente.

Declaración

```
Procedure SetFont (NewFont: Word);
```

Parámetros

NewFont: La identificación del nuevo tipo de letra.

Operación

Aunque Turbo Pascal provee del procedimiento standard *SetTextStyle* con propósitos similares, bajo ciertas circunstancias puede ser más conveniente utilizar *SetFont*. Específicamente, éste ajusta el tamaño del tipo de letra de tal manera que las letras tengan el mismo aspecto aproximado (con excepción del tipo *GothicFont*).

Véase También

PrintText (Pag. 49)

3.5. Conversión de Coordenadas

3.5.1. Point3DToPoint

Propósito

Convertir coordenadas cúbicas a rectangulares

Declaración

```
Type Point3DType = Record
                   X,Y,Z: Integer;
                   End;

Procedure Point3DToPoint (Point3D: Point3DType; WindowWidth,DistFromOrig,DistFromWindow: Word;
                          Var Point: PointType);
```

Parámetros

Point3D: Coordenadas originales a convertir.

WindowWidth: Ancho de la ventana de observación.

DistFromOrig: Distancia del punto de observación al origen del sistema de coordenadas.

DistFromWindow: Distancia del punto de observación a la ventana de observación.

Point: Variable donde se almacenará el resultado.

Operación

Siendo ésta la conversión más compleja, la fórmula utilizada asume ciertas condiciones:

- El punto de observación está sobre el eje X del sistema de coordenadas cúbicas.

- El eje Z del sistema de coordenadas cúbicas apunta al "norte".

Véase También

Polar3DToPoint3D (Pag. 56)

3.5.2. Point3DToPolar3D

Propósito

Convertir coordenadas cúbicas a esféricas.

Declaración

```
Type Point3DType = Record
                   X,Y,Z: Integer;
                   End;

Type Polar3DType = Record
                   R: Word;
                   AgX,AgZ: Real;
                   End;

Procedure Point3DToPolar3D (Point: Point3DType; Var Polar: Polar3DType);
```

Parámetros

Point: Coordenadas originales a convertir.

Polar: Variable donde se almacenará el resultado.

Véase También

Polar3DToPoint3D (Pag. 56)

3.5.3. PointDist

Propósito

Calcular la distancia entre dos puntos.

Declaración

```
Function PointDist (Var P1,P2: PointType): Real;
```

Parámetros

P1,*P2*: Coordenadas rectangulares de los dos puntos.

Retorna

La distancia en pixels del punto *P1* al punto *P2*.

Véase También

PointInView (Pag. 54)

3.5.4. PointInView

Propósito

Verificar que una coordenada se encuentre dentro de cierto rango.

Declaración

```
Function PointInView (Var P: PointType; Var V: ViewPortType): Boolean;
```

Parámetros

P: Coordenada rectangular del punto a verificar.

V: ViewPort que define las coordenadas dentro de las cuales debe encontrarse el punto *P*.

Retorna

TRUE - La coordenada P está dentro de los límites del ViewPort V.

FALSE - La coordenada P está fuera del rango especificado.

Operación

Utilícese para verificar que un punto va a ser visible luego de una determinada operación.

Véase También

PointDist (Pag. 53)

3.5.5. PointToPolar

Propósito

Convertir coordenadas rectangulares a circulares.

Declaración

```
Type PolarType = Record
                 R: Word;
                 Ag: Real;
                 End;

Procedure PointToPolar (Point: PointType; Var Polar: PolarType);
```

Parámetros

Point: Coordenadas originales a convertir.

Polar: Variable donde se almacenará el resultado.

Véase También

PolarToPoint (Pag. 57)

3.5.6. Polar3DToPoint3D

Propósito

Convertir coordenadas esféricas a cúbicas.

Declaración

```
Type Polar3DType = Record
                   R: Word;
                   AgX,AgZ: Real;
                   End;

Type Point3DType = Record
                   X,Y,Z: Integer;
                   End;

Procedure Polar3DToPoint3D (Polar: Polar3DType; Var Point: Point3DType);
```

Parámetros

Polar: Coordenadas originales a convertir.

Point: Variable donde se almacenará el resultado.

Véase También

Point3DToPolar3D (Pag. 52)

3.5.7. PolarToPoint

Propósito

Convertir coordenadas circulares a rectangulares.

Declaración

```
Type PolarType = Record
                 R: Word;
                 Ag: Real;
                 End;

Procedure PolarToPoint (Polar: PolarType; Var Point: PointType);
```

Parámetros

Polar: Coordenadas originales a convertir.

Point: Variable donde se almacenará el resultado.

Véase También

PointToPolar (Pag. 55)

3.6. Control del Selector

3.6.1. DeSelectPosition

Propósito

Mover la posición seleccionada a la posición corriente del selector, cancelando cualquier posición seleccionada previamente.

Declaración

```
Procedure DeSelectPosition;
```

Operación

En modo automatico, éste procedimiento se invoca cuando el usuario presiona la tecla de deselección (Del). La marca de selección, si está activa, es inactivada.

Observaciones

- Si la marca de selección no es visible, el procedimiento únicamente mueve la posición de selección a la posición corriente.

Véase También

SelectPosition (Pag. 67)

3.6.2. DoubleClick

Propósito

Verificar si se ha realizado una "doble" selección.

Declaración

```
Function DoubleClick: Boolean;
```

Retorna

TRUE - La posición corriente del selector está seleccionada

FALSE - La posición corriente del selector es distinta a la posición de la marca de selección.

Operación

Por lo general, las aplicaciones no necesitan hacer distinciones con respecto a las posiciones del selector y de la marca; sin embargo, dado el caso, ésta función puede ser útil.

Observaciones

- La proximidad de ambas posiciones, necesaria para ser consideradas la misma, está controlada directamente por el rango de selección definido corrientemente.

Véase También

GetPosition (Pag. 60), *GetSelectionRange* (Pag. 61), *SetSelectionRange* (Pag. 69)

3.6.3. GetPosition

Propósito

Obtener la posición del selector o de la marca de selección.

Declaración

```
Procedure GetPosition (PosType: Boolean; Var Coord: PointType);
```

Parámetros

PosType: Flag que indica cúal posición se debe retornar: si es TRUE, se retorna la posición corriente, de lo contrario se retorna la posición de la marca de selección.

Coord: Variable donde se almacenará la coordenada solicitada.

Operación

Por lo general las coordenadas del selector y/o de la marca se deben obtener luego de ejecutar *ReadSelector*, a fin de determinar sobre cúal objeto se ha solicitado una acción.

Observaciones

- Por facilidad al programador, en la sección de interface se han definido las siguientes constantes:

```
Const CurrentPos = True;
      SelectedPos = False;
```

Cualquiera de ellas puede usarse en una llamada a *GetPosition*, aumentando la legibilidad del código.

Véase También

MoveSelector (Pag. 64), *SetPosition* (Pag. 68)

3.6.4. GetSelectionRange

Propósito

Obtener el rango de selección corriente.

Declaración

```
Function GetSelectionRange: Byte;
```

Retorna

El valor corriente del rango de selección (entre 0 y 126).

Operación

El rango de selección afecta a las operaciones de selección y a la función *DoubleClick*; también afecta el aspecto de la marca de selección en pantalla. Esta función puede ser útil para salvar el valor corriente del rango, previo a un cambio temporal del mismo.

Véase También

SetSelectionRange (Pag. 69)

3.6.5. HideMark

Propósito

Borrar la marca de selección.

Declaración

```
Procedure HideMark;
```

Operación

Al llamar a este procedimiento, si la marca de selección es visible, ésta se oculta y el fondo previo a su colocación se restaura.

Véase también

HideSelector (Pag. 63), *SelectPosition* (Pag. 67), *ShowMark* (Pag. 70).

3.6.6. HideSelector

Propósito

Borrar la imagen del selector.

Declaración

```
Procedure HideSelector;
```

Operación

Al llamar a este procedimiento, si el selector es visible, éste se oculta y el fondo previo a su colocación se restaura.

Véase también

HideMark (Pag. 62), *SelectPosition* (Pag. 67), *ShowSelector* (Pag. 71).

3.6.7. MoveSelector

Propósito

Mover el selector en una dirección determinada.

Declaración

```
Type MoveType = (MoveNull, MoveUp, MoveDown, MoveLeft, MoveRight);

Procedure MoveSelector (Direction: MoveType);
```

Parámetros

Direction: Dirección en la que deberá moverse el selector.

Operación

Al llamar al procedimiento, el selector se mueve un determinado número de pixels en la dirección indicada. El número de pixels que se mueve el selector está determinado por la variable de interface *StepSelector*, de tipo Byte (inicialmente ajustada a 10 pixels).

El fondo es preservado durante la operación.

Observaciones

- El procedimiento no permite que el selector sea movido fuera de los límites del ViewPort corriente; el movimiento no se ve restringido por los valores de las variables *GrMinimun* y *GrMaximun*.

Véase también

SetPosition (Pag. 68)

3.6.8. ReadSelector

Propósito

Leer el dispositivo de entrada e interpretar los códigos que correspondan al control del selector.

Declaración

```
Function ReadSelector: Word;
```

Retorna

El código de selector, en formato de dos bytes. Si el código es ASCII normal, se coloca en el byte inferior (derecho) y el superior se ajusta a cero; si se trata de ASCII extendido (teclas funcionales, combinaciones, etc.) el orden se invierte.

Operación

El procedimiento puede opcionalmente ejecutar acciones predefinidas para ciertas teclas. Para esto, se ha definido una variable de interface de la siguiente forma:

```
Var AutoControl: Boolean;
```

Si dicha variable es TRUE al momento de ejecutar el procedimiento, se ejecuta una acción determinada para cada una de las siguientes teclas:

Tecla	Variable	Valor	Acción/Procedimiento
Enter	ConfirmKey	$000D	Ninguna
Ins	SelectKey	$5200	SelectPosition
Del	UnSelectKey	$5300	DeSelectPosition
+	IncStepKey	$002B	Incrementa StepSelector
-	DecStepKey	$002D	Decrementa StepSelector
Esc	CancelKey	$001B	Ninguna
Izquierda	LeftKey	$4B00	Mover selector izquierda
Derecha	RightKey	$4D00	Mover selector derecha
Arriba	UpKey	$4800	Mover selector arriba
Abajo	DownKey	$5000	Mover selector abajo

Cada tecla tiene una variable asociada, la cual es inicializada con el valor indicado. La aplicación puede cambiar las definiciones de estas teclas modificando los valores asignados a las variables de control.

La variable AutoControl se inicializa a TRUE.

Observaciones

- Recuérdese que el valor de la tecla es retornado siempre, sin tomar en cuenta el estado de *AutoControl*; esto permite que la aplicación realice acciones adicionales a las ejecutadas automáticamente.

- Se recomienda que todas las lecturas de teclado se hagan a través de esta función; esto es con el fin de que haya cierta independencia entre el fomato de los códigos y el dispositivo de entrada (teclado, ratón, etc.).

Véase también

GetPosition (Pag. 60)

3.6.9. SelectPosition

Propósito

Activar la marca de selección sobre la posición corriente.

Declaración

```
Procedure SelectPosition;
```

Operación

Cualquier posición previamente seleccionada es deseleccionada. Luego, la marca de selección se coloca sobre la posición corriente del selector.

Véase también

DeSelectPosition (Pag. 58), *SetSelectionRange* (Pag. 69)

3.6.10. SetPosition

Propósito

Ajustar la posición del selector o de la marca de selección.

Declaración

```
Procedure SetPosition (PosType: Boolean; Var Coord: PointType);
```

Parámetros

PosType: Flag que indica cúal posición se debe ajustar: si es TRUE, se ajusta la posición corriente, de lo contrario se ajusta la posición de la marca de selección.

Coord: Coordenada que indica la nueva posición.

Operación

Si las coordenadas finales del selector/marca son conocidas, úsese este procedimiento para ajustar la posición deseada.

El método alternativo es utilizar *MoveSelector* para realizar desplazamientos relativos del selector.

Observaciones

- Por facilidad al programador, en la sección de interface se han definido las siguientes constantes:

```
Const CurrentPos = True;
      SelectedPos = False;
```

Cualquiera de ellas puede usarse en una llamada a *SetPosition*, aumentando la legibilidad del código.

Véase También

GetPosition (Pag. 60), *MoveSelector* (Pag. 64)

3.6.11. SetSelectionRange

Propósito

Ajustar el rango de selección.

Declaración

```
Procedure SetSelectionRange (NewRange: Byte);
```

Parámetros

NewRange: El nuevo valor del rango de selección (debe ser entre 0 y 126).

Operación

El rango de selección afecta a las operaciones de selección y a la función *DoubleClick*; también afecta el aspecto de la marca de selección en pantalla. Si se cambia el rango mientras la marca de selección es visible, ésta se redibuja.

Véase También

GetSelectionRange (Pag. 61)

3.6.12. ShowMark

Propósito

Dibujar la marca de selección.

Declaración

```
Procedure ShowMark;
```

Operación

Al invocar a este procedimiento se dibuja la marca de selección, que consiste en un circulo de puntos. El radio del círculo (y del área seleccionada) depende del rango de selección corriente.

Observaciones

- El procedimiento sólo se ejecuta si la marca no está visible, de lo contrario no se realiza ninguna acción.

Véase también

HideMark (Pag. 62), *ShowSelector* (Pag. 71)

3.6.13. ShowSelector

Propósito

Dibujar el selector.

Declaración

```
Procedure ShowSelector;
```

Operación

Al invocar a este procedimiento se dibuja el selector, que consiste en una flecha que apunta arriba a la izquierda.

Observaciones

- El procedimiento sólo se ejecuta si el selector no está visible, de lo contrario no se realiza ninguna acción.

Véase también

HideMark (Pag. 62), *ShowMark* (Pag. 70)

3.7. Ventanas

3.7.1. CloseWindow

Propósito

Borrar la ventana superior de la lista de ventanas.

Declaración

```
Procedure CloseWindow;
```

Operación

Cada vez que se ejecute este procedimiento se elimina la ventana superior de la "pila". El fondo previo a la creación de la ventana es restaurado y cualquier ventana que existiere inmediatamente "abajo" de la eliminada es reactivada.

Observaciones

- Si se llama al procedimiento cuando no hay ventanas activas, no se toma ninguna acción.

Véase también

OpenWindow (Pag. 75)

3.7.2. ExistWindow

Propósito

Verificar si existen ventanas activas.

Declaración

```
Function ExistWindow: Boolean;
```

Retorna

TRUE - Existe al menos una ventana activa (o "abierta").

FALSE - No hay ventanas activas.

Operación

Utilícese para verificar si hay ventanas activas antes de realizar operaciones con coordenadas relativas (las ventanas activas cambian el origen de coordenadas para algunas rutinas).

Véase también

CloseWindow (Pag. 72), *OpenWindow* (Pag. 75)

3.7.3. MoveWindow

Propósito

Mover la ventana en una dirección determinada.

Declaración

```
Type MoveType = (MoveNull, MoveUp, MoveDown, MoveLeft, MoveRight);

Procedure MoveWindow (Direction: MoveType; Step: Integer);
```

Parámetros

Direction: Dirección en que deberá moverse la ventana.

Step: Número de pixels que deberá moverse la ventana.

Operación

El fondo "debajo" de la ventana se conserva durante la operación. Si el parámetro *Step* es igual a cero, la ventana se desplazará el espacio equivalente a ún caracter; si se desea, se puede utilizar la siguiente constante de interface:

```
Const MoveOneChar = 0;
```

Véase también

OpenWindow (Pag. 75)

3.7.4. OpenWindow

Propósito

Dibujar ("abrir") una ventana gráfica.

Declaración

```
Procedure OpenWindow (C1,F1,C2,F2: Byte; Title: String);
```

Parámetros

C1,F1: Coordenadas texto (columna, fila) de la esquina superior izquierda de la ventana.

C2,F2: Coordenadas texto (columna, fila) de la esquina inferior derecha de la ventana.

Title: Título a imprimir en la primera fila de la ventana.

Operación

Cualquier ventana activa previamente se inactiva (cambia el color de su titulo). El área de trabajo se restringe al espacio interno de la ventana y se almacena en las variables *GrMinimun* y *GrMaximun* para coordenadas gráficas, y *TxMinimun* y *TxMaximun* para coordenadas texto (todas de Tipo *PointType*).

Observaciones

- Las coordenadas especificadas son absolutas, con referencia a la pantalla física.

Véase también

CloseWindow (Pag. 72)

3.8. Botones

3.8.1. CreateButton

Propósito

Crear un botón de respuesta.

Declaración

```
Procedure CreateButton (Col,Row: Byte; StrLabel: String; Transp,Delim: Boolean);
```

Parámetros

Col,*Row*: Coordenadas texto (columna, fila) a partir de la cual se dibuja el botón.

StrLabel: Etiqueta a colocarle al botón; debe tener al menos ún caractér.

Transp: Flag; si es TRUE el boton es transparente, es decir, no es visible.

Delim: Flag; si es TRUE el botón es delimitado (se le dibuja una linea de borde).

Operación

Un menú de botones se crea con llamadas sucesivas a este procedimiento. Si al primer botón se le crea un delimitador (*Delim* = TRUE) se le dibuja un borde doble; éste botón se denomina principal y es el que se activa primero al llamar a *ReadButtons*. Los demás botones que se creen se posicionan con relación al principal, a izquierda/derecha o arriba/abajo.

Aparte de constituir la identificación visual del botón, *StrLabel* también proporciona una tecla de activación para el botón (ésta es una letra que se puede presionar para seleccionar el botón durante la lectura). Esta tecla se toma de la primera letra que se encuentre en el string.

Observaciones

- Cuando se crea un botón transparente, *StrLabel* debe contener exactamente ún caractér (que se convierte en la tecla de activación del mismo).

- Un uso recomendado para botones transparentes es la creación de *icons* (imágenes) que funcionen como opciones de menú: dibújese la imagen y créese un botón transparente directamente sobre la imagen.

- Se pueden crear hasta 255 botones concurrentes.

Véase también

DeleteButtons (Pag. 78), *ReadButtons* (Pag. 79)

3.8.2. DeleteButtons

Propósito

Eliminar todos los botones activos.

Declaración

```
Procedure DeleteButtons;
```

Operación

Llámese a este procedimiento antes de crear un nuevo menú de botones, a fin de eliminar los creados previamente (leer los botones no los elimina).

El fondo previo a la creación de los botones se restaura.

Véase también

CreateButton (Pag. 76), *ReadButtons* (Pag. 79)

3.8.3. ReadButtons

Propósito

Leer el menú de botones y obtener la selección del usuario.

Declaración

```
Function ReadButtons: Byte;
```

Retorna

El número secuencial (de creación) del botón seleccionado. Si el usuario presiona la tecla de cancelación se retorna 0.

Operación

La llamada a esta función activa los botones creados previamente, permitiendo al usuario seleccionar uno de los mismos de la siguiente forma:

- Presionando la tecla de activación asignada a cada botón, la cual usualmente es la primera letra de su etiqueta; o

- Moviendo un campo de selección de un botón a otro, por medio de las teclas de flechas; en este caso, el usuario debe presionar la tecla de confirmación o la tecla de selección, para confirmar su selección.

Véase también

CreateButton (Pag. 76), *DeleteButtons* (Pag. 78)

3.8.4. WaitForOk

Propósito

Efectuar una pausa hasta que el usuario presione una tecla, por medio de un botón etiquetado "Ok".

Declaración

```
Procedure WaitForOk (Col,Row: Byte);
```

Parámetros

Col,*Row*: Coordenadas texto (columna, fila) donde se dibujará el botón.

Operación

El procedimiento presenta el botón etiquetado "Ok" y luego espera que el usuario presione la tecla de confirmación o de selección, tras lo cual el botón es borrado. A diferencia de lo que pasa con los botones normales, el espacio ocupado no es restaurado, lo cual queda entonces bajo responsabilidad de la aplicación.

El método más común para realizar una pausa por medio de un botón es borrar los botones anteriores, crear uno, leerlo y después borrarlo. Este procedimiento combina todas las acciones necesarias en una sola llamada, con el dividendo adicional de que los botones creados previamente no son alterados, como lo serían con la secuencia antes descrita. En consecuencia, es perfectamente posible llamar a este procedimiento dentro de un bucle que contenga un menú de botones sin necesidad de estar creando y borrando dicho menú en cada iteración.

Observaciones

- Siendo el procedimiento independiente de los botones ya creados, se puede ejecutar aunque se exceda el límite máximo de botones activos.

Véase También

CreateButton (Pag. 76)

3.9. Menús

3.9.1. CreateMenu

Propósito

Crear un menú general en la primera linea de la pantalla.

Declaración

```
Procedure CreateMenu (Heading: String);
```

Parámetros

Heading: Debe ser un string con la siguiente forma: 'OPCION1¦OPCION2¦...¦OPCIONn'

Operación

Para conformar el menú coloque en un string los nombres de las opciones principales del menú, separadas por el caracter '¦' (ASCII 124). Por ejemplo, para formar un menú con tres opciones principales INGRESO, ACTUALIZACION y REPORTES, llamamos al procedimiento de la siguiente forma:

```
CreateMenu ('ingreso¦actualización¦reportes');
```

Se pueden colocar espacios en blanco iniciales, embebidos y/o finales; sin embargo, no es necesario ya que el procedimiento coloca un espacio de separación en la pantalla antes y después de cada opción.

Las opciones del menú se pueden seleccionar utilizando las flechas izquierda y derecha, o presionando la primera letra de cada opción.

Observaciones

- El procedimiento asume que la pantalla está en modo gráfico. Si no lo está se generará un error de ejecución.

- Las opciones se hacen visibles en la primera linea del monitor; sin embargo el menú no es activado. Para permitir la entrada de datos, utilice *ReadInput*.

- El tipo de letra utilizado para el menú está especificado por la variable DefaultFont.

- Una vez que se ha creado el menú, las coordenas gráficas superiores (*GrMinimum*) se restringen a la línea inferior del mismo.

Véase También

CreateSubMenu (Pag. 83), *ReadInput* (Pag. 85)

3.9.2. CreateSubMenu

Propósito

Crear un submenú asociado a una opción del menú general establecido previamente con CreateMenu

Declaración

```
Procedure CreateSubMenu (HeadNum: Byte; Options: String);
```

Parámetros

HeadNum: Número secuencial de la opción del menú general a la que va asociarse el submenú; se cuenta desde uno, empezando desde la extrema izquierda.

Heading: Debe ser un string con la siguiente forma: 'OPCION1¦OPCION2¦...¦OPCIONn'

Operación

Llámese a este procedimiento después de haber definido el menú general. El string de definición se especifica de la misma forma que para *CreateMenu*.

No es obligatorio crear submenús para todas las opciones, pero en la práctica ayuda a la organización de la aplicación.

Observaciones

- El tipo de letra utilizado para el menú está especificado por la variable *DefaultFont*.

- El submenú no aparece fisicamente en la pantalla hasta que el menú general se activa (durante *ReadInput*).

Véase También

CreateMenu (Pag. 81), *ReadInput* (Pag. 85), *ToggleOption* (Pag. 90), *ToggleWhenSelect* (Pag. 91)

3.9.3. GetMenuTitle

Propósito

Obtener el titulo de la opción seleccionada el el menú.

Declaración

```
Function GetMenuTitle: String;
```

Retorna

El nombre (definido por *CreateSubMenu*) de la opción seleccionada en el submenú activo. Si el menú activo no tiene submenú, se retorna el nombre del menú (definido por *CreateMenu*). Si no se ha definido el menú general o éste no está activo, se retorna el título por omisión, especificado en la variable *DefaultTitle*.

Operación

Puede usarse para mostrar por pantalla la opción corriente, como una confirmación visual para el usuario.

Observaciones

- El titulo por omisión definido inicialmente es '(sin titulo)', el cual puede modificarse fácilmente alterando la variable *DefaultTitle*.

Véase También

CreateMenu (Pag. 81), *CreateSubMenu* (Pag. 83)

3.9.4. ReadInput

Propósito

Controlar la entrada de datos desde el teclado.

Declaración

```
Procedure ReadInput;
```

Operación

Se utiliza para permitir al usuario la selección de un objeto en pantalla y de una opción del menú.

En el momento de llamar al procedimiento el selector es activado y aparece en su última posición; si la variable booleana *AutoDeSelect* es TRUE, cualquier objeto seleccionado previamente es de-seleccionado.

Las siguientes teclas están definidas:

[Esc] - Activa/desactiva el menú

[Enter] - Si el menú está activo selecciona la opción corriente, de lo contrario activa la opción por omisión

[letra] - Selecciona la opción (del menú principal o de un submenú) que empiece con la letra indicada

[Ins] - Si el menú está activo selecciona la opción corriente

[izquierda],[derecha] - Activan el menú izquierdo/derecho al actual, respectivamente

[arriba],[abajo] - Seleccionan la opción superior/inferior a la actual en el submenú activo

Adicionalmente a las funciones mencionadas, cuando el menú se encuentra inactivo las teclas de control del selector están vigentes (véase *ReadSelector* bajo "Control del Selector").

El código de la opción seleccionada se retorna en la variable *InputCode*, en la siguiente forma:

- 1er. digito: número del submenú dentro del menú
- 2do. digito: número de la opción dentro del submenú

Por ejemplo: 36 indica tercer submenú, sexta opción.

Observaciones

- Cuando se activa el menú mediante [Esc] la opción seleccionada previamente se conserva (la primera vez se selecciona la primera opción del primer submenú); la aplicación puede controlar cual será esta opción ajustando el valor de *InputCode* antes de llamar al procedimiento.

Véase También

CreateMenu (Pag. 81), *ReadStr* (Pag. 87), *SetDefaultOption* (Pag. 89)

3.9.5. ReadStr

Propósito

Leer un string del teclado, con control de edición, en modo gráfico.

Declaración

```
Function ReadStr (Col,Row,Width: Byte): String;
```

Parámetros

Col,*Row*: Coordenada texto (columna, fila) donde se colocará el campo de ingreso.

Width: Longitud máxima, en caracteres, del string a ingresar.

Retorna

El string ingresado. Si el usuario presiona [Esc] durante el ingreso, se retorna un string nulo (longitud 0).

Operación

El procedimiento stándard *ReadLn* no funciona adecuadamente cuando la pantalla está en modo gráfico; utilícese esta función en cambio.

Para delimitar visualmente el área de ingreso, un rectángulo es dibujado alrededor de ésta.

Los únicos caracteres permitidos son letras, números y espacios en blanco.

La única tecla de edición definida es [BackSpace], la cual borra el último caractér ingresado. El ingreso termina presionando [Enter].

Observaciones

- EL procedimiento no restaura el área cubierta por el string durante el ingreso. Por lo tanto, la aplicación es responsable por el mantenimiento de alguna imagen de fondo que pudiera cubrirse.

- Todas las letras ingresadas se convierten a mayúsculas durante el ingreso.

Véase También

ReadInput (Pag. 85)

3.9.6. SetDefaultOption

Propósito

Ajustar la opción por omisión (default).

Declaración

```
Procedure SetDefaulOption (MenuNum,OptionNum: Byte);
```

Parámetros

MenuNum: Número de identificación del submenú; se cuenta desde uno, empezando desde la extrema izquierda.

OptionNum: Número de identificación de la opción; se cuenta desde uno, empezando desde el extremo superior.

Operación

La opcion por omisión se es aquella que se retorna cuando el usuario presiona la tecla [Enter] durante ReadInput, estando el menú inactivo. Aunque no es obligatorio su uso, sí es recomendado.

Observaciones

- Si alguno de los parámetros está fuera de rango, el procedimiento no ajusta la opción.

- La opción por defecto siempre se retorna al presionar [Enter], aunque dicha opción se haya declarado inactiva mediante ToggleOption.

Véase También

ReadInput (Pag. 85), *ToggleOption* (Pag. 90)

3.9.7. ToggleOption

Propósito

Cambiar el estado de una opción en un submenú

Declaración

```
Procedure ToggleOption (MenuNum,OptionNum: Byte);
```

Parámetros

MenuNum: Número de identificación del submenú; se cuenta desde uno, empezando desde la extrema izquierda.

OptionNum: Número de identificación de la opción; se cuenta desde uno, empezando desde el extremo superior.

Operación

Una opción puede tener dos estados: activa o inactiva. Una opción inactiva se caracteriza por una línea gruesa que cruza su nombre en el menú; cuando está en ese estado la opción no puede ser seleccionada (a menos que sea la opción por omisión).

Llamadas consecutivas e este procedimiento cambian el estado de la opción especificada de activa a inactiva y viceversa.

Observaciones

- El procedimiento sólo se puede llamar mientras cuando el menú se ha creado y mientras no esté activo. El nuevo estado de la opción se reflejará en la siguiente activación del menú.

Véase También

SetDefaultOption (Pag. 89), *ToggleWhenSelect* (Pag. 91)

3.9.8. ToggleWhenSelect

Propósito

Designar el estado de una opción en un submenú como dependiente del estado del indicador de selección.

Declaración

```
Procedure ToggleWhenSelect (MenuNum,OptionNum: Byte);
```

Parámetros

MenuNum: Número de identificación del submenú; se cuenta desde uno, empezando desde la extrema izquierda.

OptionNum: Número de identificación de la opción; se cuenta desde uno, empezando desde el extremo superior.

Operación

Una opción puede tener dos estados: activa o inactiva. Una opción inactiva se caracteriza por una línea gruesa que cruza su nombre en el menú; cuando está en ese estado la opción no puede ser seleccionada (a menos que sea la opción por omisión).

Este procedimiento ajusta la opción deseada de tal manera que sólo está activa si el usuario activa a su vez el indicador de selección durante ReadInput. Para retornar la opción a su estado normal vúelvase a ejecutar el procedimiento.

Observaciones

- El procedimiento sólo se puede llamar mientras cuando el menú se ha creado y mientras no esté activo. El nuevo estado de la opción se reflejará en la siguiente activación del menú.

Véase También

ToggleOption (Pag. 90)

4. Detalles internos de cada procedimiento

4.1. After

Unidad

STRINGS

Algoritmo

Se realiza una búsqueda del string Target en Source. Si no se lo encuentra, se retorna un string nulo; de lo contrario se retorna una copia de Source desde la posición inmediatamente posterior a la primera ocurrencia de Target hasta el último caractér.

Definiciones

P: Variable Byte utilizada como índice de la primera posición de Target en Source.

4.2. Before

Unidad

STRINGS

Algoritmo

Se realiza una búsqueda del string *Target* en *Source*. Si no se lo encuentra, se retorna *Source*; de lo contrario se retorna una copia de *Source* desde el primer caractér hasta la posición inmediatamente anterior a la primera ocurrencia de Target.

Definiciones

P: Variable Byte utilizada como índice de la primera posición de *Target* en *Source*.

4.3. BiosDate

Unidad

FUNCTIONS

Algoritmo

La variable es inicializada por el procedimiento *GetBiosDate*, el cual simplemente mueve la fecha de su localización prederminada (desplazamiento FFF5 hex dentro del segmento ROM F hex) a una variable string interna, que es posteriormente retornada.

Definiciones

StrAux: Variable string de 8 posiciones utilizada como almacenamiento intermedio.

Date: Variable byte asumida en la posición inicial de la fecha en el ROM.

Comentarios

Ya que la información deseada es constante para una aplicación, se consideró innecesario colocar la función *GetDosVersion* en la sección de interface, prefiriendo el más simple acceso por medio de una variable de interface.

4.4. BitOn

Unidad

FUNCTIONS

Algoritmo

El parámetro *Valor* es rotado a la derecha de tal manera que el bit solicitado se encuentre en la posición extrema derecha. Este valor resultante es ANDeado con la máscara $01, el resultado de lo cual se compara con el mismo valor $01; el resultado de esta comparación es retornado por la función.

Definiciones

Ninguna.

4.5. BlinkArea

Unidad

GRLIBR

Algoritmo

Luego de validar que las coordenadas sean correctas, el procedimiento toma la imagen que se encuentra en dichas coordenadas y realiza una operación de NOT lógico para cada uno de sus pixels; para esto se vale de las facilidades provistas por el procedimiento standard *PutImage*.

Luego se realiza una pausa de 1.5 décimas de segundo (aproximadamente), antes de restaurar el aspecto original del área.

Definiciones

Buff: Variable Pointer que apunta a un bloque donde se almacena temporalmente la imagen original del área.

Size: Variable Word que almacena el tamaño en bytes del área apuntada por *Buff*.

4.6. ByteToHex

Unidad

FUNCTIONS

Algoritmo

La función se basa en un arreglo de 16 caracteres con los 16 símbolos hexadecimales, que se utiliza como una tabla de búsqueda. El parámetro pasado se descompone en dós números con los cuatro bits superiores e inferiores respectivamente, los cuales son utilizados para extraer de la tabla los símbolos deseados.

Definiciones

Hex: Arreglo constante de caracteres, que se utiliza como tabla.

4.7. Center

Unidad

STRINGS

Algoritmo

Primero se verifica si la longitud del string a centrar es mayor que el campo *N* a utilizar; si es así, se retornan los primeros *N* caracteres del string.

De lo contrario, se calculan cuantos caracteres se necesitan a cada lado del string para llenar el campo; se forma el campo con un string de *N* espacios en blanco, en el cual se inserta el string original en la posición apropiada. Finalmente se retorna el campo.

Definiciones

LenS: Variable byte que contendrá la longitud del string original.

Fill: Variable byte que contiene el número de caracteres de relleno del campo.

Filler: Variable string que contendrá el campo.

4.8. CenterIn

Unidad

STRINGS

Algoritmo

Primero se verifica si la longitud del string a centrar es mayor que el campo *N* a utilizar; si es así, se retornan los primeros *N* caracteres del string.

De lo contrario, se calculan cuantos caracteres se necesitan a cada lado del string para llenar el campo; se forma el campo con un string de *N* caracteres *Ch*, en el cual se inserta el string original en la posición apropiada. Finalmente se retorna el campo.

Definiciones

LenS: Variable byte que contendrá la longitud del string original.

Fill: Variable byte que contiene el número de caracteres de relleno del campo.

Filler: Variable string que contendrá el campo.

4.9. ClearArea

Unidad

GRLIBR

Algoritmo

El área se borra "pintándola" con el color de fondo; para esto, se ajusta temporalmente el seteo de relleno (*Fill*), para luego utilizar el procedimiento standard *Bar*.

Definiciones

OldFill: Variable FillSettingsType que almacena el seteo de relleno al comienzo del procedimiento.

4.10. ClearView

Unidad

GRLIBR

Algoritmo

El Área se borra ajustando temporalmente el ViewPort a las coordenadas proporcionadas y ejecutando el procedimiento standard *ClearViewPort*.

Definiciones

OldView: Variable ViewPortType que almacena el seteo de ViewPort al comienzo del procedimiento.

4.11. CloseWindow

Unidad

GRWINDOW

Algoritmo

Primeramente se realiza el "cierre" visual de la pantalla sobre el fondo previo que después de finalizar el efecto es restaurado. Luego se restauran las condiciones previas a la creación de la ventana (nótese que esto es necesario aunque no haya una ventana activa debajo de la actual).

A continuación se ajusta el tope del stack hacia la ventana inferior (o se anula si no hay ventana inferior), y si es necesario se activa la ventana inferior. Finalmente se libera la memoria ocupada por la vieja ventana.

Definiciones

OldWindow: Variable WindowPtr que apunta a la vieja ventana.

Xo,*Yo*,*StepX*,*StepY*,*Xp*,*Yp*: Variables Integers utilizadas en el cálculo del cierre.

4.12. CreateButton

Unidad

GRBUTTON

Algoritmo

Primero se valida el string *StrLabel*, y se busca el primer caractér alfabético con el cual se incializará el HotKey. Luego se incrementa el contador de botones y se calculan las coordenadas del nuevo botón.

A continuación se crea el registro correspondiente y se incializan sus campos respectivos. Si el botón no es transparente se almacena el área a ocuparse y si es necesario, se dibuja el borde correspondiente con el procedimiento interno *DrawButton*, procediendo luego a escribir la etiqueta con el procedimiento interno *WriteButtonLabel* (recuérdese que, por definición, un botón transparente no tiene borde ni etiqueta visible).

El siguiente paso, si el botón no es el principal o primer botón, es buscar la posición relativa del botón con respecto al botón principal, lo cual se hace comparando las coordenadas de su esquina superior izquierda, y recorriendo la cadena de botones correspondiente hasta que se encuentre un lugar vacío. Una vez colocado el botón en su nueva posición el procedimiento finaliza.

Definiciones

NewButt: Variable ButtonPtr que almacenará los datos del nuevo botón.

SearchPtr: Variable ButtonPtr que servirá para buscar la posición del nuevo botón.

Pt1,*Pt2*: Variables PointType que almacenan las coordenadas gráficas de las esquinas superior izquierda e inferior derecha del nuevo botón, respectivamente.

Dir: Variable MoveType que servirá durante la búsqueda de la posición del nuevo botón.

Len: Variable Byte que almacena la longitud de la etiqueta.

P: Variable Byte que se utiliza como índice en el bucle de búsqueda del HotKey.

4.13. CreateMenu

Unidad

GRENV

Algoritmo

Primeramente se almacenan las condiciones actuales de texto y ViewPort, las cuales se ajustan a los valores apropiados para dibujar el menú.

Luego se valida que *Heading* contenga al menos tres caracteres (cantidad mínima aceptable); se contabilizan los caracteres de separación que se encuentren en

Heading y, tomando como base este conteo, se reserva la memoria necesaria para el menú.

A continuación se entra en un bucle que recorre las opciones del menú y realiza las siguientes acciones:

- Extrae el nombre de la opción de las primeras posiciones de *Heading*, eliminándolo del mismo.

- Inicializa el número y las coordenadas de inicio de la opción.

- Busca el primer caractér alfabético en el nombre e inicializa con él el *HotKey* o tecla de activación. También coloca dicha tecla en el conjunto de *HotKeys* activos.

- Encera los campos de control del submenú; y

- Escribe el nombre en la posición apropiada.

Para finalizar, se dibuja una línea de separación en la fila inmediatamente inferior al menú, y se restauran las condiciones previas de texto y ViewPort.

Definiciones

P,N: Variables Byte que se utilizan como índices de los bucles, para *Heading* y el menú respectivamente.

LenHead,LenItem: Variables Byte que almacenan las longitudes del heading y de cada nombre de opción, respectivamente.

StartCol: Variable Word que almacena la columna inicial para cada opción.

OldStyle: Variable TextSettingsType que almacena el seteo de texto al comienzo del procedimiento.

OldView: Variable ViewPortType que almacena el seteo de ViewPort al comienzo del procedimiento.

4.14. CreateSubMenu

Unidad

GRENV

Algoritmo

Primeramente se almacenan las condiciones actuales de texto, las cuales se ajustan a los valores apropiados para dibujar el menú.

Luego se contabilizan los caracteres de separación que se encuentren en *Options* y, tomando como base este conteo, se reserva la memoria necesaria para el submenú.

A continuación se entra en un bucle que recorre las opciones del submenú y realiza las siguientes acciones:

- Extrae el nombre de la opción de las primeras posiciones de *Options*, eliminándolo del mismo.

- Inicializa el número y las coordenadas de inicio de la opción.

- Busca el primer caractér alfabético en el nombre e inicializa con él el *HotKey* o tecla de activación.

- Ajusta los flags a los valores normales, y

- Verifica y si es necesario reajusta el ancho máximo del conjunto opción-menú.

Para finalizar, se almacena el ancho máximo del submenú y se restauran las condiciones previas de texto.

Definiciones

P,N: Variables Byte que se utilizan como índices de los bucles, para *Options* y el submenú respectivamente.

LenOptions: Variable Byte que almacena la longitud de Options.

Width,MaxWidth: Variables Byte que almacenan los anchos (en pixels) de la opción corriente y el ancho máximo encontrado, respectivamente.

StartRow: Variable Byte que almacena la fila inicial para cada opción.

OldStyle: Variable TextSettingsType que almacena el seteo de texto al comienzo del procedimiento.

4.15. DefaultDrive

Unidad

FILES

Algoritmo

Simplemente se hace una llamada al DOS para obtener la información, la cual se toma del registro AX y se retorna.

Definiciones

Regs: Variable Registers utilizada en la llamada al DOS.

4.16. DegToRad

Unidad

FUNCTIONS

Algoritmo

El parámetro se multiplica por el resultado de dividir la función standard *Pi* sobre 180 (grados), y el resultado se retorna.

Definiciones

Ninguna.

4.17. DeleteButtons

Unidad

GRBUTTON

Algoritmo

El procedimiento consiste básicamente en un bucle que recorre las cuatro direcciones a partir del botón principal, eliminando las cuatro posibles listas enlazadas simples de botones que puede haber (para cada botón que tenga un fondo grabado, éste es restaurado antes de disponer del botón.

Finalmente se dispone del propio botón principal y se enceran las variables de conteo y control.

Definiciones

Dir: Variable MoveType que servirá de índice al bucle de eliminación.

Old,*Nxt*: Variables ButtonPtr que sirven para recorrer las listas de botones.

4.18. DeSelectPosition

Unidad

GRSELECT

Algoritmo

Se ocultan los símbolos del selector y de la marca; se actualiza la coordenada seleccionada y, si estaba originalmente activo, se vuelve a displayar el selector.

Definiciones

Visible: Variable Boolean utilizada para conservar el ajuste original de visibilidad del selector (dicho ajuste varía cuando el selector es ocultado).

4.19. DiskError

Unidad

FILES

Algoritmo

Si el byte en la dirección adecuada es 0, se retorna este valor, de lo contrario se convierte el valor deseado al formato decimal y se retorna el resultado más uno (se suma uno para poder retornar 0 en caso de que no haya error).

Definiciones

Ln2: Constante Real igual al logaritmo natural de 2, utilizada en la transformación de base 2 a base 10.

Flag: Variable byte asumida en la posición de memoria 41 hex dentro del área de datos del BIOS (segmento 40 hex).

4.20. DnCase

Unidad

STRINGS

Algoritmo

Si el caractér a convertir se encuentra en el rango de letras mayúsculas, se retorna el caractér que se encuentra 32 posiciones más "adelante" en la tabla ASCII (es decir, su equivalente minúscula); de lo contrario se retorna el mismo caractér.

Definiciones

Ninguna.

4.21. DosVersion

Unidad

FUNCTIONS

Algoritmo

La variable es inicializada por el procedimiento *GetDosVersion*, el cual simplemente ejecuta una llamada al DOS para solicitar la versión. El resultado se toma del registro AX, se convierte al formato string y se almacena.

Definiciones

Regs: Variable Registers utilizada en la llamada al DOS.

N1,*N2*: Variables strings de 2 caracteres utilizadas durante la concatenación del string resultado.

Comentarios

Ya que la información deseada es constante para una aplicación, se consideró innecesario colocar la función *GetDosVersion* en la sección de interface, prefiriendo el más simple acceso por medio de una variable de interface.

4.22. DoubleClick

Unidad

GRSELECT

Algoritmo

Simplemente se retorna el resultado de verificar que las coordenadas corrientes (variable *Current*) estén a no más de *Range* pixels de las coordenadas seleccionadas (variable *Selected*) correspondientes.

Definiciones

Ninguna.

4.23. DownCase

Unidad

STRINGS

Algoritmo

Se recorre con un bucle el string original y se pasa cada uno de sus elementos por la función *DnCase*. Luego se retorna el string resultante.

Definiciones

P: Variable byte utilizada como índice del bucle de conversión.

4.24. DrawArrow

Unidad

GRLIBR

Algoritmo

La flecha se forma simplemente combinando en el mismo espacio los caracteres > y -, en distintas orientaciones según la dirección que se haya indicado.

Definiciones

Txt: Variable TextSettingsType que almacena el seteo de texto al comienzo del procedimiento, parte del cual se utiliza para dibujar el caractér flecha.

4.25. DriveAtStart

Unidad

FILES

Algoritmo

La variable es inicializada por medio de la función *DefaulDrive*, el resultado de la cual es convertida a la letra mayúscula correspondiente.

Definiciones

Ninguna.

4.26. Equal

Unidad

FUNCTIONS

Algoritmo

Para fines de comparación se trata a las dos variables como si fueran arreglos byte de longitud igual al parámetro *Size*. EL algoritmo consiste entonces simplemente en un bucle que recorre los dos arreglos desde la posición 0 hasta *Size* - 1 y que termina cuando se haya procesado el último elemento o cuando los dos bytes correspondientes son distintos. Dependiendo de las condiciones en que finalice el bucle, se retorna el resultado deseado.

Definiciones

Block: Tipo arreglo byte de longitud indeterminada.

Par1: Variable Block que se asume en la misma dirección que el parámetro V1.

Par2: Variable Block que se asume en la misma dirección que el parámetro V2.

I: Variable Word que se utiliza como índice del bucle de comparación.

Comentarios

Definir el parámetro *Size* como Word impone el límite de 64K para el tamaño de las variables comparadas. En la versión corriente de Turbo Pascal éste el el tamaño máximo de una variable, por lo que no debe ser restricción. Sin embargo, de cambiar el límite, se puede redefinir la variable como entero largo (*LongInt*).

4.27. EscPressed

Unidad

FUNCTIONS

Algoritmo

Si no hay una tecla pendiente en el buffer se retorna FALSE, de lo contrario se recupera la tecla y se retorna el resultado de la comparación del dicha tecla con el valor de [Esc].

Definiciones

Esc - Constante caractér que contiene el código ASCII de [Esc] (27 decimal).

Tecla - Variable caractér que contendrá el código de la tecla leída del buffer.

Comentarios

Cambiando el valor asignado a la constante *Esc* es posible hacer procedimientos sensitivos a otras teclas. En tal caso, lo aconsejable sería pasar dicho valor como parámetro y definir Esc en la sección de interface (a fin de que la aplicación pueda utilizarla si desea).

4.28. ExistFile

Unidad

FILES

Algoritmo

Se intenta abrir el supuesto nombre de archivo; si la operación fur normal el archivo se cierra y se retorna TRUE, de lo contrario se retorna FALSE.

Definiciones

Archivo: Variable Archivo texto utilizado para la prueba.

4.29. ExistWindow

Unidad

GRWINDOW

Algoritmo

Se retorna el resultado de verificar que el apuntador a la primera ventana no sea nulo.

Definiciones

Ninguna.

4.30. FirstWord

Unidad

STRINGS

Algoritmo

Se busca desde la primera posición del string, hacia adelante, un caractér que se encuentre en el conjunto de inválidos. Luego se retorna una copia del string original, desde el primer caractér hasta la posición inmediatemente anterior al caractér inválido encontrado.

Definiciones

P: Variable byte utilizada como índice del bucle de búsqueda.

4.31. GetMenuTitle

Unidad

GRENV

Algoritmo

Si no se ha creado aún el menú principal o si el código de la opción seleccionada no es válido se retorna el titulo por omisión; de lo contrario se calcula el número del menú y submenú correspondientes: si el menú no tiene un submenú se retorna el nombre del menú, de lo contrario se retorna el nombre del submenú.

Definiciones

MenuNum,*SubNum*: Variables Byte que almacenan los números de menú y submenú, respectivamente.

4.32. GetPosition

Unidad

GRSELECT

Algoritmo

Si el tipo de posición solicitada es la corriente se ajusta el parámetro al valor de la variable *Current*, de lo contrario se ajusta el parámetro al valor de la variable *Selected*.

Definiciones

Ninguna.

4.33. GetSelectionRange

Unidad

GRSELECT

Algoritmo

Simplemente se retorna el valor de la variable interna *Range*.

Definiciones

Ninguna.

Comentarios

Definir la variable *Range* en la sección de interface hubiera hecho innecesario este procedimiento (y su contrario *SetSelectionRange*); sin embargo debido a las diversas condiciones y variables que dependen de este valor, y al hecho de que se deben monitorear los cambios realizados, se decidió declarla interna.

4.34. GetView

Unidad

GRLIBR

Algoritmo

Simplemente se colocan en los campos proporcionados los valores corrientes de las variables *GrMinimun* y *GrMaximun*.

Definiciones

Ninguna.

4.35. HexToByte

Unidad

FUNCTIONS

Algoritmo

La función se basa en un string de 15 caracteres con 15 simbolos hexadecimales (sin incluir 0), que se utiliza como una tabla de búsqueda. Cada uno de los caracteres hexadecimales del parámetro se busca en esta tabla, y su posición se multiplica por su peso a fin de formar las partes del resultado.

Definiciones

Hex: String constante de 15 posiciones que almacena la tabla de búsqueda.

4.36. HideMark

Unidad

GRSELECT

Algoritmo

Primeramente se calcula las coordenadas del área que se alteraró con la marca; con esta información se restaura la imagen previa a la colocación del selector, luego se libera la memoria ocupada por dicha imagen, y se inactiva el flag de control correspondiente.

Definiciones

Ninguna.

4.37. HideSelector

Unidad

GRSELECT

Algoritmo

Simplemente se restaura la imagen previa a la colocación del selector, se libera la memoria ocupada por dicha imagen, y se inactiva el flag de control correspondiente.

Definiciones

Ninguna.

4.38. InvertArea

Unidad

GRLIBR

Algoritmo

Luego de validar que las coordenadas sean correctas, el procedimiento toma la imagen que se encuentra en dichas coordenadas y realiza una operación de NOT lógico para cada uno de sus pixels; para esto se vale de las facilidades provistas por el procedimiento standard *PutImage*.

Definiciones

Buff: Variable Pointer que apunta a un bloque donde se almacena temporalmente la imagen original del área.

Size: Variable Word que almacena el tamaño en bytes del área apuntada por *Buff*.

4.39. LTrim

Unidad

STRINGS

Algoritmo

Se busca desde la primera posición del string, hacia adelante, un caractér que no se encuentre en el conjunto de inválidos. Luego se retorna una copia del string original, desde el primer caractér válido encontrado hasta el final del mismo.

Definiciones

P: Variable byte utilizada como índice del bucle de búsqueda.

L: Variable byte utilizada para almacenar la longitud del string original.

4.40. MoveSelector

Unidad

GRSELECT

Algoritmo

Depenciendo de la dirección en que se mueva el selector, se verifica si la noeva posición no queda fuera de la pantalla; si no es así, se oculta el selector, se varía el valor de la coordenada correspondiente en *StepSelector* unidades, y si es necesario se vuelve a displayar el selector.

Definiciones

View: Variable ViewPortType que almacena el seteo de ViewPort corriente.

Visible: Variable Boolean utilizada para conservar el ajuste original de visibilidad del selector (dicho ajuste varía cuando el selector es ocultado).

4.41. MoveWindow

Unidad

GRWINDOW

Algoritmo

Luego de validar las condiciones de trabajo, lo primero que hace el procedimiento es verificar si se ha pasado el flag que indica el movimiento de ún caractér; si es así, se ajusta el paso convenientemente.

El siguiente paso es verificar si mover la ventana en la dirección y cantidad indicada colocará a la ventana fuera de los límites de la pantalla física; si es así el procedimiento termina, de lo contrario se calcula el valor del desplazamiento que deberá aplicarse a las coordenadas de la ventana.

A continuación se toma la imagen de la ventana y se graba en una variable dinámica temporal, tras lo cual se restaura el fondo almacenado en el registro de la pantalla.

Luego se varían las coordenadas columna y fila apropiadamente y se almacena el fondo previo de la nueva posición, colocando a continuación la ventana en la pantalla.

Finalmente se almacenan las nuevas coordenadas de la pantalla y se libera la memoria utilizada para almacenar la imagen de la ventana.

Definiciones

WindowImage: Variable Pointer que apuntará a un bloque utilizado como almacenamiento temporal de la imagen de la ventana.

X1,Y1,X2,Y2: Variables Integers que almacenan las nuevas coordenadas izquierda, superior, derecha e inferior de la ventana, respectivamente.

Dx,Dy: Variables Integers que almacenan los desplazamientos respectivos horizontales y verticales.

Comentarios

El procedimiento es simple de ampliar para que abarque casos de movimientos diagonales; si embargo el autor no los ha implementado debido a que no los considera muy útiles.

4.42. OpenWindow

Unidad

GRWINDOW

Algoritmo

Si existe una ventana previa, esta se inactiva. Luego se calculan las coordenadas gráficas de la nueva ventana y se reserva memoria para la misma.

Luego se almacenan las condiciones en las cuales se abre la ventana y el fondo previo. La ventana es colocada en la parte superior del stack.

A continuación se realiza el efecto de "apertura" y se almacenan las condiciones actuales de texto, las cuales se ajustan a los valores apropiados para dibujar la ventana.

Finalmente se dibujan los bordes y el título, se restauran las condiciones del texto, y se ajustan las coordenadas mínimas y máximas de trabajo al interior de la ventana.

Definiciones

DefaultCharSize: Constante Byte igual al tamaño del texto a utilizar en el título.

X1,*Y1*,*X2*,*Y2*: Variables Integer que almacenan las coordenadas izquierda, superior, derecha e inferior de la ventana, respectivamente.

Y3,*Xo*,*Yo*,*StepX*,*StepY*,*Xp*,*Yp*: Variables Integers utilizadas en el cálculo de la apertura.

NewWindow: Variable WindowPtr que apunta a la nueva ventana.

OldStyle: Variable TextSettingsType que almacena el seteo de texto al comienzo del procedimiento.

4.43. PointInView

Unidad

GRCOORD

Algoritmo

Simplemente se retorna el resultado de validar que las coordenadas X y Y del punto estén dentro de las coordenadas del ViewPort indicado.

Definiciones

Ninguna.

Comentarios

Nótese que no se ha utilizado el operador IN para realizar la validación. Esto es debido a que el operador IN está diseñado para trabajar con conjuntos, los cuales sólo tienen un rango de 1 byte (0 - 254); los valores de las coordenadas son Integer (dos bytes), por lo que, aunque sintácticamente se puede utilizar el operador IN con ellos, los resultados obtenidos serían erróneos ya que sólo se tomaría el byte más bajo para la comparación.

4.44. PointsH

Unidad

GRLIBR

Algoritmo

Se resta uno de la coordenada proporcionada (a fin de empezar el conteo desde 0) y se multiplica por el número de pixels horizontales que ocupa un caractér, retornándose el resultado.

Definiciones

Ninguna.

4.45. PointsV

Unidad

GRLIBR

Algoritmo

Se resta uno de la coordenada proporcionada (a fin de empezar el conteo desde 0) y se multiplica por el número de pixels verticales que ocupa un caractér, retornándose el resultado.

Definiciones

Ninguna.

4.46. PrintText

Unidad

GRLIBR

Algoritmo

Se calculan las coordenadas gráficas correspondientes a las coordenadas texto proporcionadas; luego se usan éstas para imprimir el texto por medio del procedimiento standard *OutTextXY*.

Definiciones

X,Y: Variables Integer que almacenan las coordenadas gráficas calculadas.

4.47. RadToDeg

Unidad

FUNCTIONS

Algoritmo

El parámetro se multiplica por el resultado de dividir 180 (grados) sobre la función standard *Pi*, y el resultado se retorna.

Definiciones

Ninguna.

4.48. ReadButtons

Unidad

GRBUTTON

Algoritmo

Como primer paso se desconecta el control del selector; también se deseleccionan todos los botones que hayan quedado activos de una llamada anterior, por medio del procedimiento interno *UnSelectAll*. Luego se selecciona el botón principal.

A continuación se realiza un bucle de lectura de la entrada: los códigos correspondientes a las teclas de

movimiento se manejan con el procedimiento interno *ChangeSelect* (que básicamente actualiza un apuntador al botón seleccionado y lo invierte); si se presiona una tecla normal (el byte bajo del código es distinto de 0) el procedimiento interno *SearchHotKey* verifica si se trata de uno de los *HotKeys* o teclas de activación; el bucle termina si se presiona la tecla de Cancelar (Esc), Confirmar (Enter) o Seleccionar (Ins).

Finalmente se ajusta el control de selector a su estado original y, si el bucle terminó con la tecla de cancelación, se retorna 0, de lo contrario se retorna el número del botón seleccionado.

Definiciones

KeyCode: Variable Word que almacena el código de la tecla presionada.

Final: Variable Boolean utilizada como señal para el fin del bucle de lectura.

OldControl: Variable Boolean que almacena el estado original de la variable de control automático del selector (GrSelect.AutoControl).

SelectButt: Variable ButtonPtr que apunta al botón corrientemente seleccionado.

4.49. ReadInput

Unidad

GRENV

Algoritmo

Se comienza por validar que el menú haya sido creado (si no lo está, simplemente se retorna 0). Luego, si es necesario, se ajusta la opción inicial según el valor de InputCode.

Luego se almacenan las condiciones actuales de texto y ViewPort, las cuales se ajustan a los valores apropiados para dibujar el menú.

A continuación se activa el control automático del selector y, si es necesario, se de-selecciona la posición corriente del selector.

El siguiente paso es mostrar el selector en pantalla, tras lo cual se entra en un bucle de lectura de entrada. Ya que el movimiento del selector se controla durante la misma lectura, sólo resta cambiar (si es

necesario) el estado de las opciones de selección única por medio del procedimiento interno ChangeSelectOnly, siendo el resto de las posibles órdenes manejadas por dos procedimientos internos: *HandleSpecialKeys* y *HandleNormalKeys*; éstos procedimientos son básicamente sentencias CASE que llaman a los correspondientes procedimientos para activar/desactivar los distintos submenús y seleccionar opciones. El control automático del selector se activa sólo si el menu no ha sido activado. El bucle termina cuando alguno de los procedimientos de manejo de teclas determina que se ha seleccionado una opción válida.

Para terminar, el procedimiento coloca el código de la opción seleccionada en la variable *InputCode*, oculta el selector, si es necesario inactiva el menú y se restauran las condiciones previas de texto y ViewPort.

Definiciones

Null: Constante caractér que almacena el valor de un caractér nulo.

MainHotKeys: Variable estática Conjunto de caracteres, que almacena temporalmente las teclas de activación del menú principal, mientras estén activas las de un submenú.

KeyCode: Variable Word que almacena el código de la tecla presionada.

M,N: Variables Byte que indican cual es la opción corrientemente seleccionada (Menú/SubMenú).

Final: Variable Boolean utilizada como señal para el fin del bucle de lectura.

OldStyle: Variable TextSettingsType que almacena el seteo de texto al comienzo del procedimiento.

OldView: Variable ViewPortType que almacena el seteo de ViewPort al comienzo del procedimiento.

4.50. ReadSelector

Unidad

GRSELECT

Algoritmo

Primeramente se lee un byte de teclado; si éste es nulo se lee el siguiente byte colocándolo en la parte superior de la variable correspondiente. A continua-

ción, si el flag de control automático está activado, se verifican las teclas que corresponden al control del selector.

Si el código es una de las teclas de movimiento, se llama al procedimiento *MoveSelector* con el parámetro sorrespondiente.

Si el código es el de la tecla de Selección, entonces si la posición del selector es la misma que la de la marca y la marca es visible se asume que se ha presionado la tecla de Confirmación, de lo contrario se selecciona la posición corriente.

Si el código es el de la tecla de De-Selección, entonces se llama al procedimiiento *DeSelectPosition*.

Si el código es el de la tecla de Confirmación y la marca no es visible, entonces se mueve la coordenada seleccionada a la posición corriente.

Si el código es el de la tecla de Incremento/Decremento y el incremento del selector (variable *StepSelector*) está dentro del rango permitido, entonces se Incrementa/Decrementa la variable *StepSelector* en una unidad.

Finalmente se retorna el código leído.

Definiciones

Null: Constante Byte que almacena el valor de un carac-tér nulo.

InKey: Variable Word que almacena el código leído del teclado y convertido.

Comentarios

La modificación más deseable para esta rutina es la incorporación de otros medios de entrada (por ejemplo, un ratón). Para ello, sería necesario modificar las dos primeras líneas para vericicar de dónde se debe leer la entrada, y aumentar una sentecia CASE que convierta los códigos del nuevo dispositivo a los códigos ya establecidos que se desee.

4.51. ReadStr

Unidad

GRENV

Algoritmo

Primeramente se calculan las coordenadas del área necesaria para la lectura,, la cual se borra y se rodea de un rectángulo. Luego se desconecta el control automático del selector.

A continuación se entra en un bucle de lectura de caracteres, los cuales se filtran de tal manera que sólo se reciben las teclas normales y mayúsculas. Se verifica el caractér: si es uno de los caracteres permitidos en el string y no se ha alcanzado la longitud máxima, el caractér es concatenado al final del string y displayado; si se trata de [BackSpace] y el string no está vacio el último caracter del string es eliminado y borrado de la pantalla; si se trata de [Esc] el string y el área de entrada son blanqueados, finalizando el bucle; el bucle termina normalmente si la tecla presionada es [Enter], tras lo cual el control del selector es restaurado a su valor original y el string ingresado es retornado.

Definiciones

Enter,Esc,BackSpace: Constantes caractér que almacenan los códigos de la teclas [Enter], [Esc] y [BackSpace] respectivamente.

Sep: Constante Byte que almacena los pixels de separación entre el área de entrada y el rectángulo delimitador.

Ch: Variable caractér que contendrá el caractér ingresado.

S: Variable string que contendrá la concatenación de los caracteres ingresados.

NumCh: Variable Byte que se utiliza como contador de los caracteres en S.

A,B: Variables PointType que almacenan las coordenadas gráficas de las esquinas superior izquierda e inferior derecha del área de entrada, respectivamente.

OldControl: Variable Boolean que almacena el estado original de la variable de control automático del selector (*GrSelect.AutoControl*).

4.52. ReplaceAll

Unidad

STRINGS

Algoritmo

Se recorre con un bucle el string original; si el elemento correspondiente es igual al primer caractér, se coloca el segundo caractér en su lugar. Luego se retorna el string resultante.

Definiciones

P: Variable byte utilizada como índice del bucle de conversión.

4.53. Replicate

Unidad

STRINGS

Algoritmo

Si el parámetro *N* es 0 se retorna un string nulo; de lo contrario se llena un string auxiliar con el caractér indicado, se ajusta ls longitud apropiada y se retorna dicho string.

Definiciones

Aux: Variable string utilizada como almacenamiento temporal.

Comentarios

Nótese que es necesario asignar la longitud del nuevo string directamente en el caractér 0 del mismo. Esto es debido a que se modifica directamente (por medio de *FillChar*) las posiciones de memoria asignadas a dicho string, de lo cual se tiene que informar explícitamente a Turbo Pascal.

4.54. RTrim

Unidad

STRINGS

Algoritmo

Se busca desde la última posición del string, hacia atrás, un caractér que no se encuentre en el conjunto de inválidos. Luego se retorna una copia del string original, desde el comienzo del mismo hasta el primer caractér válido encontrado.

Definiciones

P: Variable byte utilizada como índice del bucle de búsqueda.

4.55. SelectPosition

Unidad

GRSELECT

Algoritmo

Primeramente se oculta el selector. Luego se deselecciona la posición corriente (lo que a su vez actualiza las coordenadas del punto selecionado), se displaya la marca y, si es necesario, también el selector.

Definiciones

Visible: Variable Boolean utilizada para conservar el ajuste original de visibilidad del selector (dicho ajuste varía cuando el selector es ocultado).

4.56. Separate

Unidad

STRINGS

Algoritmo

Se realiza una búsqueda del string *Separator* en *Source*. Si no se lo encuentra, se retorna *Source* en *LeftPart* y un string nulo en *RightPart*; de lo contrario se coloca en *LeftPart* una copia de *Source* desde el primer caractér hasta la posición inmediatamente anterior a la primera ocurrencia de *Target*, y en *RightPart* se retorna una copia de *Source* desde la posición inmediatamente posterior a la primera ocurrencia de Target hasta el último caractér.

Definiciones

P: Variable byte utilizada como índice de la primera posición de *Target* en *Source*.

4.57. SetBit

Unidad

FUNCTIONS

Algoritmo

Se forma una máscara rotando a la izquierda el valor $01 hasta que el bit 1 esté en la posición específica. Luego, si se desea encender el bit, se retorna esta máscara OReada con el valor original, de lo contrario se retorna el valor original ANDeado con el negativo de la máscara.

Definiciones

NewValor: Variable byte que contendrá la máscara.

4.58. SetDefaultOption

Unidad

GRENV

Algoritmo

Simplemente se verifica que el menú se haya creado y que los parámetros estén en los rangos correctos. Si las condiciones se dan, entonces se ajustan los campos de la variable *DefOption* a los parámetros proporcionados.

Definiciones

Ninguna.

4.59. SetFont

Unidad

GRLIBR

Algoritmo

Se verifica si el nuevo tipo de letra necesita algún ajuste de tamaño a fin de que ocupe aproximadamente el mismo espacio que el tipo normal; luego se llama a los procedimientos standard para ajuste de tipo de letra.

Definiciones

FontSize: Variable Word que se utiliza para pasar los parámetros de tamaño de letra.

4.60. SetPosition

Unidad

GRSELECT

Algoritmo

Primeramente se toman las coordenadas de trabajo actuales y se verifica que la coordenada solicitada esté dentro de dichos límites. Luego, dependiendo de cúal sea la posición a cambiar, se verifica el flag de control correspondiente, se oculta el símbolo si es necesario, se cambia el valor de la coordenada y se vuelve a displayar el símbolo en su nueva posición si es necesario.

Definiciones

View: Variable ViewPortType que almacena el seteo de ViewPort utilizado para validar la coordenada.

Visible: Variable Boolean utilizada para conservar el ajuste original de visibilidad del símbolo correspondiente (dicho ajuste varía cuando el símbolo es ocultado).

4.61. SetSelectionRange

Unidad

GRSELECT

Algoritmo

Primeramente se valida que el nuevo rango se encuentre entre 0 y 126; esto es debido a que el máximo tamaño en pixels previsto para la marca es 255 x 255, y el rango es por definición la mitad del área de selección.

El siguiente paso es ocultar la marca si es visible. Se calculan y almacenan los nuevos parámetros. Luego se verifica que no haya un fondo grabado (si es necesario, el área ocupada se libera) y se vuelve a calcular el tamaño necesario para las subsecuentes grabaciones. Finalmente, si es necesario, se dibuja de nuevo la marca.

Definiciones

MarkVisible: Variable Boolean utilizada para conservar el ajuste original de visibilidad de la marca (dicho ajuste varía cuando la marca es ocultada).

4.62. ShowMark

Unidad

GRSELECT

Algoritmo

Primero se calcula las coordenadas del área que se va a alterar con la marca; con esta información se reserva la memoria necesaria para almacenar el fondo y se graba el mismo.

A continuación se ajusta el tipo de línea a punteado y se dibuja la marca en las coordenadas seleccionadas (grabadas en la variable *Selected*); el tipo de línea corriente se reajusta al previamente establecido, se activa el flag de control correspondiente y se finaliza.

Definiciones

Pt: Variable PointType que almacena las coordenadas de la esquina superior izquierda del área ocupada por la marca.

OldLine: Variable LineSettingsType que almacena el seteo de línea al comienzo del procedimiento.

Comentarios

Nótese que no se ha considerado necesario pregrabar la imagen de la marca en una variable, como se lo ha hecho con el selector, debido a que es una figura simple y se la puede recrear con una simple instrucción (procedimiento standard *Circle*).

4.63. ShowSelector

Unidad

GRSELECT

Algoritmo

Primero se reserva la memoria necesaria para almacenar el fondo del selector y se graba el mismo.

Luego simplemente se coloca la imagen del selector, creada durante la inicialización, en las coordenadas corrientes (grabadas en la variable *Current*), se activa el flag de control correspondiente y se finaliza.

Definiciones

Ninguna.

4.64. Spaces

Unidad

STRINGS

Algoritmo

Si el parámetro *N* es 0 se retorna un string nulo; de lo contrario se llena un string auxiliar con espacios, se ajusta ls longitud apropiada y se retorna dicho string.

Definiciones

Aux: Variable string utilizada como almacenamiento temporal.

Comentarios

Nótese que es necesario asignar la longitud del nuevo string directamente en el caractér 0 del mismo. Esto es debido a que se modifica directamente (por medio de *FillChar*) las posiciones de memoria asignadas a dicho string, de lo cual se tiene que informar explicitamente a Turbo Pascal.

4.65. StrToVal

Unidad

STRINGS

Algoritmo

Simplemente se llama al procedimiento standard *Val* utilizando el string proporcionado. Si éste procedimiento no indica error se retorna el número correspondiente, de lo contrario se retorna 0.

Definiciones

N: Variable Real que contiene el resultado de la conversión efectuada por *Val*.

Result: Variable Integer que contiene el código de error de *Val*.

4.66. SwapBytes

Unidad

FUNCTIONS

Algoritmo

Se utiliza una variable temporal para almacenar los datos de la primera variable, mientras los datos de la segunda variable se mueven a la primera. Finalmente, los datos de la primera variable se mueven del almacenamiento temporal a la segunda variable.

Definiciones

Tmp: Variable arreglo de 255 bytes usada como almacenamiento intermedio.

Comentarios

Si las variables a intercambiar son de menos de 255 bytes, este procedimientop es más rápido que su equivalente *SwapVars*, ya que la variable Tmp es dimensionada automáticamente por Turbo Pascal en el stack.

4.67. SwapVars

Unidad

FUNCTIONS

Algoritmo

Se utiliza una variable temporal para almacenar los datos de la primera variable, mientras los datos de la segunda variable se mueven a la primera. Finalmente, los datos de la primera variable se mueven del almacenamiento temporal a la segunda variable.

Definiciones

Tmp: Apuntador a un bloque de memoria usado como almacenamiento intermedio.

Comentarios

Aun siendo ligeramente más lento que *SwapBytes*, *SwapVars* tiene la ventaja de ser más general y de utilizar (dinámicamente) sólo la memoria necesaria.

4.68. ToggleOption

Unidad

GRENV

Algoritmo

Simplemente se verifica que el menú se haya creado y que los parámetros estén en los rangos correctos. Si las condiciones se dan, entonces se cambia el flag de habilitación (*Enabled*) al estado contrario al que se encuentra corrientemente.

Definiciones

Ninguna.

4.69. ToggleWhenSelect

Unidad

GRENV

Algoritmo

Simplemente se verifica que el menú se haya creado y que los parámetros estén en los rangos correctos. Si las condiciones se dan, entonces se cambia el flag de selección única (*SelectOnly*) al estado contrario al que se encuentra corrientemente, y el flag de habilitación (*Enabled*) al estado contrario al de selección única; es decir, si se declara a la opción como activa por selección, se ajusta el flag de habilitación a FALSE y viceversa.

Definiciones

Ninguna.

4.70. Trim

Unidad

STRINGS

Algoritmo

Primero se busca desde la primera posición del string, hacia adelante, un caractér que no se encuentre en el conjunto de inválidos. Luego se busca desde la última posición del string, hacia atrás, un caractér que no se encuentre en el conjunto de inválidos. Finalmente se retorna una copia del string original, desde el primer caractér válido hasta el último encontrado.

Definiciones

P1,*P2*: Variables byte utilizadas como índice del primer y segundo bucles de búsqueda, respectivamente.

L: Variable byte utilizada para almacenar la longitud del string original.

4.71. UpperCase

Unidad

STRINGS

Algoritmo

Se recorre con un bucle el string original y se pasa cada uno de sus elementos por la función *UpCase*. Luego se retorna el string resultante.

Definiciones

P: Variable byte utilizada como índice del bucle de conversión.

4.72. ValToStr

Unidad

STRINGS

Algoritmo

Simplemente se llama al procedimiento standard *Str* utilizando los parámetros proporcionados. El string resultante se retorna.

Definiciones

S: Variable string utilizada como almacenamiento temporal del resultado de *Str*.

4.73. WaitForOk

Unidad

GRBUTTON

Algoritmo

Primeramente se calculan las coordenadas gráficas del botón, convirtiendo las coordenadas texto proporcionadas y adicionando las coordenadas mínimas corrientes.

Luego se dibuja el borde del botón por medio del procedimiento interno *DrawButton*. Dado que sólo hay un botón activo se le dibuja un borde doble, como corresponde a un botón principal.

A continuación se calculan las coordenadas del centro del botón y se dibuja la etiqueta ' Ok ' por medio del procedimiento interno *WriteButtonLabel*.

Luego se desconecta el control del selector y se procede a leer la entrada hasta que se presiona una de las siguientes teclas: Confirmar (Enter), barra espaciadora, letra O (mayúscula o minúscula), o Seleccionar (Ins).

Se notará en el listado que todas las teclas son probadas por medio del operador IN con la excepción de Ins. Esto es debido a que el operador IN está diseñado para trabajar con conjuntos, los cuales sólo tienen un rango de 1 byte (0 - 254); el código de selector retornado para todas las teclas que se prueban con IN caen dentro del rango de un byte, con la excepción del correspondiente a *SelectKey*, el cual tiene que probarse por separado con el operador =.

Una vez que el usuario ha presionado una de las teclas se conecta el control del selector y se hace blinkear el área que ocupa el botón, como confirmación visual.

Definiciones

OKey, *UpOKey*: Constantes Word que almacenan los códigos de la tecla O miunúscula y mayúscula, respectivamente.

P1,*P2*: Variables PointType que almacenan las coordenadas gráficas de las esquinas superior izquierda e inferior derecha del botón, respectivamente.

Center: Variable PointType que almacena la coordenada gráfica del centro del botón (necesaria para dibujar la etiqueta centrada).

Key: Variable Word que almacenará el código de la tecla leída.

Comentarios

Nótese que en realidad lo que se crea es la imagen de un botón, sin la estructura de datos que se utiliza para almacenar un botón. Es por esto que el procedimiento no altera otros botones activo al momento de ejecutarse, pero también es por esto que no se restaura el fondo previo a la creación de la imagen.

5. Ejemplo de una aplicación: Hiparco

Hiparco es una sencilla aplicación que hace buen uso de la mayoría de las rutinas y filosofías expuestas. El propósito de Hiparco es presentar información sobre estrellas, constelaciones y datos astronómicos; la información se obtiene de simples archivos textos y mostrada en la pnatalla en forma de un mapa de estrellas.

Ya que las necesidades de la interface de usuario y de interacción con la consola son cubiertas por las rutinas presentadas, la aplicación puede concentrarse en su tarea primaria de organizar la información obtenida y presentarla en un formato útil. Sin dejar de ser tareas complejas e interrelacionadas, éstas pueden organizarse en cuatro grupos lógicos:

- Catálogo de estrellas: forma una base de datos donde se pueden consultar los datos de las estrellas.

- Graficador del mapa: toma una serie de puntos tridimensionales y los proyecta en forma bidimensional sobre la superficie de la pantalla.

- Diccionario: forma una base de datos de los términos técnicos y astronómicos que el usuario debe conocer para interpretar la información.

- Módulo principal: sirve como enlace de los otros tres módulos y de las librerías de rutinas.

5.1. Catálogo de estrellas

La base de datos o catalogo de estrellas se implementa del siguiente modo: las constelaciones o grupos se almacenan en un arreglo dinámico de *TotalGroups* elementos, apuntado por *Sky*. Los elementos del arreglo estan conformados por dos clases de datos: los datos del grupo propiamente hablando (variable *Info*) y los datos que permiten manejar los miembros del grupo (estrellas), descritos a continuación.

A cada grupo le corresponde un arreglo dinámico de estrellas, de *TotalMembers* elementos, apuntado por *Member*. Cada elemento de este arreglo contiene la información característica de una estrella (nombre, masa, etc) más la identificación del grupo o constelación a que pertenece y su rango dentro de dicho grupo (los rangos estelares se determinan por la brillantez de la estrella dentro del grupo; la estrella mas brillante recibe el rango "alfa", la siguiente "beta", y asi sucesivamente).

Dado el tipo de estrucura, los datos del catalogo se accesan basicamente por medio del número de agrupación y número de

rango (Procedure *SeekStarInfo*), lo cual permite que la estructura sea accesada rápida y eficientemente, con la excepción del caso en que se desee accesar directamente a una estrella por medio de su nombre (el cual no suele tener relación con la agrupación y/o rango). Para este caso no es eficiente la búsqueda secuencial por toda la estructura, que puede tener varios miles de estrellas, por lo que se ha implementado un árbol binario de busqueda, cuya raiz es apuntada por *SearchTree*, en donde cada nodo contiene un apuntador a la estrella correspondiente en el catalogo principal; este árbol se lo crea al inicializar la estructura principal, es liberado al finalizar la aplicación, y se lo accesa únicamente en el procedimiento *FindStarInfo*.

Se ha provisto además de un tercer procedimiento de consulta, *GetStarInfo*, en el cual se hace un acceso por nombre de rango y nombre de grupo. Dado que en este acceso la búsqueda es secuencial, es el más lento de todos y sólo se lo ha previsto como de uso ocasional.

También se han provisto algunas funciones para inspección del tamaño del catálogo y para conversión de los códigos de identificación a formato numérico.

La inicialización de las estructuras es enteramente realizada por el procedimiento Initialize. El archivo de datos de entrada debe ser un texto ASCII normal con el siguiente formato:

```
<numero-constelaciones>
<nombre-constelacion-1>
   <nombre-latino>
   <dibujo>
   <numero-miembros>
   <nombre-estrella-1>
      <distancia-al-sol>
      <declinacion>
      <angulo-horario>
      <signo-magnitud>
      <valor-magnitud>
      <masa>
      <clase-espectral>
      <numero-de-rango>
   <nombre-estrella-2>
      <distancia-al-sol>
      ...
   <nombre-estrella-n>
      <distancia-al-sol>
      <declinacion>
      ...
      <numero-de-rango>
<nombre-constelacion-2>
   <nombre-latino>
   <dibujo>
   <numero-miembros>
   <nombre-estrella-1>
         ...
   <nombre-estrella-n>
      <distancia-al-sol>
      <declinacion>
      ...
      <numero-de-rango>
<EOF>
```

Los datos deberán estar cada uno en una linea (separados por CR/LF). Espacios en blanco o tabuladores al comienzo y/o fin de las lineas son ignorados.

Los valores de <declinacion> (norte-sur) y <angulo-horario> (este-oeste) deberán estar en grados decimales.

El nombre del archivo texto se toma de la variable estatica *StarFile*. El archivo debe estar accesible al momento de la inicialización.

5.2. Graficador del Mapa

El mapa consiste básicamente en la proyección de un conjunto de puntos en el espacio (de tres coordenadas) sobre un plano, en este caso la pantalla. La estructura que se ha escogido para la implementación interna del mapa es una lista

enlazada simple, con apuntadores al inicio y final de la misma.

Cada nodo de la lista contiene un string identificador del punto, junto con sus coordenadas bidimensionales y tridimensionales equivalentes. Se ha escogido almacenar constantemente ambas coordenadas debido a la flexibilidad que se obtiene al poder separar, el momento de cálculo de las coordenadas del momento de dibujo del mapa. Si no hay cambios frecuentes en los parámetros de proyección, esto reduce considerablemente el tiempo necesario para regenerar el mapa, colocando la decisión de cuándo recalcular bajo responsabilidad de los módulos principales (y que a su vez pueden otorgar esta decisión al usuario). Por supuesto, si alguno de los parámetros que controlan la proyección del mapa es alterado, el módulo lo recalcula automáticamente antes de la siguiente regeneración.

Se ha implementado una capacidad limitada de consulta del mapa. Es posible averiguar las coordenadas de un punto dado su string de identificación, y viceversa; además es posible realizar una búsqueda del punto que se encuentre más cerca de una coordenada proporcionada (función *GetNextPointName*). Estos procesos actualmente son estrictamente secuenciales, ya que no se ha querido incrementar la complejidad del módulo adicionándole funciones de búsqueda que no caen dentro de su propósito.

La inicialización del mapa debe hacerse por medio de sucesivas llamadas al procedimiento *RegisterPoint*, proporcionando los datos de un punto por cada llamada (identificación y coordenadas esféricas). Dado que cada punto es recalculado individualmente antes de ser adicionado, es posible dibujar el mapa inmediatamente después de la adición.

Los únicos cambios que pueden forzar una recalculación del mapa son la alteración de uno o varios de los siguientes parámetros:

- Punto de observación de la proyección (ajustado con el procedimiento *SetObserverPos*).

- Ventana de proyección (ajustado con el procedimiento *SetViewWindowPos*).

- Angulo de alcance de vista (ajustado con el procedimiento *ResetDefaults*).

En el caso de que se ejecute alguno de los procedimientos mencionados, el módulo principal tiene las opciones de regenerar el mapa por medio de *PlotMap* (que hará los cálculos automáticamente) o de recalcular únicamente sin regenerar la imagen del mapa, por medio de *CalcMap*.

Como toques adicionales, el módulo puede modificar el área de visualización del mapa de tal manera que un punto específico quede a la vista (procedimiento *PutInCenter*), y tiene la capacidad de unir puntos por medio de líneas de varios tipos (procedimiento *JoinPoints*).

5.3. Diccionario

El diccionario se lo considera como un conjunto de entradas (términos), cada una de las cuales tiene asociada una descripción de hasta 16 líneas. Todos estos datos provienen de un archivo texto.

Dado que el diccionario es sólo-lectura y de consulta ocasional, se ha decidido mantener los textos explicativos en disco y tener en memoria un índice de la posición en el archivo de cada entrada; dicho índice se implementa por medio de un arreglo dinámico, el cual luego de ser inicializado es sorteado por el método llamado "QuickSort". Dadas las peculiares condiciones, éste método proporciona casi las mismas características de un índice de árbol binario, con las definitivas ventajas de mayor velocidad y facilidad de implementación.

Cada entrada del índice, además del nombre de la entrada, contiene el número de la línea del archivo de diccionario donde empieza su descripción. Ya que no se ha previsto que el archivo sea muy extenso, la localización del texto deseado no suele tomar mucho tiempo.

Sólo se ha implementado una rutina de consulta (*Browse*), la cual presenta una ventana en la que se presentan secuencialmente los términos, en orden alfabético, permitiendo que el usuario seleccione uno para que le sea presentada su información.

La inicialización del índice es enteramente realizada por el procedimiento Initialize. El archivo de datos de entrada debe ser un texto ASCII normal con el siguiente formato:

```
<numero-entradas>
<nombre-entrada>
<linea-descripcion-1>
        ...
<linea-descripcion-n>
<linea-en-blanco>
<nombre-entrada>
<linea-descripcion-1>
        ...
<linea-descripcion-n>
<linea-en-blanco>
        ...
<EOF>
```

El numero maximo de lineas de descripcion es 16; todas las lineas estan limitadas a 80 caracteres. Las descripciones deberan estar separadas por una linea en blanco (por dos secuencias CR/LF). Espacios en blanco o tabuladores al comienzo y/o fin de las lineas son ignorados

El nombre del archivo texto se toma de la variable estatica *DictFile*. El archivo debe estar accesible al momento de la inicialización.

5.4. Módulo principal

Este módulo constituye el nucleo de toda la aplicación; contiene todos los procedimientos que toman datos del catálogo y se los proporcionan al graficador, presentan los menús y selector para que el usuario pueda seleccionar los datos que desea, y muestran ventanas con los datos requeridos.

El cuerpo de Hiparco consiste únicamente de las siguientes acciones:

- Creación del sistema de menú, e inicialización de los diversos módulos (aquellos que no se inicializan automáticamente).

- Proceso de órdenes del usuario hasta que se seleccione la opción de "Fin".

Las órdenes que procesa el sistema, junto con su respectivo procedimiento, son en síntesis:

Mostrar información del programa (*ShowProgramInfo*): Muestra una pequeña pantalla con datos de identificación del programa.

Mostrar información de un objeto (*ShowObjectInfo*) y Mostrar información de un grupo (*ShowGroupInfo*): Presentan un "formulario" con la información de un objeto/grupo en una determinada posición, solicitada por medio del mapa/catálogo en base a la posición del selector. Ambos procedimientos pueden llamarse entre sí si es necesario.

Buscar información (*SearchInfo*): Realiza una consulta por nombre al catálogo, en base a una clave ingresada por el usuario.

Mostrar catálogo (*ShowCatalog*): Presenta una lista de todo el contenido del catálogo (consulta secuencial), dando al usuario oportunidad de seleccionar, directamente y en contexto, la información que desea.

Consultar diccionario (*Dictionary*): Simplemente prepara una ventana apropiada y llama al procedimiento *Browse* del diccionario.

Ver un grupo (*ViewGroup*): Dibuja líneas delimitando un grupo de estrellas; para esto solicita del catálogo un código que le indica, por medio de pares de letras, cuáles estrellas deben ser conectadas; el procedimiento analiza esta información y se la pasa al procedimiento *JoinPoints* del mapa.

Ver todos los grupos (*ViewAllGroups*): Ejecuta el procedimiento *ViewGroup* para todas los grupos del catálogo.

Ver el mapa normal (*PlotMap*): Simplemente llama al procedimiento para redibujar el mapa.

Cambiar la posición (*ChangePosition*): Posiblemente el procedimiento más complejo del módulo principal, permite al usuario cambiar semi-interactivamente la posición de la ventana de observación sobre el mapa. El usuario hace esto presionando las teclas de dirección para variar los valores de desplazamiento X y Y de dicha ventana.