UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

TURBO PASCAL

COORDINACION DE ADMINISTRACION Y DE SISTEMAS
ING. F. EUGENIO LOPEZ GUERRERO

TURBO PASCAL

PRESENTACION

El proceso evolutivo de la sociedad ha alcanzado nuestros ámbitos universitarios, de tal forma que nuestra Facultad ha visto en los últimos tiempos el proceso de rediseño curricular.

Al respecto, el Ing. Guadalupe Cedillo Garza, director actual, ha mencionado: "este proceso es de gran importancia para la FACULTAD DE INGENIERIA MECANICA Y ELECTRICA; es también importante que maestros y alumnos participen en éste proceso de actualización, ahora ya de carácter permanente. Su justificación principal radica en el compromiso universitario de formar excelentes egresados ávidos de asimilar las innovaciones científicas y tecnológicas, dispuestos a aportar su esfuerzo en una forma responsable al sector productivo y a la sociedad en general".

Conciente de las necesidades que involucra este gran cambio, la Coordinación de Administración y de Sistemas modificó sus programas de clase, entre ellos el de la clase de Análisis Numérico e introdujo el lenguaje de programación Pascal para microcomputadoras.

Como un esfuerzo de apoyo a dichas modificaciones, la coordinación y un servidor presentamos el siguiente texto a la comunidad universitaria el cual fue procesado en tipografía LASER en el Centro de Diseño Y Mantenimiento de Instrumentos de la Facultad de Ingeniería Mecánica Y Eléctrica.

Ing. F. Eugenio López Guerrero

Diciembre 1989

CONTENIDO **PAGINA**

INTRODUCCION

Pascal fue desarrollado en 1971 por Niklaus Wirth, en respuesta a la creciente necesidad de un lenguaje de programación modular, sistemático y fuertemente implementado en tipos. Wirth, quien había desarrollado anteriormente los lenguajes ALGOL y PL/1, entendía la importancia de proveer un lenguaje que diese una forma de programación estructurada, ayudase al desarrollo de sistemas y facilitara el aprendizaje de tópicos y conceptos avanzados de programación, tales como apuntadores, registros y variables dinámicas.

En 1971, Niklaus Wirth escribió la definición del lenguaje de Pascal, en su bien conocido (e imposible de leer) libro titulado "Pascal: User Manual and Report". El libro contiene una descripción meticulosamente detallada de lo que el lenguaje debe ser. Sin embargo, no existe actualmente un compilador que tome un programa fuente y produzca código ejecutable a partir de él. Algunos (muchos algunos, actualmente), basándose en el libro de Wirth, elaboraron compiladores que han dejado muy atrás su definición de lenguaje. Tales programas aparecieron primero para computadoras de tipo "mainframe", como la CDC6600, y hace algunos años para microcomputadoras.

Esos compiladores, que funcionalmente siguen el cuerpo de la definición de lenguaje de Pascal, se llaman "implementaciones" del lenguaje Pascal.

La definición formal de Pascal, como la hizo Wirth, tiene algunas "lagunas" y partes débiles, con un uso bastante limitado. La definición de Wirth no maneja strings, archivos random, llamadas al sistema operativo, ni muchas otras cosas más. La gente que implementa el lenguaje en una computadora dada, generalmente va mas allá de la definición del lenguaje, haciendo al compilador capaz de compilar programas más útiles.

Además, algunas implementaciones imponen límites al programador, que la definición de Wirth no tiene. Para poder hacer uso de sistemas de memoria pequeña, los límites del mundo real deben forzarse al tamaño de los identificadores, de las funciones y procedimientos, de los números enteros y reales máximos, etc. Ninguno de ellos está presente en la definición del lenguaje, pero son críticos en alguna implementación en particular.

Pascal, por diseño, es un lenguaje estructurado. A diferencia de BASIC y FORTRAN, Pascal impone una estructura en sus programas. No permite estatutos colocados al azar, aun si se toma cada uno y éstos son sintácticamente correctos. Hay un plan maestro detallado que todo programa de Pascal debe seguir. Un programa debe ser codificado en ciertas partes. Algunas deben estar ahí y otras aquí. Todo debe estar en cierto orden. Algunas cosas no pueden trabajar juntas y otras sí. Haciendo a un lado ciertas concesiones que los diseñadores de los compiladores han hecho (Pascal hace esas tareas fácilmente), la estructura de Pascal existe para reforzar cierta forma de pensar acerca de la programación. Esta forma de pensar, desarrollada por Wirth y otros, se denomina "programación estructurada". Representa el énfasis de Niklaus Wirth sobre la creación de programas que sean comprensibles, sin necesidad de páginas de diagramas de flujo y cientos de líneas de explicación. Aunque la programación estructurada puede estar acompañada de casi cualquier lenguaje (inclusive BASIC), Pascal es uno de los pocos que la requiere.

Desde entonces, Pascal ha sido aceptado como estándar para enseñar estructuras de datos y conceptos de programación en muchas universidades. Los graduados se han encargado de llevar el Pascal al resto del mundo. Gracias al TURBO Pascal de la casa Borland, Pascal es el lenguaje de programación para microcomputadoras más ampliamente usado. De hecho, el ambiente de TURBO Pascal es tan popular, que muchos desarrolladores de sistemas no han tenido elección para emigrar al ambiente de programación interactivo que éste provee. En adición, TURBO Pascal ha tenido un impacto significativo en el ISO Pascal estándar. Los compiladores de Pascal de todos los niveles -desde computadoras mainframe hasta micros- tienen mayor inclinación para el Pascal, ya que éste cuenta con un desarrollo de programas interactivo y una poderosa colección de rutinas de librerías. En los años que vienen, los negocios y las organizaciones verán las ventajas de un programador adiestrado en Pascal -especialmente TURBO Pascal- como herramienta para el desarrollo.

CAPITULO 1.- CONCEPTOS BASICOS

INTRODUCCION INFORMAL A PASCAL

Una estructura está hecha de algo. La estructura de un cristal consta de átomos arreglados en un orden particular. Un programa en Pascal está hecho de "átomos", los cuales son simples palabras en inglés. Esos átomos del programa caen dentro de tres categorías:

- Un pequeño núcleo de palabras (43 en TURBO Pascal) llamadas "palabras reservadas".

- Un número grande de palabras llamadas "identificadores estándar".

- Un número ilimitado de identificadores ordinarios, creados por el programador.

Las palabras reservadas tienen un significado especial dentro de Pascal. No pueden ser usadas por el programador,a no ser dentro de los límites especificados para su uso. El compilador encontrará error en el uso de una palabra reservada que no se ajuste al uso para el que fue destinada. Ejemplos: BEGIN, END, PROCEDURE, ARRAY, y ese viejo diablo: GOTO.

Todo lo que no sea una palabra reservada, es un "identificador". Algunos identificadores tienen significados predeterminados para el compilador. Esos "identificadores estándar", como las palabras reservadas, tienen un significado particular para el compilador. Sin embargo, bajo ciertas circunstancias, se pueden redefinir los significados para propósitos bastante particulares.

Cualquier nombre aplicado a una entidad en un programa, es un identificador ordinario. Los nombres de las variables, los procedimientos, las funciones y el programa en sí mismo, son identificadores. Un indentificador creado por el programador puede servir (dentro de los límites y las reglas de Pascal) para un propósito único. Esto es, si se tiene una variable llamada CONTADOR, no se puede tener un procedimiento o función llamado CONTADOR, ni tampoco otra variable llamada CONTADOR. La siguiente es una revisión a los elementos de un programa de Pascal. El programa en sí mismo debe ser construido con esos elementos, de una muy particular manera. La razón de ello es proporcionar al programador una visión sencilla de la tarea del programa, en forma clara y estructurada.

ESTRUCTURA DEL PROGRAMA

EL ESTATUTO PROGRAM.

El estatuto PROGRAM le da al programa un nombre y le dice al compilador en dónde empieza el programa. El nombre del programa no es necesariamente igual al nombre del archivo que lo contiene, ni tampoco igual al nombre del archivo que contiene el código ejecutable que el compilador hace como producto final. Este es un estatuto program:

```
PROGRAM FIME;
```

EL ESTATUTO USES

Una vez que se desarrolló y compiló una unidad, es necesario incorporar las rutinas que contienen la unidad dentro del programa. El estatuto USES es el medio para hacerlo. Debe declararse al inicio del programa, después de la declaración del identificador del programa. La sintaxis es:

```
USES Unidad1, Unidad2, Unidad3...;
```

Los nombres de la lista de unidades deben ser los de unidades ya creadas y que se encuentran en disco, por ejemplo UNIDAD1.TPU, UNIDAD2.TPU, etc. Cuando el compilador encuentra el estatuto USES, busca en el disco la unidad e incorpora las rutinas públicas en el programa.encadenar las rutinas con el programa a compilar: Unicamente las rutinas que usa el programa son las que se convierten en código en el producto final. Por ejemplo, si una unidad tiene diez rutinas públicas, pero el programa que la utiliza sólo manda llamar a tres, Turbo Pascal encadena estas tres solamente.

Si se compila un programa que involucra varios archivos de código, Turbo Pascal es muy eficiente al

El siguiente programa es un ejemplo de la declaración USES. Se asume la unidad "PRUEBA", compilada y lista para usarse en disco, y que incluye la rutina "IMPRIME":

```
program PruebaUnidad;
uses
      Crt, Prueba;                    { incluye las unidades CRT y PRUEBA }
begin
      clrscr;                         { incluye la función de CRT }
      writeln;
      Imprime ('Hola !');             { utiliza la función de PRUEBA }
end.
```

LA SECCION DE DECLARACION DE ETIQUETAS (LABEL)

La parte siguiente es la sección de declaración de etiquetas. Consiste en la palabra reservada LABEL, seguida de una lista de etiquetas. Las etiquetas son marcas en un programa, al cual debe ir el estatuto GOTO. Deben contener números no- negativos, no importando su orden ni su rango, pero cada una debe ser única. Este es un ejemplo de una declaración de etiquetas:

```
LABEL
        50,100,150,200;
```

Si no se tienen GOTO's en el programa (y ésta es una muy buena idea), no necesitará etiquetas. No tener etiquetas significa que la parte de la declaración de etiquetas es innecesaria y puede omitirse. Pero si se tiene por lo menos una, la declaración debe ser el primer estatuto del programa.

LA SECCION DE DECLARACION DE CONSTANTES (CONST)

Si se intenta usar constantes en un programa, el compilador las localiza en la sección de la declaración de constantes. Los valores de las constantes son definidos al tiempo de compilación y nunca son cambiados en el proceso de ejecución del programa. La declaración de las constantes consiste en la palabra reservada CONST, seguida de la lista de constantes. La declaración de una constante consiste en un identificador y su valor, separados por un signo de igual (=).

```
CONST
        Limite      = 255;
        GNombre     = 'Gráfica 3-D';
        Plotter     = true;
```

En el ejemplo anterior, LIMITE es una constante entera; GNombre es una constante string y Plotter una constante booleana. El Pascal estándar y el TURBO Pascal difieren significativamente por la forma en que puede ser una constante. Esto se verá posteriormente.

LA SECCION DE DECLARACION DE TIPOS (TYPE)

La declaración de tipos consiste en la palabra reservada TYPE, seguida por la lista de tipos. Una declaración de tipos es un tipo de identificador y su declaración, separado por un signo igual (=). Pascal predeclara los tipos fundamentales: entero (integer), booleano (boolean), caracter (char), byte (byte) y real (real). Estos no necesitan ser declarados. Solo se declaran aquellos que son creados por el usuario, o los que son subrangos de los tipos fundamentales ya definidos.

Se pueden definir nuevos tipos a partir de tipos ya definidos por el usuario; por lo tanto, no se puede usar un tipo como ingrediente de un nuevo tipo, a menos que el compilador conozca de qué tipo se trata. El compilador conoce un tipo cuando éste ya ha sido predeclarado (integer, byte, etc), o porque ya ha sido compilado el estatuto definiendo el tipo en términos de otros tipos ya definidos con anterioridad.

```
TYPE
        Grupo       = 'A'..'G';                 {subrango}
        Llamada     = string[8];
        USHam       = record
                        LlamadaOp  :Llamada;
                        Clase      :char;
                      end;
        Conta       = record
                        SuLlamada  :Llamada;
                        RST        :integer;
                        Banda      :integer;
                        Frecuencia :real;
                        Emision    :string[2];
                        Tiempo     :integer;
                      end;
        Inicio      = record
                        NuestraOp  :USHam;
                        Equipo     :Grupo;
                        QSO        :Conta;
                      end;
```

La anterior es una serie de declaraciones que podrían aparecer en un programa que manejase un radio de CB. Si los detalles de la declaración son extraños para ud., no se preocupe, serán cubiertos en forma minuciosa posteriormente. Lo importante es notar que todos los tipos están definidos en función de los tipos fundamentales o de subrangos (a través del signo =), antes de que sean usados. Por ejemplo, todos los tipos que forman Inicio son definidos por el usuario. Cada uno debe tener una definición anterior a Inicio, y de hecho cada uno la tiene. Si Grupo fuese definido después de Inicio, el compilador marcaría error durante la compilación del programa. No sabría qué almacenar al encontrar el primer Grupo. Grupo es un subrango de tipo char (que incluye las letras de la A a la G), y una vez definido en esos términos, el compilador ya sabe a qué se refiere. Así como la parte de etiquetas, la declaración es opcional; si no hay tipos qué definir, puede ser omitida.

Ordinariamente, la declaración de constantes es anterior a la de tipos, pero TURBO Pascal permite declarar tipos antes que constantes, si así se desea. Esto se hace en conjunto con las constantes de arreglos (arrays) y las de registro (record).

LA SECCION DE DECLARACION DE VARIABLES (VAR)

Estrictamente hablando, un programa no necesita una sección de declaración de variables para ser compilado o ejecutado. Sin embargo, a pesar de que un programa puede carecer de las secciones de declaración de tipos y de constantes, no se podría hacer gran cosa sin algunas variables para trabajar con ellas.

La declaración de variables consiste en la palabra reservada VAR, seguida por la lista de variables. Cada variable usada debe tener un tipo válido, ya sea predefinido (integer, real, boolean, etc) o definido por el usuario en la parte de declaración de tipos del programa. A diferencia de la declaración de tipos, la declaración de variables utiliza el símbolo de dos puntos (:) para separar la variable de su tipo:

```
VAR
    Contador    :integer;
    Destino     :char;
    Zona        :puntoXY;
```

Si algunas variables tienen el mismo tipo, pueden ser agrupadas en una sola línea, separándolas por comas:

```
VAR
    I,J,K : integer;
```

PROCEDIMIENTOS (PROCEDURE) Y FUNCIONES (FUNCTION)

Después de que todas las variables han sido declaradas, se deben definir todos los procedimientos y funciones que el programa va a necesitar. Los procedimientos y las funciones son las subrutinas en Pascal. Tienen la forma de un programa en miniatura e idéntica estructura que el programa principal, excepto que principian con la palabra reservada PROCEDURE (o FUNCTION, según sea el caso) en lugar de PROGRAM. No se necesita un punto, sino punto y coma para terminarlos. Las funciones, adicionalmente, regresan un valor.

El resto de la estructura es la misma. Procedimientos y funciones pueden tener sus declaraciones "privadas" de etiquetas, constantes, tipos definidos por el usuario, variables; e inclusive pueden tener sus propios procedimientos y funciones. Este anidamiento podría seguir hasta el infinito, pero aquí es donde están las restricciones de la implementación sobre la que se esté trabajando en particular.

El compilador de Pascal fuerza las llamadas a los procedimientos y las funciones en forma jerárquica. Un procedimiento o función puede llamar a cualquier procedimiento o función que se haya declarado anteriormente.

EL PROGRAMA PRINCIPAL

El programa principal, que se encuentra al final del archivo fuente, es el "jefe" que llama a cualquier procedimiento. Como se dijo anteriormente, el cuerpo principal del programa es tratado por el compilador en forma similar a los procedimientos y las funciones, excepto que siempre finaliza con un punto.

Cuando el programa es ejecutado, "entra" al programa principal en el BEGIN y no desde el principio del archivo, como podría pensarse. Cuando encuentra el END del programa principal, abandona éste para regresar al sistema operativo. Mientras lo hace, sigue el flujo de control del programa, llamando a todos los procedimientos y funciones que se indiquen en él.

IDENTIFICADORES

La computadora crea programas para su uso. Los programas son un conjunto de datos y pasos para almacenar, modificar o desplegar datos. La computadora conoce cada parte de los datos, porque ellos tienen una dirección en la memoria. Los nombres propios que se le asignan a los datos, procedimientos y programas, son para beneficio del programador. A estos nombres se les denomina "identificadores", y existen solamente en el código fuente. Los identificadores no existen en el código final.

Los identificadores son secuencias de caracteres de cualquier longitud, hasta 127, y deben seguir estas reglas:

1. Los caracteres legales son las letras, dígitos y el símbolo de subrayado (_). Espacios y símbolos como &, !, *, % no se permiten.

2. Símbolos o dígitos (0 al 9) no pueden ser usados como primer caracter. Todos los identificadores deben empezar con una letra (mayúscula o minúscula) o con el subrayado.

3. Los identificadores no deben ser iguales a ninguna palabra reservada.

4. Las diferencias entre mayúsculas y minúsculas son ignoradas. Para el compilador, "a" es igual a "A".

5. Los subrayados son legales y significantes. Nótese que difieren de algunos otros compiladores de Pascal, en donde los subrayados son legales pero se ignoran. Los subrayados se utilizan para hacer más legible el identificador. Por ejemplo, CODIGOPOSTAL y CODIGO_POSTAL. Otro método es: CódigoPostal.

6. Todos los caracteres son significantes, no importa qué tan grande sea el identificador, siempre y cuando no exceda los 127 caracteres. Muchos otros compiladores de Pascal ignoran todos los caracteres después del octavo.

```
{
  Problema con peticion de datos sencillo. Dando la velocidad inicial
  y el tiempo, calcular la distancia de caida libre de un cuerpo con
  la formula:
                    y = Vot - 1/2 gt**2
}

program VelocidadCaidaLibre;
uses
   Crt;
const
   g = 9.81;
var
    y, Vo, t : real;
begin
   clrscr;
   writeln ('** CAIDA LIBRE DE UN CUERPO **');
   writeln;
   write ('Velocidad inicial: '); readln (Vo);
   write ('Tiempo de caida  : '); readln (t);
   y := Vo*t - (1/2)*g*sqr(t);
   writeln ('La distancia total en la caida libre es ',y:10:3);
end.
```


PROCESO COMPILACION/EJECUCION

Al Sistema de TURBO Pascal se le denomina comúnmente "El ambiente del Turbo" (Turbo enviroment). Se le llama ambiente de trabajo porque es más que un compilador: Es una zona de trabajo.

Tradicionalmente, un compilador de Pascal es un programa separado. Para desarrollar programas en Pascal se debe ejecutar un procesador de textos y elaborar el programa fuente en él. Entonces se abandona el editor y se ejecuta el compilador de Pascal, el cual compila el texto que se ha editado. Después de eso, en la mayoría de los casos se debe encadenar (linker). El linker prepara el código para ser ejecutado. Solamente hasta que el linker ha finalizado su tarea, es cuando se puede ejecutar el programa y revisar si funciona adecuadamente.

Estos pasos representan un conjunto de brincos entre un programa y otro, y toman demasiado tiempo cuando se trabaja con discos flexibles. En contraste, el ambiente de TURBO Pascal es un programa que se ejecuta solamente una vez, permitiendo hacer todo dentro de él. Incluye un editor de textos, un compilador, un medio para ejecutar los programas y algunas utilerías para depurar: Todo en un solo programa.

El sistema consiste en un menú principal y conjuntos de submenus de ventana. El menú principal aparece en la parte superior de la pantalla de la siguiente forma:

File Edit Run Compile Options Debug Break/Watch

Excepto por el comando Edit, cada seleccíon del menú principal despliega una lista de opciones adicionales. En este capítulo se examinarán las más importantes.

EL SUBMENU FILE

Los comandos del submenú se utilizan para grabar un programa nuevo en el disco, cargar en memoria uno ya existente, trabajar con un directorio diferente o terminar la sesión de Turbo Pascal. A continuación se examinarán cada una de ellas:

EL COMANDO LOAD

Load, el primer comando del submenú File, lee un programa ya existente en disco y lo carga en la memoria del editor de Turbo Pascal. Una vez que el archivo está en memoria, es posible modificarlo, compilarlo y ejecutarlo.

cuando se selecciona el comando, una ventana de entrada de datos titulada "Load File Name" se habre en la pantalla preguntando el nombre del archivo que se desea cargar en el editor. Asumiendo que se sabe el nombre completo del archivo es posible teclearlo y presionando <Enter> confirmar. Turbo Pascal buscará y cargará dicho programa. Si el programa se encuentra en un drive o directorio diferente al actual, es posible teclear el nombre completo y Turbo Pascal buscará en el lugar indicado.

Si no se especifica extensión, Turbo Pascal automáticamente le coloca .PAS antes de buscar en disco.

Si Turbo Pascal no encuentra el archivo, regresará al editor. El nuevo archivo con el que se trabajará en el editor será el especificado. Este archivo no existe en disco hasta que se grabe con el comando Save.

Hay una alternativa interesante cuando se especifica el archivo a cargar en la ventana: el nombre del archivo puede contener los comodines (wildcards) asterisco (*) y el símbolo de interrogación (?). Un nombre que contenga los caracteres comodines es conocido como Mask (máscara). Por ejemplo, cuando el comando Load es invocado, aparece lo siguiente: *.PAS.

Si se presiona <Enter> aceptando la máscara, Turbo Pascal muestra una lista con todos los programas con extensión .PAS del directorio actual.

Para seleccionar un programa se utilizan las teclas de las flechas y, posicionándose en el nombre del programa, se presiona <Enter>; el archivo se cargará en el editor del Turbo Pascal.

Si en algún momento se desea cancelar la operación, basta con presionar la tecla <ESC>.

EL COMANDO PICK

El comando Pick da la facilidad de cargar en el editor los archivos que ya se habían cargado en el transcurso de la sesión actual. El comando presenta una lista de los ocho nombres mas recientes. Con las teclas de las flechas y <Enter> es posible seleccionar uno.

EL COMANDO NEW

El comando New inicializa la zona de trabajo del editor, dando oportunidad a empezar con un programa completamente nuevo. El nombre de éste es NONAME.PAS. Cuando se termine de editar y se desee grabar en disco, Turbo Pascal permitira cambiar al nombre que se prefiera. Cada vez que el editor reinicia su trabajo con un archivo nuevo, verifica que los cambios que se hayan hecho hayan sido grabados. Si no es así, avisa:

B:\EJEMPLO.PAS not saved. Save? (Y/N)

Si se presiona Y, el archivo se grabará en disco antes de ser descargado del RAM. Si se presiona N, los cambios no se actualizan a disco, pero el archivo se borrará del RAM, dando lugar a uno nuevo.

Si en algún momento se desea cancelar la operación, basta con presionar la tecla <ESC>.

EL COMANDO SAVE

El comando Save graba el programa actual del editor. Si se está grabando el programa por primera vez, y éste tiene el nombre NONAME.PAS, Turbo Pascal da la oportunidad de cambiar a cualquier otro nombre que se prefiera através de la ventana:

Rename NONAME

B:\NONAME.PAS

Inmediatamente que se presione la primera tecla del nombre nuevo, el nombre anterior desaparece de la ventana. Si se desea, se puede dar la especificación exacta del drive y subdirectorio. Si se omite la extensión, Turbo Pascal coloca .PAS.

Si ya antes se había grabado el programa, Turbo Pascal renombra la versión vieja con la extensión .BAK y la nueva adopta el nombre del programa.

EL COMANDO WRITE TO

El comando Write to da la oportunidad de cambiar el nombre del programa del editor. Cuando se selecciona, aparece una ventana preguntando por el nombre nuevo. Cuando se tecleea el nombre nuevo y se presiona <Enter>, Turbo Pascal graba una copia con el nombre nuevo, y el nombre del archivo en el editor adopta el nuevo. Cualquier versión anterior del programa permanece en disco.

EL COMANDO DIRECTORY

El comando Directory simplemente despliega en pantalla el directorio. Se puede cambiar la máscara *.* por alguna otra que se desee. Si el directorio contiene subdirectorios, es posible seleccionar con las teclas de las flechas el subdirectorio y ver los archivos que se encuentran allí. Esta característica se aplica a todos los comandos en donde interviene la ventana de selección de archivos.

EL COMANDO CHANGE DIR

Con el comando Change dir es posible cambiar el directorio actual de trabajo por uno nuevo, incluyendo el drive. Todas las operaciones subsecuentes utilizarán el directorio nuevo.

EL COMANDO OS SHELL

El comando OS shell provee una salida temporal al prompt del sistema operativo. OS shell envia la sesión al prompt normal para efectuar cualquier operación o ejecutar algún programa. Cuando se desee regresar a la sesión en Turbo Pascal, basta con teclear EXIT y aparecerá el editor de nuevo.

EL COMANDO QUIT

El comando Quit termina la sesión con Turbo Pascal. A diferencia del comando OS shell, no retiene ningún dato del compilador. Para regresar al editor es necesario ejecutar otra vez el TURBO desde el prompt del DOS.

Si hay algún archivo sin grabar, Turbo Pascal preguntará si se desea grabar en disco o no. Existe la opción de cancelar la salida presionando <Enter>.

EL SUBMENU RUN

El submenú Run ofrece varias diferentes maneras de ejecutar un programa. Hay seis comandos en el submenú, junto con sus maneras de acceso directo:

Para ejecutar un programa en forma normal se debe utilizar el primer comando. En contraste, los otros comandos del submenú representan algunas facilidades que Turbo Pascal ofrece para depurar y monitorear los programas.

EL COMANDO RUN

El comando Run inicia la ejecución normal del programa actual. Si este no ha sido previamente compilado, y si se ha modificado desde la última compilación, Run primero compila el código (haciendo el equivalente a Make en el submenú de Compile). Al final de la ejecución Turbo Pascal no retiene la pantalla de salida (Output screen). Si se desea revisar esta pantalla, presione Alt-F5 (la forma rápida del comando User screen del submenú Run).

EL COMANDO PROGRAM RESET

El comando Program reset desactiva algunos aspectos de la sesión de depuración.

Una sesión de depuración consta de los siguientes aspectos:

- Establecer punto de quiebre (break points) en las líneas en donde se desea interrumpir el código. (El comando Toggle breakpoint se encuentra en el submenú Break/watch).

- Ejecutar el código desde el inicio hasta la posición actual del cursor, usando el comando Go to cursor.

- Ejecutar el programa línea por línea usando el comando Trace into o el comando Step over.

- Examinar el valor de alguna expresión en particular. (El comando Evaluate se encuentra en el submenú Debug).

- Examinar alguna expresión específica en la ventana Watch durante una sesión de depuración. (El comando Add watch se encuentra en el submenú Break/watch).

- Interrumpir el programa presionando las teclas Ctrl-Break.

Después de interrumpir el desarrollo de una sesión de depuración, el comando Program reset deja el control del código en el punto en que fué interrumpido. Si se desea terminar la sesión de depurar e iniciar una sesión normal de trabajo desde el inicio del progama, se deben seguir los siguentes pasos:

1. Ejecutar el comando Program reset (Ctrl-F2) de tal forma que el programa pueda ser ejecutado desdeel inicio.
2. Ejecutar el comando Clear all break points para desactivar todos los puntos de quiebre que hayan sido establecidos.
3. Ejecutar el comando Remove all watches para limpiar la ventana Watch.
4. Ejecutar el comando Run para correr el programa.

El comando Program reset libera la memoria extra que Turbo Pascal requiere para la sesión de depuración. Note, sin embargo, que no quita los puntos de quiebre ni las expresiones watch; tampoco reinicializa los valores de las variables del programa.

EL COMANDO GO TO CURSOR

El comando Go to cursor ejecuta el código hasta la línea en que está el cursor del editor. (La línea del cursor no se incluye en esta ejecución parcial). Este comando permite iniciar una sesión de depuración, ejecutando el programa hasta el punto en que se presente algún problema en particular.

Si se está ejecutando el programa por primera vez (o inmediatamente después de haber ejecutado el comando Program reset), Go to cursor inicia su ejecución al inicio del programa. De otra forma Go to cursor empieza la ejecución del codigo en la línea inmediata a la última ejecución, continuando hasta la posición actual del cursor.

Al igual que el comando Run, la acción de Go to cursor se ve afectada por cualquier punto de quiebre que se haya establecido.

EL COMANDO TRACE INTO

Durante una sesión de depuración, Trace into ejecuta la siguiente línea de código del programa. Si se desea ejecutar todo el programa línea por línea, es necesario ejecutar el comando Trace into repetidas veces. La manera rápida es con F7.

EL COMANDO STEP OVER

Igual que Trace into, Step over ejecuta la siguiente línea de código. La diferencia estriba en que, mientras que Trace into incursiona en el código de los procedimientos, Step over ejecuta el procedimiento de una pasada. Step over no transfiere el control del programa a las líneas de código de los procedimientos o funciones. La manera rápida es con F8.

EL COMANDO USER SCREEN

El comando User screen despliega la ventana de salida en forma completa, quitando temporalmente el menú de Turbo Pascal. La manera rápida es con Alt-F5. La ventana de salida muestra la salida de pantalla del programa que se ejecutó anteriormente, o el ambiente del DOS antes de invocar a Turbo Pascal. Presionando cualquier tecla se regresa al ambiente Turbo.

EL SUBMENU COMPILE

El menú Compile muestra algunas opciones diferentes para compilar el programa actual, incluyendo tres formas diferentes de iniciar el proceso de compilación. Este menú contiene los diferentes comandos:

Los comandos Make y Build son formas alternas de compilar un programa que conste de varios archivos. Ya que la acción de Make y Build depende del estado actual del comando Primary file, se examinarán los comandos en diferente orden.

EL COMANDO COMPILE

El comando file inicia el proceso de compilación, trabajando siempre con el programa que se encuentra en el editor. Turbo Pascal despliega una ventana de compilación, donde despliega la información del archivo que se está compilando y el destino del código objeto. Si la compilación es exitosa, (esto es, si no hubo errores en el programa), la ventana premanece en la pantalla para mostrar que el proceso se terminó. Por otro lado, si hubo algún error de código, Turbo Pascal activa la ventana del editor de texto y posiciona el cursor en la línea en donde encontró el error. El mensaje del código del error y una breve explicación del error encontrado aparencerá en la parte superior de la pantalla. La manera rápida es con Alt-F9.

EL COMANDO PRIMARY FILE

Turbo Pascal proporciona dos formas diferentes de compilación para cuando se está trabajando con múltiples archivos. La técnica más simple permite almacenar procedimientos, funciones, o alguna otra sección de código fuente en un archivo independiente en disco. Para incorporar el archivo de inclusión en algún programa, es necesario insertar una directiva del compilador en el programa. La segunda técnica consiste en compilar unidades separadas (units); es mas compleja, pero también más poderosa.

Independientemente de la técnica usada, será necesario trabajar con varios archivos a la vez. Ya que solamente puede tenerse un archivo de código en el editor, este tipo de sistemas deben manejarse con cuidado en cuanto al manejo de archivos se refiere. El comando Primary file se utiliza para informar al compilador el nombre del programa principal del proyecto multi-archivo. Una vez que se ha hecho, los comandos Make y Build direccionan el trabajo del compilador en el archivo principal, a pesar del texto que se esté editando en ese momento.

Después de terminar de trabajar con el programa multi-archivo, y antes de intentar compilar algún otro programa, se debe quitar el nombre del comando Primary file; para hacerlo, seleccione el comando Primary file, y presione Ctrl-Y para borrar la línea del nombre.

EL COMANDO MAKE

El comando Make compila el archivo principal (primary file), si es que se seleccionó alguno. Si no, Make compila el archivo que se encuentre en ese momento en el editor.Además, Make obliga al compilador a examinar los archivos fuente de las unidades del usuario especificadas en el estatuto USES. Si se ha modificado el código fuente de alguna unidad, se recompila. (Para determinar si recompila o no, Turbo Pascal compara la fecha y la hora del archivo principal y el de la unidad).La manera rápida es con F9.

EL COMANDO BUILD

Al igual que el comando Make, Build compila el archivo principal, o el archivo que se encuentre en el editor, si es que no se ha designado alguno. Build también recompila los códigos fuentes disponibles de las unidades que forman parte del proyecto multi- archivo sin importar que hayan sido modificados o no. Use Build para forzar la recompilación completa de todo el programa.

EL COMANDO DESTINATION

Utilice el comando Destination para designar el lugar endonde se desee que Turbo Pascal almacene el código objeto producto de la compilación. El comando tiene dos estados válidos:

Destination Memory

que será lo que normalmente se vea cuando se esté compilando en el ambiente del Turbo. La otra alternativa es:

Destination Disk

El resultado es un programa ejecutable directamente desde el DOS. Turbo Pascal encadena automáticamente los recursos del lenguaje al código objeto resultante para producir un archivo .EXE.

EL COMANDO FIND ERROR

Cuando se genera un error de corrida en el ambiente del compilador Turbo Pascal localiza inmediatamente la línea en donde ocurrió. Si este programa se estuviese ejecutando en el ambiente DOS, se enviaría este mensaje a la pantalla:

Runtime error 002 at 0000:03C5.

Este mensaje contiene dos datos importantes: el número de error y la dirección de memoria en donde éste ocurrió. Es posible utilizar esta información para determinar cúal fué la causa del error: revise el número de error en la lista de códigos de error en la documentación de Turbo Pascal.

Para traducir la dirección del error en una línea de código es necesario hacer los siguientes pasos:

1. Escriba la dirección del mensaje.
2. Ejecute el ambiente TURBO.
3. Cargue el código fuente en el editor.
4. Seleccione el submenú COMPILE y seleccione el comando Find error.
5. Teclee la dirección del código del error.

Después de esto, Turbo Pascal localizará la línea de código fuente que se referencía con la dirección dada.

EL COMANDO GET INFO

El comando Get info presenta una ventana en la pantalla con la información general del programa que se encuentra en el editor:

- Nombre y tamaño del código fuente.
- Archivo principal (primary file) si hay alguno.
- Número de líneas compiladas.
- Destino del código fuente.
- Errores producto de la compilación.

CAPITULO 2.- TIPOS DE DATOS

LA SECCION DE DECLARACION DE UNIDADES

La versión 5.0 de Turbo Pascal tiene ocho unidades predefinidas, llamadas UNIDADES ESTANDAR. Estas unidades contienen procedimientos y funciones propias de Turbo Pascal.

Los nombres de estas unidades son: SYSTEM , DOS, OVERLAY, CRT, PRINTER, GRAPH, TURBO3 y GRAPH3.

La primera de ellas, la unidad SYSTEM, contiene las declaraciones y definiciones esenciales que todos los programas necesitan. Turbo Pascal encadena automáticamente los recursos que esta unidad provee cuando se compila un programa. La unidad SYSTEM no debe ser incluida explícitamente por medio del estatuto USES; esto genera un error.

El resto de las unidades estándar contienen rutinas que pueden seleccionarse en cada programa. Para usar cualquiera de estas unidades, se debe incluir el nombre de la unidad en la zona del estatuto USES. A continuación se da una breve descripción del contenido de estas unidades:

DOS.- Contiene las rutinas que ayudan a elaborar programas asociados con el sistema operativo de disco. Por ejemplo: GETDATE y GETTIME, procedimientos para accesar la fecha y la hora del sistema.

OVERLAY .- Pequeño grupo de rutinas requeridas para la implementación de código traslapable en Turbo Pascal.

CRT .- Incluye una variedad de rutinas comunes para controlar el video y el teclado. Por ejemplo: CLRSCR, CLREOL, GOTOXY, etc.

PRINTER .- Contiene la técnica para enviar datos a la impresora, en lugar del video. PRINTER define un archivo de texto con el nombre de LST. Cuando se utiliza WRITE o WRITELN direccionado a este archivo, Turbo Pascal lo redirecciona a la impresora.

GRAPH .- Es una colección grande de rutinas de graficación.

TURBO3 y GRAPH3 .- Contienen las rutinas de la versión 3.0 de Turbo Pascal. En la mayoría de los casos, las rutinas de estas unidades han sido reemplazadas por procedimientos nuevos en la versión 5.0; sin embargo, se incluyen éstas para efectos de compatibilidad de código con la versión 3.0.

Por conveniencia, cinco de estas ocho unidades están "empacadas" en un archivo de librería llamado TURBO.TPL. Estas unidades son: SYSTEM, DOS, OVERLAY, CRT y PRINTER.

CONSTANTES

Las constantes son valores que se "incluyen" en el código fuente y no cambian durante la ejecución del programa. Hay dos tipos de constantes en el Pascal estándar: literales y declaradas. TURBO Pascal provee un tipo más de constante, aunque en realidad no es tal. Se trata de las constantes estructuradas, las cuales contienen estructuras de datos, como arrays y records. Una constante literal es aquella que se representa como un valor en el código fuente. Ejemplo:

```
VolumenEsfera := (4/3)*PI*(Radio*Radio*Radio)
```

En la línea, 4 y 3 son constantes literales, representando los valores de 4 y 3. Sin embargo, hay otro tipo de constante: el identificador PI, el cual debió haber sido declarado en el programa de la siguiente manera:

```
CONST
    PI = 3.14159;
```

Se puede usar la constante literal 3.14159 en lugar del PI, pero PI es más corto y hace la expresión más legible. En cuanto a números se refiere, las constantes hacen al código fuente más legible.

Otro uso de las constantes se da cuando se tiene un parámetro que rara vez cambiará en un programa. Tal vez algún día cambie de valor, entonces resultará ventajoso modificar solamente la declaración en lugar de revisar todas las líneas de código para hacer el cambio.

En Pascal estándar las constantes pueden ser solamente de tipos simples y sets. Eso incluye números reales, enteros, bytes, caracteres, strings, sets y booleanos. Las constantes de tipo enumerado individual pueden ser consideradas como tales, aunque se declaran de manera diferente. Ejemplos:

```
CONST
  PI          = 3.14159;        {Real de punto flotante  }
CONST
  PI          = 3.14159;        {Real de punto flotante  }
  Threshold   = -71.47;         {Real negativo           }
  Direccion   = $06;            {Valor hexadecimal       }
  Usa8087     = true;           {Booleano                }
  Drive       = 'A';            {Caracter                }
  Revision    = 'V2.16B';       {String                  }
  Respuesta   = 42;             {Entero                  }
  NoRespuesta = -42;            {Entero negativo         }
  Afirmacion  = ['S','s';]      {Set                     }
  Nulo        = '';             {String nulo             }
```

La diferencia más evidente entre una constante y una variable es que el valor de la constante se asigna al tiempo de compilación. No se puede asignar un valor a una constante.

```
program MuestraFecha;
uses
  Crt;
const
  NomMes : array[1..12] of string[3] = ('ENE','FEB','MAR','ABR','MAY','JUN',
                                        'JUL','AGO','SEP','OCT','NOV','DIC');
var
  Dia, Mes, Anno : integer;


procedure Error;
begin
  writeln;
  writeln ('** ERROR ** Esta fecha no existe');
  halt
end;


begin
  repeat
    write ('Dame la fecha en formato DD MM AA : ');
    readln (Dia, Mes, Anno);
    if (Dia  31) or (Mes  12) or (Anno  99) or
      (Dia  1)  or (Mes  1)  or (Anno  1)  then Error;
    if (Mes = 2) and (Dia  28) and (Anno mod 4  0) then Error;
    writeln ('La fecha es ',Dia:2,' de ',NomMes[Mes],' de 19',Anno);
  until false;
end.
```


TIPOS

Los datos representan información que los programas manipulan. Los datos pueden ser números, caracteres u otros símbolos, como las condiciones de Boole falso y verdadero, conjuntos de conceptos, los días de la semana, o estructuras elaboradas en función de términos simples.

La riqueza de las expresiones con las que Pascal puede tratar los datos lo colocan muy lejos de lenguajes como BASIC y FORTRAN. Versiones modernas de BASIC pueden manejar strings y diferentes tipos de datos numéricos, pero no permiten manejar estructuras construidas a partir de tipos de datos simples.

El "tipo" de un dato es una serie de reglas que gobiernan el almacenamiento y el manejo de cada dato. Los datos ocupan un lugar en el RAM. El tipo de dato determina cuánto espacio es necesario y cómo el dato se representa en ese espacio. Un entero, por ejemplo, siempre ocupa dos bytes de memoria. El bit más significativo representa el signo del entero.

El tipo también determina cómo puede ser usado el dato. Los tipos CHAR y BYTE equivalen a un byte en RAM. Se pueden sumar o multiplicar dos variables de tipo BYTE, pero intentar usar dos variables de tipo CHAR en operaciones aritméticas, genera un error.

La versión 5.0 de Turbo Pascal introduce tipos de datos nuevos, tanto para escalares como para reales. Los tipos de datos escalares nuevos son: word, shortint y longint.

CHAR

El tipo CHAR (caracter) está incluido en el ISO Standar Pascal, por lo tanto, en todas las implementaciones de Pascal.

El tipo CHAR incluye el juego de 128 caracteres ASCII: letras, números, símbolos comunes y los códigos no imprimibles de control, como el retorno de carro, el backspace, etc. Pero actualmente el tipo CHAR incluye 256 caracteres de diferentes valores, ya que ocupan un byte. Los 128 caracteres restantes no tienen nombres ni significados estandarizados en el código ASCII. Cuando se imprimen en la pantalla de la IBM PC, se muestran como letras de idiomas diferentes al Inglés, marcos o símbolos matemáticos. En otras computadoras, estos caracteres se imprimen centelleando o subrayados en variantes del código ASCII estándar.

BYTE

El tipo BYTE no está presente en el ISO Standar Pascal, aunque muchas implementaciones de Pascal lo incluyen. Como CHAR, BYTE ocupa un byte en memoria. A nivel máquina, BYTE y CHAR son iguales, solamente difieren en la forma como son tratados por el compilador. El tipo BYTE puede ser concebido como un entero de "media precisión". Las variables de tipo BYTE solamente pueden compartir asignaciones con el tipo INTEGER.

SHORTINT

El tipo shortint (short integer) puede ser tomado como un byte con signo. Utiliza un byte para almacenamiento, pero el bit más significativo es el bit de signo. Esto le da a shortint un rango de -128 a 127.

INTEGER

El tipo INTEGER es parte del ISO Standar Pascal. Es la manera más eficiente de manejar números pequeños. Los enteros están en el rango de -32768 a 32767.

El identificador predefinido MAXINT es una constante conteniendo el máximo valor que puede ser expresado: 32767. MAXINT es una característica del ISO Standar Pascal, y existe en el TURBO Pascal para ayudar a los programadores a hacer los programas más portables, ya que otras computadoras manejan palabras de datos de tamaño diferente (computadoras de 32 bits definen a MAXINT como 2,147,483,647).

WORD

El tipo word es llamado algunas veces como un entero sin signo. Como el tipo integer, un word toma 2 bytes. En los enteros se usa el bit más significativo para indicar el signo, mientras que en el tipo word no se utiliza un bit para el signo. Esto le da a word un rango de 0 a 65535.

LONGINT

Es probablemente la adición más significativa de tipos de datos en Turbo Pascal. Longint (long integer) utiliza 4 bytes de almacenamiento y tiene un rango de -2,147,483,648 a 2,147,483,647. En consecuencia, se pueden usar enteros largos en muchos lugares donde antes solo se podía trabajar con reales. Esto es especialmente importante, pues los cálculos que usan enteros toman menos tiempo que los que usan reales.

BOOLEAN

El tipo BOOLEAN es parte del ISO Standar Pascal. Una variable booleana puede tener solo dos posibles valores: True y False (verdadero y falso). Las palabras TRUE y FALSE son identificadores predefinidos en Pascal. El compilador las define como constantes del tipo boolean.

Las variables booleanas se usan para manipular los resultados de expresiones, usando los operadores relacionales =, >, <, < >, > = y < =, así como los operadores de conjuntos +, * y -. Una expresión como "2 < 3" es fácil de evaluar; obviamente 2 es menor que 3 y el resultado es verdadero. Si esto fuese un estatuto en Pascal, la expresión regresaría un valor booleano True, el cual puede ser procesado como sigue:

```
OK : = 2 < 3;
```

Esta asignación guarda el valor booleano True dentro de la variable booleana OK. El valor de OK puede ser usado en estatutos IF..THEN..ELSE.

NUMEROS REALES DE PUNTO FLOTANTE

Todos los tipos de datos descritos anteriormente han sido tipos ordinales. Los tipos ordinales tienen un número limitado de valores posibles, existiendo en un orden definido. Los enteros son de tipo ordinal, ya que son exactamente 65535; y tienen un orden definido, ya que después del 6 está el 7, luego el 8, sin valores posibles entre ellos. Los enteros son de precisión absoluta; esto es, el valor del entero 6 es exactamente seis.

REAL

El mundo real demanda fracciones. ISO Standar Pascal soporta el tipo REAL, el cual expresa números con fracciones y exponentes. Los números reales, especialmente los muy pequeños o muy grandes, no tienen precisión absoluta.

Los números reales pueden ser expresados de dos maneras en Turbo Pascal: Una es la mantisa (1.6125), que da el valor, y el exponente (E10), que da el orden de la magnitud. La segunda manera es expresar el número con un punto decimal: 3.14159, 121.402, -16.66, etc.

En adición, Turbo Pascal 4.0 ofrece 5 tipos de reales: real (igual a versiones anteriores), single, double, extended y comp. Desafortunadamente, los tipos single, double, extended y comp solo pueden usarse si la computadora tiene el chip coprocesador matemático 8087.

SINGLE

El tipo single es el de menor precisión de los tipos reales. Utiliza 4 bytes y tiene un rango de 1.5E-45 a 3.4E-38

DOUBLE

El tipo double ocupa 8 bytes y tiene un rango de 5.0E-324 a 1.7E- 308

EXTENDED

El tipo extended tiene 10 bytes de longitud y un rango de - 2E+63+1 a 2E+63-1

COMP

A pesar de que el tipo de dato COMP es, técnicamente, una variable real, éste actúa como un INTEGER y contiene solamente números sin decimales. Utiliza 8 bytes y tiene un rango de -2E+63+1 a 2E+63-1

```
program DemoNumerico;
uses
   Crt;
var
   Fin : boolean;
   arg  : real;


procedure ExpLnDemo; { El procedimiento ExpLnDemo muestra
                       las funciones Exp y Ln.  }
var
   i : shortint;
begin
   writeln ('** LOGARITMOS **':50);
   write   (' ':6, 'n');
   write   (' ':6, 'Exp (n) ');
   writeln (' ':4, 'Ln (n)');
   for i : = -10 to 10 DO
   begin
      arg : = i / 2;
      write (arg: 10: 4, Exp (arg): 10: 4);
      if i < = 0 then writeln ('- ':8) else writeln (Ln(arg):10:4);
   end
end;{-------------------------------------------------------------}


procedure  TrigDemo;   { El procedimiento TrigDemo muestra el uso
                         de las funciones trigonometricas: sin,
```

```
                                cos, arctan, y PI. }
var
   i : shortint;
begin
   writeln ('** FUNCIONES TRIGONOMETRICAS **':50); writeln;
   write  (' ':4, 'n');
   write  (' ':7, 'sin (n*PI) ');
   write  (' ':4, 'cos (n*PI) ');
   writeln (' ':2, 'arctan (n) ');
   for i := -8 TO 8 DO
   begin
      arg := PI * i / 8;
      write (i / 8: 7 : 4);
      write (sin (arg): 13 : 4);
      write (cos (arg): 13 : 4);
      writeln (arctan (i / 8): 13 : 4)
   end
end; {-------------------------------------------------}


procedure IntDemo;      { El procedimiento IntDemo muestra las
                          funciones enteras: INT, ROUND, TRUNC y FRAC. }

var
   i : shortint;
begin
   writeln ('** FUNCIONES PARA ENTEROS **':60);
   write  (' n');
   write  (' ':6, 'INT (n)');
   write  (' ':3, 'ROUND (n)');
   write  (' ':2, 'TRUNC (n)');
   writeln (' ':3, 'FRAC (n)');
   for i:=-10 to 10 do
   begin
      arg := i/4;
      write  (arg:5:2);
      write  (Int(arg):10:1);
      write  (round(arg):10);
      write  (trunc(arg):10);
      writeln (frac(arg):13:2)
   end
end; {-------------------------------------------------}


procedure AritDemo;   { El procedimiento AritDemo muestra algunas
                        funciones miscelaneas: ABS, SQR, y SQRT. }
var
   i : shortint;
begin
   writeln ('** FUNCIONES ARITMETICAS **':60);
   write  (' n');
```



```
   write  (' ':6, 'ABS (n)');
   write  (' ':3, 'SQR (n)');
   writeln (' ':2, 'SQRT (n)');
   for i:=-10 to 10 do
   begin
      write  (i:3);
      write  (abs(i):10);
      write  (sqr(i):10);
      if (i) then writeln ('- ':10) else writeln (sqrt(i):12:4)
   end
end; {-------------------------------------------------}


procedure PotenciaDemo; { El procedimiento PotenciaDemo muestra
                          la forma de exponenciacion en Pascal }
var
   i     : shortint;
   Num : real;
begin
   writeln ('** EXPONENCIAL **':50);
   write  (' ':8, 'x');
   write  (' ':13, 'y');

   writeln (' ':13, 'x^Y');
   for i := -10 to 10 DO
   begin
      Num := i / 2;
      write (Num:10:4, Num:15:4);
      if Num0 then writeln (Exp(Num*ln(abs(Num))):15:4) else writeln ('-':13);
   end
end;{-------------------------------------------------}


function EsPrimo (num : integer) : boolean;
var
   Primo  : boolean;
   Prueba : integer;
begin
   Primo := true;
   Prueba := 2;
   while Primo and (Prueba < = Imt (sqtr(num))) do
      if num mod prueba = 0 then primo: false else inc(prueba);
   EsPrimo := Primo;
end; { fin de funcion EsPrimo }


function Factorial (num : integer) : real;
begin
   if num = 0 then
      Factorial:=1
   else
```

```
    Factorial := num * Factorial(num-1);
end; { fin de funcion Factorial }


procedure PrimosFactorialDemo; { El procedimiento PrimosFactorialDemo muestra
                                 como calcular un factorial y determinar si
                                 es numero primo o no. }
var
  I : shortint;
begin
  writeln ('** FUNCIONES FACTORIAL Y PRIMOS **'); writeln;
  write   (' N');
  write   (' ':6, 'N es numero primo?');
  writeln (' ':6, 'Factorial (N)');
  for I:= 1 to 20 do
  begin
    write   (I:2);
    if EsPrimo(I) then write ('SI':15) else write ('No':15);
    writeln (Factorial(I):25:0);
  end
end;{-------------------------------------------------------}


procedure HiperbolicaDemo; { El procedimiento HiperbolicaDemo muestra
                             como calcular funciones hiperbolicas sinh, cosh. }
var
  I : shortint;
begin
  writeln ('** FUNCIONES HIPERBOLICAS **':60);
  write   (' n':5);
  write   ('sinh (n)':17);
  writeln ('cosh (n)':16);
  for I:=-10 to 10 do
  begin
    arg:= I/4;
    write   (arg:6:3);
    write ((exp(arg)-exp(-arg))/2:15:4);
    writeln ((exp(arg)+exp(-arg))/2:15:4);
  end
end;{-------------------------------------------------------}


  { El procedimiento Menu despliega un menu en la pantalla y responde
    apropiadamente a la seleccion del usuario. El parametro Salir
    regresa un valor TRUE cuando el usuario selecciona la opcion Terminar. }

procedure Menu (var Salir : boolean);
const
  col = 25;
var
  Seleccion : char;
```



```
begin
  Irscr;
  Salir := false;
  gotoxy (col,5);
  writeln ('Funciones Numericas');
  gotoxy (col,7);  writeln ('1. Exponencial');
  gotoxy (col,8);  writeln ('2. Trigonometrica');
  gotoxy (col,9);  writeln ('3. Enteros');
  gotoxy (col,10); writeln ('4. Aritmeticas');
  gotoxy (col,11); writeln ('5. Potencias');
  gotoxy (col,12); writeln ('6. Primos y factoriales');
  gotoxy (col,13); writeln ('7. Hiperbolicas');
  gotoxy (col,14); writeln ('8. Terminar');
  writeln; gotoxy (col-5,18);
  write (' *** Seleccion de Menu: ');
  repeat
    Seleccion := readkey;
  until Seleccion in ['1'..'8'];
  clrscr;
  case Seleccion of
    '1' : ExpLnDemo;
    '2' : TrigDemo;
    '3' : IntDemo;
    '4' : AritDemo;
    '5' : PotenciaDemo;
    '6' : PrimosFactorialDemo;
    '7' : HiperbolicaDemo;
    '8' : Salir := true;
  end;
  if Seleccion  '8' then
  begin
    writeln;
    write ('Presiona la <barra Espaciadora> para continuar...');
    repeat
    until readkey = ' ';
  end;
  clrscr;
end; {-------------------------------------------------------}


begin
  repeat
    Menu (Fin)
  until Fin
end.
```

SUBRANGOS Y TIPOS ENUMERADOS

Todos los tipos explicados anteriormente son tipos simples, predefinidos en TURBO Pascal y listos para ser usados. Gran parte de la fuerza de Pascal es su habilidad para crear estructuras de datos fuera de esos tipos simples. Las estructuras de datos pueden hacer los programas fáciles de escribir y, posteriormente, de leer.

Definir tipos para datos particulares es bastante fácil. La palabra reservada TYPE inicia la parte de la definición de tipos del programa, y es aquí precisamente donde se necesita algo de planeación para estructurar los datos:

```
TYPE
ElTipo = LaDefinicion;
```

De aquí en adelante los tipos que se discutan deben ser declarados y definidos en la parte de definición de tipos del programa.

Una definición de tipos no ocupa un lugar en el área de datos del programa, únicamente provee de instrucciones al compilador, para el manejo de datos que sean de ElTipo en el programa fuente.

SUBRANGOS

La estructura de datos más sencilla que se puede definir es un subconjunto de un tipo ordinal llamado "subrango". Si se escogen dos valores legales cualquiera, del tipo ordinal, esos dos valores, más los de enmedio, definen el subrango de tipo ordinal. Por ejemplo:

```
TYPE
    Mayusculas      ='A'..'Z';
    Minusculas      ='a'..'z';
    Digitos         ='0'..'9';
```

Mayúsculas son el rango de caracteres A,B,C,D,E hasta Z. Dígitos incluye los caracteres de 0,1,2,3 hasta 9. Las comillas son importantes; le dicen al compilador que los valores en el subrango son de tipo CHAR. Si la declaración se hace sin comillas para Dígitos:

```
Digitos = 0..9;
```

se tiene un subrango del tipo entero, pues '7' no es lo mismo que 7.

Una expresión en la forma 'A'..'Z' o 3..6 se llama "intervalo cerrado". Un intervalo cerrado es un rango de valores ordinales, incluyendo los dos que marcan los límites y todos los que caen en él.

TIPOS ENUMERADOS

Los novatos en Pascal frecuentemente encuentran difícil de digerir la notación de los tipos enumerados. El "TURBO Pascal Reference Manual" llama a los tipos enumerados "tipos escalares definidos por el usuario". Muchos textos de Pascal se refieren a ellos como tipos enumerados. Un tipo enumerado es un tipo ordinal que el programador define. Uno de los mejores ejemplos que el lenguaje ofrece es el tipo BOOLEAN.

El tipo BOOLEAN, de hecho, es un tipo enumerado predefinido por el compilador y usado de manera especial. El tipo BOOLEAN es una lista ordenada de dos valores con nombres únicos: False y True. No es un par de cadenas de caracteres conteniendo las palabras en Inglés "False" y "True". El tipo BOOLEAN es un número binario con los valores de 00 y 01. Se ha nombrado el código 00 dentro del tipo BOOLEAN como False y el código 01 dentro del tipo BOOLEAN como True.

Considere una lista de valores con nombres únicos: los colores del espectro luminoso. Para crear un tipo enumerado de él se declara:

```
TYPE
    Espectro = (Rojo, Naranja, Amarillo, Verde, Azul, Indigo, Violeta);
```

La lista de un tipo enumerado siempre se marca entre paréntesis. El orden en que se colocan los valores define el valor ordinal, el cual puede ser probado usando la función ORD(X). Por ejemplo, Ord(Amarillo) regresará el valor 2. Ord(Rojo) 0.

Se pueden comparar valores de un tipo enumerado con otros del mismo tipo. El estatuto Amarillo > Rojo (2 > 0) regresará el valor booleano True.

```
Verde = Violeta          o          Azul < Naranja
```

Regresarán ambos el valor booleano False.

Los valores del tipo Espectro son todos constantes. Pueden ser asignados a variables de tipo Espectro. Por ejemplo:

```
VAR
    Color1, Color2 : Espectro;
BEGIN
    Color1:=Amarillo;
    Color2:=Indigo;
```

Una gran desventaja de los tipos enumerados es que no pueden ser impresos en la pantala o en la impresora. No puede codificar Writeln (Naranja), y esperar ver la palabra "Naranja" en la pantalla. TURBO Pascal genera el error 66:I/O no permitido. En esos casos se puede crear un array de strings indexados por el tipo enumerado:

```
Nombres          : array[Rojo..Violeta] of String[80];
Nombres[Rojo]    :='Rojo';
Writeln (Nombres[Rojo]);
```

ARRAYS

Un array (vector o arreglo) es una estructura de datos consistente en un número determinado de elementos idénticos, con un solo identificador como referencia para el grupo. El programa se refiere a cada elemento por medio de un número o índice. En Pascal, los índices no son necesariamente números. Tipos enumerados, caracteres y subrangos, pueden actuar algunas veces como índices, permitiendo una enorme riqueza de expresiones, imposible de obtener con algún otro lenguaje computacional.

Un elemento de un array puede ser de cualquier tipo, excepto de tipo FILE (archivo). Los arrays pueden constar de estructuras completas de datos. Es posible tener arrays de records (registros) o arrays de arrays. Un índice debe ser parte de algún subrango o grupo de tipo ordinal, o un tipo enumerado definido por el usuario.

Los siguientes son ejemplos para mostrar algunas declaraciones válidas:

```
CONST
     Distritos      =14;
TYPE
     String80       =string[80];
     Grados         ='A'..'F';
     Porcentaje     =1..99;
     Niveles        =(K,G1,G2,G3,G4,G5,G6,G7,G8,G9);
     Materias       =(Ingles, Matematicas, Historia, Arte, Espanol);
     Ficha          =record
                        Nombre      :String80;
                        Matrícula   :String80;
                        IQ          : integer;
                        Promedio    :Porcentaje;
                        Finales     :array[Materias] of Distritos;
                    end;
     DefGrados      =array[Grados] of String80;
VAR
     ArchivoK12          :array[Niveles] of Ficha;
     Aprobados           :array[Niveles] of boolean;
     SubTotal            :array[1..24] of integer;
     PorcArea            :array[1..Distritos] of Porcentaje;
     NivelArea           :array[1..Distritos] of array[Niveles] of Porcentaje;
```

Las declaraciones anteriores son parte de un programa en Pascal, para el control escolar de calificaciones. Cuando TURBO Pascal separa área en memoria para un arreglo, no inicializa con cero los elementos. Si hay basura en el RAM al tiempo de compilar, los valores de los elementos del arreglo tendrán valores indefinidos. Es responsabilidad del programador iniciar el arreglo con algún valor (aun con cero).


```
program MiPrimerArray;

uses
  Crt;
var
  Numero : array[1..100] of integer;
  N, i   : byte;
  Suma,
  Prom   : real;
begin
  writeln ('Cuantos numeros son? ');
  readln (N);
  Suma := 0;
  for i:=1 to N do
  begin
     write ('Numero : ');
     readln (Numero[i]);
     Suma := Suma + Numero[i];
  end;
  Prom := Suma / N;
  clrscr;
  writeln ('LOS NUMEROS FUERON :');
  for i:=1 to N do writeln (Numero[i]:10);
  writeln;
  writeln ('La suma es = ',Suma:10:3,' y el promedio es = ',Prom:10:3);
end.
```

RECORDS

Un registro es una estructura compuesta de muchos elementos de diferentes tipos, agrupados por cada elemento. Estos elementos son llamados "campos" de un registro. Un taller de reparación de vehículos podría necesitar un archivo conteniendo el inventario de sus partes de repuesto. Para cada parte de repuesto se necesita un registro con el número de parte, la descripción, el costo de venta, precio de compra, mínimo y existencia. Todos esos elementos están íntimamente encadenados en un grupo físico (la pieza del carro), así que para simplificar el proceso de programación, es necesario reunir todos los campos en un registro:

```
TYPE
    ParteReg = record
                 NumParte,Clase:integer;
                 Descripcion:string[80];
                 NuestroCosto:real;
                 PrecioLista:real;
                 Minimo:integer;
                 Existencia:integer;
               end;
VAR
    ParteActual, ParteProx    :ParteReg;
    ArchivoPartes             :file of ParteReg;
    Almacenados               :integer;
    DebeOrdenarse             :boolean;
```

La nueva entidad se convierte en un nuevo tipo con su propio nombre. Los datos del registro pueden ser asignados, escritos en archivos y otras operaciones como una sola unidad, sin necesidad de especificar ningún campo del registro.

```
ParteActual := ParteProx;                  { asigna una parte a otra }
read (ArchivoPartes, ParteProx);           { lee una parte del archivo }
```

Cuando se necesite trabajar con campos individuales dentro de un registro, la notación consistirá en el nombre del registro seguido de un punto y el identificador del campo.

```
Almacenados := ParteActual.Existencia;
IF Almacenados < ParteActual.Mínimo THEN DebeOrdenarse := True;
```

Los operadores relacionales NO pueden ser usados con los registros. Decir que un registro es "mayor que" o "menor que" otro, carece de sentido. En el ejemplo anterior se compararon dos campos de los registros, no los registros propiamente. Los campos eran numéricos, por lo que pueden ser comparados con el operador "<".

STRINGS

Manipular palabras y líneas de texto es una función fundamental en un programa de computadora. Como mínimo, el programa debe ser capaz de mostrar mensajes tales como "Presione una tecla para continuar...". En Pascal, como en la mayoría de los lenguajes, los caracteres de una línea pueden tomarse juntos y manejarse como una entidad llamada "string". Muchos implementadores de Pascal, incluyendo aquellos que diseñaron TURBO Pascal, han ampliado la definición en cuanto al tratamiento de strings, con strings de "longitud variable", los cuales son tratados por el compilador de manera especial. Los strings de longitud variable tienen una longitud "lógica", misma que varía dependiendo de lo que contenga el string. Strings de diferentes longitudes lógicas pueden ser asignados a otros, siempre y cuando las longitudes físicas no se excedan.

Una variable string se define usando la palabra reservada STRING. Una longitud física debe especificarse entre paréntesis cuadrados (con un máximo de 255). Ejemplo:

```
VAR
    Mensaje    :string[80];      { longitud física de 80 }
    Nombre     :string[30];      { longitud física de 30 }
    Dir        :string[30];      { longitud física de 30 }
    Estado     :string[2];       { longitud física de 2 }
BEGIN
    Mensaje    := 'Esta es una prueba...';
    Nombre     := 'F. Eugenio López G.';
    Dir        := 'Malinche #427 San Nicolás';
    Estado     := 'NL';
```

CAPITULO 3.- ESTATUTOS DE CONTROL

Hasta ahora se han cubierto los diferentes tipos de datos que Pascal ofrece. Los datos son representaciones de conceptos que deseamos que un programa manipule. Este punto se refiere a cómo Pascal manipula los datos.

Como ya se explicó, una variable es una localidad de memoria manejada por el compilador. Su tamaño y su manejo dependen del tipo de ella.

El operador fundamental en Pascal es el de asignación, cuyo símbolo es ":=", con el cual probablemente usted ya esté familiarizado. El operador de asignación es (generalmente) la forma como los valores son puestos en las variables. Considere este estatuto de simple asignación:

```
I := 17;
```

Un valor del lado derecho del operador de asignación se asigna a la variable que está del lado izquierdo. En un estatuto de asignación, siempre hay una variable del lado izquierdo del operador. Del lado derecho puede haber una constante, una variable o una expresión.

Una expresión en Pascal es una combinación de datos y operadores que dan como resultado un valor. Los datos pueden ser constantes o variables. Los operadores son símbolos especiales que desarrollan alguna acción involucrando el valor o valores que se le han dado. Estos valores se llaman "operandos" de los operadores.

El más simple y familiar ejemplo de expresión es el aritmético:

```
17 + 3
```

El operador de adición desarrolla una operación de suma con los dos operandos: 17 y 3. El valor de la expresión es 20.

OPERADORES RELACIONALES

Los operadores relacionales se usan para construir expresiones booleanas, que son las más ampliamente usadas en Pascal. Todos los estatutos de lazos y brincos en Pascal dependen de las expresiones booleanas.

La siguiente tabla resume los operadores relacionales implementados en TURBO Pascal. Todos los resultados son booleanos.

OPEARADORES RELACIONALES EN Turbo Pascal

Operador	Símbolo	Tipo de Operando
Igual	=	escalar, string, apuntador, registro
No igual	< >	escalar, string, apuntador, registro
Menor que	<	escalar, string
Mayor que	>	escalar, string
Menor o igual	< =	escalar, string
Mayor o igual	> =	escalar, string
Es miembro	IN	conjunto, miembros de conjuntos
El izq. está en el der.	< =	conjuntos
El der. está en el izq.	> =	conjuntos
Negación	NOT	booleanos
Conjunción	AND	booleanos
Disjunción	OR	booleanos
EXOR	XOR	booleanos

IGUALDAD

Si dos valores se comparan y son iguales, la expresión que se evalúa es verdadera (True). En general, TURBO Pascal considera dos valores iguales comparando bit por bit. Esto es cierto para comparaciones de tipos iguales.

La excepción son las comparaciones hechas entre valores numéricos de diferentes tipos. TURBO Pascal permite comparaciones entre enteros, reales y bytes con libertad, pero deben hacerse con mucho ciudado.

Los conjuntos son considerados iguales si tienen los mismos elementos. Dos apuntadores se consideran iguales si ambos apuntan a NIL. Dos registros son iguales si ambos son del mismo tipo y todos sus campos son bit por bit idénticos. Dos strings se consideran iguales si tienen la misma longitud lógica y contienen los mismos caracteres.

NO IGUALDAD

Las reglas para las no-igualdades son las mismas que las de la igualdad. La única diferencia es que el valor booleano resultante es el contrario.

En general, se puede usar el operador de no igualdad en las mismas situaciones que el de igualdad.

```
17   =   17   { true }
17   <>  17   { false }
42   =   17   { false }
42   >   16   { true }
```

MAYOR QUE / MENOR QUE

Los operadores MAYOR QUE, MENOR QUE, MAYOR O IGUAL A y MENOR O IGUAL A, añaden una nueva dimensión a las notaciones de comparación. Se asume que los operandos existen en algún orden definido para que la comparación pueda ser hecha.

Esto inmediatamente descalifica a los apuntadores, conjuntos y registros. Decir que un apuntador es más grande que otro no tiene sentido. Lo mismo se aplica a conjuntos y registros. Ordenarlos no tiene sentido lógico, así que los operadores involucrados en el orden no pueden usarse.

Los tipos CHAR y BYTE están limitados a 256 posibles valores, y ambos tienen un orden implícito. El tipo CHAR está ordenado por el juego de caracteres ASCII, lo que convierte estas expresiones en verdaderas:

```
'A'   <    'B'
'a'   >    'A'
'@'   < >  '['
```

Los tipos enumerados están limitados a no más de 255 valores diferentes. El orden en que se declararon son los valores de prioridad que se les asigna.

```
TYPE
Espectro = (Rojo, Naranja, Amarillo, Verde, Azul,Indigo, Violeta);
```

Este orden hace que las siguientes expresiones se evalúen como verdaderas:

```
Rojo   <    Verde
Azul   >    Amarillo
Indigo < >  Violeta
```

NOT, AND, OR, XOR

Estos cuatro operadores trabajan solamente con operadores Booleanos. No comprueban que exista una relación entre dos operandos, sino que combinan los operandos según las reglas del álgebra de Boole, para producir un nuevo valor y evaluar la expresión.

El más sencillo de ellos es el operador NOT. El operando debe seguir del operador. NOT niega el valor del operando.

```
NOT False        { esta expresión es veradera }
NOT True         { esta expresión es falsa }
NOT (6 < > I)    { verdadera si I = 6 }
NOT (J = K)      { verdadera si J < > K }
```


OPERADORES ARITMETICOS

Los números se manipulan con los operadores aritméticos, los cuales, junto con las variables numéricas, forman las expresiones. El único operador que en Turbo Pascal no existe, es el de exponenciación. La siguiente tabla muestra los operadores aritméticos válidos en Pascal:

	Operador	Tipo operando	Resultado
Suma	+	entero, real, byte	igual
Inversión de signo	-	entero, real	igual
Resta	-	entero, real, byte	igual
Multiplicación	*	entero, real, byte	igual
División de enteros	DIV	entero, byte	igual
División de reales	/		real
Módulo	MOD		igual

La lista de operandos son los tipos que el operador puede manejar. El compilador da la libertad de mezclar los tipos de las variables numéricas que están en una expresión dada. En otras palabras, se pueden multiplicar bytes por enteros, sumar reales y bytes, etc. Las siguientes son expresiones válidas en Pascal:

```
VAR
  I,J,K   : integer;
  R,S,T   : real;
  A,B,C   : byte;

  I * B          { entero multiplicado por byte }
  R + J          { entero sumado con un real }
  C + (R * I)    { etc }
  J * (A / S)
```

La columna del tipo resultante indica el tipo de dato que se obtiene al ser evaluada la expresión. Los tipos numéricos deben observar estas reglas:

1) Cualquier expresión que incluya un número real, solamente puede ser asignada a una variable real.

2) Las expresiones que contengan la división de reales (/), pueden ser asignadas a variables reales.

Si no se siguen estas reglas, el compilador genera el error #44: Inconcordancia de tipo.

LA INVERSION DE SIGNO

El signo de inversión es un operador unario, esto es, toma solamente un operando. Cambia el signo del operando, haciendo positiva una cantidad negativa, y viceversa. Nótese que la inversión de signo no se puede usar con el tipo byte, pues éste es absoluto (sin signo). Un byte nunca es considerado negativo.

DIVISION

Hay tres tipos de división en Pascal: Una para los números reales y las otras dos para los enteros y bytes. La división real (/) puede llevar operandos de cualquier tipo, pero siempre produce valores reales. Intentar asignar una división para reales a un tipo no-real genera el error #44, aun si todos los operandos son del mismo tipo:

```
VAR
    I,J,K : Integer;

I := J / K;          { este estatuto no compila! }
```

Cuando se realiza una división, resultan dos números: Uno representa la parte entera de la operación y el otro la parte fraccionaria. En Pascal, la división entera se separa en dos funciones independientes: DIV, que produce la parte entera de la división; y MOD, que produce la parte fraccionaria:

```
J := 17;
K := 3;
I := J DIV K;        { se asigna 5 a I }
I := J MOD K;        { se asigna 2 a I }
```

```
program ParesyNones;
var
  N, i, Numero,
  NumPares, NumNones : integer;
begin
  NumNones := 0;
  NumPares := 0;
  write ('Cuantos numeros se van a analizar? ');
  readln (N);
  writeln;
  for i:=1 to N do
  begin
     write ('Numero: ');
     readln (Numero);
     write ('El ',Numero,' es un numero ');
     if (Numero mod 2) = 0 then
     begin
        inc(NumPares);
        writeln ('PAR.')
     end
     else
     begin
        inc(NumNones);
        writeln ('NON.')
     end
  end;
  writeln;
  writeln ('Pares=',NumPares,' Nones=',NumNones);
end.
```


LOS OPERADORES DE CONJUNTOS

Los operadores de conjuntos se usan para manipular los valores de tipo SET. La tabla resume los operadores de conjuntos implementados en TURBO Pascal. Todos ellos toman operandos y producen valores de tipo SET.

Operador	Símbolo
Set union	+
Set diference	-
Set intersection	*

SET UNION

El resultado de la unión de dos conjuntos es el conjunto en donde están reunidos todos los miembros de los dos conjuntos. El símbolo que lo representa es el de suma (+). Ejemplo:

```
VAR
    SetX, SetY, SetZ : set of char;
    SetX := ['Y', 'y', 'M', 'm'];
    SetY := ['N', 'n', 'M', 'm'];
    SetZ := SetX + SetY;
```

La variable SetZ contiene los caracteres 'Y', 'y', 'N', 'n', 'M', 'm'. Aunque 'M' y 'm' se repiten en los dos conjuntos, en el resultante están solo una vez.

SET DIFERENCE

La diferencia entre dos conjuntos es aquella que contiene solamente los miembros que NO son comunes. Este operador es lo contrario de SET INTERSECTION, que se verá más adelante. Su símbolo es el de resta (-). Ejemplo:

```
VAR
    SetX, SetY, SetZ : set of char;
    SetX := ['Y', 'y', 'M', 'm'];
    SetY := ['N', 'n', 'M', 'm'];
    SetZ := SetX - SetY;
```

La variable SetZ contiene 'Y', 'y', 'N', 'n', que son los elementos no comunes.

SET INTERSECTION

La intersección de dos conjuntos es la que contiene como miembros aquellos que están en ambos conjuntos. Es lo contrario que SET DIFERENCE y su símbolo es el asterisco (*). Ejemplo:

```
VAR
    SetX, SetY, SetZ : set of char;
    SetX := ['Y', 'y', 'M', 'm'];
    SetY := ['N', 'n', 'M', 'm'];
    SetZ := SetX * SetY;
```

La variable SetZ contiene 'M' y 'm', que son los únicos miembros comunes en ambos conjuntos.

Controlar el flujo del programa es la más importante faceta de un programa de computación. Estatutos condicionales que cambien el sentido de flujo de control, lazos que repitan líneas un número determinado de veces y estatutos que tomen un curso dado de acuerdo a un valor de control, incrementan la utilidad de los programas.

Pascal permite que el programador direccione el flujo de control de una manera estructurada y racional, para que los programas sean fáciles de escribir, leer y cambiar cuando sea necesario. Por esta razón, incursionar en un programa de Pascal puede ser en un principio difícil.

La sintaxis sugiere por sí misma que los programas empiecen en el inicio de la página y progresen hacia abajo, terminando abajo cuando el trabajo está concluido. Múltiples puntos de entradas, salidas y brincos incondicionales vía el GOTO provocan más problemas que los beneficios que se obtienen.

BLOQUES BEGIN/END

En Pascal, frecuentemente es necesario que un grupo o número de estatutos sean tratados como un solo bloque o como un solo estatuto. Esto se logra con las palabras reservadas BEGIN y END. Un grupo de estatutos entre un BEGIN y un END se convierten en un estatuto compuesto. Los cuerpos de los programas, procedimientos y funciones, son estatutos compuestos:

```
PROGRAM Raiz;
    VAR
    R, S : real;
BEGIN
    writeln ('Calculo de una raiz cuadrada');
    writeln;
    writeln ('Dame un numero:');
    readln (R);
    S := sqrt (R);
    writeln ('La raiz de',R:7:7,' es ',S:7:7,'.');
END.
```

Los estatutos compuestos pueden ser usados en cualquier lugar como un estatuto simple. Todo lo que esté entre un BEGIN y un END es tratado por el compilador como un estatuto sencillo.


```
program Seno;
uses
   Crt;
const
   Eje = 15;
   Escala = 10;
var
   Angulo : real;
   radian  : integer;
   i       : byte;
begin
   clrscr;
   write ('< GRAFICA DE UNA FUNCION SENOIDAL >':60);
   Angulo := 0;

   for i:=1 to 24 do { EJE Y }
   begin
      gotoxy (1,i);
      write ('|')
   end;

   for i:=1 to 80 do { EJE X }
   begin
      gotoxy (i,Eje);
      write ('-')
   end;

   for i:=1 to 80 do
   begin
      radian := round(Eje-Escala*sin(Angulo));
      gotoxy (i,radian);
      write ('*');
      Angulo := Angulo + 0.1
   end
end.
```

```
program Coseno;
uses
   Crt;
const
   Eje = 15;
   Escala = 10;
var
   Angulo : real;
   radian  : integer;
   i       : byte;
begin
   clrscr;
   write ('< GRAFICA DE UNA FUNCION COSENOIDAL >':60);
   Angulo := 0;
```

```
for i:=1 to 24 do { EJE Y }
begin
   gotoxy (1,i);
   write ('|')
end;

for i:=1 to 80 do { EJE X }
begin
   gotoxy (i,Eje);
   write ('-')
end;

for i:=1 to 80 do
begin
   radian := round(Eje-Escala*cos(Angulo));
   gotoxy (i,radian);
   write ('*');
   Angulo := Angulo + 0.1
end
end.
```

DECISIONES

ESTATUTO IF/THEN/ELSE

Un estatuto condicional redirecciona el programa en una de dos direcciones, en base a un valor booleano. En Pascal el estatuto condicional es IF/THEN/ELSE. La claúsula ELSE es opcional, pero las palabras IF y THEN siempre van asociadas. En su forma más simple, cada estatuto se construye así:

IF expresión booleana THEN estatuto1 ELSE estatuto2

La forma de evaluación es la misma de como lo sugiere la frase en inglés (de la misma forma que BASIC): Si la condición booleana es verdadera, se ejecuta el estatuto #1; si es falsa y el estatuto #2 existe, éste se ejecuta.

Cualquiera de los dos (o ambos) estatutos asociados con IF/THEN/ELSE pueden ser estatutos compuestos. Recuerde que un estatuto compuesto puede ser usado en cualquier lugar como un estatuto simple. Por ejemplo:

```
IF I < 0 THEN
begin
      Negativo := true;
      I := abs(I)
end
      ELSE Negativo := false;
```

Nota importante: No hay punto y coma después del estatuto compuesto BEGIN/END. El fragmento completo es considerado un solo estatuto IF. NUNCA debe haber un punto y coma antes de la palabra reservada ELSE.

Ya que IF es un estatuto perfectamente válido, es posible que uno o varios estatutos contengan IF/THEN/ELSE. Los IF's pueden ser "anidados" tanto como se quiera, pero eso hace el código menos legible. La forma estructurada es:

```
IF [expresión booleana1]THEN
      IF [expresión booleana2]THEN
            IF [expresión booleana3]THEN
                  estatuto;
```

que puede ser transformado con el operador AND de la siguiente forma:

IF expresión booleana1 AND expresión booleana2 AND expresió booleana3 THEN estatuto

La discusión anterior no incluye cláusulas ELSE. Anidar IF's no excluye ELSE, pero cambia totalmente el significado de los estatutos:

```
IF expresión booleana1 THEN estatuto1
ELSE
      IF expresión booleana2 THEN estatuto2
      ELSE
            IF expresión booleana3 THEN estatuto3
```

La mejor forma de tratar este caso es con el estatuto CASE OF, que se verá enseguida. El estatuto CASE OF es una forma compacta de representar IF's anidados.

ESTATUTO CASE/OF

Escoger entre algunas alternativas de caminos de control, resulta crítico en un programa de computación. Se ha visto cómo IF /THEN/ELSE, en su forma más simple, sirve para escoger entre dos alternativas, basado en una expresión booleana. El problema de legibilidad aparece cuando se anidan más de dos o tres. Considere el problema de enviar un mensaje a la pantalla, en base a un número clave de entrada. El reporte de problemas de un sistema computarizado puede ser algo así:

```
Beep;
writeln ('****PRECAUCION****');
IF CodProblema = 1 THEN
      writeln ('[001] Combustible menor del 10%')
ELSE IF CodProblema = 2 THEN
      writeln ('[002] La presión de aceite abajo del mínimo')
ELSE IF CodProblema = 3 THEN
      writeln ('[003] Temperatura de operación muy alta')
ELSE IF CodProblema = 4 THEN
      writeln ('[004] Voltaje de baterías bajo')
ELSE IF CodProblema = 5 THEN
      writeln ('[005] Líquido de frenos menor al especificado')
ELSE IF CodProblema = 6 THEN
      writeln ('[006] Fluido de transmisión faltante')
ELSE IF CodProblema = 7 THEN
      writeln ('[007] Radiador con bajo nivel de agua')
ELSE
      writeln ('[***] Falla de lógica en el reportador')
```

Esto trabaja adecuadamente, pero el código es un poco caótico al final. Esta selección se puede codificar en un solo estatuto CASE OF creado para seleccionar en base a una sola evaluación. Reescribiendo resulta:

```
Beep;
writeln ('****PRECAUCION****');
CASE CodProblema OF
    1: writeln ('[001] Combustible menor del 10%');
    2: writeln ('[002] La presion de aceite abajo del minimo');
    3: writeln ('[003] Temperatura de operacion muy alta');
    4: writeln ('[004] Voltaje de baterias bajo');
    5: writeln ('[005] Liquido de frenos menor al especificado');
    6: writeln ('[006] Fluido de transmision faltante');
    7: writeln ('[007] Radiador con bajo nivel de agua')
ELSE
    writeln ('[***] Falla de logica en el reportador')
end; { final de CASE }
```

Aquí la variable CodProblema se llama "el selector del CASE". Puede ser una variable o una expresión. El valor se mantiene durante la selección de alternativas del estatuto. Los números en las líneas del CASE se llaman "etiquetas del CASE". Cada etiqueta va seguida de dos puntos (:) y después de un estatuto. El estatuto puede ser simple o compuesto.

Cuando el CASE se ejecuta, el selector es evaluado y comparado, uno por uno, contra las etiquetas del mismo. Si una etiqueta es igual al valor del selector del CASE, el estatuto asociado con la etiqueta se ejecuta. Una vez que se ejecutó un estatuto, el CASE OF transfiere el control al resto del programa. Solo uno (o ninguno) de los estatutos del CASE OF se ejecuta cada vez que se evalúa un estatuto CASE OF.

Si ninguna de las etiquetas concuerda con el valor del selector, se ejecuta el estatuto del ELSE (si existe). ELSE es opcional; si no hay el control se pasa al resto del programa.

La forma general del CASE OF es:

```
CASE OF
    ante1 : estatuto1;
    ante2 : estatuto2;
    ante3 : estatuto3;
    .....
    anteN : estatutoN;
ELSE estatuto
END;
```

Se pueden tener tantas etiquetas como se quiera, hasta 256. En el ejemplo, cada etiqueta es una constante numérica. Una etiqueta NO puede ser una variable. Se puede componer de una lista de constantes separadas por comas.

El siguiente ejemplo es de un sistema de análisis por correo. Las respuestas son agrupadas en distribuciones geográficas de los estados de la república. Estado es un tipo enumerado de los estados en orden alfabético. Este orden toma el número de respuestas para cada región en particular:

```
TYPE
    Estado =  (AGS,  BCN, BCS,CAM, COH, COL, CHP, CHI, DF,
               DGO,  GTO,  GRO,  HGO,  JAL, MEX, MCH, MOR,
               NAY,  NL, OAX, PUE, QRO, QR, SLP, SIN, SON,
               TAB, TAM, TLX, VER, YUC, ZAC)
VAR
  Norte, California,
  Centro Sur, Golfo, Peninsula      :byte;
  DelEstado                         :Estado;
BEGIN
  CASE DelEstado OF
      SON, CHI COH,
      DGO, NL, TAM                  :Norte     := Norte + 1;
      BCN, BCS                      :California := California + 1;
      ZAC, SLP, AGS,
      GTO, QRO, HGO,
      TLX, DF, PUE,
      MOR, MEX, VER                 :Centro    := Centro + 1;
      COL, MCH, GRO,
      OAX, CHP                      :Sur       := Sur + 1;
      VER, TAB                      :Golfo     := Golfo + 1;
      CAM, YUC, QR                  :Peninsula := Peninsula + 1;
  END;
END.
```

Lo más importante que debemos recordar en lo que se refiere a la lista de etiquetas, es que éstas deben ser constantes. Un valor en particular debe aparecer solamente una vez en dicha lista.

```
program Horno;
uses
  Crt;
const
  Abierta       = true;
  Cerrada       = false;
  ZonaMuerta    = 4;
  PuntoOp       = 12;
  LimSup        = PuntoOp + ZonaMuerta;
  LimInf        = PuntoOp-ZonaMuerta;
  InerciaTermica = 1;
var
  tiempo,
  Temperatura : integer;
  Valvula     : boolean;
```

```
procedure Ejes;
var
   Cont : byte;
begin
   gotoxy (1,24);
   for Cont := 1 to 80 do write ('-');
   for Cont := 1 to 24 do
   begin
      gotoxy (1,Cont);
      write ('|');
   end; { for }
   for Cont := 1 to 80 do
   begin
      gotoxy (Cont,LimSup);
      write ('-');
      gotoxy (Cont,PuntoOp);
      write ('-');
      gotoxy (Cont,LimInf);
      write ('-');
   end;
end; {-------------------------------------------------}


begin
   clrscr;
   Ejes;
   Temperatura := 0;
   Valvula := Cerrada;
   for tiempo: = 1 to 80 do
   begin
      if Temperatura  LimSup then Valvula := Cerrada;
      if Temperatura  LimInf then Valvula := Abierta;


      gotoxy (tiempo, 24-Temperatura);
      write ('*');
      sound (2000);
      delay (10);
      nosound;
      delay (200);
      case Valvula of
         Cerrada : Temperatura := Temperatura - InerciaTermica;
         Abierta : Temperatura := Temperatura + InerciaTermica;
      end; { case }
   end; { for }
end.
```


CICLOS

LAZOS FOR

En ocasiones es necesario ejecutar un estatuto varias veces con un rango de valores. El ejemplo más usado es un programa que genere una lista de las raíces cudradas de los números comprendidos entre uno y cien. Pascal provee de un estatuto que ejecuta el mismo código para cada valor, y continúa con el resto del programa cuando todos los casos han sido evaluados. Se llama lazo FOR, y es uno de los tres lazos disponibles en Pascal. Su funcionamiento es similar al FOR de BASIC.

La forma general del estatuto es la siguiente:

```
FOR variable de control := valor inicial TO valor final DO
    estatuto
```

Los valores iniciales y finales pueden ser expresiones. La variable de control es de cualquier tipo ordinal, incluyendo tipos enumerados.

Es necesario aclarar que el estatuto FOR no especifica que la variable de control "se incremente en uno". La variable de control se asigna al SIGUIENTE valor en su estado ordinal. Esto es muy importante cuando la variable de control es de tipo enumerado.

Imprimir las raíces es bastante fácil:

```
FOR I := 1 TO 100 DO
begin
      J := sqrt (I);
      writeln ('La raíz cuadrada de ',I,' es: ',J)
end;
```

El estatuto FOR, en su forma pura, incrementa la variable de control; esto es, asigna el SIGUIENTE valor "hacia arriba" del estado ordinal. Algunas veces es necesario hacerlo al revés: Que a la variable de control se le asigne el valor precedente "hacia abajo" (decremento) en el estado ordinal:

```
FOR I := 17 DOWNTO 7 DO estatuto;
FOR Color := Indigo DOWNTO Naranja DO estatuto;
FOR Ch := 'Z' DOWNTO 'X' DO estatuto;
```

LAZOS WHILE/DO

Como se ha visto, el lazo FOR ejecuta un estatuto un número determinado de veces; ni más ni menos. La variable de control no puede ser alterada durante la ejecución del lazo. Hay muchos casos en que el lazo se repite hasta que una condición dada ocurre. La variable de control debe ser cambiada durante la ejecución del lazo. Pascal ofrece dos maneras de construir este tipo de lazos: WHILE/DO y REPEAT/UNTIL. La forma general del lazo WHILE/DO es:

```
WHILE expresión booleana DO estatuto
```

Como en los lazos FOR, el estatuto puede ser simple o compuesto. Si el valor de la expresión es verdadero, entonces el estatuto es ejecutado.

Asumiendo que la expresión booleana sea verdadera la primera vez que se evalúa, se ejecuta el estatuto y, al terminar, se evalúa nuevamente la condición booleana. Si continúa verdadera, el estatuto se ejecuta de nuevo; si se encuentra falsa, el lazo termina, transfiriendo el control del programa al estatuto siguiente del lazo. Por ejemplo:

```
var
    pH : real;
begin
    LlenaTanque;                { llenar el tanque de agua }
    Muestrear (pH);             { tomar la medida de pH inicial }
    While pH < 7.2 do
    begin
        PonerAlcalino;          { mezclar sustancia en el tanque }
        AgitarTanque;           { agitar }
        Muestrear (pH);         { tomar lectura del sensor }
    end;
end.
```

Esta parte del código es de un sistema de control de un aparato de proceso químico. Todo lo que hace es llenar un tanque de agua y asegurarse de que el pH sea cuando menos de 7.2. Si el agua del tanque se vuelve muy ácida, el pH sube de valor y el agua no sería potable.

Primero se llena el tanque. Después se toma el valor inicial de pH. El agua puede ser potable desde el principio, en cuyo caso el lazo no se ejecutaría jamás. Pero si el agua es ácida, el lazo se ejecuta, añadiendo una cantidad de sustancia alcalina y tomando la muestra de pH otra vez. Si el pH sobrepasa el valor de 7.2, el lazo termina; de lo contrario, el lazo se ejecuta otra vez, añadiendo alcalinidad al agua y muestreando el pH una vez más.

La más importante propiedad del lazo WHILE/DO es que la expresión booleana se revisa ANTES de que el estatuto se ejecute, al contrario de lo que sucede con el lazo REPEAT/UNTIL.

LAZOS REPEAT/UNTIL

El lazo REPEAT/UNTIL es similar al lazo WHILE/DO. Como el otro, REPEAT/UNTIL ejecuta un estatuto hasta que la expresión booleana sea verdadera. La forma general es:

```
REPEAT estatuto UNTIL expresión booleana;
```

Lo más importante es que el estatuto se ejecuta cuando menos una vez. A diferencia del lazo WHILE/DO, no es necesario inicializar las variables. Es correcto inicializar las variables necesarias dentro del estatuto.

Otra cosa interesante es que las palabras reservadas REPEAT/UNTIL actúan como separadores de bloque, igual que lo harían las palabras BEGIN/END, de tal manera que el estatuto puede ser simple o compuesto, sin cambiar NADA de la sintaxis dada anteriormente.

Estos dos tipos de lazos son bastante similares, y es válido cuestionar cuál de los dos es el necesario en un momento dado y para una situación en particular. WHILE/DO puede hacer todo lo que hace REPEAT/UNTIL, sin embargo, en un pricipio puede ser confuso el uso de ellos. Recomendación: Use REPEAT/UNTIL en todos los casos en que el estatuto deba ejecutarse cuando menos una vez. Cuando codifique lazos, siempre pregúntese qué pasaría si el lazo no se ejecutara jamás; si no hay una respuesta coherente, codifique un REPEAT/UNTIL en vez de un WHILE/DO.

CAPITULO 4.- PROCEDIMIENTOS Y FUNCIONES

Mucha gente piensa que los lazos como WHILE/DO y REPEAT/UNTIL (y la innecesaria posición del GOTO) son la piedra angular de una programación estructurada, pero no hay tal. En el fondo, la programación estructurada se compone de detalles más sutiles. La capacidad humana de englobar un sistema complejo se ve disminuida rápidamente, a menos que se encuentre una estructura o patrón. Cuando se hace un programa en FORTRAN, por ejemplo, de más de 1000 líneas (lo cual puede tomar unas seis semanas de trabajo), y se intenta recordar cómo funciona exactamente la parte inicial de dicho programa, resalta la diferencia de la estructura que un lenguaje como Pascal puede ofrecer.

Uno de los detalles escondidos de la programación estructurada es la secuencia de código que ejecuta tareas discretas. Tales secuencias de código son llamadas "subprogramas".

En Pascal hay dos tipos de subprogramas: procedimientos y funciones. Estos subprogramas son secuencias de estatutos colocados fuera del cuerpo principal del programa. Ambos, para ser ejecutados, se invocan llamándolos simplemente por su nombre. La única diferencia entre ellos es que la función regresa un valor de un tipo declarado, mientras que los procedimientos no.

Los procedimientos y funciones son, en efecto, programas en miniatura. Pueden tener declaraciones de etiquetas, constantes, tipos, variables, procedimientos y funciones. Por supuesto que tienen estatutos de código. La única diferencia entre un procedimiento y un programa es la palabra PROGRAM y la puntuación final después del END.

```
PROCEDURE Mensaje (Linea : String80);
const
        Xpos = 50;
        Ypos = 24;
begin
        gotoxy (Xpos, Ypos);
        write (Linea);
end;
```

Las funciones son un poco diferentes. Una función tiene un tipo y toma un valor que regresa a la lógica del programa que la llamó:

```
FUNCTION Area (R : real) : real;
const
        Pi = 3.14159;
begin
        Area := Pi * R * R;
end;
```

El tipo de la función Area es real. Como se puede ver en la única línea de código de la función, se evalúa una expresión con el valor del radio R y el valor asignado al nombre de la función. Fuera de esas dos distinciones, los procedimientos y las funciones son iguales.

Asumiendo que la expresión booleana sea verdadera la primera vez que se evalúa, se ejecuta el estatuto y, al terminar, se evalúa nuevamente la condición booleana. Si continúa verdadera, el estatuto se ejecuta de nuevo; si se encuentra falsa, el lazo termina, transfiriendo el control del programa al estatuto siguiente del lazo. Por ejemplo:

```
var
    pH : real;
begin
    LlenaTanque;                { llenar el tanque de agua }
    Muestrear (pH);             { tomar la medida de pH inicial }
    While pH < 7.2 do
    begin
        PonerAlcalino;          { mezclar sustancia en el tanque }
        AgitarTanque;           { agitar }
        Muestrear (pH);         { tomar lectura del sensor }
    end;
end.
```

Esta parte del código es de un sistema de control de un aparato de proceso químico. Todo lo que hace es llenar un tanque de agua y asegurarse de que el pH sea cuando menos de 7.2. Si el agua del tanque se vuelve muy ácida, el pH sube de valor y el agua no sería potable.

Primero se llena el tanque. Después se toma el valor inicial de pH. El agua puede ser potable desde el principio, en cuyo caso el lazo no se ejecutaría jamás. Pero si el agua es ácida, el lazo se ejecuta, añadiendo una cantidad de sustancia alcalina y tomando la muestra de pH otra vez. Si el pH sobrepasa el valor de 7.2, el lazo termina; de lo contrario, el lazo se ejecuta otra vez, añadiendo alcalinidad al agua y muestreando el pH una vez más.

La más importante propiedad del lazo WHILE/DO es que la expresión booleana se revisa ANTES de que el estatuto se ejecute, al contrario de lo que sucede con el lazo REPEAT/UNTIL.

LAZOS REPEAT/UNTIL

El lazo REPEAT/UNTIL es similar al lazo WHILE/DO. Como el otro, REPEAT/UNTIL ejecuta un estatuto hasta que la expresión booleana sea verdadera. La forma general es:

```
REPEAT estatuto UNTIL expresión booleana;
```

Lo más importante es que el estatuto se ejecuta cuando menos una vez. A diferencia del lazo WHILE/DO, no es necesario inicializar las variables. Es correcto inicializar las variables necesarias dentro del estatuto.

Otra cosa interesante es que las palabras reservadas REPEAT/UNTIL actúan como separadores de bloque, igual que lo harían las palabras BEGIN/END, de tal manera que el estatuto puede ser simple o compuesto, sin cambiar NADA de la sintaxis dada anteriormente.

Estos dos tipos de lazos son bastante similares, y es válido cuestionar cuál de los dos es el necesario en un momento dado y para una situación en particular. WHILE/DO puede hacer todo lo que hace REPEAT/UNTIL, sin embargo, en un pricipio puede ser confuso el uso de ellos. Recomendación: Use REPEAT/UNTIL en todos los casos en que el estatuto deba ejecutarse cuando menos una vez. Cuando codifique lazos, siempre pregúntese qué pasaría si el lazo no se ejecutara jamás; si no hay una respuesta coherente, codifique un REPEAT/UNTIL en vez de un WHILE/DO.

CAPITULO 4.- PROCEDIMIENTOS Y FUNCIONES

Mucha gente piensa que los lazos como WHILE/DO y REPEAT/UNTIL (y la innecesaria posición del GOTO) son la piedra angular de una programación estructurada, pero no hay tal. En el fondo, la programación estructurada se compone de detalles más sutiles. La capacidad humana de englobar un sistema complejo se ve disminuida rápidamente, a menos que se encuentre una estructura o patrón. Cuando se hace un programa en FORTRAN, por ejemplo, de más de 1000 líneas (lo cual puede tomar unas seis semanas de trabajo), y se intenta recordar cómo funciona exactamente la parte inicial de dicho programa, resalta la diferencia de la estructura que un lenguaje como Pascal puede ofrecer.

Uno de los detalles escondidos de la programación estructurada es la secuencia de código que ejecuta tareas discretas. Tales secuencias de código son llamadas "subprogramas".

En Pascal hay dos tipos de subprogramas: procedimientos y funciones. Estos subprogramas son secuencias de estatutos colocados fuera del cuerpo principal del programa. Ambos, para ser ejecutados, se invocan llamándolos simplemente por su nombre. La única diferencia entre ellos es que la función regresa un valor de un tipo declarado, mientras que los procedimientos no.

Los procedimientos y funciones son, en efecto, programas en miniatura. Pueden tener declaraciones de etiquetas, constantes, tipos, variables, procedimientos y funciones. Por supuesto que tienen estatutos de código. La única diferencia entre un procedimiento y un programa es la palabra PROGRAM y la puntuación final después del END.

```
PROCEDURE Mensaje (Linea : String80);
const
        Xpos = 50;
        Ypos = 24;
begin
        gotoxy (Xpos, Ypos);
        write (Linea);
end;
```

Las funciones son un poco diferentes. Una función tiene un tipo y toma un valor que regresa a la lógica del programa que la llamó:

```
FUNCTION Area (R : real) : real;
const
        Pi = 3.14159;
begin
        Area := Pi * R * R;
end;
```

El tipo de la función Area es real. Como se puede ver en la única línea de código de la función, se evalúa una expresión con el valor del radio R y el valor asignado al nombre de la función. Fuera de esas dos distinciones, los procedimientos y las funciones son iguales.

```
program Piramide;
uses
   Crt;

procedure HacerPiramide (Ch : char);
var
   I,j : char;
begin
   writeln ('Piramide de la ',Ch);
   for I: = 'A' to Ch do
   begin
      gotoxy (105-ord(I),wherey);
      for j: = 'A' to pred(I) do write (j);
      for j: = i downto 'A' do write (j);
      writeln;
   end;
   gotoxy (40,25);
   write ('Presione <Enter> para continuar...');
   readln;
end;


var
   Ch : char;
begin
   clrscr;
   repeat
      writeln ('** PIRAMIDE DE LETRAS **');
      writeln;
      writeln ('Utilice * para terminar...');
      writeln; writeln;
      Ch : = upcase(readkey);
      if Ch in ['A'..'Z'] then HacerPiramide (Ch);
      clrscr;
   until Ch = '*';
end.
```



```
program ValidaNumeros;
uses
   Crt;


function Digito (Numero : word; Posicion : byte) : integer;
var
   NumStr : string;
begin
   str (Numero,NumStr);
   if length (NumStr)  Posicion then Digito : = -1
   else Digito : = ord(NumStr[Posicion])-48;
end;

var
   Numero   : word;
   Posicion  : byte;
   R         : integer;


begin
   clrscr;
   repeat
      writeln;
      write ('Dame el numero  : '); readln (Numero);
      write ('Dame la posicion: '); readln (Posicion);
      R : = Digito(Numero,Posicion);
      if R = -1 then writeln ('**ERROR**')
      else writeln ('El digito #',Posicion,' de el numero es ',R);
   until R = -1
end.
```

PARAMETROS FORMALES Y ACTUALES

Pasar datos a los procedimientos y a las funciones puede hacerse de dos formas:

1) usando variables globales para que cualquier procedimiento y/o función pueda leerse.

2) a través de una lista de parámetros para cada procedimiento y/o función.

La primera es una muy mala idea, por no mencionar las limitantes que el hardware pueda imponer. Es una buena práctica que un procedimiento tenga solamente que manejar una lista de valores.

Los parámetros definidos en la declaración del procedimiento se llaman "parámetros formales". Los valores que están presentes en una lista de parámetros de la invocación en particular se llaman "parámetros actuales". Si los parámetros son pasados por referencia, los parámetros actuales deberán tener variables de tipos iguales para guardar en ellos los parámetros formales.

TRANSFERENCIA DE DATOS POR VALOR Y REFERENCIA

PARAMETROS PASADOS POR VALOR

Un parámetro pasado por valor es justamente eso: Un valor copiado del parámetro actual al parámetro formal. El movimiento del valor hacia el procedimiento es en un sentido. Nada regresa ni puede ser usado por el programa que lo llamó.

Hay ventajas poderosas en el movimiento de valores dentro del procedimiento. El procedimiento puede guardar, modificar y mutilar el parámetro en la medida que las necesidades del procedimiento lo exijan, y no existen efectos colaterales fuera del procedimiento. La copia del parámetro actual es una copia privada, estrictamente local para el mismo procedimiento.

La forma de declarar pase de valores por valor es:
PROCEDURE (variable : tipo);

PARAMETROS PASADOS POR REFERENCIA

En muchas ocasiones el punto central de pasar un parámetro a un procedimiento o función es tener ese mismo, pero modificado para su uso. Para que un procedimiento pueda modificar un parámetro y regresarlo así, éste debe pasarse por referencia.

A diferencia de los parámetros pasados por valor, los valores pasados por referencia NO pueden ser literales, constantes o expresiones. Los valores de constantes y literales, por definición, no pueden ser cambiados.

Para pasar un valor por referencia, un parámetro actual debe ser una variable del mismo tipo que el parámetro formal. No hay tipos compatibles; deben ser exactamente del mismo tipo. La única excepción a esta regla son los tipos string, los cuales pueden ser definidos por una longitud de hasta 255.

La forma para definir valores pasados por referencia es:
PROCEDURE (VAR variable : tipo);

CAPITULO 5 ARREGLOS (Arrays)

Las variables que se han venido manejando hasta ahora son todas tipos simples. Iniciaremos ahora el estudio de algunos tipos estructurados que posee Pascal. Se puede decir que un tipo estructurado difiere de un tipo simple en que el tipo estructurado posee mas de un componente. Y cada componente del tipo estructurado es una variable, la cual a su vez puede ser de tipo estructurado o simple. Lo importante con este tipo de variables es la forma en como se tiene acceso a cada uno de sus componentes.

ARREGLO DE UNA DIMENSION (VECTORES)

Un arreglo no es mas que una colección de variables, en donde todas son del mismo tipo. Se puede decir que una línea de texto es un arreglo de caracteres; un vector se puede representar como un arreglo de números; una matriz se puede pensar como un arreglo, cuyos elementos son a su vez vectores.

Un arreglo se declara en términos de un índice y un tipo base, o sea, el tipo de cada uno de sus componentes.
Visto de otra forma, podemos decir que un arreglo nos va a permitir:

- Denotar un grupo de variables mediante un solo identificador.
- Poder distinguir un elemento en particular del grupo mediante un índice. De esta forma podemos referenciar el iésimo componente del arreglo.

El índice podrá ser cualquier tipo ordinal. Así, por ejemplo, una declaración del tipo array podrá quedar de la siguiente forma:

```
Numeros : array[1..100] of real;
```

Esta declaración permite disponer de un arreglo de 100 elementos. En este caso el índice es del tipo entero y el tipo de cada elemento es real.

Según se mencionó antes, el índice no es necesariamente numérico; puede ser cualquier tipo ordinal:

```
Grupo : array['A'..'G'] of integer;
```

Un elemento individual de un arreglo variable se denotará como una variable indexada, escribiendo el nombre (identificador) del arreglo seguido por el correspondiente valor del índice, enmarcado entre corchetes cuadrados. Ejemplo:

```
Grupo['E']
```

está haciendo referencia a aquél elemento del arreglo Grupo cuyo subíndice es 'E', el cual tiene o puede tener almacenado un valor del tipo integer.

Con esta variable (Grupo['E']) es posible hacer todas las operaciones en las cuales el operando sea de tipo entero:

```
Acumulado := Grupo['A'] + Grupo['E'];

Prom      := Grupo['B'] + Grupo['C'] / 2;
```

PARAMETROS FORMALES Y ACTUALES

Pasar datos a los procedimientos y a las funciones puede hacerse de dos formas:

1) usando variables globales para que cualquier procedimiento y/o función pueda leerse.

2) a través de una lista de parámetros para cada procedimiento y/o función.

La primera es una muy mala idea, por no mencionar las limitantes que el hardware pueda imponer. Es una buena práctica que un procedimiento tenga solamente que manejar una lista de valores.

Los parámetros definidos en la declaración del procedimiento se llaman "parámetros formales". Los valores que están presentes en una lista de parámetros de la invocación en particular se llaman "parámetros actuales". Si los parámetros son pasados por referencia, los parámetros actuales deberán tener variables de tipos iguales para guardar en ellos los parámetros formales.

TRANSFERENCIA DE DATOS POR VALOR Y REFERENCIA

PARAMETROS PASADOS POR VALOR

Un parámetro pasado por valor es justamente eso: Un valor copiado del parámetro actual al parámetro formal. El movimiento del valor hacia el procedimiento es en un sentido. Nada regresa ni puede ser usado por el programa que lo llamó.

Hay ventajas poderosas en el movimiento de valores dentro del procedimiento. El procedimiento puede guardar, modificar y mutilar el parámetro en la medida que las necesidades del procedimiento lo exijan, y no existen efectos colaterales fuera del procedimiento. La copia del parámetro actual es una copia privada, estrictamente local para el mismo procedimiento.

La forma de declarar pase de valores por valor es:
PROCEDURE (variable : tipo);

PARAMETROS PASADOS POR REFERENCIA

En muchas ocasiones el punto central de pasar un parámetro a un procedimiento o función es tener ese mismo, pero modificado para su uso. Para que un procedimiento pueda modificar un parámetro y regresarlo así, éste debe pasarse por referencia.

A diferencia de los parámetros pasados por valor, los valores pasados por referencia NO pueden ser literales, constantes o expresiones. Los valores de constantes y literales, por definición, no pueden ser cambiados.

Para pasar un valor por referencia, un parámetro actual debe ser una variable del mismo tipo que el parámetro formal. No hay tipos compatibles; deben ser exactamente del mismo tipo. La única excepción a esta regla son los tipos string, los cuales pueden ser definidos por una longitud de hasta 255.

La forma para definir valores pasados por referencia es:
PROCEDURE (VAR variable : tipo);

CAPITULO 5 ARREGLOS (Arrays)

Las variables que se han venido manejando hasta ahora son todas tipos simples. Iniciaremos ahora el estudio de algunos tipos estructurados que posee Pascal. Se puede decir que un tipo estructurado difiere de un tipo simple en que el tipo estructurado posee mas de un componente. Y cada componente del tipo estructurado es una variable, la cual a su vez puede ser de tipo estructurado o simple. Lo importante con este tipo de variables es la forma en como se tiene acceso a cada uno de sus componentes.

ARREGLO DE UNA DIMENSION (VECTORES)

Un arreglo no es mas que una colección de variables, en donde todas son del mismo tipo. Se puede decir que una línea de texto es un arreglo de caracteres; un vector se puede representar como un arreglo de números; una matriz se puede pensar como un arreglo, cuyos elementos son a su vez vectores.

Un arreglo se declara en términos de un índice y un tipo base, o sea, el tipo de cada uno de sus componentes.
Visto de otra forma, podemos decir que un arreglo nos va a permitir:

- Denotar un grupo de variables mediante un solo identificador.
- Poder distinguir un elemento en particular del grupo mediante un índice. De esta forma podemos referenciar el iésimo componente del arreglo.

El índice podrá ser cualquier tipo ordinal. Así, por ejemplo, una declaración del tipo array podrá quedar de la siguiente forma:

```
Numeros : array[1..100] of real;
```

Esta declaración permite disponer de un arreglo de 100 elementos. En este caso el índice es del tipo entero y el tipo de cada elemento es real.

Según se mencionó antes, el índice no es necesariamente numérico; puede ser cualquier tipo ordinal:

```
Grupo : array['A'..'G'] of integer;
```

Un elemento individual de un arreglo variable se denotará como una variable indexada, escribiendo el nombre (identificador) del arreglo seguido por el correspondiente valor del índice, enmarcado entre corchetes cuadrados. Ejemplo:

```
Grupo['E']
```

está haciendo referencia a aquél elemento del arreglo Grupo cuyo subíndice es 'E', el cual tiene o puede tener almacenado un valor del tipo integer.

Con esta variable (Grupo['E']) es posible hacer todas las operaciones en las cuales el operando sea de tipo entero:

```
Acumulado := Grupo['A'] + Grupo['E'];

Prom      := Grupo['B'] + Grupo['C'] / 2;
```

Como lo deja entrever la definición, el índice dentro de una variable indexada no necesita ser una constante, bien puede ser una variable, e incluso una expresión, contal de que el tipo de la expresión y/o de la variable concuerde con el tipo del índice. De forma que si por ejemplo tenemos la siguiente declaración:

```
Numeros : array[1..100] of real;
```

podemos denotar el i-esimo elemento de la variable Numeros con una variable o una expresión de tipo entero, con tal de que su valor esté en el rango permitido para el índice (1-100), de forma que podrá escribirse:

```
Numeros[j];

Numeros[i+j];

Numeros[trunc(x*4.2/12) div 6];
```

Si el valor de j, i+j, trunc(x*4.2/12) div 6 cae fuera del rango 1-100, se estará tratando de hacer referencia hacia un elemento que no existe y el compilador marcará error.

En ocasiones necesitamos guardar durante la ejecución de un programa el contenido de un arreglo en otro arreglo. Turbo Pascal permite hacer lo siguiente:

```
Numeros1 := Numeros2;
```

lo cual sería equivalente a hacer lo siguiente:

```
Numero1[1] := Numeros2[1];

Numero1[2] := Numeros2[2];

Numero1[3] := Numeros2[3];

Numero1[100] := Numeros2[100];
```



```
program Orden1;
uses
  Crt;
var
  Temporal,
  N, i, j  : integer;
  Numero   : array[1..50] of integer;
begin
  clrscr;
  writeln ('** CAPTURA DE NUMEROS ENTEROS **');
  write ('Cuantos numeros? ');
  readln (N);
  for i:=1 to N do
  begin
     write ('Numero ',i:2,': ');
     readln (Numero[i])
  end;

  writeln;
  writeln ('** IMPRESION DE NUMEROS COMO SE CAPTURARON **');
  for i:=1 to N do writeln ('Numero ',i:2,' = ',Numero[i]);


  for i:=1 to (N-1) do
     for j:=(i+1) to N do
        if (Numero[i]Numero[j]) then
        begin
           Temporal  := Numero[i];
           Numero[i] := Numero[j];
           Numero[j] := Temporal;
        end;
  writeln;
  writeln ('** IMPRESION DE NUMEROS EN ORDEN **');
  for i:=1 to N do writeln ('Numero ',i:2,' = ',Numero[i])
end.
```

```
{
    Hay que ser cuidadoso con las mayusculas/minusculas, pues eso puede
    alterar el orden de los nombres. Se recomienda probar primero con
    puras mayusculas.
}

program Orden2;
uses
   Crt;
var
   N, i, j  : integer;
   Temporal : string;
   Nombre   : array[1..50] of string;
begin
   clrscr;
   writeln ('** CAPTURA DE NOMBRES **');
   write ('Cuantos nombres?');
   readln (N);
   for i:=1 to N do
   begin
      write ('Nombre ',i:2,': ');
      readln (Nombre[i])
   end;
   writeln;
   writeln ('** IMPRESION DE NOMBRES COMO SE CAPTURARON **');
   for i:=1 to N do writeln ('Nonbre ',i:2,' = ',Nombre[i]);

   for i:=1 to (N-1) do
      for j:=(i+1) to N do
         if (Nombre[i]Nombre[j]) then
         begin
            Temporal  := Nombre[i];
            Nombre[i] := Nombre[j];
            Nombre[j] := Temporal;
         end;
   writeln;
   writeln ('** IMPRESION DE NOMBRES EN ORDEN **');
   for i:=1 to N do writeln ('Nombre ',i:2,' = ',Nombre[i])
end.
```


ARREGLOS DE DOS DIMENSIONES (MATRICES)

En los ejemplos de los arreglos manejados hasta ahora sólo se han hecho uso de arreglos de una dimensión (sólo usan un subíndice). Sin embargo Pascal permite arreglos de más de una dimensión. Por ejemplo, cada una de las páginas de este escrito se puede considerar como un arreglo de renglones, en donde cada renglón consta de 65 caracteres, o sea, que se puede definir la variable página de la siguiente manera:

```
pagina : array[1..52,1..65] of char;
```

o bien:

```
pagina : array[1..52] of array[1..65] of char;
```

así, para el i-esimo caracter de la j-esima línea de PAGINA se puede referenciar:

```
pagina[i,j]
```

o bien:

```
pagina [i][j]
```

Cualquier arreglo definido con más de una dimensión se considera un arreglo multidimensional, aunque es muy raro que se exceda de tres dimensiones. Los arreglos bidimensionales, llamados matrices, son bastante comunes, especialmente en estadística e ingeniería. Por ejemplo, cuando se está midiendo la conductividad de un metal, una matriz se adapta perfectamente para llevar el registro de la temperatura con respecto a la conductividad medida:

```
Temp_Conductividad : array[1..200,1..2] of real;
```

La mejor manera de visualizar una matriz es pensar en una tabla con filas y columnas: el primer rango significa las filas (1..200) y el segundo las columnas (1..2).

Las asignaciones de valores pueden ser:

```
Temp_Conductividad [1,1] := -99.34;

Temp_Conductividad [1,2] := 12.3;
```

	1	2
	Temperatura	Conductividad
1	-99.34	12.3
2	-97.76	12.2
3	-96.01	11.9
.	.	.
.	.	.
200	99.01	2.9

Concentrándose en el manejo de las matrices, y dejando de lado el significado de cada localidad de ella, es posible manejarlas como cualquier sistema de ecuaciones y obtener sus resultados tales como el determinante, los elementos de las diagonales, resolver el sistema, etc. A continuación se presenta un programa que maneja los métodos para la solución de sistemas por medio de los algoritmos de Montante y Gauss-Jordan. Obsérvese el tratamiento tanto para la captura como la impresión de la matriz.

Los diagramas de flujo de los algoritmos se dejan de lado para concentrar la atención el el código.


```
Program Test;
uses
  Crt;
type
  rango  = 1..20;
  tipo   = real;
  Matriz = array[rango,rango] of tipo;


procedure CapturaMatriz (var M : Matriz; var Orden : rango);
var
  i,j : rango;
begin
  writeln; writeln;
  writeln ('** CAPTURA DE MATRIZ **');
  writeln;
  write ('Orden de la matriz: ');
  readln (Orden);
  for i:=1 to Orden do
      for j:=1 to Orden do
          begin
              write ('Elemento (',i,',',j,'): ');
              readln (M[i,j])
          end;
  writeln
end; {-------------------------------------------------}


procedure ImprimeMatriz (M : Matriz; Fil,Col : rango);
var
  i,j : rango;
begin
  writeln; writeln;
  writeln ('** MATRIZ **');
  writeln;
  for i:=1 to Fil do
  begin
      for j:=1 to Col do write (M[i,j]:10:3);
      writeln;
  end;
  writeln
end; {-------------------------------------------------}


procedure ConcatenaMatrizIdentidad (var M : Matriz; Orden : rango);
var
  i,j : rango;
begin
  for i:=Orden+1 to 2*Orden do
      for j:=1 to Orden do
          if (i-Orden=j) then M[j,i]:=1 else M[j,i]:=0;
end; {-------------------------------------------------}
```

```
procedure CambiaFilas (var M : Matriz; Orden,Fila,Pivote : rango);
var
   Factor : tipo;
   i      : rango;
begin
   Factor := -M[Fila,Pivote];
   for I:=1 to Orden do
      M[Fila,I] := Factor * M[Pivote,I] + M[Fila,I];
end; {-------------------------------------------------}


procedure TransformacionElemental (var M : Matriz; Fil, Col : rango);
var
   Diagonal,
   ColCero  : rango;
   Factor   : tipo;
begin
   for Diagonal:=1 to Fil do
      if (M[Diagonal,Diagonal]0) then
      begin
         Factor := M[Diagonal,Diagonal];
         for ColCero:=1 to Col do
            M[Diagonal,ColCero] := M[Diagonal,ColCero]/Factor;
         ImprimeMatriz (M,Fil,Col);
         readln;
         for ColCero:=1 to Fil do
            if (ColCeroDiagonal) then
               CambiaFilas(M,Col,ColCero,Diagonal);
      end;
end; {-------------------------------------------------}


var
   UnaMatriz : Matriz;
   N         : rango;
begin { main }
   CapturaMatriz (UnaMatriz,N);
   ConcatenaMatrizIdentidad (UnaMatriz,N);
   ImprimeMatriz (UnaMatriz,N,2*N);
   TransformacionElemental (UnaMatriz,N,N*2);
   ImprimeMatriz (UnaMatriz,N,2*N);
   writeln ( 'Presione <Enter> para continuar ...');
end..
```



```
{
            MANEJO DE OPERACIONES ELEMENTALES CON MATRICES
}

program ManejoDeMatrices;
uses
   Crt;
const
   MAX = 20;
type
   rango  = 1..MAX;
   tipo   = real;
   Matriz = array [rango,rango] of tipo;
var
   MatrizA,
   MatrizB,
   MatrizC  : Matriz;

   Fil_A, Fil_B, Fil_C,
   Col_A, Col_B, Col_C  : rango;

   DeterminanteA,
   DeterminanteB,
   DeterminanteC : real;


procedure Captura (var A : Matriz; var Fil,Col : rango);
var
   I,J : rango;
begin
   clrscr;
   write ('Numero de filas    : '); readln (Fil);
   write ('Numero de Columnas : '); readln (Col);
   for i:=1 to Fil do
      for j:=1 to Col do
      begin
         write ('Elemento (',I:2,',',J:2,'): ');
         readln (A[I,J])
      end
end; {-------------------------------------------------}


procedure ImprimeMatriz (A : Matriz; Fil,Col : rango);
var
   I,J : rango;
begin
   writeln;
   writeln ('** MATRIZ **');
   for I:=1 to Fil do
   begin
      for J:=1 to Col do write (A[I,J]:10:3);
      writeln;
   end;
end; {-------------------------------------------------}
```

```
function MultiplicaMatriz (   A : Matriz;    Fil_A,    Col_A : rango;
                              B : Matriz;    Fil_B,    Col_B : rango;
                           var R : Matriz; var Fil_R : rango; var Col_R : rango) : boolean;
var
   i,j,k,n : rango;
 begin
   MultiplicaMatriz := false;
   if (Col_A = Fil_B) then
   begin
      N := Col_A;
      MultiplicaMatriz := true;
      Fil_R := Fil_A;
      Col_R := Col_B;
      for i:=1 to Fil_R do
        for j:=1 to Col_R do
        begin
           R[i,j] := 0;
           for k:=1 to N do R[i,j] := A[i,k] * B[k,j] + R[i,j]
        end { for j }
   end { if }
end; {-------------------------------------------------}


procedure ImprimeMatrizAdyacente (A : Matriz; Fil,Col : rango);
var
   i,j : rango;
begin
   writeln;
   writeln ('** MATRIZ **');
   for i:=1 to Fil do
   begin
      for j:=1 to (Col*2) do write (A[i,j]:10:3);
      writeln;
   end;
end; {-------------------------------------------------}


procedure AgregaMatrizIdentidad (var A : Matriz; N : rango);
var
   i,j : rango;
begin
   for i:=1 to N do
      for j:=N+1 to (N*2) do
         if ((i+N)=j) then A[i,j] := 1 else A[i,j] := 0;
end; {-------------------------------------------------}
```



```
procedure Montante (var A : Matriz; var Deter : real; N : rango);
var
   i,j,k,
   M      : rango;
   Factor,
   PivAnt,
   Pivote : tipo;
begin
   AgregaMatrizIdentidad (A,N);
   PivAnt := 1;
   for i:=1 to N do
   begin
      Pivote := A[i,i];
      for k:=1 to N do
         if (ik) then
         begin
            Factor := A[k,i];
            for j:=1 to (N*2) do
               A[k,j]:= (A[k,j]*Pivote - A[i,j]*Factor) / PivAnt;
         end; { if }
      PivAnt := Pivote;
   end; { for }
   Deter := Pivote;
end; {-------------------------------------------------}


function CalculaDeterminante (A : Matriz; N : rango) : real;
var
   DeterAux  : real;
   MatrizAux : Matriz;
begin
   MatrizAux := A;
   Montante (MatrizAux,DeterAux,N);
   CalculaDeterminante := DeterAux;
end; {-------------------------------------------------}


begin
   Captura (MatrizA, Fil_A, Col_A);
   Captura (MatrizB, Fil_B, Col_B);
   ImprimeMatriz (MatrizA, Fil_A, Col_A);
   ImprimeMatriz (MatrizB, Fil_B, Col_B);
   if MultiplicaMatriz(MatrizA, Fil_A, Col_A,
                       MatrizB, Fil_B, Col_B,
                       MatrizC, Fil_C, Col_C) then ImprimeMatriz (MatrizC, Fil_C, Col_C);
   writeln ('Determinante calculado por funcion  : ',CalculaDeterminante(MatrizC,Fil_C):10:3);
   Montante (MatrizC,DeterminanteC,Fil_C);
   writeln ('Determinante calculado por Montante : ',DeterminanteC:10:3);
   ImprimeMatrizAdyacente (MatrizC, Fil_C, Col_C);
end.
```

CAPITULO 6.- ARCHIVOS

Los archivos de pascal, independientemente de su tipo, tienen características comunes. A priori, todos los archivos se utilizan tanto para entrada como para salida de datos. Se le llama "entrada" a la operación que ejecuta un programa cuando toma datos de un archivo y los utiliza en sus cálculos; se le llama "salida" cuando los resultados del proceso se envían a un archivo.

Los archivos pueden ser grabados en discos flexibles o en discos duros. También hay otros medios de almacenamiento: cintas, discos ópticos, discos en RAM, etc...; pero son menos comunes.

El DOS requiere que los nombres de los archivos tengan de 1 a 8 caracteres; puede incluirse una extensión de 1 a 3 caracteres, que sirve para ayudar a describir el contenido del archivo.

ARCHIVOS DE TEXTO

Los archivos de texto constan de líneas terminadas por un retorno de carro (carriage return), de una línea (linefeed CR/LF) y de los caracteres que contiene (palabras, oraciones, números,etc.).

Nombre del archivo: Texto.dat

Línea 1.- Este es un ejemplo de texto. [CR/LF]
Línea 2.- Cada línea de texto termina con [CR/LF].
Línea 3.- Retorno de carro y un alimentador de línea.[CR/LF]
Línea 4.- [CR/LF]
Línea 5.- Aun las líneas vacías, como la anterior. [CR/LF]
Línea 6.- 50 12.23 40343 332324 [CR/LF]

Antes de empezar a trabajar con archivos de texto, se debe declarar un IDENTIFICADOR DE ARCHIVO DE TEXTO en el programa. Los identificadores para los archivos de texto se declaran como cualquier identificador, utilizando la palabra reservada en Turbo Pascal TEXT. Ejemplo:

```
Program ArchivoTexto;

var

  Arch : Text;
```

El identificador del archivo es Arch (de tipo Text). Antes de utilizar el archivo para Entrada o Salida es necesario asignar la variable a un nombre de archivo en el disco; a esta operación se le llama asignación:

```
assign (Arch, 'TEXTO.DAT');
```

Una vez que el archivo se asignó, no vuelve a ser referenciado por su nomber otra vez.

Después de asignar el nombre del archivo, es necesario prepararlo para la operación que se desa hacer con él : RESET, REWRITE o APPEND.

Reset "abre" el archivo y lo prepara para lectura; solo se pueden utilizar comandos de entrada cuando Reset se utiliza. Cualquier intento de escriturá generará un error de I/O.

El comando Reset posiciona al inicio el apuntador de archivo, (el cual es un contador que mantiene la posición del programa en el archivo). Esto hace que las operaciones se empiecen desde el inicio del archivo.

Intentar aplicar Reset a un archivo que no existe genera un error de I/O.

Rewrite y Append preparan el archivo para salida (escritura), pero funcionan de diferente manera. Cuando se prepara un archivo ya existente con Rewrite, se borra su contenido y el apuntador de archivo se posiiona al inicio de este. Si Rewrite se utiliza con un archivo que no existe, se crea uno nuevo con el nombre utilizado en el comando Assign.

Por otro lado, el comando Append preserva el contenido del archivo y posiciona el apuntador de archivo al final de éste. Como resultado, cualquier dato añadido al archivo se escribe al final, junto con lo que ya había.

Cuando se termina de utilizar el archivo, es necesario "cerrarlo". El comando Close cierra el archivo y se asegura que todos los datos almacenados en el buffer temporal se almacenen en el disco (a esto se le conoce como FLUSH).

Una vez cerrado, un archivo no puede utilizarse ni para entrada, ni para salida, hasta que sea abierto con Reset, Rewrite o Append. Sin embargo, el encadenamiento con el sistema operativo permanece, de tal forma que para volver a abrir el archivo no es necesario utilizar el comando Assign.

LEYENDO UN STRING DE ARCHIVOS DE TEXTO

Una vez que un archivo de texto se abrió para lectura, se puede extraer información de él con los procedimientos Read y Readln;

```
Program Text1;
Var
        ArchTxt      : Text;
        [S]          :string[10];
begin
        assign (ArchTxt,'TEST.DAT');
        reset (ArchTxt);
        readln (ArchTxt,S);
        writeln(S);
        close (ArchTxt);
end.
```

ArchTxt se prepara para lectura con el comando Reset. La primera operación de entrada en el programa es el estatuto.

```
Readln(ArchTxt,S);
```

Este le dice a Turbo Pascal que lea los caracteres de la línea actual en el archivo y los coloque en la variable S. Después de que los caracteres sean leídos, el apuntador de archivo brinca los caracteres remanentes de la línea y va hacia el inicio de la siguiente.

Cuando se lee un string de un archivo de texto con el procedimiento Readln, pueden ocurrir tres cosas:

1) Que haya en la línea exactamente los caracteres necesarios para llenar la variable en toda su longitud.

2) Que no haya suficientes caracteres en la línea para llenar la variable de lectura.

3) Que haya demasiados caracteres en la línea para la longitud de la variable de lectura.

En los primeros dos casos, Turbo Pascal lee todos los caracteres de la línea, asignándolos a S y moviendo el apuntador al inicio de la siguiente línea. La longitud del string S es igual al número de caracteres leído.

En el tercer caso, Turbo Pascal lee tantos caracteres como sea necesario para llenar la variable de lectura completamente, y después mueve el apuntador del archivo a la siguiente línea. Todos los caracteres sobrantes en la línea son descartados.

El procedimiento Read opera como el Readln, pero después de leer al string, Read coloca el apuntador del archivo enseguida del último caracter leído; no mueve el apuntador al inicio de la siguiente línea. Si el procedimiento Read encuentra un [CR/LF], indicando que se encontró al final de la línea, detiene la lectura de caracteres y no avanza el apuntador de archivo hasta que se utilice un procedimiento Readln.

LEYENDO VARIOS STRINGS POR LINEA

Un procedimiento Readln sencillo puede leer varios strings a la vez. Por ejemplo, el estatuto readln (ArchTxt,S1,S2,S3) lee caracteres de la línea actual, llenando las variables S1, S2 y S3, en ese orden. Si la línea a leer es:

" Esta es una línea de caracteres "

y las variables srtings son todas del tipo string[5], los strings asignados serían:

[Esta]	[es un]	[a lín]	[ea de caracteres]
S1	S2	S3	sin usar
S1	S2	S3	sin usar

LEYENDO NUMEROS DE LOS ARCHIVOS DE TEXTO

Los archivos de texto no solamente pueden almacenar strings o sentencias, sino también datos numéricos. Los números, sin embargo, no se almacenan en el archivo en forma binaria, sino como caracteres. Por ejemplo, en RAM el valor entero 20,545 es almacenado en 2 bytes con el valor binario 0101 0000 0100 0001. Pero en un archivo de texto, el número se almacena con los caracteres "2", "0", "5", "4", "5", requiriendo un total de 5 bytes. Cuando se lee un número de un archivo de texto, Turbo Pascal convierte el número de un string de caracteres a formato binario de enteros.

Asimismo, cuando se lee un número de un archivo de texto, Turbo Pascal brinca los caracteres de espacio de la línea hasta que encuentra un caracter no-espacio. Entonces lee los caracteres siguientes hasta que encuentra un caracter de espacio otra vez. De esta manera sigue leyendo hasta que encuentra un [CR/LF]. Turbo Pascal combina los caracteres leídos en un string que convierte a un valor entero o real, dependiendo del tipo de variable que se use. Si la conversión tiene éxito, el número resultante se asigna a la variable; si no, se genera un error de I/O.

Para observar en detalle cómo leer datos numéricos de un archivo de texto, examinemos el archivo de texto TEST.DAT.

11	27.53	6.4144900000E+02
21	50.83	1.1843390000E+03
31	74.13	1.2547890000E+03
41	97.43	2.1259870000E+03
51	120.73	2.8139002400E+03
61	144.03	3.8654741700E+03
71	167.33	4.4412450000E+03
81	190.63	4.9852140000E+03
91	213.93	5.0214578825E+03
101	273.23	5.5274590000E+03

Este archivo contiene 3 columnas de números. La primera columna es de enteros, la segunda de reales en formato decimal y la tercera de reales en notación científica. Se pueden leer 3 números por línea, empleando los siguientes estatutos:

```
Read  (ArchTxt,I2);
Read  (ArchTxt,R1);
Read  (ArchTxt,R2);
```

o bien usando el estatuto equivalente:

```
Readln (ArchTxt,I2,R1,R2);
```

Turbo Pascal asigna el primer número encontrado en la línea a la variable entera I2; y los siguientes 2 números a las variables reales R1 y R2.

El siguiente programa contiene una rutina que lee los datos numéricos contenidos en el archivo de texto TEST.DAT, y calcula el promedio de cada columna.

```
program CalculaPromedios;
var:
      f              :Text;
      I, count       : integer;
      imean,
      r1, r2,
      r1mean,
      r2mean         : real;
begin
      assign (f,'TEST.DAT');
      reset (f);
      count          := 0;
      imean          := 0;
      r1mean         := 0;
      r2mean         := 0;
      while not eof(f) dp
      begin
            readln (f,I, r1, r2);
            writeln (I:10,' ',r1:10:3,' ',r2:10:3);
            count          := count + 1;
```

```
            imean      := imean + i;
            r1mean     := r1mean + r1;
            r2mean     := r2mean + r2;
        end;
        imean := imean / count;
        r1mean := r1mean / count;
        r2mean := r2mean / count;
        writeln;
        writeln (imean:10:3,' ', r1mean:10:3,' ',  r2mean:10:3);
        close (f);
    end.
```

Si el formato de un número leído de un archivo de texto es incorrecto, el programa produce un error de I/O.

Por ejemplo, leer el número 50,000 con una variable del tipo INTEGER causa error, ya que el número más grande permitido para ese tipo es de 32,767. Similarmente, leer el número 32.1 en una variable de tipo INTEGER causa error por el punto decimal (es ilegal para el tipo Integer). El tipo REAL tiene menos restricciones, ya que estos números pueden ser leídos con o sin punto decimal y en notación normal o científica.

APENDICE A.-REFERENCIA COMPLETA DE FUNCIONES

La definición de ISO Standar Pascal incluye un número de "funciones estándar" construidas dentro del lenguaje, y no necesitan ser declaradas ni codificadas dentro del programa fuente. Esas funciones caen dentro de dos grupos básicos: 1) Las funciones matemáticas, las cuales proveen operaciones fundamentales, como la raíz cuadrada, el valor absoluto, los logaritmos naturales, las funciones trigonométricas; y 2) las funciones de transferencia, las cuales permiten la relación entre tipos incompatibles, como enteros y de caracter.

Muchos libros refieren los valores pasados a una función estándar como "argumentos". Este término es común dentro del argot matemático, por lo que puede ser confundido con el término parámetro. No hay diferencia mas que de término.

Turbo Pascal implementa todas las funciones del ISO Standar Pascal. A continuación se muestra un glosario completo. También se han añadido las propias del IBM PC.

```
            imean      := imean + i;
            r1mean     := r1mean + r1;
            r2mean     := r2mean + r2;
        end;
        imean := imean / count;
        r1mean := r1mean / count;
        r2mean := r2mean / count;
        writeln;
        writeln (imean:10:3,' ', r1mean:10:3,' ',  r2mean:10:3);
        close (f);
    end.
```

Si el formato de un número leído de un archivo de texto es incorrecto, el programa produce un error de I/O.

Por ejemplo, leer el número 50,000 con una variable del tipo INTEGER causa error, ya que el número más grande permitido para ese tipo es de 32,767. Similarmente, leer el número 32.1 en una variable de tipo INTEGER causa error por el punto decimal (es ilegal para el tipo Integer). El tipo REAL tiene menos restricciones, ya que estos números pueden ser leídos con o sin punto decimal y en notación normal o científica.

APENDICE A.-REFERENCIA COMPLETA DE FUNCIONES

La definición de ISO Standar Pascal incluye un número de "funciones estándar" construidas dentro del lenguaje, y no necesitan ser declaradas ni codificadas dentro del programa fuente. Esas funciones caen dentro de dos grupos básicos: 1) Las funciones matemáticas, las cuales proveen operaciones fundamentales, como la raíz cuadrada, el valor absoluto, los logaritmos naturales, las funciones trigonométricas; y 2) las funciones de transferencia, las cuales permiten la relación entre tipos incompatibles, como enteros y de caracter.

Muchos libros refieren los valores pasados a una función estándar como "argumentos". Este término es común dentro del argot matemático, por lo que puede ser confundido con el término parámetro. No hay diferencia mas que de término.

Turbo Pascal implementa todas las funciones del ISO Standar Pascal. A continuación se muestra un glosario completo. También se han añadido las propias del IBM PC.

ABS

SINTAXIS: function Abs (r:real): real;

function Abs (I:integer): integer;

DESCRIPCION: Abs regrese el valor absoluto del argumento. El resultado de la funcion es del mismo tipo (real o integer) como el parámetro.

Addr

SINTAXIS: function Addr (Var Variable):Pointer;

DESCRIPCION:Addr regresa la dirección de una variable, type, constante o procedimiento. El resultado es un tipo Pointer .

Append

SINTAXIS: procedure Append (Var F:Text);

DESCRIPCION Append abre un archivo de texto de escritura y posiciona el apuntador de archivo al final de este.

Arc [GRAPH UNIT]

SINTAXIS: procedure Arc(x,y:integer; StAngle, EndAngle, Radius:word);

DESCRIPCION:Arc dibuja un círculo alrededor de las coordenadas x:y con un radio de Radius.El círculo empieza a dibujarse en sentido de las manecillas del reloj desde StAngle y termina en EndAngle.

ArcTan

SINTAXIS: function ArcTan(R:real):real;

DESCRIPCION:ArcTan regresa el arctangente del parámetro.

Assign

SINTAXIS: procedure Assign(Var F:File;Name:string);

DESCRIPCION:Assign une el archivo de la variable F al archivo llamado Name.

AssignCrt [CRT UNIT]

SINTAXIS: procedure AssignCrt(Var F: Text);

DESCRIPCION:AssignCrt permite al usuario direccionar la salida a la pantalla escribiendo al archivo F.

Bar [GRAPH UNIT]

SINTAXIS: procedure Bar(x1,y1,x2,y2:integer);

DESCRIPCION:Bar llena el área del rectángulo de la pantalla.

Bar3D [GRAPH UNIT]

SINTAXIS: procedure Bar3D(x1,y1,x2,y2:integer; Depth:word; Top:boolean);

DESCRIPCION:Bar3D llena en tres dimensiones el área del rectángulo de la pantalla. La dimension de la profundidad se especifica con Depth. Si Top es TRUE,el procedimiento dibuja en tercera dimension en la parte superior del rectángulo.

BlockRead

SINTAXIS: procedure BlockRead(Var F:File; Var B:Type;
NumRecs:integer;Var RecsRead:integer);

DESCRIPCION:BlockRead intenta leer NumRecs registros del archivo F sin tipo en el buffer B.RecsRead indica el número de los registros ya leídos.Note que el parámetro RecsRead es solo usado en las versiones PC/MS-DOS.

BlockWrite

SINTAXIS: procedure BlockWrite(Var F:File; Var B:Type; NumRecs:integer);

DESCRIPCION:BlockWrite escribe los NumRecs registros desde el buffer B al archivo sin tipo F.

ChDir

SINTAXIS: procedure ChDir(S:string);

DESCRIPCION:ChDir cambia el directorio actual a S.

Chr

SINTAXIS: function Chr(I:integer);

DESCRIPCION:Chr regresa el caracter ASCII que corresponde a I.

Circle [GRAPH UNIT]

SINTAXIS: procedure circle(x,y:integer; Radius:word);

DESCRIPCION:Circle dibuja un círculo con un radio de Radius alrededor de la coordenada x:y.

ClearDevice [GRAPH UNIT]

SINTAXIS: procedure ClearDevice;

DESCRIPCION:ClearDevice borra la pantalla de las gráficas.

ClearViewPort [GRAPH UNIT]

SINTAXIS: procedure Clear View Port;

DESCRIPCION:Borra la pantalla actual.

Close

SINTAXIS: procedure Close (var F : File);

DESCRIPCION:Close drena el buffer del archivo F y entonces lo cierra.

CloseGraph [GRAPH UNIT]

SINTAXIS: procedure CloseGraph;

DESCRIPCION:Reestablece el video en el modo anterior al modo gráfico actual. El procedimiento tambien libera la memoria usado por el sistema de gráficos.

ClrEol [CRT UNIT]

SINTAXIS: procedure ClrEol;

DESCRIPCION:Limpia la pantalla desde la posición del cursor al final de la línea en la pantalla.

ClrScr [CRT UNIT]

SINTAXIS: procedure ClrScr;

DESCRIPCION:Limpia la pantalla y posiciona el cursor en la esquina superior izquierda (localización 1,1).

Concat

SINTAXIS: function Concat (S1,S2,...,Sn) : string;

DESCRIPCION:Combina cualquier número de strings y los regresa como uno solo. Si la longitud de la concatenación del string es mayor que 255, Turbo Pascal genera un Run-Time error.

Copy

SINTAXIS: function Copy (S : string; P,L : integer) : string;

DESCRIPCION:Regresa una porción del string S el cual empieza con el caracter número P y tiene L caracteres.

Cos

SINTAXIS: function Cos (R : real) : real;

DESCRIPCION:Regresa el coseno de R.

Cseg

SINTAXIS: function Cseg : word;

DESCRIPCION:Regresa la dirección del segmento del código del programa.

Dec

SINTAXIS: procedure Dec (var X : escalar; N : longint);

DESCRIPCION:Decrementa la variable X por N y si se omite N, X se decrementara en uno.

Delay [CRT UNIT]

SINTAXIS: procedure Delay (MS : word);

DESCRIPCION:Detiene la ejecución del progama por MS milisegundos.

Delete

SINTAXIS: procedure Delete (S : string; P,L : integer);

DESCRIPCION:Quita los L caracteres desde el string S empezando con el caracter número P.

DelLine [CRT UNIT]

SINTAXIS: procedure DelLine;

DESCRIPCION:Borra la línea de la pantalla en donde el cursor esta localizado. Las líneas que se encuentren abajo de ésta, se recorren.

DetectGraph [GRAPH UNIT]

SINTAXIS: procedure DetectGraph (var Gd,Gm : integer);

DESCRIPCION:Regresa el driver del tipo de gráficas detectado (Gd) y el modo gráfico (Gm) para la instalación adaptada.

DiskFree [DOS UNIT]

SINTAXIS: function DiskFree (Drive : word) : longint;

DESCRIPCION:Regresa la cantidad de espacio libre en disco en bytes en el drive (0 = default, 1 = A, 2 = B, etc).

DiskSize [DOS UNIT]

SINTAXIS: function DiskSize (Drive : word) : longint;

DESCRIPCION:Regresa el tamaño en bytes del disco en el drive (0 = default, 1 = A, 2 = B, etc).

Dispose

SINTAXIS: procedure Dispose (P : pointer);

DESCRIPCION:Libera la memoria del HEAP en donde se localiza la variable pointer.

DosExitCode [DOS UNIT]

SINTAXIS: function DosExitCode : word;

DESCRIPCION:Regresa el código de salida de un subproceso (0 = terminación normal, 1 = terminación por Ctrl-C, 2 = terminación por error de dispositivo, 3 = terminación por procedimiento Keep).

DosVersion [DOS UNIT]

SINTAXIS: function DosVersion : word;

DESCRIPCION:Regresa la versión del sistema operativo. El digito mayor en el byte mas significativo y el menor en el menos significativo.

DrawPoly [GRAPH UNIT]

SINTAXIS: procedure DrawPoly (NumPoints : word; var PolyPoints);

DESCRIPCION:Dibuja un polígono definido por el número de puntos NumPoints. El array PolyPoints contiene las coordenadas de los puntos del polígono.

Dseg

SINTAXIS: function Dseg : word;

DESCRIPCION:Dseg regresa la dirección del segmento de datos del programa.

EnvCount [DOS UNIT]

SINTAXIS: function EnvCount : integer;

DESCRIPCION:Regresa el número de strings definidas en el ambiente del DOS.

EnvStr [DOS UNIT]

SINTAXIS: function EnvStr (i : integer) : string;

DESCRIPCION:Regresa el string número i del ambiente DOS.

Eof

SINTAXIS: function Eof (F : File) : boolean;

DESCRIPCION:Regresa TRUE cuando el apuntador del archivo F encuentra el final de este.

Eoln

SINTAXIS: function Eoln(F : File) : boolean;

DESCRIPCION:Regresa TRUE cuando el apuntador del archivo F encuentra el final de una línea (marcado con los caracteres de retorno de carro y alimentador de línea) o el final del archivo.

Erase

SINTAXIS: procedure Erase (F : File);

DESCRIPCION:Borra de disco el archivo F y quita la información del directorio.

Exec [DOS UNIT]

SINTAXIS: procedure Exec (Path, CmdLine : string);

DESCRIPCION:Ejecuta el archivo especificado en Path con los párametros de línea definidos en CmdLine.

Exit

SINTAXIS: procedure Exit;

DESCRIPCION:Causa la terminación abrupta del bloque que esta siendo ejecutado.

Exp

SINTAXIS: function Exp (R : real) : real;

DESCRIPCION:Regresa el logaritmo base E de R.

FExpand [DOS UNIT]

SINTAXIS: function FExpand (P : PathStr);

DESCRIPCION:Acepta el nombre del archivo P, y regresa el nombre completo con el Path incluyendo el drive.

FilePos

SINTAXIS: function FilePos (F : File) : integer;

DESCRIPCION:Regresa el número de registro hacia donde apunta el apuntador del archivo F.

FileSize

SINTAXIS: function FileSize (F : File) : integer;

DESCRIPCION:Regresa el número de registros que contiene F.

FillChar

SINTAXIS: procedure FillChar (Variable : tipo; I,Code : escalar);

DESCRIPCION:Llena los I bytes de memoria con el valor del código Code, el cual puede ser de cualquier tipo escalar, empezando con la dirección de la variable.

FillEllipse [GRAPH UNIT]

SINTAXIS: procedure FillEllipse (x,y : integer; Xradio,Yradio : word);

DESCRIPCION:Dibuja un elipse centrado en las coordenadas x,y con un radio vertical Yradio y un radio horizontal Xradio. El elipse se rellena con el color y estilo de llenado actuales y el borde con el color actual.

FillPoly [GRAPH UNIT]

SINTAXIS: procedure FillPoly (NumPoints : word; var PolyPoints);

DESCRIPCION:Dibuja un polígono con un número de puntos NumPoints. El array PolyPoints contiene las coordenadas para cada uno de los puntos del polígono.

FindFirst [DOS UNIT]

SINTAXIS: procedure FindFirst(Path:string;Attr :word;var S: SearchRec);

DESCRIPCION:Regresa información en el primer archivo encontrado en el directorio PATH que tiene el atributo ATRR.Los valores estandar de ATRR son:

```
Const
ReadOnly   = $01;
Hidden     = $02;
SysFile    = $04;
VolumeId   = $08;
Directory  = $10;
AnyFile    = $3f;
```

El tipo de dato SearchRec es definido como

```
Type
SearchRec = Record
              Fill   : Array [1..21] of byte;
              Atrr   : byte;
              Time   : longint;
              Size   : longint;
              Name : string[12];
            End;
```

Si la búsqueda es exitosa, el valor de DosError será cero.

FindNext [DOS UNIT]

SINTAXIS: procedure FindNext(Var S : SearchRec);

DESCRIPCION:Regresa información en el siguiente archivo encontrado en el directorio,PATH definido en FindFirst .Si la búsqueda es exitosa,el valor de DosError será cero.

Floodfill [GRAPH UNIT]

SINTAXIS: procedure FloodFill(x,y : integer; Border : word);

DESCRIPCION : Rellena y encierra el área de despliegue de gráficas con el color y modelo común. El área debe ser completamente encerrada por el color Border, y las coordenadas x:y deben posicionarse con el área a llenar.

Flush

SINTAXIS: procedure Flush(Var F : Text);

DESCRIPCION:Manda al disco toda la salida del buffer al archivo F.

Frac

SINTAXIS: function Frac(R:real):real;

DESCRIPCION:Regresa la porción no entera de R.

FreeMem

SINTAXIS: procedure FreeMem(Var P :Pointer,I: integer);

DESCRIPCION:Libera I bytes de la memoria HEAP asociada con la variable P, que ya fué previamente localizada por GetMem.

FSearch [DOS UNIT]

SINTAXIS: function FSearch(Path :PathStr;DirList : string): PathStr;

DESCRIPCION:Busca en la lista de directorios incluido en el DirList para un nombre de archivo que une el PATH. Si el archivo se encontró, el resultado regresará como un string.Si no se encontró, la función regresa un string vacío.

FSplit [DOS UNIT]

SINTAXIS: procedure FSplit(Path : PathStr;Var Dir :Dirstr;Var Name : NameStr;Var Ext : ExtStr);

DESCRIPCION:Acepta un nombre de archivo PATH y regresa sus componentes. Se usan los siguientes tipos.

```
Type
PathStr    = string[79];
DirStr     = string[67];
NameStr    = string[8];
ExtStr     = string[4];
```

GetArcCoords [GRAPH UNIT];

SINTAXIS: procedure GetArcCoords(Var ArcCoords :ArcCoordsType);

DESCRIPCION:Regresa las coordenadas mas recientes que se usaron en los comandos Arc o Ellipse.La estructura es como se muestra a continuación:

```
type
ArcCoordsType = record
                  y              : integer;
                  Xstart,Ystart  : integer;
                  Xend,Yend      : integer;
                end;
```

GetAspectRatio [GRAPH UNIT]

SINTAXIS: procedure GetAspectRatio(Var Xasp,Yasp :word);

DESCRIPCION:Regresa en Xasp y Yasp la resolución efectiva de la pantalla de gráficas.El radio es calculado como Xasp dividido entre Yasp.

GetBkColor [GRAPH UNIT]

SINTAXIS: function GetBkColor:word;

DESCRIPCION:Regresa el color de fondo de la paleta actual.

GetCBreak [DOS UNIT]

SINTAXIS: procedure GetCBreak(Var Break:Boolean);

DESCRIPCION:Regresa el estado actual del CTRL-BREAK checado en el DOS. Cuando el Break es FALSE, el DOS checa el CTRL-BREAK mediante la consola,impresora, o I/O. Cuando es TRUE, el DOS checa el CTRL-BREAK en cualquier sistema llamado.

GetColor [GRAPH UNIT]

SINTAXIS: function GetColor :word;

DESCRIPCION:Regresa el color del dibujo en el modo gráfico.

GetDate [DOS UNIT]

SINTAXIS: procedure GetDate (Var Year,Month,Day,DayOfWeek :word);

DESCRIPCION:Regresa la fecha determinado por el reloj del sistema.

GetDefaultPalete [GRAPH UNIT]

SINTAXIS: procedure GetDefaultPalette (Var Pal: PaletteType);

DESCRIPCION:Regresa en Pal la paleta por default para el drive de gráficas.La estructura del Palette Type es como sigue:

```
type
PaletteType = record
                Size   : byte;
                Colors : array [0..MaxColor]of shortint;
              end;
```

GetDir

SINTAXIS: procedure GetDir(d:byte;Var s: string);

DESCRIPCION:Trae el directorio del drive especificado por d.El directorio lo regresa en s. Si d es cero, GetDir busca el drive actual.

GetDriverName [GRAPH UNIT]

SINTAXIS: function GetDriverName :string;

DESCRIPCION:Regresa el nombre del driver de gráficas.

GetEnv [DOS UNIT]

SINTAXIS: function GetEnv(EnvVar :string): string;

DESCRIPCION:Regresa el string a la varaiable especificada en EnvVar.

GetFAttr [DOS UNIT]

SINTAXIS: procedure GetFAttr(Var F;Var Attr:word);

DESCRIPCION:Regresa el atributo para el archivo F.Antes de llamar a este procedimiento,F debe ser asignado pero no abierto.

GetFillPattern [GRAPH UNIT]

SINTAXIS: procedure GetFillPattern(Var FP : FillPatternType);

DESCRIPCION:Regresa en FP la definición del modelo. La estructura del FillPatternType es :

```
Type
FillPatterntype = Array [1..8] of byte;
```

GetFillSettings [GRAPH UNIT]

SINTAXIS: procedure GetFillSettings(Var FS : FillSettingsType);

DESCRIPCION:Regresa en FS el diseño completo y el color.La estructura del FillSettingsType es:

```
type
FillSettingsType = record
                     Pattern : word;
                     Color   : word;
                   end;
```

GetFTime [DOS UNIT]

SINTAXIS: procedure GetFTime (Var F;var Time : LongInt);

DESCRIPCION:Regresa en Time la hora en que fué creado el archivo F. El archivo F debe ser asignado antes de usar este procedimiento.La variable Time es un valor de packed y debe ser desempacada con el procedimiento UnPackTime.

GetGraphMode [GRAPH UNIT]

SINTAXIS: function GetGraphMode : integer;

DESCRIPCION:Regresa el modo gráfico. El valor numérico del modo gráfico debe ser interpretado en combinación con el graphics driver.

GetImage

SINTAXIS: procedure Getimage (x1,y1,x2,y2 : integer;Var BitMap);

DESCRIPCION:Amacena una porción rectangular de una pantalla gráfica, definida por x1:y1 y x2:y2 en BitMap.

GetIntVec [DOS UNIT]

SINTAXIS: procedure GetIntVec(IntNo :byte;Var: Vector :Pointer);

DESCRIPCION:Regresa en Vector el contenido actual de el vector de interrupción IntNo.

GetLineSettings [GRAPH UNIT]

SINTAXIS: procedure GetLineSettings (Var LST : LineSettingsType);

DESCRIPCION : Regresa en el LST el conjunto actual del estilo de la línea, diseño y el espesor. La estructura es como sigue:

```
type
 LineSettingsType = record
                      Linestyle   : word;
                      Pattern     : word;
                      ThickNess : word;
                    end;
```


GetMaxColor [GRAPH UNIT]

SINTAXIS: function GetMaxColor : word;

DESCRIPCION:Regresa el valor más alto que representa el color en la paleta actual .

GetMaxMode [GRAPH UNIT]

SINTAXIS: function GetMaxMode : word;

DESCRIPCION:Regresa un volor que indica la resolución más alta en el modo gráfico para un adaptador instalado.

GetMaxX [GRAPH UNIT]

SINTAXIS: function GetMaxX : integer;

DESCRIPCION:Regresa la máxima coordenada horizontal para el modo gráfico actual.

GetMaxY

SINTAXIS: function GetMaxY : integer;

DESCRIPCION:Regresa la máxima coordenada vertical para el modo gráfico actual.

GetMem

SINTAXIS: procedure GetMem(Var P:Pointer;I:integer);

DESCRIPCION:Reserva I bytes en el HEAP y almacena la dirección en la variable P.

GetModeName [GRAPH UNIT]

SINTAXIS: function GetModeName(ModeNumber : word) : string;

DESCRIPCION:Regresa un string describiendo el modo gráfico denotado en el ModeNumber.

GetModeRange [GRAPH UNIT]

SINTAXIS: procedure GetModeRange(GraphDriver : integer;Var LoMode,HiMode :integer);

DESCRIPCION:Regresa el mas alto (HiMode) y el más bajo (LoMode) modo de resolución para el driver gráfico denotado por GrapDriver.

GetPalette [GRAPH UNIT]

SINTAXIS: procedure GetPalette(Var P:PaletteType);

DESCRIPCION:Regresa en P la paleta actual. La estructura es:

```
const
 MaxColors = 15;
 type
 PaletteType = record
                 Size   : byte;
                 Colors : array [0..MaxColors] of shortint;
               end;
```

GetPaletteSize [GRAPH UNIT]

SINTAXIS: function GetPaletteSize : word;

DESCRIPCION:Regresa el máximo número de la entrada de una paleta que el modo gráfico actual puede soportar.

GetPixel [GRAPH UNIT]

SINTAXIS: function GetPixel(x,y : integer);

DESCRIPCION:Regresa el color del Pixel en las coordenadas x:y.

GetTextSettings [GRAPH UNIT]

SINTAXIS: procedure GetTextSettings(Var TS :TextSettingsType);

DESCRIPCION:Regresa en TS el conjunto textual actual.La estructura es como sigue:

```
type
TextSettingsType = record
                     Font      : word;
                     Direction : word;
                     Charsize  : word;
                     Horiz     : word;
                     Vert      : word;
                   end;
```

GetTime [DOS UNIT]

SINTAXIS: procedure GetTime(Var Hour,Minute,Second,Sec100 : word);

DESCRIPCION:Regresa la hora de acuerdo al reloj del sistema.

GetVerify [DOS UNIT]

SINTAXIS: procedure GetVerify(Var Verify : Boolean);

DESCRIPCION:Regresa el estatuto de la verificación escrita en DOS. Cuando el Verify es TRUE,DOS verifica toda la escritura del disco.

GetViewSettings [GRAPH UNIT]

SINTAXIS procedure GetViewSettings (Var VP : ViewPortType);

DESCRIPCION:Regresa en VP los conjuntos de Viewport.La estructura es como sigue:

```
type
ViewportType = record
                 x1,y1,x2,y2 : integer;
                 Clip        :Boolean;
               end;
```

GetX [GRAPH UNIT]

SINTAXIS: function GetX : integer;

DESCRIPCION : Regresa la coordenada horizontal de la posición actual.

GetY [GRAPH UNIT]

SINTAXIS: function GetY : integer;

DESCRIPCION : Regresa la coordenada vertical de la posición actual.

GotoXY [CRT UNIT]

SINTAXIS: procedure GotoXY(x,y:integer);

DESCRIPCION:Coloca el cursor en las coordenadas de la pantalla X:Y

GraphDefaults [GRAPH UNIT]

SINTAXIS: procedure GraphDefaults;

DESCRIPCION:Borra el conjunto de gráficas a sus valores por default.

GraphErrorMsg [GRAPH UNIT]

SINTAXIS: function GraphErrorMsg(Code:integer):string;

DESCRIPCION:Regresa un mensaje de error a la condición de error denotada por el Code.

GraphResult [GRAPH UNIT]

SINTAXIS: function GraphResult:integer;

DESCRIPCION:Regresa un código de error para los últimos procedimientos gráficos.

Halt

SINTAXIS: procedure Halt;

DESCRIPCION : Termina un programa.

Hi

SINTAXIS: function Hi(I:integer):byte;

DESCRIPCION:Regresa al byte mas significativo del entero I.

HighVideo [CRT UNIT]

SINTAXIS: procedure HighVideo;

DESCRIPCION:Habilita la intensidad del video.

ImageSize [GRAPH UNIT]

SINTAXIS: function ImageSize(x1,y1,x2,y2:Integer);

DESCRIPCION:Regresa el número de bytes requeridos para almacenar el mapa de bit a la parte de la pantalla definida por x1:y1 y x2:y2.

Inc

SINTAXIS: procedure Inc(Var x;n:longint);

DESCRIPCION : Incrementa el valor escalar de x por n. Si n se omite de la lista del parametros,x es incrementada por 1.

InitGraph [GRAPH UNIT]

SINTAXIS: procedure InitGraph(Var GraphDriver :integer;var GraphMode : integer;DriverPath : string);

DESCRIPCION:Inicializa al ambiente de gráficas con el driver de las gráficas GraphDriver y el modo GraphMode.Si el GraphDriver es cero,el procedimiento automáticamente detecta el adaptador y pone el modo en la mas alta resolución.

Insert

SINTAXIS: procedure Insert(Source :string;Var Target :string;index:integer);

DESCRIPCION:Inserta un string source en un string Target en la posición Index.

InsLine [CRT UNIT]

SINTAXIS: procedure InsLine;

DESCRIPCION:Inserta una línea en blanco en la pantalla en la posición actual del cursor

InstallUserDriver [GRAPH UNIT]

SINTAXIS: function InstallUserDriver (Name:string;AutoDetectPtr:Pointer):integer;

DESCRIPCION:Instala un Driver gráfico non-Borland.Name contiene el nombre de archivo que contiene el Driver,y AutoDetectPtr es un apuntador hacia una función opcional auto detecta.El Driver debe estar en formato .BGI.

InstallUserFont [GRAPH UNIT]

SINTAXIS: function InstallUserFont (FontFileName:string):integer;

DESCRIPCION:Permite al usuario instalar un alfabeto non- Borland.El nombre de archivo FontFileName contiene la informacion del alfabeto.

Int

SINTAXIS: function Int(R:real):integer;

DESCRIPCION:Regresa la porción entera de R.

Intr [DOS UNIT]

SINTAXIS: procedure Intr(Func:byte;Var Regs:Registers);

DESCRIPCION:Llama al número de interrupción BIOS con los registros definidos por Regs.

IOresult

SINTAXIS: function IOresult:word;

DESCRIPCION:Reporta un código de error cuando las operaciones de I/Oson ejecutadas.Si el IOresult no es igual a cero, ocurre un error.

Keep [DOS UNIT]

SINTAXIS: procedure Keep(ExitCode:word);

DESCRIPCION:Termina el programa, pero lo mantiene residente.El procedimiento pasa al ExitCode como un código de error DOS.

KeyPressed [CRT UNIT]

SINTAXIS: function KeyPressed:Boolean;

DESCRIPCION:Regresa un valor TRUE cuando se presiona una tecla y ésta esta pendiente de procesarse.

Length

SINTAXIS: function Length(S:string):integer;

DESCRIPCION:Regresa la longitud de un string S.

Line [GRAPH UNIT]

SINTAXIS: procedure Line(x1,y1,x2,y2:integer);

DESCRIPCION:Dibuja una línea desde x1:y1 a x2:y2.

LineRel [GRAPH UNIT]

SINTAXIS: procedure LineRel(Dx,Dy:integer);

DESCRIPCION:Dibuja una línea desde el apuntador actual al punto definido por Dx y Dy.Por ejemplo, si el apuntador actual esta posicionado en 1:2,entonces el comando LineRel(100,100) dibujará una línea desde 1:2 a 101:102.

LineTo [GRAPH UNIT]

SINTAXIS: procedure LineTo(x,y:integer);

DESCRIPCION:Dibuja una línea desde el punto actual x:y.

Ln

SINTAXIS: function Ln(Var R:real):real;

DESCRIPCION:Regresa el logaritmo natural de R.

Lo

SINTAXIS: function Lo(I:Integer):byte;

DESCRIPCION:Regresa el LSB de un entero de I.

LowVideo [CRT UNIT]

SINTAXIS: procedure LowVideo;

DESCRIPCION:Fija el video a menor intensidad.

Mark

SINTAXIS: procedure Mark(P:Pointer);

DESCRIPCION:Almacena la dirección superior del Heap en el apuntador P.

MaxAvail

SINTAXIS: function MaxAvail:longint;

DESCRIPCION:Regresa el tamaño del bloque más largo en la memoria no -localizada en el Heap.

MemAvail

SINTAXIS: function MemAvail:longint;

DESCRIPCION:Regresa la cantidad total de la memoria no localizada en el Heap.

MkDir

SINTAXIS: function MkDir(S:string);

DESCRIPCION:Hace un directorio con el nombre almacenado en el string S.

Move

SINTAXIS: procedure Move (Var V1,V2;I:integer);

DESCRIPCION:Copia I bytes de la memoria desde la localizacion de la variable V1 a la localizacion V2.

MoveRel [GRAPH UNIT]

SINTAXIS: procedure MoveRel(Dx,Dy:integer);

DESCRIPCION:Mueve el apuntador a la posicion horizontal Dx pixels y el vertical Dy pixels, relativo a la posición actual del cursor.

MoveTO [GRAPH UNIT]

SINTAXIS: procedure Moveto(x,y:integer);

DESCRIPCION:Posiciona el cursor actual en el pixel x:y.

MsDos [DOS UNIT]

SINTAXIS: procedure MsDos(Var Regs:Registers);

DESCRIPCION:Ejecuta los servicios del DOS usando los valores del conjunto en REGS.

New

SINTAXIS procedure New(Var P:Pointer);

DESCRIPCION:Coloca la memoria en el Heap del apuntador P.Después que la memoria es colocada,la variable es referida como P^.

NormVideo [CRT UNIT]

SINTAXIS: procedure NormVideo;

DESCRIPCION:Almacena los atributos por default de la pantalla hacia aquellos que estuvieron presentes en la posición del cursor cuando se empezo el programa.

NoSound [CRT UNIT]

SINTAXIS: procedure NoSound;

DESCRIPCION:Detiene cualquier sonodo generado en la bocina..

Odd

SINTAXIS: function Odd(I:integer):Boolean;

DESCRIPCION:Regresa el valor True cuando I es impar y False cuando I es par.

Ofs

SINTAXIS: function Ofs(Variable ,procedure, Or function):integer;

DESCRIPCION:Regresa el offset de la dirección de memoria para cualquier variable, procedimiento o función.

Ord

SINTAXIS: function Ord(S:Scalar):integer;

DESCRIPCION:Regresa el valor entero de cualquier variable escalar.

OutText [GRAPH UNIT]

SINTAXIS: procedure OutText(TextString:string);

DESCRIPCION:Despliega el string TextString usando los conjuntos de pilas,justificacion,altura y anchura.

OutTextXY

SINTAXIS: procedure OutTextXY (x,y:integer,TextString:string);

DESCRIPCION:Despliega el string TextString en la posicion x;y usando los conjuntos actuales de las pilas, justificación,altura y anchura en modo gráfico.

OvrClearBuf [OVERLAY UNIT]

SINTAXIS: procedure OvrClearBuf;

DESCRIPCION:Vacía el Buffer para el código traslapable.

OvrGetBuf [OVERLAY UNIT]

SINTAXIS: function OvrGetBuf:longint;

DESCRIPCION:Regresa el tamaño del buffer para el código traslapable.

OvrInit [OVERLAY UNIT]

SINTAXIS: procedure OvrInit(FileName:string);

DESCRIPCION:Inicializa el administrador del overlay.El FileName contiene el nombre del archivo overlay.

OvrInitEMS [OVERLAY UNIT]

SINTAXIS: procedure OvrInitEMS;

DESCRIPCION:Carga el archivo de overlay en memoria, si hay suficiente disponible.

OvrSetBuf [OVERLAY UNIT]

SINTAXIS: procedure OvrSetBuf(BufSize:integer);

DESCRIPCION:Coloca los bytes del BufSize hacia el buffer de overlay.El BufSize no debe de ser mas pequeño que el buffer overlay por default.

PackTime [DOS UNIT]

SINTAXIS: procedure PackTime(Var DT:DateTime;Var Time:longint);

DESCRIPCION:Acepta la variable DT,que contine información de fecha y hora, y regresa Time, que contiene la misma información en forma de paquete.

ParamCount

SINTAXIS: function ParamCount:word;

DESCRIPCION:Regrasa el número de parámetros de comandos de entrada.

ParamStr

SINTAXIS: function ParamStr(I:word:string);

DESCRIPCION:Regresa parámetros que eran de entrada en líneas de comando.Por ejemplo, ParamStr (1) regresa el primer parámetro. En el DOS 3.x,ParamStr (0) regresa el PATH y el nombre de archivo del archivo ejecutado.

Pi

SINTAXIS: function Pi:real;

DESCRIPCION:Regresa el valor de una constante matematica Pi.La presicion del numero depende en si el modo 8087 está activado.

PieSlice [GRAPH UNIT]

SINTAXIS: procedure PieSlice(x,y:integer;StAngle,EndAngle,Radius:word);

DESCRIPCION:Dibuja un pedazo de un Pie centrado en x:y,con radio Radius,y empezando en StAngle y terminando en EndAngle.

Pos

SINTAXIS: function Pos(SubS,S:string):integer;

DESCRIPCION:Regresa la posición de SubS en S.Si SubS no se encuentra en S,Pos regresa un 0.

Pred

SINTAXIS: function Pred(Var S:Scalar):integer;

DESCRIPCION:Decrementa cualquier variable escalar.

Ptr

SINTAXIS: function Ptr(Segment,Offset:integer):Pointer;

DESCRIPCION:Acepta dos enteros que contiene un segmento y un offset y regresa un valor de apuntador de 32 bytes.

PutImage [GRAPH UNIT]

SINTAXIS procedure PutImage(x,y:Integer;Var BitMap;BitBlt:word);

DESCRIPCION:Despliega el contenido del BitMap empezando en x:y.El BitBlt especifica el proceso a usar para el despliegue del bit map,y puede tomar los siguientes valores:

```
const
CopyPut     = 0;
XORPut      = 1;
OrPut       = 2;
AndPut      = 3;
NotPut      = 4;
```

PutPixel [GRAPH UNIT]

SINTAXIS: procedure PutPixel(x,y :integer;Pixel:word);

DESCRIPCION:dibuja un punto de color,definido por Pixel,en la posicion x:y.

Random

SINTAXIS: function Random(l:word):word;

function Random:real;

DESCRIPCION:Regresa un número aleatorio generado por Turbo Pascal.Si pasa un parámetro entero,el Random regresa un entero mayor o igual a cero y menor que el parámetro. Sin un entero,el Random regresa un valor real mayor o igual a cero y menor que 1.

Randomize

SINTAXIS: function Randomize;

DESCRIPCION:Inicializa el valor original del número aleatorio generado.El valor original es almacenado en la variable Lougint RandSeed.

Read (ReadIn)

SINTAXIS: procedure Read({Var F:File,}Parameters);

procedure ReadLn({Var F:File,}Parameters);

DESCRIPCION:Recibe una entrada de cualquiera de las entradas estandar o de un archivo especificado por F.ReadIn,que puede ser usado en archivo de textos,recibe entradas de la misma manera que lo hace el Read,pero despues de leerlo en el dato,el ReadLn mueve el apuntador del archivo hacia el próximo retorno de carre/delimitacion de la línea de alimentación.

ReadKey [CTR UNIT]

SINTAXIS: function ReadKey:char;

DESCRIPCION:Lee un caracter desde el tablero sin eco.Si el resultado es # 0 entonces una tecla especial ha sido presionada y deberia llamar al ReadKey otra vez para capturar la segunda parte del código de la tecla.

Rectangle [GRAPH UNIT]

SINTAXIS: procedure Rectangle(x1,y1,x2,y2:integer);

DESCRIPCION:Dibuja un rectángulo con su esquina superior izquierda el x1:y1 y en la esquina inferior derecha en el punto x2:y2.

RegisterBGIdriver [GRAPH UNIT]

SINTAXIS: function RegisterBGIdriver (Driver:Pointer):integer;

DESCRIPCION:Permite al usuario cargar un archivo Driver BGI (lee del disco al heap o une en el programa usando BINOBJ) y registra el driver con los sistemas gráficos.El driver es un apuntador hacia la localidad del driver BGI.Si ocurre un error,la función regresa un valor menos que 0;de otra manera regresa el número asignado del driver.

RegisterBGIfont [GRAPH UNIT]

SINTAXIS: function RegisterBGIfont(Font:Pointer):integer;

DESCRIPCION:Permite al usuario cargar un archivo de alfabeto driver BGI (lee desde el disco en el heap o une en el programa usando en el programa BINOBJ) y registra el alfabeto con el sistema gráfico. Font es un apuntador hacia la localización del driver BGI.Si ocurre un error, la función regresa un valor menor que 0 por otra manera regresara el número asignado del alfabeto.

Release

SINTAXIS: procedure Release (Var P:Pointer);

DESCRIPCION:Libera la memoria que ha sido cargada desde el comando Mark.Usado para almacenar la dirección top-of-heap en P.

Rename

SINTAXIS: procedure Rename(Var F:File;S:string);

DESCRIPCION:cambia el nombre de archivo F que contiene en S.

Reset

SINTAXIS: procedure Reset(Var F:File{;I:integer});

DESCRIPCION:Abre el archivo F para lectura.Si el archivo no esta sin tipo, puede especificar el tamano en I.

RestoreCRTMode [GRAPH UNIT]

SINTAXIS: procedure RestoreCRTMode;

DESCRIPCION:Almacena el despliegue del video hacia el modo que existía antes que se utilizaran gráficas.

Rewrite

SINTAXIS: procedure Rewrite(Var F:File {;I:integer});

DESCRIPCION:Prepara un archivo para ser escrito.Si el archivo no existe;Turbo Pascal lo crea, si existe, lo destruye si el archivo es sin tipo,usted lo puede especificar con el tamaño de registros en I.

RmDir

SINTAXIS: procedure RmDir(S:string);

DESCRIPCION:Borra el directorio especificado en S.

Round

SINTAXIS: function Round(R:real):longint;

DESCRIPCION:Regresa el valor entero redondeado de R.

RunError

SINTAXIS procedure RunError;
procedure RunError(ErrorCode:word);

DESCRIPCION:Para la ejecución del programa y genera un run- time error.Si el ErrorCode esta incluído, Turbo Pascal interpretara este como el tipo de run-time error que ocurrió.

Sector [GRAPH UNIT]

SINTAXIS procedure Sector(x,y:integer;StAngle,EndAngle,XRadius,Yradius:word);

DESCRIPCION:Dibuja un sector centrodo en x:y,empezando en StAngle ,y terminando en EndAngle,con un radio horizontal XRadius y un radio vertical YRadius.

Seek

SINTAXIS: procedure Seek(Var F:File;P:integer);

DESCRIPCION:Mueve el apuntador archivo al comienzo del numero de registro P en el archivo F.

SeekEof

SINTAXIS: function SeekEof(Var F:File)Boolean;

DESCRIPCION:Es similar al EOF,excepto que omite blancos,cuenta y marcas de secuencia end-of-line (CR/LF) antes que se pruebe para una marca end-of-file.El tipo del resultado es booleano.

SeekEoln

SINTAXIS: function SeekEoln(Var F: File):Boolean;

DESCRIPCION:Es similar a Eoln, excepto que este omite blancos y cuenta antes de que pruebe la marca end-of-line.El tipo de resultado es booleano.

Seg

SINTAXIS: function Seg(Var variable)word;
function Seg(or function):word);

DESCRIPCION:Regresa la dirección del segmento de una variable, procedimiento, o función.

SetActivePage [GRAPH UNIT]

SINTAXIS: procedure SetActivePage(Page:word);

DESCRIPCION:Selecciona el número de página de los gráficos.

SetAllPAlette [GRAPH UNIT]

SINTAXIS: procedure SetAllPalette (Var Palette);

DESCRIPCION:Cambia todas las paletas hacia la definición contenida en Palette.

SetAspectRatio [GRAPH UNIT]

SINTAXIS: procedure SetAspectRatio(XAsp,YAsp:word);

DESCRIPCION:Cambia el aspecto del radio usado en el despliegue de gráficas para XAsp dividido por YAsp.

SetBkColor [GRAPH UNIT]

SINTAXIS: procedure SetBkColor(Color: word);

DESCRIPCION:Activa el color del background para el modo gráfico utilizando de entrada Color de la paleta actual.

SetCBreak [DOS UNIT]

SINTAXIS: procedure SetCBreak (Break:Boolean);

DESCRIPCION:Activa el CTRL-BREAK (cuando Break es TRUE) o desactiva (caundo el Break es FALSE).

SetColor [GRAPH UNIT]

SINTAXIS: procedure SetColor(Color:word);

DESCRIPCION:Cambia el color actual del dibujo a el Color de la paleta.

SetDate [DOS UNIT]

SINTAXIS: procedure SetDate(Year,Month,Day:word);

DESCRIPCION:Actualiza el reloj del sistema a la fecha pasada como parámetro.Por ejemplo, el comando SetDate (1990,12,1) coloca la fecha a diciembre 1 ,1990.

SetFAttr [DOS UNIT]

SINTAXIS: procedure SetFAttr(Var F;Attr:word);

DESCRIPCION:Coloca el byte de atributo del archivo F a Attr.El archivo F debe ser asignado pero no abierto antes de la llamada del procedimiento.

SetFillPattern [GRAPH UNIT]

SINTAXIS: procedure SetFillPattern(Pattern:FillPartternType;Color:word);

DESCRIPCION:Define el diseño de gráficas usado para llenar las porciones de la pantalla con los comandos tales como Fill- Poly y FloodFill.

SetFillStyle [GRAPH UNIT]

SINTAXIS: procedure SetFillStyle(Pattern: word;Color:word);

DESCRIPCION:Coloca el diseño usado para llenar las áreas de un despliegue de gráficas .

SetFTime [DOS UNIT]

SINTAXIS: procedure SetFTime(Var f;Time: longint);

DESCRIPCION:Coloca la hora del archivo f al valor de Time,que es una representación empaquetada de la hora y fecha. Time es creado con el procedimiento PackTime.

SetGraphBufSize [GRAPH UNIT]

SINTAXIS: procedure SetGraphBufSize(BufSize:word);

DESCRIPCION:Delimita el tamaño del buffer para el modo gráfico.

SetGraphMode [GRAPH UNIT]

SINTAXIS: procedure SetGraphMode (Mode:integer);

DESCRIPCION:Cambia el modo gráfico actual a aquel especificado por Mode.

SetIntVec [DOS UNIT]

SINTAXIS: procedure SetIntVec(IntNo:byte;Vector:Pointer);

DESCRIPCION:Coloca el valor del Vector de interrupción en la número IntNo en el interruptor de la tabla de vectores de interrupción.

SetLineStyle [GRAPH UNIT]

SINTAXIS: procedure SetLineStyle(LineStyle:word;Pattern :word;Thickness :word);

DESCRIPCION:Determina el estilo,diseño y el espesor de las líneas de los dibujos en el modo gráfico.

SetPalette [GRAPH UNIT]

SINTAXIS: procedure SetPalette(ColorNum:word;Color:shortint);

DESCRIPCION:Coloca el número de colores de ColorNum de la paleta activa al color.

SetRGBPalette [GRAPH UNIT]

SINTAXIS: procedure SetRGBPalette(ColorNum,RedValue,GreenValue,BlueValue:integer);

DESCRIPCION:Cambiaa la paleta de entrada ColorNum para que consista en cualquiera de las combinaciones rojo , verde y azul.

SetTextBuf

SINTAXIS: procedure SetTextBuf(Var f:Text;Var Buf);
procedure SetTextBuf(Var f:Text;Var Buf;Size:word);

DESCRIPCION:Asigna el archivo de texto f al buffer Buf.Si el tamaño no se especifica el tamano del buffer es aquel del Buf. Size puede ser usado para cambiar el tamaño del buffer.

SetTextJustify [GRAPH UNIT]

SINTAXIS: procedure SetTextJustify(Horiz,Vert:word);

DESCRIPCION:Define el formato del despliegue usado por OutText y OutTextXY.

SetTextStyle [GRAPH UNIT]

SINTAXIS: procedure SetTextStyle(Font: word;Direction:word;CharSize:word);

DESCRIPCION:Determina cómo los caracteres deberan desplegarse en el modo gráfico. Las características incluyen el alfabeto, la dirección en donde la escritura toma lugar, y el tamaño de los caracteres.

SetTime [DOS UNIT]

SINTAXIS: procedure SetTime(Hour,Minute,Second,Sec100:word);

DESCRIPCION:Coloca el reloj del sistema de acuerdo con los valores pasados como parámetros.

SetUserCharSize [GRAPH UNIT]

SINTAXIS: procedure SetUserCharSize(MultX,DivX,MultY,DivY:word);

DESCRIPCION:Cambia las proporciones de anchura y altura para las pilas. Por ejemplo,si MultX es 1 y DivX es 2, entonces los caracteres seran desplegados con un medio de anchura que normalmente tendrían.

SetVerify [DOS UNIT]

SINTAXIS: procedure SetVerify(Verify:Boolean);

DESCRIPCION: Activa la verificación de escritura (cuando Verify es TRUE) o (desactiva cuando Verif es FALSE).

SetViewPort [GRAPH UNIT]

SINTAXIS: procedure SetViewPort(x1,y1,x2,y2:integer;Chip:Boolean);

DESCRIPCION:Selecciona la porcion rectangular de la. pantalla de gráficos para usarlo como parte activa de pantalla. Cuando el clipping es TRUE,los dibujos son clippped en la fronteras del viewport.

SetVisualPage [GRAPH UNIT]

SINTAXIS: procedure SetVisualPage(Page:word);

DESCRIPCION:Selecciona la páginas de gráficos.

SetWriteMode [GRAPH UNIT]

SINTAXIS: procedure SetWriteMode(WriteMode:integer);

DESCRIPCION:Selecciona uno de los dos modos para dibujar líneas.Las líneas del modo CopyPut(0) son dibujados usando el comando de ensamblador Mov.En las líneas del modo XORPut(1) son dibujados usando el comando XOR.

SIN

SINTAXIS: function Sin(R:real):real;

DESCRIPCION:Regresa el seno de R.

SizeOf

SINTAXIS: function SizeOf(Var Variable):word;

DESCRIPCION:Regresa el número de bytes requeridos por una variable o un tipo de dato.

Sound

SINTAXIS: procedure Sound(Freq:word);

DESCRIPCION:Genera un tono desde la bocina de la PC's en una frecuencia especificada por Freq. El tono continua hasta que se use el comando NoSound .

SPtr

SINTAXIS: function SPTr :word;

DESCRIPCION:Regresa el valor actual del registro del apuntador del stack.(SP).

Sqr

SINTAXIS: function Sqr(R:real):real;

DESCRIPCION:Regresa el cuadrado de R.

Sqrt

SINTAXIS: function Sqrt(R:real):real;

DESCRIPCION:Regresa la raíz cuadrada de R.

SSeg

SINTAXIS: function SSeg:word;

DESCRIPCION:Regresa el valor actual del registro del segmento del stack(SS).

Str

SINTAXIS: procedure Str(I:integer;[:Length;]Var S:string);

procedure Str(I:integer;[:Length:Decimals,]Var S:string);

DESCRIPCION:Convierte un número real o un entero en un string.

Succ

SINTAXIS: function succ(S:Scalar):integer;

DESCRIPCION:Incrementa en uno el valor escalar.

Swap

SINTAXIS: procedure Swap(I:integer):integer;

DESCRIPCION:Intercambia la posición de los bytes de un entero.Por ejemplo,si I es igual a 00FFh,Swap regresara FF00h.

SwapVectors [DOS UNIT]

SINTAXIS: procedure SwapVectors;

DESCRIPCION:Intercambia el valor actual del vector de tabla de interrupciones con aquellos que fue grabado cuando el programa empezó la ejecucion.

TextBackground [CRT UNIT]

SINTAXIS: procedure TextBackground(Clor:byte);

DESCRIPCION:Cambia el color del fondo al color especificado por Color.

TextColor [CRT UNIT]

SINTAXIS: procedure TextColor(Color:byte);

DESCRIPCION:Cambia el color del texto al color especificado por Color.

TextHeight [GRAPH UNIT]

SINTAXIS: function TextHeight(TextString:string):word;

DESCRIPCION:Determina cuánto espacio vertical requerirá TextString para el alfabeto actual y el factor de multiplicación.

Textmode [CRT UNIT]

SINTAXIS: procedure Textmode(mode:word);

DESCRIPCION:Activa uno de los modos del texto que provee Turbo Pascal.Esos incluye

```
const
 BW40   = 0;      (* 40 x 25 black & White / color adapter *)
 C040   = 1;      (* 40 x 25 color / color adapter*)
 C40    = C040;   (* For 3.0 compatibility*)
 BW80   = 2;      (* 80 x 25 black & White / color adapter*)
 C080   = 3;      (* 80 x 25 color /color adapter*)
 C80    = C080;   (* For 3.0 compatibility*)
 MONO   = 7;      (* 80 x 25 monochrome adapter*)
 Font8x8 = 256;   (* EGA and VGA 43-AND 50- line mode*)
```

TextWidth [GRAPH UNIT]

SINTAXIS: function TextWidth(TextString:string);

DESCRIPCION:Regresa la anchura en pixels requerida para el despliegue TextWidth dando el alfabeto actual y el factor multiplicativo.

Trunc

SINTAXIS: function Trunc(R:real):integer;

DESCRIPCION:Regresa la porción entera de R.El resultado debe estar en el rango legal de un entero.

Truncate

SINTAXIS: procedure Truncate(Var f);

DESCRIPCION:trunca el archivo en la posición actual del apuntador de archivo. El contenido fuera del archivo del apuntador se pierde.

UnPackTime [DOS UNIT]

SINTAXIS: procedure UnPackTime(Time:longint;Var DT :DateTime);

DESCRIPCION:Decodifica la variable Time y regresa el resultado en DT.La estructura de DateTime es:

```
type
  Datetime = record
               Year,Month,Day,Hour,Min,Sec:word;
             end;
```

Upcase

SINTAXIS: function Upcase(C:char):char;

DESCRIPCION:Regresa la mayúscula de C SI C es minúscula.

Val

SINTAXIS: procedure Val(S:string;Var R:real;Var Code:integer);

procedure Val(S:string;Var I:integer;Var Code:integer);

DESCRIPCION:Procura convertir S en un valor número(R o I). Si la conversión tiene éxito,Turbo Pascal coloca el Code igual a cero.Si no hay éxito ,Code contiene un entero que representa el caracter en el string en donde ocurrió el error.

WhereX [GRAPH UNIT]

SINTAXIS function WhereX :byte;

DESCRIPCION:Regresa la columna en la ventana actual en donde está el cursor.

WhereY [GRAPH UNIT]

SINTAXIS: function WhereY :byte;

DESCRIPCION:Regresa la fila en la ventana actual en donde está el cursor.

Window [CRT UNIT]

SINTAXIS: procedure Window(x1,y1,x2,y2:byte);

DESCRIPCION:Limita la pantalla al rectángulo definido por las coordenadas x1:y1(arriba a la izquierda y x2:y2(Abajo a la derecha).Turbo Pascal trata la esquina superior izquierda de la ventana como cordenadas 1:1.

Write (Writeln)

SINTAXIS: procedure Write({VarF:File,}Parameters);

procedure Writeln({VarF:File,}Parameters);

DESCRIPCION:Acepta una lista de parámetros, para escribirlos. Cuando el primer parámetro es un archivo variable,la salida es directamente a quel archivo,Writeln,que puede ser usado en el archivo de textos, opera de la misma manera que el Write, pero adiciona un retorno de carro a la salida.

APENDICE B PALABRAS RESERVADAS

PALABRAS RESERVADAS EN TURBO PASCAL

ABSOLUTE	END	LABEL	REPEAT	WHILE
AND	EXTERNAL	MOD	SET	WITH
ARRAY	FILE	NIL	SHL	XOR
BEGIN	FOR	NOT	SHR	
CASE	FORWARD	OF	STRING	
CONST	FUNCTION	OR	THEN	
DIV	GOTO	PACKED	TO	
DO	IF	PROCEDURE	TYPE	
DOWNTO	IN	PROGRAM	UNTIL	
ELSE	INLINE	RECORD	VAR	

Write (Writeln)

SINTAXIS: procedure Write({VarF:File,}Parameters);

procedure Writeln({VarF:File,}Parameters);

DESCRIPCION:Acepta una lista de parámetros, para escribirlos. Cuando el primer parámetro es un archivo variable,la salida es directamente a quel archivo,Writeln,que puede ser usado en el archivo de textos, opera de la misma manera que el Write, pero adiciona un retorno de carro a la salida.

APENDICE B PALABRAS RESERVADAS

PALABRAS RESERVADAS EN TURBO PASCAL

ABSOLUTE	END	LABEL	REPEAT	WHILE
AND	EXTERNAL	MOD	SET	WITH
ARRAY	FILE	NIL	SHL	XOR
BEGIN	FOR	NOT	SHR	
CASE	FORWARD	OF	STRING	
CONST	FUNCTION	OR	THEN	
DIV	GOTO	PACKED	TO	
DO	IF	PROCEDURE	TYPE	
DOWNTO	IN	PROGRAM	UNTIL	
ELSE	INLINE	RECORD	VAR	

IDENTIFICADORES ESTANDAR EN Turbo Pascal

Addr	Delete	Length	ReadIn
Append	Dispose	LN	Real
ArcTan	EOF	Lo	Release
Assign	EOLn	LowVideo	Rename
Aux	Erase	Lst	Reset
AuxInPtr	Execute	LstOutPtr	Rewrite
AuxOutPtr	Exit	Mark	Round
BlockRead	Exp	MaxAvail	Seek
BlockWrite	External	MaxInt	SeekEOF
Boolean	False	Mem	SeekEOL
BufLen	FilePos	MemAvail	Sin
Byte	FileSize	Move	SizeOf
Chain	FillChar	New	Sqr
Char	Flush	NormVideo	Sqrt
Chr	Frac	Odd	Str
Close	FreeMem	Ord	Succ
ClrEol	GetMem	OutPut	Swap
ClrScr	GotoXY	OvrDrive	Text
Con	Halt	OvrPath	Trm
ConInPtr	HeapPtr	Paramcount	True
ConOutPtr	Hi	ParamStr	Trunc
Concat	IOResult	Pi	Truncate
ConstPtr	Input	Port	UpCase
Copy	InsLine	Pos	Usr
Cos	Insert	Pred	UsrInPtr
CrtExit	Int	Ptr	UsrOutPtr
CrtInit	Integer	Random	Val
DelLine	Kbd	Randomize	Write
Delay	Keypressed	Read	WriteIn

IDENTIFICADORES ESPECIFICOS PARA LA IBM PC EN Turbo Pascal

Black	Brown	DarkGray	
Blink	BW40	CW40	Draw
Blue	BW80	CW80	GraphBackGround
GraphColorMode	LightCian	Plot	WhereY
GraphMode	LightGray	Red	White
GraphWindow	LightGreen	Sound	Window
Green	LightMagenta	TextBackGround	Yellow
HiRes	LightRed	TextColor	
HiResColor	Magenta	TextMode	
LightBlue	NoSound	WhereX	

APENDICE C MENSAJES Y CODIGOS DE ERROR

MENSAJES DE ERROR DEL COMPILADOR

La siguiente es una lista de los posibles mensajes de error que se generan durante el proceso de compilación. Es posible, sin embargo, que el compilador complemente la información con identificadores y/o nombres de archivos, por ejemplo:

ERROR 15: FILE NOT FOUND [WINDOW.TPU]
Error 15: No se encontró archivo [WINDOW.TPU]

Cuando se detecta el error, Turbo Pascal carga el archivo fuente y coloca el cursor en donde se encontró. Nótese que algunos errores no se detectan hasta que se han compilado algunos estatutos más. Por ejemplo, un error de inconcordancia de tipo (type mismatch) en un estatuto de asignación no puede ser detectado hasta que se evaluó completamente la expresión después del símbolo ": =". En tales casos el error está a la izquierda o arriba del cursor.

1)Out of memory (Memoria insuficiente).

Este error ocurre cuando el compilador no tiene memoria suficiente para ejecutar el programa. Algunas soluciones a este problema son:

- Compilar a disco, generando un archivo .EXE
- Encadenar (linkear) a disco, a través de la directiva del compilador {$L-} al inicio del programa.
- Remover de memoria los programas residentes, como el SideKick o SuperKey.
- Utilizar el compilador TPC.EXE, en lugar del TURBO.EXE.

Si ninguna de estas sugerencias ayuda, significa que el programa o unidad es demasiado grande y que debe particionarse en dos o más unidades.

2)Identifier expected (Se espera un identificador).

En este punto se espera un identificador, o probablemente se está tratando de declarar una palabra reservada.

3)Unknown identifier (Identificador desconocido).

El identificador no ha sido declarado.

4)Duplicate identifier (Identificador duplicado).

El identificador ya ha sido utilizado en el bloque actual.

5)Sintax error (Error de sintaxis).

Se encontró un caracter ilegal en el texto fuente, o probablemente falta la comilla que encierra a un string.

6)Error in real constant (Error en la constante real).

Error en la sintaxis declarativa de la constante real.

7)Error in integer constant (Error en la constante entera).

Error en la sintaxis declarativa de la constante entera.

8)String constant exceeds line (La constante string excede la línea).

Probablemente se olvidó poner la comilla final en la constante string.

9)Too many nested files (Muchos archivos anidados).

El compilador permite no más de cinco archivos anidados.

10)Unexpected end of file (Fin de archivo inesperado).

Este error se genera cuando el archivo fuente termina antes del END principal. La mayor parte de las veces el número de BEGIN y END está desbalanceado.

11)Line too long (Línea muy larga).

El número máximo de una línea es de 126 caracteres.

12)Type identifier expected (Se espera un identificador de tipo).

El identificador no denota un tipo como debiera.

13)Too many open files (Demasiados archivos abiertos).

Cuando ocurre este error, significa que en el archivo CONFIG.SYS no se incuyó el comando FILES = XXX, o se especificó un número muy pequeño de archivos. Incremente el valor a un número mayor, por ejemplo 20.

14)Invalid file name (Nombre de archivo inválido).

El nombre del archivo es inválido, o no existe el subdirectorio especificado.

15)File not found (No se encontró el archivo).

No se pudo encontrar el archivo en el directorio actual, o en cualquiera de los subdirectorios de búsqueda.

16)Disk full (Disco lleno).

El disco está lleno. Borre algunos archivos o utilice un disco nuevo.

17)Invalid compiler directive (Directiva del compilador inválida).

La letra que especifica la directiva del compilador no pudo ser reconocida, o alguno de los parámetros no es válido, o se está usando una directiva de compilador global cuando ya ha empezado la compilación del bloque principal.

18)Too many files (Demasiados archivos).

Hay muchos archivos involucrados en el proceso de compilación del programa o de la unidad. Trate de utilizar menos, uniéndolos y utilizando nombres cortos para ellos.

19)Undefined type in pointer definition (Tipo indefinido en la definición del apuntador).

El tipo fue referenciado en la declaración de tipos, pero nunca fue declarado.

20)Variable identifier expected (Se espera identificador de variable).

El identificador no denota una variable.

21)Error in type (Error en tipo).

Este símbolo no puede empezar la definición de un tipo.

22)Structure too large (Estructura demasiado grande).

El tamaño de memoria máximo es de 45520 bytes.

23)Set base type out of range (El tipo set está fuera de rango).

La base del tipo set debe tener su rango en los límites de 0 a 255, o un tipo enumerado que no contenga más de ese número de elementos.

24)File componentes may not be files (Los campos de un archivo no pueden ser archivos).

La estructura de un archivo no puede contener archivos.

25)Invalid string length (Longitud del string inválida).

La declaración de la longitud máxima de un string no debe entrar fuera del rango de 1..255.

26)Type mismatch (Inconcordancia de tipo).

Esto puede deberse a lo siguiente:

- Tipos incompatibles de la variable y la expresión en el estatuto de asignación.

- Tipos incompatibles en la llamada a una función o procedimiento, entre los parámetros formales y los actuales.

- El tipo de la expresión y el tipo índice de un arreglo no concuerdan.

- Tipos incompatibles entre la expresión y los operandos.

27)Invalid subrange base type (Tipo base de subrango inválido).

Todos los tipos ordinales son tipos base válidos.

28)Lower bound greater than upper bound (El límite inferior es más grande que el superior).

La declaración de un tipo subrango especifica un límite inferior mayor al superior.

29)Ordinal type expexted (Se espera tipo ordinal).

No se permiten aquí los tipos reales, strings, estructurados y apuntadores.

30)Integer constant expected (Se espera constante del tipo integer).

31)Constant expected (Se espera constante).

32)Integer o real constant expected (Se espera constante entera o real).

33)Type identifier expected (Se espera el identificador de un tipo).

El identificador no denota un tipo como debiera.

34)Invalid function result type (Tipo inválido en el resultado de la función).

Los tipos de resultados válidos para una función son los tipos simples, string y apuntadores.

35)Label indentifier expected (Se espera un identificador de etiqueta).

El identificador no denota una etiqueta como debiera.

36)BEGIN expected (Se espera la palabra BEGIN).

37)END expected (Se espera la palabra END).

38)Integer expression expected (Se espera una expresión entera).

La expresión anterior debe ser de tipo INTEGER.

39)Ordinal expression expected (Se espera una expresión ordinal).

La expresión anterior debe ser de tipo ordinal.

40)Boolean expresion expected (Se espera una expresión booleana).

La expresión anterior debe ser de tipo BOOLEAN.

41)Operand types do not match operator (Los tipos de los operandos no concuerdan con el operador).

El operador no puede aplicarse a operandos de este tipo, por ejemplo: 'A' div '2'.

42)Error in expression (Expresión con error).

Este símbolo no puede participar en la expresión de la forma en que lo hace. Es posible que se haya olvidado escribir un operador entre dos operandos.

43)Illegal assignament (Asignación ilegal).

Esto puede deberse a que:

- No se les puede asignar valor a los archivos y a las variables sin tipo.

- A un identificador de una función solamente se le pueden asignar valores dentro del estatuto que es parte de la función.

44)Field identifier expected (Se espera identificador de campo).

El identificador no denota un campo en la variable anterior de tipo registro.

45)Object field too large (El archivo objeto es demasiado grande).

Turbo Pascal no puede encadenar archivos .OBJ más grandes de 64K.

46)Undefined external (External no definido).

Una función o procedimiento del tipo EXTERNAL no tiene una definición de tipo PUBLIC que concuerde con el archivo objeto.

Asegúrese de que están especificados todos los archivos objeto con la directiva {$ L nomarch}, y revise los identificadores en los archivos .ASM.

47)Invalid object file record (Registro del archivo objeto inválido).

El archivo .OBJ contiene un registro inválido; asegúrese de que dicho archivo es realmente un archivo .OBJ.

48)Code segment too large (Segmento de código demasiado grande).

El tamaño máximo del código de un programa o una unidad es de 65520 bytes. Si está compilando un programa, cambie algunos procedimientos o funciones en una unidad. Si está compilando una unidad, particiónela en dos o más unidades.

49)Data segment too large (Segmento de datos demasiado grande).

El tamaño máximo del segmento de datos de un programa es de 65520 bytes, incluyendo los datos declarados en las unidades a usar. Si necesita más datos globales, declare las estructuras grandes como apuntadores y almacénelas dinámicamente usando el procedimiento NEW.

50)DO expected (Se espera la palabra DO).

51)Invalid PUBLIC definition (Definición del tipo PUBLIC inválida).

Esto puede deberse a que:

- El identificador fue hecho público por medio de la directiva PUBLIC en lenguaje ensamblador, pero no concuerda con la declaración externa del programa o unidad de Pascal.

- Dos o más directivas PUBLIC en lenguaje ensamblador definen al mismo identificador.

- El archivo .OBJ define símbolos PUBLIC que no residen en el segmento CODE.

52)Invalid EXTRN definition (Definición EXTRN inválida).

Esto puede deberse a que:

- Se referenció el identificador por medio de la directiva EXTRN en lenguaje ensamblador, pero no está declarada en el programa o unidad de Pascal, ni en la parte de interfase de cualquiera de las unidades que se usan.

- El identificador denota una variable absoluta.

- El identificador denota un procedimiento o función del tipo INLINE.

53)Too many EXTRN definitions (Demasiadas definiciones de tipo EXTRN).

Turbo Pascal no puede manejar archivos .OBJ con más de 256 definiciones EXTRN.

54)OF expected (Se espera la palabra OF).

55)INTERFACE expected (Se espera la palabra INTERFACE).

56) Invalid relocatable reference (Referencia relocalizable inválida).

Esto puede deberse a que:

- El archivo .OBJ contiene datos y referencias relocalizables en segmentos diferentes al CODE. Por ejemplo, se está intentando declarar variables inicializadas en el segmento DATA.

- El archivo .OBJ contiene referencias medidas en bytes para símbolos relocalizables. Este error ocurre cuando se usan los operadores HIGH y LOW con símbolos relocalizables, o cuando se refieren los símbolos relocalizables en directivas.

- Un operando refiere a un símbolo relocalizable que no fue definido en el segmento CODE o en el segmento DATA.

- Un operando refiere a un procedimiento o función EXTRN con un offset, por ejemplo, CALL MiFuncion + 8.

57)THEN expected (Se espera la palabra THEN).

58)TO or DOWNTO expected (Se espera la palabra TO o DOWNTO).

59)Undefined FORWARD (FORWARD indefinido).

Esto puede deberse a que:

- El procedimiento o función fue declarado en la parte INTERFACE de una unidad, pero su definición nunca ocurrió en la parte de la implementación.

- El procedimiento o función fue declarado con un FORWARD, pero su definición nunca se encontró.

60)Too many procedures (Demasiados procedimientos).

Turbo Pascal no permite más de 512 procedimientos o funciones por módulo. Si está compilando un programa, cambie algunos procedimientos o funciones en una unidad. Si está compilando una unidad, particiónela en dos o más unidades.

61)Invalid typecast (Tipo forzado inválido).

Esto puede deberse a que:

- Los tamaños de la variable referencial y el tipo destino difieren en la variable de tipo forzado.

- Está intentando forzar el tipo de una expresión, cuando solamente se permite referenciar a una variable.

62)Division by zero (División entre cero).

63)Invalid file type (El tipo del archivo es inválido).

El procedimiento que maneja el archivo no soporta archivos de este tipo. Por ejemplo, no se puede utilizar READLN en un archivo con registros o SEEK en un archivo de texto.

64)Cannot READ or WRITE variables of this type (No se puede utilizar un READ o un WRITE con variables de este tipo).

Esto puede deberse a que:

- READ y READLN pueden aceptar variables de tipo CHAR, INTEGER, REAL y STRING.

- WRITE y WRITELN pueden aceptar variables de tipo CHAR, INTEGER, REAL, STRING y BOOLEAN.

65)Pointer variable expected (Se espera una variable de tipo apuntador).

La variable anterior debe ser de tipo apuntador.

66)String variable expected (Se espera variable de tipo STRING).

La variable anterior debe ser de tipo STRING.

67)String expression expected (Se espera una expresión STRING).

La expresión anterior debe ser de tipo STRING.

68)Circular unit reference (Referencia circular en unidad).

Dos unidades no pueden usarse la una a la otra.

69)Unit name mismatch (Inconcordancia en el nombre de la unidad).

El nombre de la unidad en el archivo .TPU no concuerda con el nombre que se usó en la cláusula USES.

70)Unit version mismatch (Versión incorrecta de unidad).

Una o más unidades se han modificado desde que la unidad fue compilada. Utilice la opción COMPILE/MAKE o COMPILE/BUILD para compilar automáticamente aquellas unidades que lo necesiten.

71)Duplicate unit name (Nombre de unidad duplicado).

El nombre de la unidad ya se está usando en otra parte.

72)Unit file format error (Error en el formato del archivo de la unidad).

El archivo .TPU contiene un error en alguna de sus partes. Revise las unidades involucradas.

73)Implementation expected (Se espera la palabra IMPLEMENTATION).

La palabra reservada IMPLEMENTATION es parte del desarrollo de unidades en Turbo Pascal.

74)Constant and case do not match (La constante y el estatuto case no concuerdan).

75)Record variable expected (Se espera una variable de tipo registro).

76)Constant out of range (La constante está fuera de rango).

77)File variable expected (Se espera una variable de tipo FILE).

78)Pointer expression expected (Se espera una expresión de tipo POINTER).

79)Integer or real expression expected (Se espera una expresión de tipo INTEGER o REAL).

80)Label not within current block (La etiqueta no está en el bloque actual).

Un estatuto goto no puede referenciar a una etiqueta fuera del bloque actual.

81)Label already defined (La etiqueta ya está definida).

82)Undefined label in preceding statement part (Etiqueta indefinida en el estatuto anterior).

La etiqueta fue declarada y referenciada en el estatuto anterior, pero nunca fue definida.

83)Invalid @ argument (Argumento @ inválido).

Los argumentos válidos son referencias de variables e identificadores de funciones y/o procedimientos.

84)UNIT expected (Se espera la palabra UNIT).

85)";" expected (Se espera un ";")

86)":" expected (Se espera un ":")

87)"," expected (Se espera un ",")

88)"(" expected (Se espera un "(")

89)")" expected (Se espera un ")")

90)"=" expected (Se espera un "=")

91)":=" expected (Se espera un ":=")

92)"[" or "(." expected (Se espera un "[" o un "(.")

93)"]" or ".)" expected (Se espera un "]" o un ".)")

94)"." expected (Se espera un ".")

95)".." expected (Se espera un "..")

96)Too many variables (Demasiadas variables)

Esto puede deberse a que:

- La cantidad total de variables globales declaradas en un programa o una unidad no puede exceder a 64K.

- La cantidad total de variables locales declaradas en un procedimiento o una función no puede exceder a 64K.

97)Invalid FOR control variable (La variable para el control del ciclo FOR no es válida).

La variable de control del estatuto FOR debe ser una variable simple definida en la sección de declaración del subprograma actual.

98)Integer variable expected (Se espera una variable entera).

La variable anterior debe ser del tipo INTEGER.

99)Files are not allowed here (Aquí no se permiten archivos).

Una constante con tipo forzado no puede ser de tipo FILE.

100)String length mismatch (Inconcordancia en longitudes del string).

La longitud de una constante string no concuerda con el número de componentes del arreglo de caracteres.

101)Invalid ordering of fields (Orden de campos inválido).

Los campos de un registro deben escribirse en el orden en que fueron declarados.

102)String constat expected (Se espera una constante STRING).

103)Integer or real variable expected (Se espera una variable del tipo INTEGER o REAL).

La variable precedente debe ser del tipo INTEGER o REAL.

104)Ordinal variable expected (Se espera una variable ordinal).

La variable precedente debe ser de tipo ordinal.

105)INLINE error (Error en estatuto INLINE).

El operador no se permite junto con referencias a variables relocalizables.

106)Character expression expected (Se espera una expresión de tipo caracter).

La expresión precedente debe ser de tipo CHAR.

107)Too many relocation items (Demasiados elementos relocalizables).

El tamaño de la tabla de relocalizaciones en un archivo .EXE excede de 64K, que es el límite superior de Turbo Pascal. Este error indica que el programa es muy grande para el programa LINK residente del sistema. Lo más recomendable es particionar el programa en un padre y un hijo para el sistema DOS, a través de la instrucción EXEC que está en la unidad DOS.

108)Not enough memory to run the program (No hay memoria suficiente para ejecutar el programa).

No hay memoria suficiente en el sistema de Turbo Pascal. Si se están utilizando programas residentes (SideKick o SuperKey, por ejemplo), quítelos del RAM. Si esto no ayuda, compile el programa a disco y finalice el programa TURBO para ejecutarlo.

109)Cannot find EXE file (No se encontró el archivo EXE).

Por alguna razón desapareció el archivo .EXE previamente generado por el compilador.

110)Cannot run a unit (No se puede ejecutar una unidad).

No es posible ejecutar una unidad. Para probar alguna, es necesario crear un programa que la utilice.

111)Compilation aborted (Compilación abortada).

El proceso de compilación se abortó al presionar Ctrl-Break.

112)CASE constat out of range (La constante del CASE está fuera de rango).

113)Error in statement (Error en el estatuto).

Un estatuto no puede empezar con un símbolo.

114)Cannot call an interrupt procedure (No es posible ejecutar el procedimiento de interrupción).

No es posible llamar directamente a un procedimiento de interrupción.

115)Have an 8087 to compile this (Es necesario tener el 8087 para compilar).

El compilador requiere del coprocesador 8087 para compilar programas y/o unidades en el estado de {N+}.

116)Must be in 8087 mode to compile this (Se debe tener el modo 8087 para compilar esto).

Esta construcción solamente se puede compilar con el coprocesador matemático 8087 en el sistema y con la directiva {N+}. Las operaciones con el 8087, con tipos SINGLE, DOUBLE, EXTEND y COMP, no se permiten con la directiva {N-}

117)Target address not found (No se encontró la dirección destino).

El comando /Find error no puede localizar un estatuto que corresponda a la dirección especificada.

118)Include files are not allowed here (No se permite inclusión de archivos aquí).

119)TPM file format error (Error en el formato del archivo TPM).

El archivo TPM tiene un error en alguna parte.

120)NIL expected (Se espera un NIL).

121)Invalid qualifier (Cualificador inválido).

Esto puede deberse a que:

-Está tratando de indexar una variable que no es un arreglo dimensionado.

-Está tratando de especificar un campo en una variable que no es un registro.

-Está tratando de referenciar una variable que no es un apuntador.

122)Invalid variable reference (Referencia a una variable inválida).

La construcción precedente sigue la sintaxis de una referencia a variable, pero no denota una localidad de memoria. Generalmente se llama una función de apuntador, pero se ha olvidado referenciar el resultado.

123)Too many simbols (Demasiados símbolos)

El programa o la unidad declara más de 64K de símbolos.

124)Statement part too large (Estatuto muy grande).

El límite de Turbo Pascal para el tamaño de un estatuto es de 24K. Particione la sección en dos o más procedimientos.

125)Module has no debug information (El módulo no tiene información para depurarse).

Ocurrió un error de ejecución en el módulo (programa o unidad), el cual no tiene información para depurarse; por esta razón Turbo Pascal no puede mostrar el estatuto correspondiente.

126)Files must be VAR parameters (Los archivos deben ser parámetros de tipo VAR).

Se está intentando declarar un archivo como parámetro de tipo valor. Los archivos, cuando son parámetros, deben ser del tipo VAR.

127)Too many conditional symbols (Demasiados símbolos condicionales).

No hay espacio suficiente para definir el resto de los símbolos condicionales. Intente eliminar algunos o recortar el nombre de ellos.

128)Misplaced conditional directive (Directiva condicional fuera de lugar).

El compilador encontró un {$ELSE} o {$ENDIF} sin su correspondiente {$IFDEF}, {$IFNDEF} o {$IFOPT}.

129)ENDIF directive missing (Directiva ENDIF perdida).

El archivo fuente terminó dentro de la construcción de una compilación condicional. Debe haber un número igual de {$IFxxx} y {$ENDIF} en el archivo fuente.

130)Error in initial conditional define (Error en la condición inicial).

Los símbolos especificados en la condición inicial son inválidos. Turbo Pascal utiliza cero o más identificadores separados por comas, espacios o punto y coma.

131)Header does not match previous definition (El encabezado no concuerda con la definición previa).

Esto puede deberse a que:

-El encabezado del procedimiento o función que se especificó en la sección INTERFACE no concuerda con el encabezado.

-El encabezado del procedimiento o función que se especificó en la declaración FORWARD no concuerda con el encabezado.

132)Critical disk error (Error de disco crítico).

Ocurrió un error crítico de disco durante la compilación; por ejemplo, error de disponibilidad del drive.

133)Old map file (Archivo MAP viejo).

El archivo .TPM es más viejo que el correspondiente .EXE.

Esto indica que la última vez que se compiló el programa, no se produjo el .TPM.

ERRORES DE EJECUCION

Ciertos errores de ejecución causan que el programa termine y se despliegue en el video el siguiente mensaje de error: Runtime error nnn at xxxx:yyyy, donde nnn es el número de error y xxxx:yyyy es la dirección de dicho error (segmento y offset).

Los errores de ejecución están divididos en las categorías siguientes:

Errores del DOS 1-99

Errores de I/O 100-149

Errores críticos 150-199

Errores fatales 200-255.

ERRORES DEL DOS.

2)File not found (No se encontró archivo).

Reportado por RESET, APPEND, RENAME o ERASE. El nombre asignado a la variable archivo no especifica un archivo existente.

3)Path not found (No se encontró camino).

Esto puede deberse a lo siguiente:

-Reportado por RESET, REWRITE, APPEND, RENAME o ERASE, si el nombre asignado a la variable FILE es inválido o apunta a un subdirectorio inexistente.

-Reportado por ChDir, MkDir o RmDir, si el camino (PATH) es inválido o apunta a un subdirectorio inexistente.

4)Too many open files (Demasiados archivos abiertos).

Reportado por RESET, REWRITE o APPEND si el programa utiliza demasiados archivos. El DOS nunca permite más de 15 archivos abiertos por proceso. Si se tiene este error con menos de 15 archivos abiertos, es indicativo de que el archivo CONFIG.SYS no incluye el estatuto FILES = xxx, o especifica muy pocos.

5)File access denied (Acceso al archivo denegado).

Esto puede deberse a lo siguiente:

-Reportado por RESET o APPEND si el modo de operación es de escritura y el nombre asignado a la variable FILE apunta a un directorio o a un archivo exclusivo para lectura.

-Reportado por REWRITE si el directorio está lleno o si el nombre asignado a la variable FILE apunta a un directorio o a un archivo exclusivo para lectura.

-Reportado por RENAME si el nombre asignado a la variable FILE apunta a un directorio, o si el nombre nuevo especifica un archivo ya existente.

-Reportado por ERASE si el nombre asignado a la variable FILE apunta a un directorio o a un archivo exclusivo de lectura.

-Reportado por MKDIR si un archivo con el mismo nombre existe en el directorio padre, si no hay espacio en el directorio padre, o si el camino (PATH) especifica un dispositivo.

-Reportado por RMDIR si el directorio no está vacío, si el camino (PATH) no especifica a un directorio o si el camino especifica al directorio raíz.

-Reportado por READ o BLOCKREAD en un archivo con tipo o sin tipo, si el archivo no está abierto para lectura.

-Reportado por WRITE o BLOCKWRITE en un archivo con tipo o sin tipo, si el archivo no está abierto para escritura.

6)Invalid file handle (Apuntador de archivo inválido).

Este error es reportado si el apuntador numérico de archivo es inválido y se pasa al sistema DOS.

12)Invalid file access code (Código de acceso a archivo inválido).

Reportado por RESET o APPEND en un archivo con tipo o sin tipo, si el valor del modo de operación es inválido.

15)Invalid drive number (Número de drive inválido).

Reportado por GETDIR si el número de drive es inválido.

16)Cannot remove current directory (No se puede borrar el directorio actual).

Reportado por RMDIR si el camino (PATH) especifica al directorio actual.

17)Cannot rename acrooss drives (No se puede renombrar en diferentes lectores de disco).

Reportado por RENAME si ambos nombres no están en el mismo drive.

ERRORES DE I/O.

Estos errores causan la terminación si el estatuto en particular fue compilado con la directiva {$I+}. Si la directiva es {$I-}, el programa continúa la ejecución y el error se reporta en la función IORESULT.

100)Disk read error (Error de lectura en disco).

Reportado por READ en un archivo con tipo, si se intenta leer mas allá del final del archivo.

101)Disk write error (Error de lectura en disco).

Reportado por CLOSE, WRITE, WRITELN, FLUSH o PAGE, si el disco está lleno.

102)File not assigned (Archivo no asignado).

Reportado por RESET, REWRITE, APPEND, RENAME y ERASE, si la variable de tipo FILE no ha sido asignada a un nombre a través de una llamada al procedimiento ASSIGN.

103)File not open (Archivo sin abrir).

Reportado por CLOSE, READ, WRITE, SEEK, EOF, FILEPOS, FILESIZE, FLUSH, BLOCKREAD o BLOCKWRITE, si el archivo no ha sido abierto.

104)File not open for input (Archivo no abierto para lectura).

Reportado por READ, READLN, EOF, EOLN, SEEKEOF o SEEKEOLN en un archivo de texto, si dicho archivo no ha sido abierto para entrada.

105)File not open for output (Archivo no abierto para escritura).

Reportado por WRITE, WRITELN y PAGE en un archivo de texto, si dicho archivo no ha sido abierto para salida.

106)Invalid numeric format (Formato numérico inválido).

Reportado por READ o READLN si el valor numérico leído del archivo de texto no está conformado en el formato numérico apropiado.

ERRORES CRITICOS

150)Disk is write protected (El disco está protegido contra escritura).

151)Unknown unit (Unidad desconocida).

152)Write not ready (El disco no está listo).

153)Unknown command (Comando desconocido).

154)CRC error in data (Error CRC en datos).

155) Bad drive request structure length (Requerimiento de longitud de estructura del drive equivocada).

156)Disk seek error (Error en la localización en el disco).

157)Unknown media type (Tipo de media desconocido).

158)Sector not found (No se encontró sector).

159)Printer out of paper (Impresora sin papel).

160)Device write fault (Error al escribir en dispositivo).

161)Device read fault (Error al leer en dispositivo).

162)Hardware failure (Falla física).

ERRORES FATALES

Estos errores terminan inmediatamente el programa.

200)Division by zero (División entre cero).

201)Range check error (Error al revisar rango).

Este error lo reportan los estatutos compilados en el estado {$R+}, cuando ocurre cualquiera de las situaciones siguientes:

-La expresión indexada de un arreglo está fuera de rango.

-Se intentó asignar a una variable un valor que está fuera de rango.

-Se intentó pasar un valor fuera de rango, como parámetro a un procedimiento o a una función.

202)Stack overflow error (Error de sobreflujo en el stack).

Este error lo reporta la entrada de un procedimiento o función compilado en el estado {$S+}, cuando no se encontró espacio suficiente para almacenar todas las variables locales del subprograma. Incremente el tamaño del stack usando la directiva del compilador $M.

203)Heap overflow error (Error de sobreflujo en la memoria para variables).

Este error lo reporta NEW o GETMEM cuando no hay espacio suficiente para almacenar un bloque del tamaño requerido.

204)Invalid pointer operation (Operación de apuntadores inválidos).

Este error es reportado por DISPOSE o FREEMEM si el apuntador es nulo, o si apunta a una localidad fuera del HEAP, o si la lista no puede expandirse.

205)Floating point overflow (Sobreflujo en operación de punto flotante).

206)Floating point underflow (Bajoflujo en operación de púnto flotante).

Una operación de punto flotante puede producir bajoflujo.

Este error se reporta si se está usando el 8087 con una palabra de control que no enmascare las excepciones de bajo flujo. Por default, un bajo flujo da como resultado un cero.

207)Invalid floating point operation (Operación con punto flotante inválida).

Esto puede deberse a que:

-El valor real pasado a la función TRUNC o ROUND no puede ser convertido a entero dentro del rango (-2147483648 a 2147483647).
-El argumento pasado a la función SQRT fue negativo.
-El argumento pasado a la función LN fue cero o negativo.
-Ocurrió un error de sobreflujo en el STACK del 8087.

APENDICE D.- COMANDOS DEL EDITOR

ORDENES DEL EDITOR

En la tabla se resumen la totalidad de las órdenes que comprenden al editor deTurbo Pascal. A continuacíon, este apéndice describe los detalles de cada orden. La tabla se utiliza la notación siguiente: Una palabra o

unas palabras en mayúsculas, tales como LEFT ARROW, indica que debe utilizar una "tecla unica" del nombre dado. Cuando un circumplejo (^) precede a una letra, tal como ^S, debe mantener pulsada la tecla CTRL cuando pulse la tecla coorespondiente a la letra. Cuando dos letras siguen a un circunflejo, tal como ^OS, debíra mantener pulsada CTRL cuando pulse la tecla para la primera letra y de manera opcional,puede mantener pulsada la tecla CTRL cuando pulse la tecla para la segunda letra. Aunque todas las órdenes se muestran en la letras mayúsculas, pueden utilizarse también las minúsculas

ORDENES PARA DESPLAZAR EL CURSOR	ORDEN
A la izquierda un caracter	^S o FLECHA IZQUIERDA
A la derecha un caracter	^d o FLECHA DERECHA
A la izquierda al comienzo de palabra	^A o ^FLECHA IZQUIERDA
A la derecha al comienzo de palabra	^F o ^FLECHA DERECHA
A la izquierda al comienzo de línea	^QS o HOME
A la derecha a final de la línea	^QD o END
Arriba en una línea	^E o FLECHA ARRIBA
Abajo en una línea	^X o FLECHA ABAJO
Arriba a la primera línea en la ventana	^QE o ^HOME
Arriba a la última línea en la ventana	^QX o ^END
Arriba a una nueva página	^P o PgUp
Abajo a una nueva página	^C o PgDn
Desplazamineto arriba una línea	^W
Desplazamiento abajo una línea	^Z
Desplazamiento a comienzo de bloque	^QB
Desplazamiento a final de bloque	^QK

DESPLAZAMIENTO DEL CURSOR	ORDEN
Desplazamiento a última posición del cursor	^QP
Estacionamiento marcador lugar n (n=0-3)	^Kn(no ^n)
Determinación marcador lugar n (n=0-3)	^Qn(no ^n)
Ordenes para suprimir texto	
Suprimir caracter izquierda del cursor	^H o RETROCESO
Suprimir caracter en la posición del cursor	^G o DEL
Suprimir desde cursor a final de palabra	^T
Suprimir desde cursor a final de línea	^QY
Suprimir línea completa	^Y
Ordenes para en bloque de texto	
Marcar comienzo de bloque	^KB o F7
Marcar final de bloque	^KK o F8
Marcar una palabra	^KT
Imprimir bloque marcado	^KP
Suprimir bloque marcado	^KY
Copiar bloque marcado en posición cursor	^KC
Desplazar bloque marcado	^KV
Escribir bloque marcado en archivo disco	^KW
Leer archivo en disco como un bloque	^KR
Esconder o visualizar bloque marcado	^KH
Comutaciones de modos	
Modo de sangrado automático act o desact	^OI
Modo de inserción activado o desactivado	^V o INT
Modo de tabulación	^OT
Ordenes de búsqueda y sustitución	
Búsqueda de texto	^OF
Búsqueda y sustitución de texto	^QA
Repetición última búsqueda o sustitución	^L

DESPLAZAMIENTO DEL CURSOR	ORDEN
Ordenes para guardar, cargar y abandonar	
Guardar programa actual en disco archivo	^KS o F2
Cargar nuevo programa desde disco archivo	F3
Salir del editor al menú principal	^KD, ^KQ, ^KX o ESC
Ordenes diversas	
Operaciones de terminación anormal ("abortar")	^U
Introdución de carácter de control	^P
Desplazar fin de línea a nueva línea	^N
Tabular	^I o TAB
Restaurar menensaje error	^QW
Restaurar línea	^QL

EN ESTA SECCION SE DESCRIBEN LAS ORDENES DEL EDITOR DE TURBO PASCAL QUE DESPLAZAN EL CURSOR.

A la izquierda en un carácter (^S o FLECHA IZQUERDA) Esta orden desplaza el cursor en un caracter a la izquierda. Turbo Pascal ignorará esta orden cuando el cursor esté al comienzo de la línea.

A la derecha de un caracter (^D o FLECHA DERECHA) Esta orden desplaza el cursor en un caracter a la derecha. No considerará el último carácter no en blanco en una línea como el final de dicha línea, sino que continuará desplazando el cursor a la derecha hasta que alcance la longitud de línea máxima (248 caracteres)

A la izquierda al comienzo de palabra (^A o ^FLECHA IZQUIERDA) Esta orden desplaza el cursor al comienzo de la primera palabra a la izquierda. Una palabra se define como una secuencia de caracteres que se produce entre cualquier pareja de los 17 separadores siguientes: espacio $'( ) * + - / \ : [ ] ^.

Si el cursor está en una palabra, pero no al principio de ella, se desplazará al comienzo de esta palabra. Si el cursor no está en una palabra,se desplazará al comienzo de la primera palabra que está a la izquierda. Si ninguna palabra está a la izquierda del cursor, y si una línea esta por encima de la línea actual, el cursor se desplazará a la posición que sigue al último carácter no en blanco de la línea siguente.

A la derecha al comienzo de palabra (^ F o ^FLECHA DERECHA) Esta orden desplaza el cursor al comienzo de la primera palabra a la derecha. Si el cursor está en la última palabra de una línea, se desplazará a la posición que sigue a la línea actual, el cursor se desplazará al comienza de la primera palabra en la línea siguiente.

A la izquierda al comienza de línea (^QS o HOME) Esta orden desplaza el cursor la a la columna 1 de la línea actual.

A la derecha al final de línea (^QD o END) Esta orden desplaza el cursor a la posición que sigue al ultimo carácter no en blanco en la línea. En algunos casos, esta orden puede desplazar el curso a la izquierda.

Arriba en una línea (^E o FLECHA ARRIBA) Si existe una línea por encima de línea actual, esta orden desplaza el cursor hasta esa línea. La orden preserva la columna del cursor y si encuentra un carácter de tabulación, el cursor se deplaza mas a la derecha.

Abajo en una línea (^X o FLECHA ABAJO) Si existe una línea por debajo de la línea actual, esta desplaza el cursor a dicha línea. La orden preserva la columna del cursor,no ser que encuentre un carácter de tabulación, en cuyo caso el cursor se desplazará más a la derecha.

Arriba a la primera línea en la ventana (^QE o ^HOME) Esta orden desplaza el cursor a la primera línea visible en la ventana de edición. La orden preserva la columna del cursor, a no ser que encuentre un carácter de tabulación, en el cursor se que encuentre un carácter de tabulación, en cuyo caso el cursor se desplazará más a la derecha.

Abajo a la ultima línea en la ventana (^QX o ^END) Esta orden desplaza el cursor a la última línea visible en la ventana de edición. La orden preserva la columna del cursor, a no ser que encuentre un carácter de tabulación, en cuyo caso el cursor se desplazará más a la derecha.

Arriba a una nueva página (^R o ^PgUn) Esta orden visualiza una sección anterior del programa en la ventana de edición. En general, lo que era la primera línea visible en la ventana pasa a ser la última línea en la misma.

Abajo a una nueva página (^R o PgDn) Esta orden visualiza la última un sección del programa en la ventana de edición. por la general,lo que era la última línea visible superior de la edición.

Abajo a la última línea de programa (^QR o ^PgUp) Esta orden desplaza el cursor al comienzo de la primera línea y visualiza esta línea en la parte superior de la ventana de edición.

Desplazamineto hacia arriba en una línea (^W) Si existen varias líneas de programa por encima de la primera línea visualizada en la ventana de edición, esta orden desplazará cada línea en dicha ventana abajo en una línea para hacer aparecer un nueva línea del programa por la parte superior de la ventana. Preserva la posición del cursor, a no ser que el cursor esté en la líneas visible en la ventana de edición.

Desplazamiento hacia abajo en una línea (^Z) Si existe más líneas de programa por debajo de la última línea visualizada en la ventana de edición, esta orden desplazará cada línea en la ventana hacia arriba en una línea para hacer aparecer una nueva línea del programa por la parte interior de la ventana. Preserva la posición del cursor, a no ser que el cursor que en la primera línea visible en la ventana de edición.

Desplazamiento al comienzo del bloque (^QB) Si marcó el comienzo de un bloque de texto anterior, esta orden desplazará el cursor a la posición de esta marca de "comienzo bloque".

Desplazamiento a final de bloque (^QK) Si marcó el final de un bloque de texto anterior, esta orden desplazará al cursor a la posición de esta marca de "fin de bloque".

Desplazamiento a la última posición del cursor (^QP) Esta orden desplaza el cursor a la posición en donde estaba antes de que emitiera la última orden o tecleara el último caracter. Es de útilidad después de que emita una orden de movimiento del cursor, incluyendo ^QF, para volver a la posición en la estaba trabajando.

Establecer marcador en el lugar n(^Kn) Esta orden coloca un marcador temporal en la posición del cursor actual, pudiendo ser n cualquiera de los 4 digitos simples de 0 a 3. (nota: No hay que mantener pulsada la tecla CTRL cuando sepulse la tecla del digito.) Despues de establecer un marcador temporal, puede retomar imediatamente desde cualquier otro lugar en el programa a la posición en donde esta el marcador utilizando la orden Encontrar Marcador Lugar n con el valor adecuado de n. Puede desplazar los marcadores de lugar simplemente fijándoles en una posicion nueva. Turbo Pascal no guarda los marcadores con el proceso.

Encontrar Marcador Lugar n(^Qn) Esta orden desplaza el cursor a la posición en donde estableció el último marcador de lugar n, pudiendo ser n cualquier de los 4 digitos individuales 0 a 3.

ESTA SECCION SE DESCRIBE LAS ORDENES DE EDITOR DE TURBO PASCAL QUE SUPRIME TEXTO.

Suprimir caracter a la izquierda del cursor (^H o RETROCESO) Esta orden suprime el carácter immediatamente a la izquierda del cursor. Si el cursor está al principio de la línea y hay una línea por encima de la actual, esta une a la línea actual al final de la línea anterior.

Suprimir caracteres cursor (^G o DEL) Esta orden suprime todos los caracteres a partir del carácter situado en la posición del cursor actual. Si el cursor esta situado en un carácter del espacio, la orden suprime todos los espacios desde la posición del cursor al siguiente carácter en la línea actual y hay una línea debajo de la línea actual, esta orden une la línea siguiente al final de la línea actual.

Suprimir desde el cursor hasta fin de línea (^QY) Esta orden suprime todos los caracteres desde el caracter situado en la posición del actual al último caracter en la línea.

Suprimir línea completa (^Y) Esta orden suprime todos los caracteres en la línea que contiene el cursor y desplaza las líneas posteriores hacia arriba para rellenar el hueco.

EN ESTA SECCION SE DESCRIBEN LAS ORDENES DEL EDITOR DE TURBO PASCAL QUE ACTUA SOBRE BLOQUES DE TEXTO.

Marcar el comienzo de un bloque (^KB) Esta orden marcá el caracter situado en la posición actual del cursor como el primer caracter en un bloque de texto. Cuando marca también el final del bloque, esta orden destaca visualmente el texto dentro del bloque. Puede impartir, desplazar o suprimir bloques marcados de texto en uma sola operación, según se describe más adelante en esta misma sección.

Marcar al final de un bloque (^KK) Esta orden marca el carácter situado inmediatamente a la izquierda de la posición actual del cursor como el último carácter en un bloque de texto. Cuando marque también el comienzo del bloque, al orden destacara visualmente al texto dentro del bloque. Puede imprimir, desplazar, copiar o suprimir bloques marcados de texto en una solo operación, segun se describe mas adelante en eata sección.

Marca una palabra que contenga el cursor como bloque (^KT) Esta orden marca la palabra que contiene el cursor como un bloque. Si el cursor no esta dentro de una palabra, esta orden marca la primera palabra a la derecha del cursor.

Imprimir bloque marcado (^KP) Esta orden imprime, en la impresora, el texto dentro del bloque acualmente marcado. Si no marca un bloque o si bloque marcado esta oculto, la orden imprimirá el programa completo en memoria.

Suprimir bloque marcado(^KY) Esta orden suprime de la memoria el bloque detrexto completo que marco en el programa actual.

Copiar bloque marcado en posición del cursor (^KC) Esta orden copia el, bloquede texto que marcó en el programa actual hasta el punto en donde reside actualmente el cursor. Cualquier texto que esté en la posición del cursor o después de ella se empuja hacia abajo para dejar espacio para el bloque desplazado. El cursor se deja al momienzo del bloque desplaza.

Escribir bloque marcado a posicion del cursor (^KV) Esta orden desplaza el bloque de texto que marcó en el programa actual al punto en donde reside actualmente el cursor y suprime el bloque de texto desde su posición original. La orden impulsa hacia abajo cualquier texto que este en la posición del cursor o después de ella para dejar espacios para el bloque desplazado. El cursor deja al comienzo del bloque desplazado.

Escribir bloque marcado en archivo de disco (^KW) Esta orden copia el bloque de texto que marco en el programa actual en un archivo de disco y dide el nombre del archivo a crear (o sobreescribir). Si existe el archivo de disco, la orden no afecta al bloque del texto originalmente marcado.

Lectura de archivo desde disco un bloque (^KR) Esta orden solicita el nombre de un archivo de disco y luego inserta el contenido de este archivo en el programa actual en la posición del cursor en ese momento. La orden impulsa hacia abajo cualquier texto que este en la posición del cursor, o después de ella, para hacer espacio para el texto insertado desde el archivo de disco. La orden marca automáticamente el texto insertado como un bloque y el cursor quedara al comienzo de este bloque. Esta no afectará al archivo de disco desde que fue leído el texto.

Ocultar o visualizar bloque marcado (^KH) Si un bloque marcado de texto se esta visualizado en el programa actual, esta orden oculta el hecho de que exista. Podrá ejecutar órdenes de bloques, tales como copia de bloque oculto. Si un bloque marcado existe pero esta oculto, esta orden volvera a destacar visualmente el bloque y le permitirá ejecutar órdenes de bloques.

ESTA SECCION SE DESCRIBEN LAS ORDENES DE EDITOR DE TURBO PASCAL QUE CONMUTAN LOS DIVERSOS MODOS.

Modo sangrado automático activado/desactivado (^OI) Esta orden cambia el estado del modo de sangrado automático desde desactivado (OFF) a activado (ON) o viceversa. Cuando el modo de sangrado automático esta activado, Turbo Pascal proporciona automáticamente el cursor en la misma columna que el primer carácter no en blanco de la línea anterior siembre que puse <Enter> para crear una nueva línea (Nota: La pulsación de <Enter> puede crear una nueva línea solamente si esta activado el modo de inserción)

Modo de inserción activado/ desactivado (^V o INS) Esta orden cambia el estado del modo de inserción desde desactivado (OFF) a activado (ON) o viceversa. Cuando el modo de insercion está activado, Turbo Pascal desplaza el texto existente a la derecha para dejar espacio para el nuevo texto que está introduciendo. Cuando el modo de inserción está desactivado. el texto recientemente introducido sustituye o sobreescribe el texto existente. (Nota: Puede añadir nuevas líneas a la parte inferior de un programa solamente cuando el modo de inserción ésta activado.)

Modo de tabulación activado/desactivado(^OT) Esta orden cambia el estado del modo de tabulación desde desactivado a activado o viceversa. Cuando el modo de tabulación está activado, podra utilizar TAB para colocar tabuladores en su programa con miras a una mejor legilibilidad. Turbo Pascal preestablece topes de tabulación en la columna 9,17,25,33,41, etc. Cuando el modo de tabulación está desactivado, Turbo Pascal ignora el empleo de los tabuladores.

EN ESTA SECCION SE DESCRIBEN LAS ORDENES DEL EDITOR DE TURBO PASCAL QUE EFECTUA LAS OPERACIONES DE BUSQUEDA Y SUSTITUCION.

Búsqueda de texto (^QF) Esta orden solicita una cadena de caracters a buscar, pide las opciones de búsqueda y encuentra la cadena en el programa. Las opciones de búsqueda puede ser cualquier número, o ninguno de los siguientes:

B Búsqueda hacia atras a través del programa.

G Busqueda global (a través del programa completo) en lugar de solamente la sección después (o antes) de la posicón actual del cursor.

n Ignorar las n-1 primeras condiciones para la cadena de búsqueda y desplaza el cursor a la enésima coincidencia encontrada. Por ejemplo,si se da 4 como una opción de búsqueda. Turbo Pascal ignora las 3 primeras coincidencias con la cadena de busqueda y desplazá el cursor a la cuarta coincidencia

U Ignora si las letras son mayúsculas o minúsculas cuando se compueba una coincidencia con la cadena de búsqueda. Por ejemplo,si da la opción U, una búsqueda de tiempo establecerá una coincidencia con Tiempo, TIEMPO, etcetera.

W Coincidencia de la cadena de búsqueda con las palabras completas solamente.Turbo Pascal no establecerá una coincidencia cuando la cadena búsqueda sea parte de una palabra más grande. Por ejemplo, si da la opción W, una búsqueda de "end" no será coincidente con "ends", "send" o "friend".

Dos ejemplos de opciones de búsqueda son: GB para búsqueda desde el final al principio del programa de una coincidencia de la cadena de búsqueda y 3W para la búsqueda hacia adelante desde las posición actual del cursor con respecto a la tercera palabra completa que coincida con la cadena de búsqueda.

Busqueda y sustitución de texto (^QA) Esta orden una cadena de caracteres, pide una cadena de caracteres de sustitución, solicita las opciones de búsqueda y sustitución, encuentra la cadena y búsqueda en el programa de acuerdo con las opciones de búsqueda y sustituye la cadena coincidente por la cadena de sustitución. Las opciones de Búsqueda son las que se describieron para la orden de búsqueda de texto. Tambien se dispone de la opción de sustitución siguiente:

N Realizar la sustitución sin pedir primero una comprobación de que debe sustituirse la cadena encontrada.

Repetir última búsqueda o sustitución (^L) Esta orden ejecuta de nuevo la últina orden de búsqueda y sustitución con el ejemplo de la misma cadena de búsqueda, cadena de sustitución (si fuera adecuado) y opcion.

EN ESTA SECCION SE DESCRIBEN LAS ORDENES DEL EDITOR DE TURBO PASCAL PARA LAS OPERACIONES DE GRABAR, CARGAR Y SALIR.

Guardar programa actual en archivo de disco (^KS o F2) Esta orden guarda el programa acualmente en memoria en una archivo de disco con el nombre del programa. Sin el nombre del programa es NOMAME.PAS la orden n.

Cargar nuevo programa desde archivo de disco (F3) Esta orden pregunta el nombre de un archivo de disco a cargar en memoria. Si no ha guardado el programa actual en memoria, la orden le proporciona la opcion de hacerlo. Cualquier texto acualmente en memoria se perderá cuando la orden carge el nuevo archivo en memoria.

EN ESTA SECCION SE DESCRIBE LAS RESTANTES ORDENES DEL EDITOR DE TURBO PASCAL.

Abortar operación (^U) Esta orden interrumpe la ejecución de la orden que se esté introduciendo o ejecutando en ese momento. Debe pulsar ESC a continuación. Esta orden es de utilidad sobre todo para interrumpir una orden de búsqueda y sustitución global antes de que acabe.

Introducir carácter de control (^P) Esta orden interpreta el siguente carácter de control, en lugar de como un carácter estándar. Por ejemplo, el editor interprerta la pulsación CTRL-P y luego el teclado de M como carácter ^M (carácter ASCII 13). Ha de utiliza esta orden sobre todo cuando espesifique cadenas de búsqueda y sustitución. Permite que las cadenas contengan caracteres de control, tales como el carácter de tablulación ^I y los caracters de linea ^M ^J.

Desplazar fin de línea nueva línea (^N) Esta orden crea una nueva línea debajo de las líneas actuales y desplaza cualquier texto que esté en la posicion del cursor, o a su derecha, al comienzo de esta nueva línea.

Tabulación (^i o TAB) Si el modo de tabulación esta activado, esta orden coloca un caráctrer de tabulación en el programa y desplaza el cursor al siguiente tope de tabulación. Turbo Pascal preestablece topes de tabulación en las columnas 9,17,23,33,41, etcétera.

Restaurar mensaje de error (^QW) Esta orden vuelve a visualizar el último mensaje de error del compilador en la parte superior de la ventana de edicion y desplaza el cursor al lugar en el texto de programa en donde se produjo este error.

Restaurar línea (^QL) Esta orden restaura la línea actual a su forma "original", que es la forma que tenía la línea actual cuando desplazó a la linea desde otra. Utilice esta orden para deshacer los cambios que acaba de hacer en la línea actual.