***Unidad I***

***<u>Palabras reservadas:</u>***

Para poder usar cualquier lenguaje, debemos conocer los códigos que representan las actividades a realizar. Turbo Pascal cuenta con su propio conjunto, los cuales llamaremos palabras reservadas.

***<u>Palabras reservadas de Pascal Estándar y Turbo Pascal:</u>***

***ABSOLUTE***

***AND***

***ARRAY***

***ASM***

***ASSEMBLER***

***BEGIN***

***BOOLEAN***

***BYTE***

***CASE***

***CHAR***

***COMP***

***CONST***

***CONSTRUCTOR***

***DESTRUCTOR***

***DIV***

***DO***

***DOUBLE***

***DOWNTO***

***ELSE***

***END***

***EXPORT***

***EXPORTS***

***EXTENDED***

***EXTERNAL***

***FAR***

***FILE***

***FOR***

***FORWARD***

***FUNCTION***

***GOTO***

***IF***

***IMPLEMENTATION***

***IN***

***INDEX***

***INHERITED***

***INLINE***

***INTEGER***

***INTERFACE***

***INTERRUPT***

***LABEL***

***LIBRARY***

***LONGINT***

***MOD***

***NAME***

***NEAR***

***NIL***

***NOT***

***OBJECT***

***OF***

***OR***

***PACKED***

***POINTER***

***PRIVATE***

***PROCEDURE***

***PROGRAM***

***PUBLIC***

***REAL***

***RECORD***

***REPEAT***

***RESIDENT***

***SET***

***SHL***

***SHORTINT***

***SHR***

***SINGLE***

***STRING***

***THEN***

***TO***

***TYPE***

***UNIT***

***UNTIL***

***USES***

***VAR***

***VIRTUAL***

***WHILE***

***WITH***

***WORD***

***XOR***

Ninguna de estas palabras reservadas puede ser usada como identificador, ya que cada una de ellas tiene predefinida una función.

***Identificadores:***

Son etiquetas que representan variables, constantes, procedimientos, tipos de datos, funciones. Existen dos tipos de identificadores, los predefinidos por Turbo Pascal y los que define el programador. Los identificadores son una secuencia de 1 a 127 caracteres, donde el primer carácter debe ser alfabético, y el resto no debe contener espacios en blanco y caracteres especiales como son: !, %, $, &...

***Tipos de datos:***

Existen diferentes tipos de datos, los cuales son utilizados para manipular la información:

| ***Tipo*** | ***Descripción*** | ***Rango*** |
|---|---|---|
| ***Boolean*** | ***Valores*** | ***True o False*** |
| ***Byte*** | ***Números enteros*** | ***0 .. +255*** |
| ***Char*** | ***Caracteres ASCII*** | ***'$', '%', '¦', 'ß', '¢'*** |
| ***Comp*** | ***Números reales*** | ***−9.2E18 a 9.2E18*** |
| ***Double*** | ***Números reales*** | ***5.0E −324 a 1.7E +308*** |
| ***Extended*** | ***Números reales*** | ***1.9E −4851 a 1.1E +4932*** |
| ***Integer*** | ***Números enteros*** | ***−32768 .. +32767*** |
| ***Longint*** | ***Números enteros*** | ***−2147483648 .. +2147483647*** |
| ***Real*** | ***Números con decimales*** | ***2.9E −39 a 1.7E +38*** |
| ***Shortint*** | ***Números enteros*** | ***−128 .. + 127*** |
| ***Single*** | ***Números reales*** | ***1.5E −45 a 3.4E +38*** |
| ***String*** | ***Conjunto de caracteres*** | ***'La casa', 'El toro', 'Camión'*** |
| ***Word*** | ***Números enteros*** | ***0 .. +65535*** |

***Variables y constantes:***

Los tipos de datos que manejaremos son constates y variables. Las variables son las que tienen la capacidad de cambiar su valor a lo largo del programa, a diferencia de las constantes que permanecen con el mismo valor desde su inicio hasta el final del programa. Un ejemplo sencillo de una constante es la gravedad 9.81, el cual es un valor fijo, a diferencia de una variable, la cual cambia con facilidad su valor durante la ejecución de

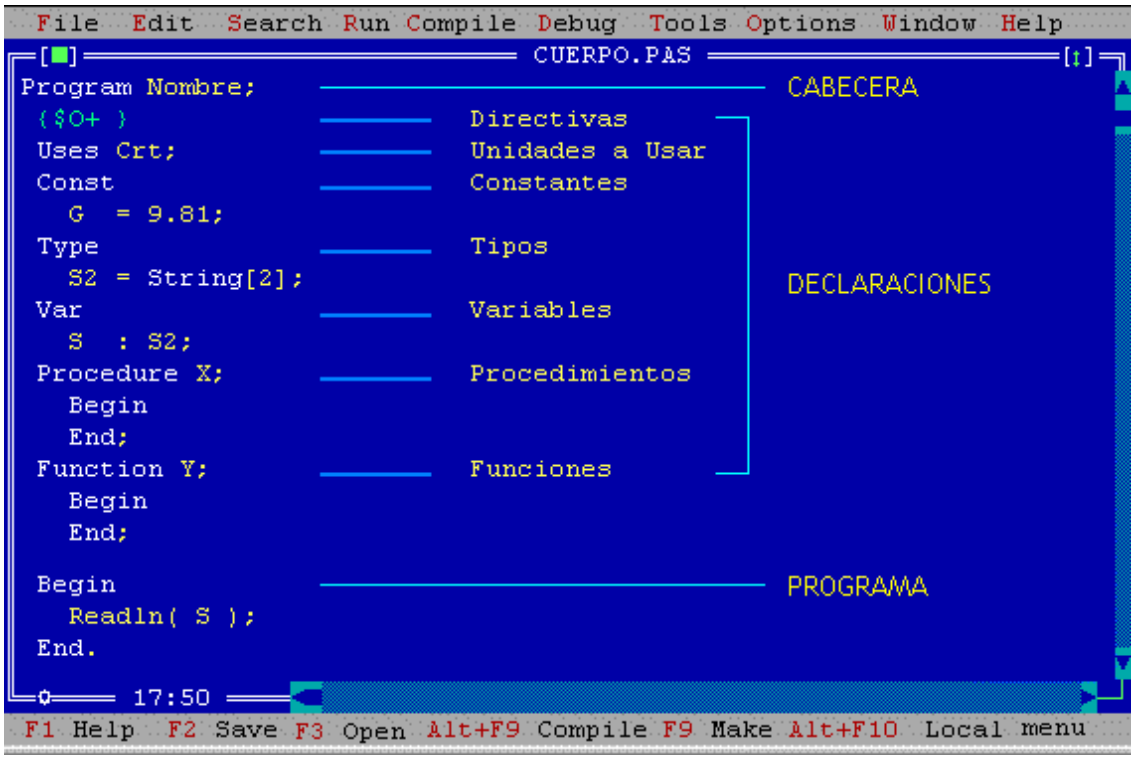
un programa. No obstante los valores almacenados, pueden ser de tipo numérico, alfanuméricos, o carácter.

***Comentarios:***

Los comentarios nos sirven para dar una apropiada documentación al programa, logrando con esto una mejor comprensión del código en futuras revisiones, o en su defecto a los programadores que tengan que modificar nuestro programa. Los comentarios no son tomados en cuenta por el compilador, es decir que durante la ejecución del programa, estos textos serán completamente ignorados. Con el uso de las llaves {comentario}, ó paréntesis con asteriscos (* Comentarios *).

***Estructura de los programas:***

Un lenguaje como Turbo Pascal, cuenta con una estructura rígida, para de esta manera poder ser ejecutado por el compilador, de lo contrario no se podrá ejecutar el programa. La siguiente gráfica muestra su estructura.




***Sentencia PROGRAM:***

La sentencia PROGRAM se utiliza para declarar el nombre del programa. Su sintaxis es la siguiente:

***Program Nombre;***

***Declaración de Unidades:***

Las unidades son módulos independientes del programa, los cuales pueden ser utilizados por cualquier programa, utilizando la palabra USES, automáticamente se incorpora el código correspondiente de la unidad. Su sintaxis es la siguiente:

***Uses CRT;***

***Declaración de constantes y variables:***

La palabra CONST es la que utilizaremos para iniciar el área de constantes, usando una lista de identificadores para almacenar los valores. Su sintaxis es la siguiente:

***Const***

***PI = 3.1415926;***

***L1 =`Hola `;***

La palabra VAR es la que utilizaremos para iniciar el área de Variables, usando una lista de identificadores para definir su tipo de información. Su sintaxis es la siguiente:

***Var***

***A :Integer;***

***L :String;***

***Declaración de procedimientos y funciones:***

La palabra PROCEDURE es la que utilizaremos para declarar un procedimiento. Su sintaxis es la siguiente:

***Procedure Nombre (parámetros);***

***Begin***

***...***

***End;***

La palabra FUNCTION es la que utilizaremos para declarar una función. Su sintaxis es la siguiente:

***Fuction Nombre (parámetros):Tipo;***

***Begin***

***...***

***Nombre := valor;***

***End;***

***Programa principal:***

En esta sección, es donde el programador estructura la secuencia de pasos a realizar, es decir escribe el conjunto de sentencias que el compilador ejecutara. su sintaxis es la siguiente:

***Begin***

***Sentencias;***

***...***

***Sentencias;***

***End.***

***<u>Compilación y ejecución en memoria:</u>***

La compilación de un programa es el paso mediante el cual traducimos dicho programa al lenguaje maquina entendible por la computadora. Existen varias combinaciones de teclas para realizar esta operación, y son las siguientes:

***ALT + F9***

***ALT + C y luego escoger la opción Compile.***

La ejecución de un programa en memoria, se realiza con la opción de RUN o en su defecto con la combinación de las siguientes teclas:

***Control + F9***

***<u>Compilación a Disco:</u>***

Este tipo de compilación nos permite crear un archivo ejecutable, es decir que una vez terminado nuestro programa, podremos generar un archivo que se pueda ejecutar desde el sistema operativo, con esto evitaríamos entrar al compilador para ejecutarlo. Esto se realiza simplemente cambiando el destino de compilación, sustituyendo la memoria por disco.

***<u>Asignación o igualación:</u>***

La asignación permite darle un valor a las variables declaradas dentro de nuestro programa, es decir asignarle algún valor del mismo tipo de información. El símbolo que realiza dicha Operación es :=. Su sintaxis es la siguiente:

***Suma := Numero A + Numero B;***

***{A Suma se le asignara el resultado de sumar Numero A + Numero B}***

***Nombre := 'Carlos Ramírez';***

***{A Nombre se le asignara la cadena 'Carlos Ramírez'}***

La asignación nos permite manipular la información que existe, pero hace falta que el usuario introduzca sus propios valores, realizando con ellos los procesos, para posteriormente mostrar los resultados obtenidos.

***<u>Salida de datos a la pantalla:</u>***

Unos de los puntos mas importantes, es poder mostrar los resultados obtenidos en la pantalla de la computadora. Esto se realiza usando los procedimientos WRITE y WRITELN. Su sintaxis es la siguiente.

***Write ( Lista de Identificadores);***

***WriteLn (Lista de Identificadores);***

La lista de identificadores, son todos los identificadores y valores que deseamos mandar a pantalla, los cuales deberán estar separados por una coma (,). Con un simple ejemplo mostraremos la diferencia de estos procedimientos.

***Program fuente***

***Begin***

***Write ('Carlos ');***

***Write ('Ramírez ');***

***Write ('González ');***

***End.***

***Program fuente***

***Begin***

***WriteLn ('Carlos');***

***WriteLn ('Ramírez');***

***WriteLn ('González');***

***End.***

***<u>Entrada de datos desde teclado:</u>***

La entrada de datos por teclado es un punto vital en cualquier lenguaje, puesto que la mayor parte del tiempo, los usuarios requieren obtener información a partir de una serie de datos, los cuales serán proporcionados por ellos mismos. Esto se realiza usando los procedimientos READ y READLN. Su sintaxis es la siguiente.

***Read (Lista de Identificadores);***

***ReadLn (Lista de Identificadores);***

La lista de identificadores, son todas las variables para leer en la pantalla utilizando el teclado, las cuales deberán estar separados por una coma (,). Con un simple ejemplo mostraremos la diferencia de estos procedimientos.

***Program fuente***

***Var***

***A, B, C: Integer;***

***Begin***

***Read (A, B);***

***Read (C);***

***End.***

***Program fuente***

***Var***

***A, B, C: Integer;***

***Begin***

***ReadLn (A, B);***

***ReadLn(C);***

***End.***

La diferencia entre el Read y el ReadLn, si se escriben mas datos que el número de identificadores especificados en un Read, serán guardados en un buffer y se le asignaran a los identificadores del siguiente Read o ReadLn del programa. En cambio ReadLn ignora todos los datos que rebasan el número de identificadores. En caso que en alguna captura de una variable, se captura un dato de diferente tipo, el Read o ReadLn nos marcara un error de tipo de variable, terminando la ejecución del mismo.

***Unidad II***

***Operaciones Básicas:***

Las Operaciones se realizan por medio de operadores y operandos, donde el operador es un símbolo que indica al compilador la operación a realizar, los operandos son los datos a quienes afectará el operador. Existen cuatro operadores básicos y son los siguientes:

| ***Operadores*** | ***Operación*** |
|---|---|
| ***** | ***Multiplicación*** |
| **/** | ***División*** |
| **+** | ***Suma*** |
| **–** | ***Resta*** |

Cuando se realiza cualquier operación, el resultado arrojado será del tipo capaz de soportarla. Por ejemplo la suma de números reales arroja un número real, no obstante la suma de un número real con uno entero, arrojará un real.

***Operadores DIV y MOD:***

En el caso de la división, no importando los números que dividamos, el resultado será un número real, para obtener un resultado entero, debemos de utilizar el operador DIV, de la misma manera usaremos el operador MOD, para obtener el residuo entero. Su sintaxis es la siguiente:

***N_Entero := Dividendo DIV Divisor;***

***N_Entero := Dividendo MOD Divisor;***

Este programa es un ejemplo sencillo de operaciones:

```
Program Operaciones;

Var

Sum, Res, Mul, Div: Real;

C_Ent, R_Ent: Integer;

Begin

Sum := 5 + 2;

Res := 5 − 2;

Mul := 5 * 2;

Div := 5 / 2;

C_Ent := 5 DIV 2;

R_Ent := 5 MOD 2;

WriteLn (`La suma de 5 + 2 = ', Sum);

WriteLn (`La Resta de 5 − 2 = ', Res);

WriteLn (`La Multiplicación de 5 * 2 = ', Mul);

WriteLn (`La División de 5 / 2 = ', Div);

WriteLn ('La división Entera 5 / 2 = ', C_Ent);

WriteLn ('El Residuo Entera 5 / 2 = ', R_Ent);

End.
```

***Prioridad de Operadores:***

Cuando realizamos alguna expresión muy compleja, nos encontraremos con mas de un operador, esto no representa ningún problema, debido a que los operadores tienen una jerarquía de ejecución, es decir se realizan de izquierda a derecha los operadores de mayor a menor jerarquía. El orden de jerarquía es el siguiente:

| *Cuentan con mayor jerarquía.* | *Cuentan con mediana jerarquía.* | *Cuentan con menor jerarquía.* |
|---|---|---|
| **( )** | *** / DIV MOD** | **+ −** |

Este programa es un ejemplo sencillo de la jerarquía de operadores:

```
Program Operaciones;
```

***Var***

***Op1, Op2: Real;***

***Begin***

***Op1 := 3 + 5 * 2;***

***Op2 := (3 + 5) * 2;***

***WriteLn ('El Resultado de 3 + 5 * 2 = ', Op1);***

***WriteLn ('El Resultado de (3 + 5) * 2 = ', Op2);***

***End.***

***Constantes Variables:***

Este término suena medio absurdo, debido a que las constantes no pueden cambiar su valor, al comprenderlo desde otro punto de vista nos daremos cuenta que es una variable, con la única diferencia de tener un valor inicial asignado. Estas se usan para evitar la tarea de inicializar nuestras variables en el programa, esto se debe a que una variable al ser declarada, trae consigo el valor ocupado en la memoria antes de su asignación. Un ejemplo similar es cuando adquirimos una caja nueva, esta puede venir vacía o con objetos, para poderla usar limpiaríamos primero su interior y posteriormente guardaríamos nuestros objetos en ella. Su sintaxis es la siguiente:

***Const***

***Var_Con_I: Real = 3.1415926;***

***Var_Con_S: String = `Hola';***

Este programa es un ejemplo sencillo de las constantes variables:

***Program Constantes_Variables;***

***Const***

***Num1: Integer = 10;***

***Nom1: String = 'Carlos';***

***Var***

***Num2: Integer;***

***Nom2: String;***

***Begin***

***WriteLn ('Constante Variable con 10 = ', Num1);***

***WriteLn ('Constante Variable con Carlos = ', Nom1);***

***WriteLn ('Variable numérica = ', Num2);***

***WriteLn ('Variable Cadena = ', Nom2);***

***End.***

***Operadores Lógicos:***

Son todas las operaciones comparativas que podemos realizar con nuestros datos, obteniendo como único resultado un True o False. La siguiente tabla muestra estos operadores:

| ***Operadores*** | ***Descripción*** |
|---|---|
| = | **Igual a** |
| <> | **Distinto a** |
| < | **Menor que** |
| > | **Mayor que** |
| <= | **Menor igual que** |
| >= | **Mayor igual que** |
| **Not** | **Negación de condición** |

***Bloques o Sentencias Compuestas:***

Un bloque es un conjunto de sentencias designadas a realizar una tarea en común, las sentencias están separadas por un ";", y el bloque es delimitado por las palabras reservadas Begin y End, esto le permite al compilador reconocer todo lo que tiene que ejecutar en algún caso determinado. El ejemplo mas claro que podemos ver, es el cuerpo de un programa.

***Program Bloque;***

***BEGIN***

***WriteLn (`Buenos días');***

***WriteLn (`Buenas tardes');***

***END.***

***Unidad III***

Cuantas veces en nuestra vida cotidiana realizamos la misma tarea varias veces, creo que casi todo el tiempo pasa esto, en la codificación de programas existe una forma de realizar la misma tarea sin tener que escribir N veces la misma sentencia.

***Ciclo FOR:***

Casi todos los lenguajes de programación no brindan sentencias de ciclos, Turbo Pascal cuenta con el ciclo FOR, este nos permite realizar un numero exacto de veces una sentencia o bloque de sentencias. Su sintaxis es la siguiente:

***FOR Variable := V_Inicial To V_Final DO***

***Sentencia;***

Para hacer esto, el ciclo FOR utiliza como base una variable índice, la cual parte de un valor inicial, sufriendo un incremento o decremento, hasta llegar al valor final. Esto se verá más claro con los ejemplos siguientes:

Diferencias entre no usar y usar un Ciclo.

***Program Sin_Ciclo;***

***Begin***

***WriteLn (`Número = 1');***

***WriteLn (`Número = 2');***

***WriteLn (`Número = 3');***

***WriteLn (`Número = 4');***

***End.***

***Program Con_Ciclo;***

***Var***

***I: Integer;***

***Begin***

***For I := 1 To 4 Do***

***WriteLn (`Número = ', I);***

***End.***

La principal limitante con la sentencia FOR, es la incapacidad de salir sin completar el ciclo, obligando al compilador a ejecutar el proceso involucrado, el número de veces especificados. Existe dos maneras de ejecutar la sentencia FOR, el incremento en uno "TO" y el decremento en uno "DOWNTO", cubriendo con esto todas las necesidades de un programador en la manipulación de sus datos.

Diferencia entre TO y DOWNTO.

***Program Ciclo_1_a_4;***

***Var***

***I: Integer;***

***Begin***

***For I := 1 To 4 Do***

***WriteLn (`Número = ', I);***

***End.***

***Program Ciclo_4_a_1;***

***Var***

***I: Integer;***

***Begin***

***For I := 4 DownTo 1 Do***

***WriteLn (`Número = ', I);***

***End.***

***<u>Ciclo WHILE:</u>***

Esta sentencia nos permite realizar un proceso cíclico más flexible, nos libra de la dependencia de un número preestablecido, el compilador ejecuta la sentencia WHILE, siempre y cuando la condición booleana sea valida (True), de ser falsa (False) termina automáticamente la sentencia. Su sintaxis es la siguiente:

***WHILE Condición DO***

***Sentencia;***

Si quisiéramos realizar un proceso diez veces, tendríamos que forzar la sentencia WHILE a trabajar como FOR. Esto se muestra en el siguiente ejemplo:

***Program WHILE_a_FOR;***

***Var***

***I: Integer;***

***Begin***

***I := 1;***

***WHILE I <= 4 Do***

***Begin***

***WriteLn (`Número ', I);***

***I := I + 1;***

***End;***

***End.***

Cuando la variable I llegue al valor de 5, la condición no se cumplirá, terminando la sentencia WHILE.

***Ciclo REPEAT UNTIL:***

La sentencia REPEAT es la contra parte del WHILE, el compilador primero ejecuta el bloque de sentencias, al terminar verifica el resultado de la condición UNTIL y de ser falsa lo seguirá ejecutando, el ciclo terminará cuando sea verdadera. Su sintaxis es la siguiente:

***REPEAT***

***Sentencia;***

***...***

***Sentencia;***

***UNTIL Condición;***

Si quisiéramos realizar un proceso diez veces, tendríamos que forzar la sentencia REPEAT UNTIL a trabajar como FOR. Esto se muestra en el siguiente ejemplo:

***Program REPEAT_a_FOR;***

***Var***

***I: Integer;***

***Begin***

***I := 1;***

***REPEAT***

***WriteLn (`Número ', I);***

***I := I + 1;***

***UNTIL I = 5;***

***End.***

Cuando la variable I llegue al valor de 5, la condición se cumplirá, terminando la sentencia REPEAT UNTIL.

***Sentencia IF... THEN... ELSE...:***

Esta sentencia nos brinda la posibilidad de bifurcar la ejecución de un programa, basándose en procesos alternativos vinculados a una condición, permitiendo depurar los resultados finales. Su sintaxis es la siguiente:

***IF Condición***

***THEN***

***Sentencia***

***ELSE***

***Sentencia;***

Si deseamos determinar si un numero es par o non, haríamos un ciclo determinado verificando el residuo de la variable índice, si es cero es par y sino es non. Esto se muestra en el siguiente ejemplo:

***Program Pares_Nones;***

***Var***

***I: Integer;***

***Begin***

***FOR I := 1 TO 4 DO***

***IF (I MOD 2) = 0***

***THEN WriteLn (`Número ', I, ' Es PAR')***

***ELSE WriteLn (`Número ', I, ' Es NON');***

***End.***

El IF puede omitir la sentencia ELSE, evitando realizar un proceso al ser falsa la condición.

***Sentencia IF anidadas:***

Cuando una condición es muy compleja de plantear, se pueden realizar bloques de IF anidados, facilitando con esto el desempeño del compilador, evitando la evaluación de condiciones de dudosa eficiencia. Su sintaxis es la siguiente:

***IF Condición 1***

***THEN IF Condición 2***

***THEN Sentencia***

***ELSE Sentencia***

***ELSE Sentencia;***

En una lista de números del uno a diez, se desea encontrar los pares que estén entre el 3 y 7, una solución es aplicar dos sentencias IF anidadas verificando ambas condiciones. Esto se muestra en el ejemplo siguiente:

***Program Pares_entre_3_y_7;***

***Var***

***I: Integer;***

***Begin***

***FOR I := 1 TO 10 DO***

***IF (I MOD 2) = 0***

***THEN IF (I >= 3) AND (I <= 7)***

***THEN WriteLn (`Núm. ', I, ' PAR entre 3 y 7');***

***End.***

***Sentencia CASE:***

Una variable puede tomar una gran variedad de valores, si deseáramos tomar en cuenta cada uno de ellos, tejeríamos una gran telaraña de IF, en estos casos se puede utilizar la sentencia CASE, capaz de estructurar sus valores significativos asignándole el proceso a realizar. Su sintaxis es la siguiente:

***CASE Variable OF***

***Valor 1: Sentencia;***

***...***

***Valor N: Sentencia***

***ELSE***

***Sentencia***

***END; {CASE}***

La sentencia CASE solo soporta variables Carácter y enteras, evitando así las complejas evaluaciones, esto se comprenderá mejor con el siguiente ejemplo:

***Program Pares_Nones;***

***Var***

***I, Res: Integer;***

***Begin***

***FOR I := 1 TO 4 DO***

***Begin***

***Res := (I MOD 2);***

***CASE Res OF***

***0 :WriteLn (`Número ', I, ' Es PAR');***

***1 :WriteLn (`Número ', I, ' Es NON');***

***End {CASE}***

***End {FOR}***

***End.***

***Sentencia GOTO:***

Esta característica es del lenguaje ***BASIC***, cambia el orden secuencial de ejecución de un programa, esto representa romper el modelo estructurado de programación de Turbo Pascal, corriendo el riesgo de perder el control del mismo *(no debe usarse en la programación estructurada)*. Para Usar la sentencia GOTO, es necesario primero declarar todas las etiquetas con la palabra reservada LABEL, una etiqueta es un identificador como cualquier otro, el bloque a realizar se delimita usando ": " y END, esto nos representa el inicio y final del mismo. Su sintaxis es la siguiente:

***GOTO Etiqueta;***

Un ejemplo sencillo del uso del GOTO es el siguiente:

***Program Goto_A_Label;***

***LABEL Salto;***

***Begin***

***WriteLn (`Primera Línea');***

***GOTO Salto;***

***WriteLn (`Segunda Línea');***

***Salto: WriteLn (`Tercera Línea');***

***End.***

En este ejemplo vemos como el compilador se salta la sentencia WriteLn, evitando desplegar el letrero de "Segunda Línea".

***Sentencia HALT:***

Se utiliza para romper la ejecución de un programa, el HALT se puede aplicar cuando se presenta algún error en las variables de alguna formula, fallas físicas, o por carecer de sentido el programa. Su sintaxis es la siguiente:

***HALT;***

Un ejemplo sencillo del uso del HALT es el siguiente:

***Program Terminar;***

***Begin***

***WriteLn (`Primera Línea');***

***HALT;***

***WriteLn (`Segunda Línea');***

***WriteLn (`Tercera Línea');***

***End.***

En este ejemplo vemos como el compilador ejecuta solo la primera sentencia WriteLn, y termina su ejecución.

***Unidad IV***

***Procedimiento:***

Es un identificador ligado a un bloque de sentencias, esto nos permite usarlo varias veces, sin necesidad de repetir todo el código, llamando el bloque por medio de su identificador. Un ejemplo sencillo, es el asignar valores nulos a variables utilizadas en varios cálculos, para no repetir el código utilizaremos un procedimiento de nombre Inicio, el cual llamaríamos en las diversas partes donde sea necesario.

***Creación de Procedimientos:***

No se puede crear un procedimiento sin tener un objetivo especifico, tomando en cuenta los posibles valores de entrada, sus identificadores propios, y el bloque de sentencias capaz de cumplir con el objetivo. Su sintaxis es la siguiente:

***PROCEDURE Identificador (Parámetros);***

***Sección declarativa;***

***BEGIN***

***END;***

Cuando declaramos un procedimiento, automáticamente se incorpora como un identificador durante su compilación y ejecución.

Este es un ejemplo sencillo de la declaración y uso de un procedimiento:

***Program Procedimiento;***

***Var***

***X, Y, Z: Integer;***

***PROCEDURE Inicio (X1, Y1, Z1: Integer);***

***Begin***

***X := X1;***

***Y := Y1;***

***Z := Z1;***

***End; {PROCEDURE}***

***Begin***

***Inicio (1, 2, 3);***

***WriteLn ('X=', X,'Y=', Y,'Z=', Z);***

***Inicio (4, 5, 6);***

***WriteLn ('X= ', X, ' Y= ', Y, ' Z= ', Z);***

***End.***

***Función:***

Al igual que un procedimiento, es un identificador ligado a un bloque de sentencias, permitiendo su uso indefinidamente, sin necesidad de repetir todo el código, pero nos ofrece la ventaja de devolver un valor al final de su ejecución. Un ejemplo sencillo, es realizar una suma con dos valores de entrada, devolviendo el resultado obtenido a la función.

***Creación de Funciones:***

No se puede crear una Función sin tener un objetivo especifico, tomando en cuenta los posibles valores de entrada, sus identificadores propios, el bloque de sentencias, y el tipo de valor que devolverá la función. Su sintaxis es la siguiente:

***FUNCTION Identificador (Parámetros): Tipo;***

***Sección declarativa;***

***BEGIN***

***END;***

Este es un ejemplo sencillo de la declaración y uso de una Función:

***Program Función;***

***Var***

***Z: Integer;***

***FUNCTION Suma (X1, X2: Integer): Integer;***

***Begin***

***Suma := X1 + X2;***

***End; {FUNCTION Suma}***

***Begin***

***Z := Suma (1, 1);***

***WriteLn (`1 + 1 = ', Z);***

***Z := Suma (2, 2);***

***WriteLn (`2 + 2 = ', Z);***

***End.***

***Parámetros:***

Las funciones y procedimientos normalmente las trabajamos con variables globales, limitando su campo de acción, con el uso de parámetros logramos indicar los valores y variables iniciales de los procesos, logrando una mayor flexibilidad y portabilidad. Contamos con dos tipos de parámetros:

***Parámetros por valor:***

Cuando se pasa un parámetro por valor, se obtiene una copia temporal de la variable usada, dentro de la función o procedimiento se trabaja con la copia obtenida, no importando las operaciones que se realicen con la copia, la variable introducida como parámetro, no será afectada en su valor inicial al terminar el proceso. Su sintaxis es la siguiente:

***PROCEDURE Identificador (Ide1, Ide2: Tipo; Ide3: Tipo);***

***Parámetro por referencia:***

Cuando se pasa un parámetro por referencia, los cambios que se realicen a la variable introducida como parámetro, se mantendrán vigentes al terminar el proceso, en este caso especifico solo se admiten variables. Su sintaxis es la siguiente:

***PROCEDURE Identificador (VAR Ide1: Tipo);***

***Variables Globales:***

Estas variables son declaradas después de la sección de unidades, permitiendo su utilización tanto en los procedimientos y funciones, como en el cuerpo del programa.

Este es un ejemplo sencillo de Variables Globales:

***Program Procedimiento;***

***VAR***

***Z: Integer;***

***Begin***

***...***

***End.***

***<u>Variables Locales:</u>***

Estas variables son declaradas dentro de un procedimiento o función, quedando limitadas para ser usadas en cualquier sección del programa.

Este es un ejemplo sencillo de Variables Locales:

***Program Procedimiento;***

***PROCEDURE Local;***

***VAR***

***Z :Integer;***

***Begin***

***... {Sección valida de Z}***

***End;***

***Begin***

***...***

***End.***

Unos de los procesos más usados en los programas, son el incremento y decremento de variables, para ello contamos con los procedimientos INC () y DEC ().

***<u>Procedimiento INC ():</u>***

Comúnmente cuando queremos incrementar una variable, utilizamos la siguiente sentencia: Var := Var + Incremento; , para estos casos específicos podemos utilizar el procedimiento INC (). Su sintaxis es la siguiente:

***INC (Variable, Incremento);***

***<u>Procedimiento DEC ():</u>***

Comúnmente cuando queremos decrementar una variable, utilizamos la siguiente sentencia: Var := Var − decremento; , para estos casos específicos podemos utilizar el procedimiento DEC (). Su sintaxis es la

siguiente:

***DEC (Variable, decremento);***

***<u>Truncamiento y Redondeo:</u>***

El Truncamiento nos permite eliminar la parte decimal de un número real, la función es TRUNC (), nos devuelve un numero de tipo entero. Su sintaxis es la siguiente:

***Variable_Entera := TRUNC (Número_Real);***

El Redondeo nos permite aproximarnos a un número entero más cercano, esto se define con sus decimales, mayor o igual a 0.5 sube al numero entero consecutivo, menor a 0.5 se trunca la parte decimal, la función es ROUND (), nos devuelve un numero de tipo entero. Su sintaxis es la siguiente:

***Variable_Entera := ROUND(Número_Real);***

***Program Tru_Red;***

***Begin***

***WriteLn (`Truncar 3.89 = ', TRUNC (3.89));***

***WriteLn (`Redondea 4.6 = ', ROUND (4.6));***

***WriteLn (`Redondea 5.1 = ', ROUND (5.1));***

***End.***

***<u>Funciones Exponenciales y Logarítmicas:</u>***

La función SQR (), nos brinda el cuadrado de un número, el tipo de esta función es real o entera, según se requiera en la sentencia. Su sintaxis es la siguiente:

***Variable := SQR (Número);***

La función SQRT (), nos brinda la raíz cuadrada de un número, el tipo de valor que devuelve esta función es real. Su sintaxis es la siguiente:

***Variable_Real := SQR (Número);***

La función EXP (), nos brinda el valor de e elevado a la X, el tipo de valor que devuelve esta función es real. Su sintaxis es la siguiente:

***Variable_Real := EXP (Número);***

La función LN (), nos brinda el logaritmo natural de un número, el tipo de valor que devuelve esta función es real. Su sintaxis es la siguiente:

***Variable_Real := LN (Número);***

Este sencillo ejemplo, aplicará las funciones anteriores:

***Program Exp_Log;***

***Begin***

***WriteLn (`Cuadrado 3.5 = ', SQR (3.5): 3: 2);***

***WriteLn (`Raíz 6 = ', SQRT (6): 3: 2);***

***WriteLn (`e ^ 3 = ', EXP (3): 3: 2);***

***WriteLn (`Logaritmo 3 = ',LN (3): 3: 2);***

***WriteLn (`2 ^ 3 = ', Exp ( 3 * Ln (2)): 3: 2);***

***End.***

***Funciones Aritméticas:***

La función ABS (), nos permite obtener el valor absoluto de un número o variable, esto se requiere en algunos procesos matemáticos. Su sintaxis es la siguiente:

***Variable := ABS (Valor);***

La función INT(), nos permite obtener la parte entera de un número real, el valor que devuelve es de tipo real. Su sintaxis es la siguiente:

***Variable := INT (Valor);***

La función FRAC (), nos permite obtener la parte decimal de un número real, el valor que devuelve es de tipo real. Su sintaxis es la siguiente:

***Variable := FRAC (Valor);***

***Funciones Trigonométricas:***

El seno, coseno y arco tangente, son las únicas funciones trigonométricas que existen en Turbo Pascal, el resto se pueden calcular a partir de estas. El seno SIN (), coseno COS () y arco tangente ARCTAN (), la siguiente tabla nos muestra como calcular las demás:

| ***Función*** | ***Operación*** |
|---|---|
| ***Tangente (X)*** | ***Sin(x) / Cos (x)*** |
| ***Cotangente (X)*** | ***Cos (x) / Sin (x)*** |
| ***Secante (X)*** | ***1 / Cos (x)*** |
| ***Cosecante (X)*** | ***1 / Sin (x)*** |

***Números Aleatorios:***

La función RANDOM, genera un número decimal comprendido entre 0 y 1, si utilizamos un parámetro X, obtiene un numero entero entre 0 y X−1. El procedimiento RANDOMIZE, evita generar el mismo grupo de números aleatorios, renovándolo en cada ejecución.

Este sencillo ejemplo, aplicará las funciones de números Aleatorios:

***Program Random_Randomize;***

***Var***

***I: Integer;***

***Begin***

***RANDOMIZE;***

***FOR I := 1 TO 15 DO***

***WriteLn (`Número: ', RANDOM (50));***

***End.***

***Unidad V***

***Cadena de Caracteres:***

Una cadena consta de 1 a 255 caracteres, el tipo de variable que la soporta es el String, si deseamos delimitar el tamaño de la cadena, se especifica entre corchetes su longitud, teniendo como limite 255 caracteres. Su sintaxis es la siguiente:

***Var_Cadena :STRING;***

***Var_Cadena :STRING [Número];***

***Comparación de Cadenas***

Cuando comparamos dos cadenas, se toman en cuenta el valor ASCII de sus caracteres, un ejemplo sencillo puede ser las cadenas `Alas' y `alas', a simple vista pensaríamos que son iguales, pero en realidad la cadena `Alas' es menor que `alas', debido a los valores de sus letras son diferentes, `A'= 65 y `a'= 97. La longitud es una característica importante en las cadenas, la cual se determina por su número de caracteres, por ejemplo, la longitud de `Hola' es de 4 caracteres, si una cadena no contiene caracteres, la llamaremos vacía o nula.

***Manejo de los caracteres de una cadena:***

Si deseamos manipular algún carácter de una cadena, tendremos que especificar la casilla de su ubicación, para obtener o cambiar su valor actual. Aplicando esta característica de las cadenas, podríamos seleccionar y desplegar en pantalla el carácter deseado, para ello desarrollaremos el siguiente ejemplo:

***Program Mod_Cadena;***

***Var***

***Cad :String [10];***

***Begin***

***Cad := 'UNO';***

***WriteLn (Cad [1]);***

***WriteLn (Cad [2]);***

***WriteLn (Cad [3]);***

***End.***

***Longitud de las Cadenas:***

La función LENGTH (), permite saber el número de caracteres de una cadena, devuelve un número entero entre 0 a 255.Su sintaxis es la siguiente:

***Variable := LENGTH(Cadena);***

***Operador +:***

Este operador nos permite unir varias cadenas de caracteres, esto se realiza como una simple suma, de igual manera podemos agregar caracteres a una cadena.

***Función CONCAT ():***

Esta función nos permite unir varias cadenas de caracteres, devolviendo como resultado una cadena, funciona de igual manera que el operador +.Su sintaxis es la siguiente:

***Variable := CONCAT (Cadena_1, Cadena_2, ... Cadena_N);***

En este ejemplo aplicaremos los conceptos anteriores:

***Program Con_Len;***

***Var***

***Cad: String [10];***

***K: Integer;***

***Begin***

***Cad := 'U'+ 'N';***

***Cad := CONCAT (Cad, 'O');***

***FOR K := 1 TO LENGTH (Cad) DO***

***WriteLn (Cad [K]);***

***End.***

***Función POS ():***

Esta función nos permite buscar una subcadena dentro de una cadena, nos devuelve un valor entero, el cual indique la posición en donde se encuentra la subcadena. Su sintaxis es la siguiente:

***Variable := POS (SubCadena, Cadena);***

Si no se logra encontrar la subcadena, la función nos devolverá un cero.

***Función COPY ():***

Esta función nos devuelve una subcadena, tomando como base una posición de la cadena origen, y el número de caracteres a copiar. Su sintaxis es la siguiente:

***Variable := COPY (Cadena, Posición, Num_Caracteres);***

Si se rebasa la capacidad de la variable receptora, solo se copiaran los caracteres que pueda soportar.

***Procedimiento INSERT ():***

Este procedimiento inserta una subcadena, indicando la cadena destino, y la posición de la inserción. Su sintaxis es la siguiente:

***INSERT (Sub_Cadena, Cadena, Posición);***

En una inserción de subcadenas, se puede saturar la capacidad de la cadena receptora, impidiendo la inserción completa, obligando a omitir la parte restante.

***Procedimiento DELETE ():***

Este procedimiento borra una cantidad de caracteres de una cadena, indicando su posición inicial y el número de caracteres a eliminar. Su sintaxis es la siguiente:

***DELETE (Cadena, Posición, Num_Caracter);***

En muchos casos el número de caracteres a borrar, rebasa por completo la longitud de la cadena, se ignora los caracteres excedidos.

***Función UPCASE ():***

Esta función convierte a mayúsculas un carácter alfabético, el cual se proporciona como parámetro. Su sintaxis es la siguiente:

***Variable := UPCASE (Carácter);***

Si el carácter utilizado como parámetro, no es una letra del abecedario, no sufrirá ningún cambio, conservando su valor original.

***Procedimiento STR ():***

Este procedimiento convierte un valor numérico a cadena de caracteres, no importando el tipo de número, puede ser entero o con decimales. Su sintaxis es la siguiente:

***STR (Var_Numérica, Cadena);***

En la variable cadena se deposita la conversión satisfactoria de la variable numérica, utilizando el número de caracteres que sea necesario.

***Procedimiento VAL ():***

Este procedimiento convierte una cadena de caracteres a un valor numérico, siempre y cuando los caracteres de la cadena sean números. Su sintaxis es la siguiente:

***VAL (Cadena, Var_Numérica, Var_Entera);***

En la variable numérica se deposita la conversión satisfactoria de la cadena, en caso de contener caracteres especiales o letras, la variable numérica almacenara un cero, y se asignara un valor de error en la variable entera.

***Unidad VI***

***Unidades:***

Son un conjunto de identificadores compilados, los cuales podemos llamar en cualquier programa de Turbo Pascal, sin necesidad de escribirlos de nuevo. Una unidad no se puede ejecutar por si sola, estas requieren ser llamadas por un programa, donde podrán desempeñar sus tareas.

***Palabra reservada USES:***

Esta sentencia se utiliza para declarar una unidad, indicando la integración al programa, las unidades se declaran después de la sentencia PROGRAM. Las unidades estándar con que cuenta Turbo Pascal son la siguientes:

***Crt***

***Dos***

***Graph***

***Overlay***

***Printer***

***System***

***Estructura de las Unidades:***

Las unidades se componen de cuatro partes, declaración, interfaz, implementación e inicialización.

***Declaración:***

Esta sección es la que se encarga de asignar un nombre a la unidad, por consiguiente no se puede omitir. Su sintaxis es la siguiente:

***UNIT Nombre;***

***Interfaz::***

Esta sección se encarga de la declaración de todos los identificadores que pueden ser usados el programa, como son variables, constantes, procedimientos y funciones. En esta sección no se incluyen los códigos de los procedimientos ni funciones, solo se encuentra su declaración. Su sintaxis es la siguiente:

***INTERFACE***

***Uses Unidades;***

***Var***

***Let :String;***

***Procedure Nom;***

***Implementación:***

Esta sección es exclusiva para la unidad, contiene las declaraciones de etiquetas, variables, constantes, así como el contenido de los procedimientos y funciones declarados en la sección de interfaz. Su sintaxis es la siguiente:

***IMPLEMENTATION***

***Procedure Nom;***

***Begin***

***...***

***End;***

***Inicialización:***

Esta sección se utiliza para inicializar los valores de las variables, no es indispensable de las unidades. Su sintaxis es la siguiente;

***Begin***

***Let := `';***

***End.***

***Creación de una Unidad:***

Esta unida fue desarrollada para facilitar el manejo de información en la pantalla, cuenta con sus procedimientos.

***UNIT Prueba;***

***INTERFACE***

***PROCEDURE Centra (Cad: String; ren: Integer);***

***IMPLEMENTATION***

***USES Crt;***

***PROCEDURE Centra (Cad: String; ren: Integer);***

***VAR***

***C: Integer;***

***BEGIN***

***C := (80 − Length (Cad)) DIV 2;***

***GotoXY (C, ren);***

***Write (Cad);***

***END;***

***BEGIN***

***END.***

***Compilación de una Unidad:***

Las unidades se compilan con destino a disco, de esta manera el compilador generara un archivo con extensión TPU, después de verificar el código fuente. La compilación se puede ejecutar con las teclas ALT + F9, y la creación de la unidad con la tecla F9.

***Unidad VII***

***Archivos de Acceso secuencial:***

Estos archivos son llamados también archivos de texto, están formados por cadenas de caracteres separadas por un enter, su valor ASCII es de 13, su final esta delimitado por el carácter 26.

***Declaración de un Archivo secuencial:***

Un archivo de texto para ser manipulado, se requiere declara una variable de tipo TEXT, esto nos permitirá asignar la variable al archivo físico de texto. Su sintaxis es la siguiente:

***VAR***

***Archivo: TEXT;***

Una vez declarada la variable del archivo, se procede a la asignación del nombre del archivo a manipular.

***Asignación de archivos:***

La sentencia ASSIGN (), nos asignar el nombre de algún archivo a nuestra variable de tipo archivo. Su sintaxis es la siguiente:

***ASSIGN (VarArchivo, NombreArchivo);***

Donde NombreArchivo es una cadena de caracteres que contiene el nombre del archivo, la unidad de disco donde se encuentra y el directorio.

***Abrir un Archivo de Texto:***

Los archivos de texto se pueden abrir de tres formas diferentes, cuando son creados se usa el REWRITE (), cuando se abren de lectura RESET () y cuando se abren para agregar mas información APPEND ().

***Escribir datos en el Archivo de Texto:***

Para poder escribir datos en un archivo de texto, se requiere primero que el archivo se encuentre abierto en modo de escritura, usando la sentencia APPEND () o REWRITE (), posteriormente se utilizan la instrucción WriteLn (), para escribir los datos. Su sintaxis es la siguiente:

***WriteLn (Var_Archivo, Datos);***

En este ejemplo crearemos un archivo de texto, y grabaremos tres cadenas:

```
Program Graba_Texto;

Var

Arch: Text;

Begin

Assign (Arch, 'demo.txt');

ReWrite (Arch);

WriteLn (Arch, 'Primera cadena ');

WriteLn (Arch, 'Segunda cadena');

WriteLn (Arch, ' Tercera cadena ');

Close (Arch);

End.
```

***Leer datos de un Archivo de Texto:***

Para poder leer datos de un archivo de texto, se requiere primero que el archivo se encuentre abierto en modo de lectura, usando la sentencia RESET (), posteriormente se utilizan la instrucción ReadLn (), para leer los datos. Su sintaxis es la siguiente:

***ReadLn (Var_Archivo, Datos);***

En este ejemplo abriremos un archivo de texto, y leeremos su contenido:

***Program Leer_Texto;***

***Var***

***Arch: Text;***

***Cadena: String;***

***Begin***

***Assign (Arch, 'demo.txt');***

***Reset (Arch);***

***While NOT ( EOF ( Arch ) ) Do***

***ReadLn( Arch, Cadena );***

***Close( Arch );***

***End.***

***Archivos de acceso directo:***

Estos archivos también son llamados archivos con tipo, estos poseen una secuencia de componentes accesibles individualmente, a los cuales llamaremos registros, un registro es un agrupamiento de datos. Su sintaxis es la siguiente:

***Archivo: FILE OF Tipo_Registro;***

***Abrir un Archivo con Tipo:***

Los archivos de tipo se pueden abrir de dos formas diferentes, cuando son creados se usa el ReWrite (), y cuando se abren de lectura y escritura Reset ().

***Escribir datos en el Archivo con Tipo:***

Para poder escribir datos en un archivo de texto, se requiere primero que el archivo se encuentre abierto en modo de escritura, usando la sentencia Reset (), después se utilizan la instrucción Write (), para escribir los datos. Su sintaxis es la siguiente:

***WRITE (Var_Archivo, Var_Registro);***

En este ejemplo crearemos un archivo de tipo, y grabaremos tres registros:

***Program Graba_Registro;***

***Var***

***Arch :File of Integer;***

***K: Integer;***

***Begin***

***Assign (Arch, 'demo.dat');***

***ReWrite (Arch);***

***For K := 1 To 3 Do***

***Write (Arch, K);***

***Close (Arch);***

***End.***

***<u>Leer datos de un Archivo con Tipo:</u>***

Para poder leer datos de un archivo de tipo, se requiere primero que el archivo se encuentre abierto en modo de lectura, usando la sentencia Reset (), después se utilizan la instrucción Read (), para leer los datos. Su sintaxis es la siguiente:

***READ (Var_Archivo, Var_Registro);***

En este ejemplo abriremos un archivo de tipo, y leeremos su contenido:

***Program Leer_Registro;***

***Var***

***Arch: File Of Integer;***

***K: Integer;***

***Begin***

***Assign (Arch, 'demo.dat');***

***Reset (Arch);***

***While NOT (EOF (Arch)) Do***

***Read (Arch, K);***

***Close (Arch);***

***End.***

***<u>Ubicación física en un registro específico:</u>***

El procedimiento SEEK(), nos permite colocarnos en un registro especifico, mediante un parámetro numérico, pero debemos tener cuidado de que exista el número del registro. Su sintaxis es la siguiente:

***SEEK (Var_Archivo, Valor_Pos);***

***Fin de un Archivo:***

La función booleana EOF( ), nos indica cuando se llega al final de un archivo, no importando su tipo. Su sintaxis es la siguiente:

***Variable := EOF (Var_Archivo);***

***Fin de Línea:***

La función booleana EOLN (), nos indica cuando se llega al final de una línea de un archivo de texto, comúnmente se utiliza las lecturas por caracteres. Su sintaxis es la siguiente:

***Variable := EOLN (Var_Archivo);***

En este ejemplo abriremos un archivo de texto, y leeremos por caracter:

```
Program Leer_car;

Var

Arch: Text;

Cad: String;

Car: Char;

Begin

Assign (Arch, 'demo.txt');

Reset (Arch);

While NOT (EOF (Arch)) Do

Begin

Cad := '';

While NOT (EOLN (Arch)) Do

Begin

Read (Arch, Car);

Cad := Cad + Car;

End;

End;

Close (Arch);
```

***End.***

***<u>Cerrar un archivo:</u>***

Después de utilizar un archivo, es importante cerrarlo, evitando con esto que se dañe físicamente, el procedimiento que realiza esta operación es CLOSE ().Su sintaxis es la siguiente:

***CLOSE (Var_Archivo);***

***Unidad VIII***

***<u>Modos de Pantalla:</u>***

Turbo Pascal cuenta con cinco constantes predefinidas para indicar el modo de pantalla que se utilizara en el programa, estas están representados en la tabla siguiente:

| ***Constante*** | ***Valor*** | ***Modo de video*** |
|---|---|---|
| ***BW40*** | ***0*** | ***40x25 Blanco y negro en tarjeta de color*** |
| ***CO40*** | ***1*** | ***40x25 Color*** |
| ***BW80*** | ***2*** | ***80x25 Blanco y negro en tarjeta de color*** |
| ***CO80*** | ***3*** | ***80x25 Color*** |
| ***Mono*** | ***7*** | ***80x25 Monocromático*** |

***<u>Tabla de Colores:</u>***

Existen 8 constantes predefinidas para los fondos de la pantalla, este rango va de 0 a 7, estos representan los primero 3 dígitos del byte de atributo, los caracteres cuenta con 16 constantes para el color, este rango va de 0 a 15, estos representa los 4 dígitos siguientes, y por ultimo un dígito de parpadeo.

| ***Constante*** | ***Valor*** | ***Color*** |
|---|---|---|
| ***Black*** | ***0*** | ***Negro*** |
| ***Blue*** | ***1*** | ***Azul*** |
| ***Green*** | ***2*** | ***Verde*** |
| ***Cyan*** | ***3*** | ***Cyan*** |
| ***Red*** | ***4*** | ***Rojo*** |
| ***Magenta*** | ***5*** | ***Magenta*** |
| ***Brown*** | ***6*** | ***Marrón*** |
| ***Light Gray*** | ***7*** | ***Gris claro*** |
| ***Dark Gray*** | ***8*** | ***Gris oscuro*** |
| ***Light Blue*** | ***9*** | ***Azul claro*** |
| ***Light Green*** | ***10*** | ***Verde claro*** |
| ***Light Cyan*** | ***11*** | ***Cyan claro*** |
| ***Light Red*** | ***12*** | ***Rojo claro*** |
| ***Light Magenta*** | ***13*** | ***Magenta claro*** |
| ***Yellow*** | ***14*** | ***Amarillo*** |
| ***White*** | ***15*** | ***Blanco*** |
| ***Blink*** | ***128*** | ***Parpadeo*** |

***Variable CheckBreak:***

Esta variable nos permite activar o desactivar el chequeo del control break, la unión de estas dos teclas, se utilizan para quebrar la ejecución de un programa, la manera de bloquear este efecto, es asignando un valor falso a esta variable, puesto a que su valor predefinido es verdadero. Su sintaxis es la siguiente:

***CheckBreak := Valor_Booleano;***

***DirectVideo:***

Esta variable se inicializa como falsa, cuando existen problemas en la entrada y salida de texto a la pantalla, desactivando la escritura directa de caracteres a la memoria de video. Su sintaxis es la siguiente:

***DirectVideo:= Valor_Booleano;***

***ClrEol:***

Este procedimiento borra todos los caracteres de la línea actual, desde la posición del cursor hasta el final de la ventana.

***Clrscr:***

Este procedimiento limpia la ventana activa en la pantalla, dejando el cursor en la esquina superior izquierda.

***DelLine:***

Este procedimiento borra la línea en donde se encuentra el cursos, las demás líneas inferiores suben una posición.

***Delay ():***

Este procedimiento detiene la ejecución del programa el número de milisegundos indicado, estos van de 0 a 65535, la precisión del tiempo depende de los ciclos de reloj del procesador.

***GotoXY ():***

Este procedimiento sitúa al cursor en la posición indicada de la pantalla, donde X es el número de columnas deseado, tomando como base del 1 al 80, e Y es el número de renglones deseados, tomando como base del 1 al 25. Su sintaxis es la siguiente:

***GotoXY (X, Y);***

***TextBackGround ():***

Este procedimiento cambia el color del fondo de la ventana actual. Su sintaxis es la siguiente:

***TextBackGround (Color);***

***TextColor ():***

Este procedimiento cambia el color de los caracteres en la pantalla. Su sintaxis es la siguiente:

***TextBackGround (Color);***

***TextMode ():***

Este procedimiento cambia el modo de video de la pantalla, asignando los nuevos atributos con que contará. Su sintaxis es la siguiente:

***TextMode (Valor);***

***WhereX:***

Esta función nos devuelve el valor actual del cursor, conforme a su eje x.Su sintaxis es la siguiente:

***Variable := WhereX;***

***WhereY:***

Esta función nos devuelve el valor actual del cursor, conforme a su eje y. Su sintaxis es la siguiente:

***Variable := WhereY;***

***Window ():***

Este procedimiento nos permite restringir el área de trabajo en la pantalla, formando una sección delimitada por la nuevas dimensiones, para ello se define en los parámetros, la esquina superior izquierda y inferior derecha. Su sintaxis es la siguiente:

***Window (Xsi, Ysi, Xid, Yid);***

***Ejercitación***

- Realizar todos los ejemplos propuestos en el apunte y comentar como funcionan.
- Construir una unidad con procedimientos y funciones, que englobe los ejemplos propuestos, y un programa que los utilice.
- A continuación se exponen 5 problemas, las cuales deben ser resueltos con sus respectivos programas en Turbo Pascal.

***Procedimientos a Seguir:***

- Análisis y comprensión del problema.
- Planteamiento del algoritmo de solución.
- Codificación del algoritmo.
- Compilación y Verificación del resultado.

- Realizar un programa que simule una calculadora con las operaciones matemáticas básicas, el programa debe contar con un menú de opciones.
- Realizar un programa que cuente todas las palabras de un archivo de texto.
- Realizar un programa que almacene una agenda telefónica.
- Realizar un programa que convierta un número en base 10 a binario.
- Realizar un programa que calcule el valor de e.