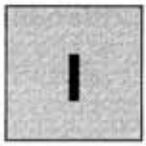
APENDICE 

GUIA DE REFERENCIA TURBO PASCAL 7.0, 6.0 y 5.5.

LENGUAJE

Palabras reservadas

and	**file**	**not**	**then**
array	**for**	**object**	**to**
asm	**function**	**of**	**type**
begin	**goto**	**or**	**unit**
case	**if**	**packed**	**until**
const	**implementation**	**procedure**	**uses**
constructor	**in**	**program**	**var**
destructor	**inherited***	**record**	**while**
div	**inline**	**repeat**	**with**
do	**interface**	**set**	**xor**
downto	**label**	**shl**	
else	**mod**	**shr**	
end	**nil**	**string**	

Directivas (*se pueden redefinir*)

absolute	**far**	**near**	**virtual**
assembler	**forward**	**private**	
external	**interrupt**	**public***	

* Sólo en versión 7.0

Identificadores

```
WriteLn      Exit      CadenaReal      System.MemAvail      Rosas
Un_mes       Nombre_Apellidos          Dia_del_Mes          Pedro_uno
```

Cadenas de caracteres

```
'Turbo'                    {Turbo}
''''                       {'}
''                         {cadena nula}
' '                        {espacio}
```

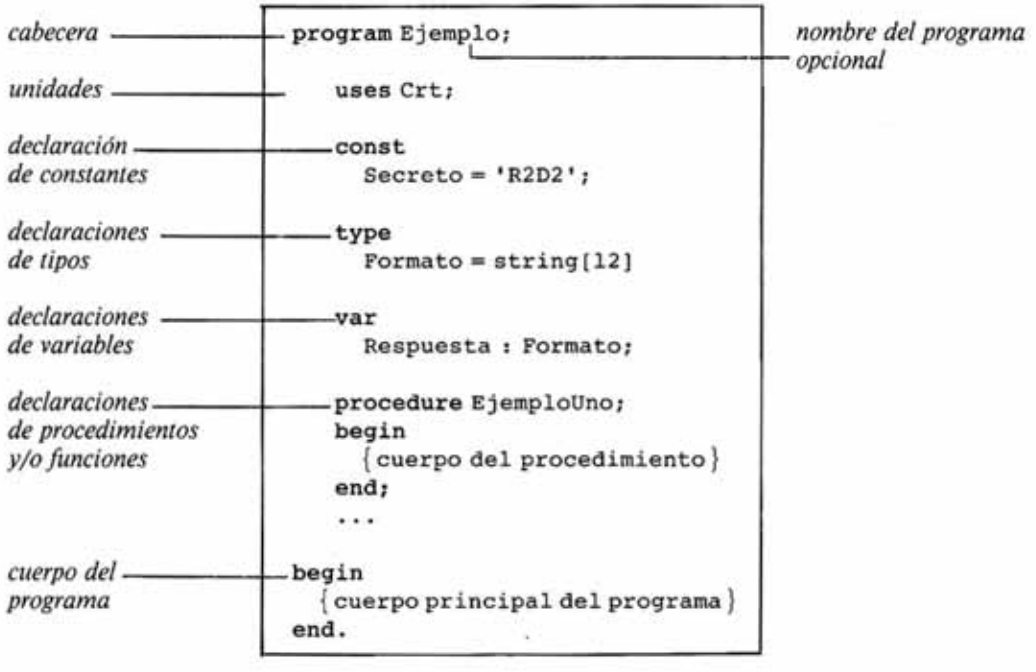
Comentarios

{ Cualquier texto encerrado entre llaves }
(* Es también un comentario válido *)

Línea de programa

Las líneas de programas Turbo Pascal tienen una longitud máxima de 126 caracteres.

Estructura de un programa



Procedimientos y funciones

cabecera procedimiento

```
procedure identificador(lista de parametros);
  {declaraciones de datos}
begin
  {cuerpo del procedimiento}
end;
```

cabecera función

```
function identificador (lista de parametros);
  {declaraciones de datos}
begin
  {cuerpo de la funcion}
end;
```

UNIDADES

Unidades estándar

Crt	Soporte de pantalla y teclados
Dos,WinDos	Funciones de propósito general DOS
Graph	Rutinas gráficas
Graph3	Gráficos compatibles versión 3
Overlay	Implementa el administrador de solapamientos
Printer	Acceso a la impresora
Strings	Tratamiento de cadenas terminadas en nulo
System	Rutinas de la biblioteca en tiempo de ejecución
Turbo3	Mantener compatibilidad con Turbo Pascal 3.0

Sintaxis de la unidad

```
unit identificador
interface lista de unidades;                    opcional
  {declaraciones públicas más}
  {cabeceras de todas las subrutinas públicas}
implementation
uses lista de unidades                          opcional
  {declaraciones privadas}
  {implementación de procedimientos y funciones}
begin
  {código de inicialización}                    opcional
end.
```

DISPOSITIVOS

`CON`	Terminal
`PRN`	Impresora
`AUX`	Dispositivo auxiliar
`LPT1`	Impresora
`NUL`	Dispositivo nulo

TIPOS DE DATOS

Tipos simples

Boolean (*lógicos*)	`true..false`
Números enteros	
Byte	`0..255`
Integer	`-32.768..32.767`
LongInt	`2.147.483.648..2.147.483.647`
ShortInt	`-128..127`
Word	`0..65.535`
Números reales	
Real	$2.9 \times 10^{-39} .. 1.7 \times 10^{38}$
Single	$1.5 \times 10^{-45} .. 3.4 \times 10^{38}$
Double	$5.0 \times 10^{-324} .. 1.7 \times 10^{308}$
Extended	$3.4 \times 10^{-4932} .. 1.1 \times 10^{4932}$
Comp	$-2^{63}+1 .. 2^{63}-1$
Carácter	
Char	Cualquier carácter ASCII
Cadenas	
String	Secuencia de hasta 255 caracteres ASCII. Si no se especifica un tamaño, se utilizará por defecto 255.
Punteros	
Pointer	Dirección de un elemento dato, procedimiento o función.

Tipos subrango

```
    0..99
 -128..127
Lunes..Domingo
```

Tipos enumerados

```
type
  Naipes = (Oros, Espadas, Bastos, Copas);
```

Tipos estructurados

Array

```
array [1.100] of Real
array [Boolean] of array [1..10] of array [Longitud] of Real;
packed array [1..10, 1..15] of Boolean
```

Registro
```
type Reg = record
   Nombre  : String [40];
   Edad    : Integer;
   Salario : Real;
         end;
```

Objeto
```
type Punto = object
                x,y : Integer;
             end;
```

Conjunto
```
Letras   := ['A','B','C','D','E']
NoLetras := [ ]
```

Puntero
```
type PunteroReal = ^Real;
New (P);
```

Pchar
```
type Pchar = ^Char;
```

Tipos procedimiento

```
type
  Proc    = procedure;
  InterX  = procedure (var X,Y : Integer);
  ProcCX  = procedure (S : string);
  FuncUna = function (X : Real) : Real;
```

VARIABLES Y CONSTANTES

Constantes predefinidas

`true`	*verdadero*
`false`	*falso*
`maxint`	*entero mayor disponible en máquina*

Declaración de variables

```
var
  x,y,z : Real;
  i,j,k : Integer;
  Digito : 0..9;
```

Variables absolutas

```
var
  Cad : string [40];
  LongCad : Byte absolute Cad;
```

Variables predefinidas

Variable	*Tipo*
`Input`	Archivo Text
`Output`	Archivo Text

Variable	*Tipo*
`ExitCode`	Integer
`FileMode`	Byte
`InOutRes`	Integer
`RandSeed`	LongInt
`StackLimit`	Word

Constantes con tipos

Tipo simple

```
const
  Maximo : Integer = 425;
  Factor : Real = -12.5;
const
  Min : Integer = 0;
  Max : Integer = 100;
type    declaración no válida
  Lista = array [Min..Max] of Integer;
```

la declaración Lista no es válida ya que Min y Max son constantes de tipos

Tipo cadena

```
const
  Cabecera     : string[10] = 'Ciudades';
  Nueva Linea  : string[2] = #13#10;
  Respuesta    : string[5] = 'Si';
```

Tipo estructurado

```
const
  Digitos : array[0..9] of Char =
  ('0','1','2','3','4','5','6','7','8','9');
const
  Digitos : array[0..9] of Char = '0123456789';

type
  Matriz = array [0..1, 0..1, 0..1] of Integer;
const
  Lista : Matriz = (((0,1), (2,3)), ((4,6), (7,9)));
```

Tipo registro

```
type
  Punto = record
            x,y : Real;
          end;
  Vector = array [0..1] of Punto;

const
  Origen : Punto = (X:0.0; Y:0.0);
  Linea  : Vecto = ((X:-4.5; Y:1.5),(X:6.4; Y:5.0));
```

Tipo objeto

```
type
  Punto = object
            x,y : Integer;
          end;

const
  Origen : Punto = (x:0; y:0);
```

Tipo conjunto

```
type
  Digitos = set of  0..9;
  Letras  = set of  'A'..'Z';
const
  Pares   : Digitos = [0,2,4,6,8];
  Vocales : Letras  = ['A','E','I','O','U'];
```

Tipo puntero

```
type
  Direccion = (Izda, Dcha, Arriba, Abajo);
  PNodo = ^NodoT;
  NodoT = record
            ...
          end;
const
  S1 : string[4] = 'ABAJO';
  N1 : NodoT = (Siguiente : nil; Symbolo : @S1; Valor : Abajo);
```

Tipo procedimiento

```
type
  ProcError = procedure (CodigoError : Integer);

procedure ErrorPorOmision (CodigoError : Integer); far;

begin
  WriteLn ('Error', CodigoError, ',');
end;

const
  ManipuladorErrores : ProcError = ErrorPorOmision;
```

OPERADORES

Aritméticos

Operador	*Sintaxis*	*Significado*
+	+expresión	Positivo (unitario)
+	expresión1 + expresión2	Suma (binaria)
-	-expresión	Negativo (unitario)

Operador	*Sintaxis*	*Significado*
-	expresión1 - expresión2	Resta (binario)
*	expresión1 * expresión2	Multiplicación
/	expresión1 / expresión2	División real
DIV	expresión1 DIV expresión2	División entera
MOD	expresión1 MOD expresión2	Resto (módulo)

Lógicos

Operador	*Sintaxis*	*Significado*
+	NOT expresión	Complemento
AND	expresión1 AND expresión2	AND
OR	expresión1 OR expresión2	OR inclusiva
XOR	expresión1 XOR expresión2	OR exclusiva

Relacionales

Operador	*Sintaxis*	*Devuelve verdadero si:*
=	expresión1 = expresión2	Expresiones son iguales
<>	expresión1 <> expresión2	Expresiones no son iguales
<	expresión1 < expresión2	expresión1 es menor que expresión2
<=	expresión1 <= expresión2	expresión1 es menor que o igual a expresión2
>	expresión1 > expresión2	expresión1 es mayor que expresión2
>=	expresión1 >= expresión2	expresión1 es mayor o igual que expresión2

De conjunto

Operador	*Sintaxis*	*Devuelve verdadero si:*
=	Conjunto1 = Conjunto2	Conjunto1 y Conjunto2 son idénticos. Cada elemento de Conjunto1 está contenido en Conjunto2, y cada elemento en Conjunto1 está contenido en Conjunto1.
<>	Conjunto1 <> Conjunto2	Uno de los conjuntos contiene al menos un elemento que no está en el otro conjunto.
<=	Conjunto1 <= Conjunto2	Cada elemento del Conjunto1 está también en Conjunto2.

Operador	*Sintaxis*	*Devuelve verdadero si:*
>=	`Conjunto1 >= Conjunto2`	Cada elemento de `Conjunto2` está también en `Conjunto1`
IN	`elemento IN Conjunto2`	El elemento `elemento` se encuentra en `Conjunto1`.
-	`Conjunto1 - Conjunto2`	Diferencia
+	`Conjunto1 + Conjunto2`	Unión
*	`Conjunto1 * Conjunto2`	Intersección

Dirección

@ Dirección de variable, procedimiento o función.

Concatenación

+ Concatena dos cadenas.

Prioridad de operadores

Prioridad	*Operadores*
1 (alta)	@ NOT *unitario* + -
2	*/ DIV, MOD, AND, SHL, SHR
3	*binario* +, -, OR, XOR
4 (baja)	=, <>, < >, <=, >=, IN

SENTENCIAS

Asignación

<identificador> := *<expresión>*

```
rango  := alto - bajo;
cuenta := cuenta + 1;
```

Compuesta

begin *<sentencias>* **end**

```
begin
  z := 5;
```

```
  GetReal ('Valor', ValorReal);
  WriteLn (ValorReal);
end
```

Selectiva (if)

if <*condición*> **then**
  <*sentencia*>
[**else**
  <*sentencia*>];

```
if Cad1 <> Cad2 then
begin
  Cad1 := Cad1 + Cad2;
  WriteLn(' nueva cadena ', Cad1);
end;
```

Selectiva (if-*anidada*)

if (*condición*) **then**
  sentencia
else if (*condición*) **then**
  sentencias
else if (*condición*) **then**
  sentencia
else
  sentencia

```
if x>y then
  if x>z then
    Write (x)
  else
    Write (z)
else
  if y>z then
    Write (y)
  else
    Write (z);
WriteLn ('es el Mayor');
```

Selectiva múltiple (*case*)

case <*selector*> **of**
  <*lista de constantes*>:
    <*sentencias 1*>;
  <*lista de constantes*>:
    <*sentencias 2*>;
    ...
  [**else**
    <*sentencia por defecto*>]
end;

```
case N of
    1,2   : WriteLn ('Primero');
    3,4,5 : WriteLn ('Bronce');
    7     : WriteLn ('Fin');
    else
      WriteLn ('Final')
end;
```

Sentencia repetitiva *for*

1. **for** <*índice*> := <*valor inicial*> **to** <*valor final*> **do** <*sentencia*>

```
for k := 1 to 10 do
  WriteLn ('cuerpo de bucle');
```

```
for j := 1 to 10 do
begin
  j := j + z;
  WriteLn ('pasos ;', pasos : 2);
  paso := paso + 1
end;
WriteLn ('fuera del bucle');
```

2. **for** <*índice*> := <*valor final*> **downto** <*valor inicial*> **do** <*sentencia*>

```
for i := 10 downto 1
begin
  Resultado := i * Resultado;
  WriteLn('Bucle', i, Resultado);
end;
```

Sentencia repetitiva *while*

while <*condición*> **do**
    <*sentencia*>;
    <*sentencia*> = *sentencia simple*
                    sentencia compuesta

```
paso := 1;
while paso <= 10 do
begin
  WriteLn ('Pasos:', paso:2);
  paso := paso + 1
end;
WriteLn('fuera del bucle');
While ch <> ' ' do Read(ch);
```

Sentencia repetitiva *repeat*

repeat
    <*sentencia*>
until <*condición*>

```
paso := 1;
repeat
   WriteLn('Pasos:',paso:2);
   paso := paso + 1
until paso > 10
WriteLn('Fuera del bucle');
```

AMBITO Y LOCALIDAD

El *ámbito* de un identificador es el conjunto de módulos (programas o subprogramas) en el que está legalmente declarado el identificador.

```
program Principal
  var
    w,x,y,z : ...                          variables globales
  procedure Uno ( var w:...; a:...);       parámetros locales w,a,x
    var
    x : ... ;
    begin
       ...
    end;
```

```
  procedure Dos ( c:...; var x...);          parámetros locales c,y,x,z,w
    var
      z,w : ... ;
    begin
      ...
    end;
begin {Principal}
  ...
  Uno (y,z);
  ...
  Dos (w,y,x);
  ...
end;
```

> Para eliminar posibles efectos laterales, evite utilizar variables globales en subprogramas o identificadores idénticos para cantidades diferentes.

PROCEDIMIENTOS Y FUNCIONES ESTANDAR

Turbo Pascal contiene procedimientos y funciones estándar (incorporadas o predefinidas) y variables predeclaradas en la unidad *System*. Todos ellos están declarados en la unidad *System*, en consecuencia no necesita ninguna sentencia **uses** cuando desee utilizar alguno/a de ellos/as.

Funciones aritméticas

Nombre	*Sintaxis*	*Descripción*
Abs	`Abs(x);`	Devuelve el valor (positivo) absoluto de su argumento.
ArcTan	`ArcTan(x:Real):Real;`	Arco tangente expresado en radianes.
Cos	`Cos(x:Real):Real;`	Coseno del argumento en radianes.
Exp	`Exp(x:Real):Real;`	Potencia exponencial del argumento (e^x).
Frac	`Frac(x:Real):Real;`	Parte decimal de un número real.
Int	`Int(x:Real):Real;`	Parte entera del argumento.
Ln	`Ln(x:Real):Real;`	Logaritmo neperiano (base e) del argumento.
Pi	`Pi:Real;`	Valor de *Pi* (3.1415926535897932385).
Sin	`Sin(x:Real):Real;`	Seno del argumento.
Sqr	`Sqr(x);`	Cuadrado del argumento.
Sqrt	`Sqrt(x:Real):Real;`	Raíz cuadrada del argumento.

Ejemplos:

```
y   := abs(x)*2;
z   := ArcTan(1.75);
Tan := Sin(X)/cos(x);
Pot := Exp(3);
R   := Frac(-245.123);
```

```
z := Int(345.678);
t := Ln(1.25);
z := ArcTan(Pi);
z := Sin(Pi);
f := Sqr(3.45);
f := Sqrt(1.2345);
```

Funciones de transferencia

Nombre	*Sintaxis*	*Descripción*
Chr	`Chr(x:Byte):Char;`	Devuelve el carácter correspondiente al código ASCII.
Ord	`Ord(x):LongInt;`	Número ordinal de un tipo ordinal.
Round	`Round(x:Real):LongInt;`	Redondea un valor real a entero largo.
Trunc	`Trunc(x:Real):LongInt;`	Trunca un valor de tipo real a entero.

Ejemplos:

```
Write(Chr(i));
Ord('A')
```

```
Round(5.449)    devuelve 5
Trunc(-3.14)    devuelve -3
Trunc(6.5)      devuelve 6
```

Procedimientos de flujo de control

Nombre	*Sintaxis*	*Descripción*
Break	`Break;`	Termina una sentencia **for**, **while** o **repeat**.
Continue	`Continue;`	Contorna con la siguiente iteración de una sentencia **for**, **while** o **repeat**.
Exit	`Exit;`	Termina inmediatamente el bloque actual (procedimiento, función o programa).
Halt	`Halt[(CodigoSalida: Word)];`	Detiene la ejecución del programa y retorna al sistema operativo.
RunError	`RunError[(CodigoError: Byte)];`	Detiene la ejecución del programa y genera un error en tiempo de ejecución.

Ejemplos:

```
if t = 's' then Exit;
```

```
if p = nil then
  RunError(204);
```

Procedimientos o funciones ordinales

Nombre	*Sintaxis*	*Descripción*
Dec	`Dec(var x[;n:LongInt]);`	Decrementa una variable.
Inc	`Inc(var x[;n:LongInt]);`	Incrementa una variable.
High*	`High(x);`	Devuelve el valor más alto en el rango del
Low*	`Low(x);`	argumento que debe ser un tipo array o un tipo cadena.
Odd	`Odd(x:LongInt):Boolean;`	Devuelve el valor más bajo en el rango del argumento (tipo array o cadena).
Pred	`Pred(x);`	Predecesor del argumento (*x* de tipo ordinal).
Succ	`Succ(x);`	Sucesor del argumento (*x* de tipo ordinal).

* Sólo está implementado en la versión 7.0.

Ejemplos:

```
Dec(z);
Inc(z);

for i := 0 to High(x) do
  s := s + x[i];
```

```
for Dia := Low(ADia) to High (ADia) do
begin
  ...
end;

Pred('z')        devuelve '4'
Succ(1946)       devuelve 1947
```

Tratamiento de cadenas

Procedimientos

Nombre	*Sintaxis*	*Descripción*
Delete	`Delete(var s:string; Pos,Len:Integer);`	Borra una subcadena a partir de una posición en una cadena.
Insert	`Insert(var s:string; var D: string;Pos:Integer);`	Inserta una subcadena en una posición de una cadena.
Str	`Str(I:Integer; var s:string);` `Str(R:Real; var s:string);`	Convierte un valor numérico a cadena.
Val	`Val(s:string; var R:Real,P: Integer);` `Val(s:string; var I,P:Integer;`	Convierte una cadena a su valor numérico.

Funciones

Nombre	*Sintaxis*	*Descripción*
Concat	`Concat(s1,s2,...,sn:string): string;`	Concatena (une) cadenas.
Copy	`Copy(s:string;Pos,Long: Integer):string;`	Copia una cadena dada.
Length	`Length(s:string):Integer;`	Longitud de una cadena.
Pos	`Pos(Patron,Fuente:string): Integer;`	Posición de la primera ocurrencia de una subcadena.

Ejemplos:

```
Cadl := 'computadora';
Delete(Cadl,4,3);

Insert('put'Cadl,4);

s := Concat('xyz','LUIS');

Str(Maxint,Cad);
WriteLn(Cad);

Val('32425',r,e);

s := Copy(s,2,3);
t := Length(Cad);
```

Punteros y direcciones

Nombre	*Sintaxis*	*Descripción*
Addr	`Addr(x):Pointer;`	Devuelve la dirección de un objeto especificado.
Assigned*	`Assigned(var P):Boolean;`	Comprueba si una variable procedimiento o puntero es **nil**.
CSeg	`CSeg:Word;`	Valor actual del registro CS.
DSeg	`Dseg:Word;`	Valor actual del registro DS.
Ofs	`Ofs(x):Word;`	Desplazamiento de un objeto especificado.
Ptr	`Ptr(Seg,Ofs:Word):Pointer;`	Convierte una base segmento y una dirección de desplazamiento a un valor tipo puntero.
Seg	`Seg(x):Word;`	Dirección del segmento de una variable o rutina.
Sptr	`Sptr:Word;`	Valor actual del registro SP.
Sseg	`Sseg:Word;`	Valor actual del registro SS.

* Sólo en la versión 7.0.

Ejemplos:

```
p := Addr(p);
if Assigned(p) then WriteLn('ok');
p := Ptr(Cseg,I);
p := Ptr(Dseg,I);
WriteLn('offset='offset=;Ofs(p2));
bajo := Ptr(0000,$046c);
WriteLn('seg(I)=',seg(I));
```

Asignación dinámica

Procedimientos

Nombre	*Sintaxis*	*Descripción*
Dispose	`Dispose(var p:pointer[,Destructor]*);`	
FreeMem	`FreeMem(var p:pointer;Tamaño:Word);`	
GetMem	`GetMem(var p:pointer;Tamaño:Word);`	
New	`New(var p:pointer[,Init:Constructor]*);`	

* La parte opcional (constructor/destructor) sólo en la versión 7.0.

Funciones

Nombre	*Sintaxis*	*Descripción*
MaxAvail	`MaxAvail:LongInt;`	Tamaño del bloque disponible mayor del montículo (*heap*).
MemAvail	`MemAvail:LongInt;`	Cantidad de memoria libre en el montículo (*heap*)

Ejemplos:

```
var
p:^string;
pl:^Real;

Dispose(p);
FreeMem(pl,10);

GetMem(pl,2);

New(p);
```

```
while MaxAvail do
  GetMem(p,65535);


write('Existen:',MemAvail);
```

Procedimientos y funciones diversas

Procedimientos

Nombre	Sintaxis	Descripción
Exclude*	`Exclude(var s:set of t; i:t);`	Excluye un elemento de un conjunto.
FillChar	`FillChar(var x; cuenta:word; valor);`	Rellena un número determinado de bytes contiguos con un valor especificado.
Randomize	`Randomize;`	Inicializa el generador de números aleatorios con una semilla.

* Sólo existe en la versión 7.0.

Funciones

Nombre	Sintaxis	Descripción
Hi	`Hi(x):Byte;`	Byte de mayor pero del argumento.
Include	`Include(var s:set of t;I:T);`	Incluye un elemento en un conjunto.
Lo	`Lo(x):Byte;`	Byte de menor pero del argumento.
Move	`Move(var Fuente,Dest;Cuenta: Word);`	Copia un número especificado de bytes contiguos de un rango fuente a un rango destino.
ParamCount	`ParamCount:Word;`	Devuelve el número de parámetros pasados en la línea de órdenes.
ParamStr	`ParamStr(Indice):string;`	Parámetro de la línea de órdenes.
Random	`Random[(Rango:Word)];`	Devuelve un número aleatorio.
Sizeof	`Sizeof(x):word;`	Número de bytes ocupado por el argumento.
Swap	`Swap(x);`	Intercambia los bytes más altos y menos altos del argumento.
TypeOf	`TypeOf(x):Pointer;`	Devuelve un puntero a una tabla de métodos virtuales de tipos objeto.
UpCase	`UpCase(ch:char):char;`	Convierte un carácter a mayúsculas.

Ejemplos:

```
Exclude(s,I);
FillChar(A,Sizeof(A),z);
Randomize;
```

```
for i := 1 to 10 do
  write(Random(MaxInt):8);
WriteLn('I=',I:5,'Byte alto=',Hi(I):3);
Include(S,I);
WriteLn('byte bajo',Lo(n):6);
Move(car[1],car[50],50);
for i := 1 to ParamCount do
  WriteLn(i:2,':',ParamStr(i))
WriteLn('Real..',Sizeof(Real));
w := 240; w := Swap(w); WriteLn('w='w);
WriteLn('Car:UpCase');
for ch := 'a' to 'z' do
  WriteLn(ch,UpCase(ch),' ');
```

Procedimientos de tratamiento de archivos

Nombre	*Sintaxis*	*Descripción*
ChDir	`ChDir(s:string);`	Cambia directorio actual.
GetDir	`GetDir(D:byte; var s:string);`	Obtiene directorio actual.
MkDir	`MkDir(s:string);`	Crea un directorio.
RmDir	`RmDir(s:string);`	Borra un directorio vacío.

Ejemplos:

```
ChDir(Camino);
GetDir(1,s);
MkDir(Camino);
RmDir('\Aux');
```

Entradas/Salidas

Procedimientos

Nombre	*Sintaxis*	*Descripción*
Append	`Append(var f:Text);`	Abre un archivo de texto, ya existente, para añadir.
Assign	`Assign(var f:Nombre);`	Asigna el nombre de un archivo externo a una variable archivo.

Nombre	*Sintaxis*	*Descripción*
BlockRead	`BlockRead(var f:file;var Buf; Cuenta:Word[;var Resultado:Word]);`	Lee uno o más registros de un archivo sin tipos.
BlockWrite	`BlockWrite(var f:file;var Buf; Cuenta:Word[;var Resultado:Word]);`	Escribe uno o más registros en un archivo sin tipos.
Close	`Close (var F);`	Cierra un archivo abierto.
Erase	`Erase(var F);`	Borra un archivo externo.
FileSize	`FileSize(var F);LongInt;`	Devuelve el tamaño actual de un archivo (no de texto).
Flush	`Flush(var F:Text);`	Limpia el *buffer* de un archivo de texto de salida.
Read	`Read(F,Vl[,V2,...Vn]);`	Lee uno o más valores de un archivo.
ReadLn	`ReadLn(var F:Text;]Vl[,V2,..., Vn]);`	Igual que `Read` y después salta al principio de la línea siguiente.
Rename	`Rename(var F:NuevoNombre);`	Renombra un archivo externo.
Reset	`Reset(var F[:file;TamaReg: Word]);`	Abre un archivo existente.
ReWrite	`ReWrite(var F[:file;TamaReg: Word]);`	Crea y abre un archivo nuevo.
Seek	`Seek(var F;N:LongInt);`	Mueve la posición actual de un archivo a un componente especificado.
SetTextBuf	`SetTextBuf(var F:Text;var Buf [; Tamaño:Word]);`	Asigna una memoria intermedia de E/S a un archivo de texto.
Truncate	`Truncate(var F);`	Trunca un archivo con tipos o sin tipos.
Write	`Write([var F:Text;]pl[,p2,..., pn]);`	Escribe en un archivo de texto.
	`Write(f,vl,[,v2,...,vn]);`	Archivos con tipos.
WriteLn	`WriteLn([var F:Text;]pl[,p2, ...,pn]);`	Igual que `Write` y luego escribe un fin de línea.

Funciones

Nombre	*Sintaxis*	*Descripción*
Eof	`Eof(var F):Boolean;`	Devuelve el estado fin de archivo.
Eoln	`Eoln[(var F:Text)]:Boolean;`	Devuelve el estado fin de línea de un archivo de texto.

Nombre	*Sintaxis*	*Descripción*
`FilePos`	`FilePos(var F):LongInt;`	Posición actual de un archivo con o sin tipos.
`IOResult`	`IOResult:Integer;`	Estado de la última operación de E/S realizada.
`SeekEof`	`SeekEof[(var F:Text)]:Boolean;`	Estado fin de archivo de un archivo.
`SeekEoln`	`SeekEoln[(var F:Text)];`	Estado fin de línea de un archivo.

Ejemplos:

```
var Tf:Text;
    F:File;
Assign(Tf,'TEST,TXT');
Rewrite(Tf);
Close(Tf);
Append(Tf);
BlockRead(F,A,10,Resultado);
BlockWrite(F,A,1,Resultado);
Erase(F)

Seek(F,FileSize(F));
Flush(F);

while not Eof(Tf) do
begin
  if Eoln(Tf)
     then writeLn;
if Length(s)>0
   then Rename(F,S);
```

```
Reset(F);

Seek(Tf,12);
Read(Tf,ch);

Set Text Buf(F,Buffer,512);

Existe := (IOResult = 0);

while not SeekEof(Tf) do
begin
  ReadLn(Tf,S);
  Inc(C2);
end

while not SeekEoln(Tf) do
begin
  ReadLn(Tf,Ch);
  write(ch);
end;
```

FUNCIONES DEFINIDAS POR EL PROGRAMA

Declaración

function <*identificador*> (<*parámetros formales*>) : <*tipo*>;
    <*declaraciones locales*>
begin
    <*cuerpo de la función*>
end;

ARCHIVOS

Preparación de un archivo

- Enlace entre un archivo interno y un archivo externo.

  Assign (<*identificador de archivo*>, <*nombre archivo externo*>);

Creación de un archivo de registros

- Preparar el sistema para enviar datos de la estructura interna al archivo externo creando y abriendo un nuevo archivo para recibir salida de la computadora.

  Rewrite (<*identificador de archivo*>);

Almacenar datos en un archivo

Write (<*identificador de archivo*>, <*identificador de salida*>);

Cerrar un archivo

Close (<*identificador de archivo*>);

Lectura de un archivo de registros

Procedimiento ***Reset*** *(Inicializar un archivo)*

Reset (<*identificador de archivo*>);

Lectura de un archivo de registros

Read (<*identificador de archivo*>, <*identificador de entrada*>);

Final de archivo

Eof (<*identificador de archivo*>);

Borrar archivos

Erase (<*identificador de archivo*>);

Cambiar registros en un archivo de registros

Función de posición ***FilePos***

FilePos (<*identificador de archivo*>);

Posicionamiento en registro

Seek (<*identificador de archivo*>, <*número de registro*>);

DIRECTIVAS DE COMPILACION

Directivas de conmutación

Directiva	*Significado*	*Valor por defecto*	*Tipo*
A	Alinear datos	{$A+}	Global
B	Evaluación lógica (*boolena*)	{$B–}	Local
D	Información de depuración	{$D+}	Global
E	Emulación	{$E+}	Global
F	Force far calls	{$F–}	Local
G	Generación de código 80286	{$G–}	Global
I	Verificación de E/S	{$I+}	Local
L	Generación de información de símbolos locales	{$L+}	Global
N	Proceso numérico	{$N–}	Global
P	Parámetros de cadenas abiertas	{$P–}	Local
O	Generación de código de solapamiento	{$O–}	Local
Q	Generación de código de verificación de desbordamiento	{$Q–}	Local
R	Verificación de rango	{$R–}	Local
S	Verificación de desbordamiento de pila	{$S+}	Local
T	Tipos de valores puntero, generados, por el operador @	{$T–}	Global
V	Verificación de tipos en cadenas pasados como parámetros variables	{$V+}	Local
X	Activa o desactiva sintaxis extendida de Turbo Pascal	{$X+}	Global
Y	Información de referencias a símbolos	{$Y+}	Global

Directivas parámetro

Directiva	*Significado*	*Sintaxis*
I	Archivo de inclusión	{$I *nombrearchivo*}
L	Enlaza el archivo reforzado con el programa o unidad que se está compilando	{$L *nombrearchivo*}
M	Requisitos de asignación de memoria: *long*, tamaño de la pila; *min* y *max*, tamaños mínimo y máximo del montículo	{$M *long,min,max*}
O	Nombre de unidad de recubrimiento (solapamiento)	{$O *nombreunidad*}

Directivas de compilación condicional

Directiva	*Significado*
{DEFINE *nombre*}	Define un símbolo condicional con el nombre dado.
{$ELSE}	Conmuta entre compilación.
{$ENDIF}	Termina la compilación condicional iniciada por la última directiva {$IFxxx}.
{$IFDEF *nombre*}	Compila el texto fuente que le sigue si está definido *nombre*.

Directiva	*Significado*
{$IFNDEF}	Compila el texto fuente que le sigue si *nombre* no está definido.
{$IFOPT *conmutador*}	Verdadero o falso de acuerdo a que la directiva *conmutador* esté activado o desactivado.
{$UNDEF *nombre*}	Indefine un símbolo condicional definido anteriormente.

```
{$IFxxx}                {$IFxxx}                {$IFDEF nombre}
  ...                     ...                     <sentencias-1>
{$ENDIF}                {$ELSE}                 {$ELSE}
                          ...                     <sentencias-2>
                        {$ENDIF}                {$ENDIF}
```

Símbolos condicionales

Símbolo	*Significado*
VER70	Siempre definido, indicando versión 7.0.
MSDOS	Siempre definido, indicando que el sistema operativo es MS-DOS o PC-DOS.
CPU86	Siempre definido, indicando que la CPU pertenece a la familia de procesadores 80 × 86.
CPU87	Definido si un coprocesador matemático está presente en tiempo de compilación.

CONTROL DE DISPOSITIVOS

Las unidades *Crt* y *Printer* son utilizadas con mucha frecuencia por los programadores. *Printer* envía salida a su impresora. La unidad *Crt* implementa una amplia y potente gama de rutinas que le proporcionan un control completo de características de su PC tales como: control del modo de pantalla, códigos de teclado extendido, colores, ventanas y sonido. Por su importancia práctica para el programador, seleccionamos las características más notables utilizadas en programación profesional.

Recuerde que para utilizar la unidad *Crt* o *Printer*, en su programa, debe incluir la cláusula **uses** como cualquier otra unidad:

```
uses Crt, Printer;
```

Caracteres de control

Carácter	*Nombre*	*Descripción*
#7	BELL	Emite un sonido (pitido) del altavoz interno.
#8	BS	Retrocede el cursor una columna.
#10	LF	Avanza el cursor una línea abajo.
#13	CR	Retorna el cursor al extremo izquierdo de la línea siguiente.

Teclas de edición de entrada de líneas

Tecla de edición	*Descripción*
RETROCESO (←┘)	Borra el último carácter introducido.
ESC	Borra la línea de entrada completa.
ENTER (Intro)	Termina línea de entrada y almacena en almacenamiento temporal. Genera marca fin de línea.
CTRL+S	Igual que Retroceso (*Backspace*).
CTRL+D	Llama un carácter de la última línea introducida.
CTRL+A	Igual que *Esc*.
CTRL+F	Llama a la última línea introducida.
CTRL+Z	Termina la última línea introducida y genera una marca fin de archivo.

Funciones

Nombre	*Sintaxis*	*Descripción*
`KeyPressed`	`KeyPressed`	Devuelve *true* si se ha pulsado una tecla del teclado; *false* en caso contrario.
`Readkey`	`Readkey`	Lee un carácter del teclado.
`WhereX`	`WhereX`	Devuelve coordenada `X` de la posición actual del cursor.
`WhereY`	`WhereY`	Devuelve coordenada `Y` de la posición actual del cursor.

Ejemplos:

```
repeat
  Write('Zz');
until KeyPressed;
```

```
Car := Readkey;

gotoxy(1, whereY-1);
```

Nombre	*Sintaxis*	*Descripción*
`InsLine`	`InsLine`	Inserta una línea vacía en posición del cursor.
`LowVideo`	`LowVideo`	Selecciona caracteres de baja intensidad.
`NormVideo`	`NormVideo`	Selecciona caracteres normales.
`NoSound`	`NoSound`	Desactiva altavoz interno de la computadora.
`Sound`	`Sound(HZ:Word)`	Arranca altavoz interno.
`TextBackground`	`TextBackground(Color:Byte)`	Selecciona color de fondo.

Nombre	*Sintaxis*	*Descripción*
TextColor	`TextColor(Color:Byte)`	Selecciona color de carácter (Primer plano).
TextMode	`TextMode(Modo:Word)`	Selecciona un modo de texto especificado.
Window	`Window(x1,y2,x2,y2:Byte)`	Define una ventana de texto en pantalla.

Procedimientos

Nombre	*Sintaxis*	*Descripción*
AssignCrt	`AssignCrt(var F:Text)`	Asocia un archivo de texto con la ventana `Crt`.
ClrEol	`ClrEol`	Borra todos los caracteres desde la posición del cursor hasta el final de la línea.
ClrScr	`ClrScr`	Borra la pantalla y cursor a posición inicial.
Delay	`Delay(ms:Word)`	Retarda un número especificado de segundos.
DelLine	`DelLine`	Borra la línea que contiene el cursor y mueve todas las líneas inferiores a él, una línea hacia arriba.
GotoXY	`GotoXY(x,y:Byte)`	Posiciona el cursor en coordenadas (x,y).
HighVideo	`HighVideo`	Selecciona caracteres de alta intensidad.

```
Gotoxy(1,10);
Write('Mackoy');
ClrEol;
ReadLn(Car);
```

```
Sound(440);
Delay(500);
NoSound;
```

```
Window(1,10,60,20);
DelLine;
Write('Mortimer');
InsLine;
```

```
repeat
  x := Succ(Random(71));
  y := Succ(Random(16));
  Window(x,y,x+Random(10),y+Random(10));
  TextBackground(Random8);
  ClrScr;
until KeyPressed;
```