3. DATOS SIMPLES

Conceptos: *Dato, Constante, Literal, Variable, Tipos de dato, Ordinal, Predefinido, Enumerado, Subrango, Compatibilidad.*

Resumen: En este capítulo se presentan los primeros elementos básicos de programación: los distintos tipos de datos que se pueden emplear en un programa en TurboPascal. Se comienza por distinguir entre constantes y variables y posteriormente se lleva a cabo una clasificación de los datos más simples según su naturaleza y formato de representación: integer, byte, real, boolean…

Objetivos específicos. Al finalizar el tema, el alumno deberá ser capaz de:

a) Definir el concepto de constante y de variable (*Conocimiento*)
b) Describir los tipos de datos simples en el lenguaje de programación Turbopascal y su formato de representación (*Conocimiento*)
c) Escribir la declaración de constantes y variables de cualquiera de los tipos de datos simples (*Comprensión*)
d) Definir el concepto de compatibilidad entre datos (*Conocimiento*)
e) Escribir la declaración de tipos de datos simples predefinidos: enumerados y subrangos (*Comprensión*)
f) Seleccionar el tipo de dato más adecuado para una aplicación determinada (*Aplicación*)

3.1. INTRODUCCIÓN

Un dato es cualquier información codificada y utilizada por algún programa que necesite un espacio de almacenamiento en el ordenador. La codificación significa que cada dato estará representado por una serie de símbolos manejables por el ordenador. Aunque, habitualmente, para facilitar su utilización por el programador cada dato tiene asignado un nombre o **identificador** en el programa fuente. Pueden realizarse distintas clasificaciones en función de diferentes características de los datos. Por ejemplo, dependiendo de:

a) si varían o no durante la ejecución de un programa, pueden ser *variables* o *constantes*.
b) su naturaleza (lo que representan), tamaño que ocupan en memoria, formato de codificación y funcionalidad (qué operaciones se pueden realizar con ellos), pueden clasificarse en distintos *tipos de dato*
c) cómo y en qué lugar de la memoria se almacenan, serán *estáticos* o *dinámicos*

A continuación se explicará cada unos de estos grupos de datos.

3.2. CONSTANTES

Las constantes son datos que no cambian de valor durante la ejecución del programa. Pueden clasificarse en: **constantes literales**, **constantes con nombre**, **expresiones constantes** y **constantes con tipo**. Por ejemplo, en la sentencia de asignación:

```
VolEsfera := (4/3)*PI*(R*R*R);
```

`4` y `3` son constantes literales, que ya tienen un significado para el lenguaje TurboPascal y `PI` podría declararse como una constante con nombre[2].

3.2.1. Constantes literales y constantes con nombre

Las constantes literales y las constantes con nombre pueden ser de diferentes *tipos*: numéricas enteras, numéricas reales, lógicas o *booleanas*, caracteres, cadenas de caracteres, de un tipo conjunto o de un tipo enumerado definido por el usuario. Si bien las constantes literales ya tienen un significado específico para el lenguaje TurboPascal, las constantes con nombre deben definirse (declararse) antes de su utilización en la sección de declaraciones de constantes del programa. En caso de declararse varias constantes puede hacerse una a continuación de la otra y después de la palabra reservada `CONST`.

Sintaxis:
```
CONST Identificador_1 = Valor_1;
      Identificador_2 = Valor_2;
      ...
      Identificador_n = Valor_n;
```

Ej.:
```
Const Pi       = 3.141592;
      Caracter = 'N';
      Numero   = 11;
      Cierto   = true;
      Saludo   = '¡Hola!';
      Letras   = ['A'..'Z','a'..'z'];
```

[2] En realidad, como se verá más adelante, el identificador `Pi` corresponde a una **función estándar** definida en TurboPascal en la unidad `system`

Mientras que las constantes numéricas no emplean ningún formato especial o utilizan caracteres adicionales, las constantes de tipo carácter o cadena de caracteres se escriben siempre entre comillas simples

Ej.: `'a' , '3' , '*' , 'Escuela' , 'Me llamo Luis'`

Las constantes de tipo conjunto se escriben delimitando los elementos (si hay varios se separan por comas) entre corchetes.

Ej.: `[3,11,25,29,34,41] , ['a','e','i','o','u']`

3.2.2. Expresiones constantes

Pueden definirse también **expresiones constantes** que se evalúan al compilar (esto es una extensión del Pascal estándar característico de TurboPascal), es decir, sin necesidad de ejecutar el programa.

Sintaxis: `CONST` *`Identificador = Expresion`*`;`

```
Ej.: Const   Pi         = 3.141592;
             DosPi      = 2 * Pi;     (* Expresion constante *)
             Largo = 45;
             Ancho      = 3;
             Alto  = 4;
             Volumen    = Largo*Ancho*Alto;      (* Expr. cte *)
             Vocales    = ['a','e','i','o','u'];
             Letras     = ['A'..'Z','a'..'z'];
             Digitos    = ['0'..'9'];
             Alfanum    = Letras + Digitos;      (* Expr. cte *)
```

Para definir las expresiones, en el segundo término de la igualdad pueden utilizarse constantes literales, constantes con nombre y algunos operadores y funciones estándar de TurboPascal (@ para expresiones de direcciones de memoria constantes exclusivamente, `Abs`, `Chr`, `Hi`, `Length`, `Lo`, `Odd`, `Ord`, `Pred`, `Ptr`, `Round`, `SizeOf`, `Succ`, `Swap`, `Trunc`).

3.2.3. Constante con tipo

Existe otro tipo de dato llamado genéricamente **constante con tipo** que, aunque se define en la sección de declaraciones como *constante*, no lo es en la práctica, ya que su valor puede variar durante la ejecución del programa. Las constantes con tipo sirven, en realidad, para *inicializar* variables (asignar valores a una *variable* antes de la primera sentencia del programa). En la declaración se especifica tanto el tipo de la "constante" como el valor inicial asignado. Como se verá más adelante, además de los tipos antes mencionados, pueden ser de tipo conjunto (`Set`), matriz (`Array`) o registro (`Record`).

Sintaxis: `CONST` *`Identificador : Tipo = Valor`*`;`

```
Ej.:   Type Letras    = set of Char;
            Combina   = array[1..7] of integer;
            Matriz3x3 = array[1..3,1..3] of real;
            Punto= record
                      X, Y : real
                      end;
            Persona   = record
                      Nombre : string[40];
                      Tlfno : string[9];
                      end;
       Const n : integer = 17;
             saludo : string = 'Buenos dias';
```

```
x : array[1..3] of char = ('?','3','a');
digitos : letras = ['0'..'9'];
pares : set of char = ['A','E','I','O','U']
MIdent: Matriz3x3 = ((1,0,0),(0,1,0),(0,0,1));
Origen : Punto = (X:0.0; Y:0.0);
Responsable : Persona =
        (Nombre:'Juan'; Tlfno:'5550000');
```

3.3. VARIABLES

Una variable es un dato almacenado en la memoria del ordenador cuyo valor puede cambiar durante la ejecución del programa. Es, en realidad, una dirección de memoria con nombre, ya que asocia al identificador de la variable un espacio de la memoria donde almacenar el valor dado a la variable. Como cualquier otro elemento en un programa de TurboPascal deben declararse antes de su utilización.

Sintaxis: `VAR` *`Ident_1, Ident_2,... Ident_n : Tipo`*`;`

Ej.:
```
Var   NumeroAlumnos   : integer;
      NumMatricula    : string[5];
      Apellidos       : string[40];
      Nota            : real;
      Aprobado        : boolean;
      GrupoPracticas  : char;
```

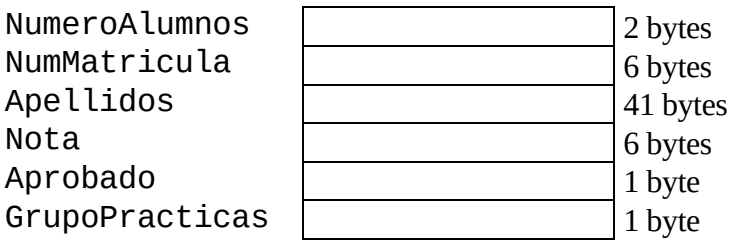

Figura 11. Espacio reservado en memoria para las variables durante la ejecución del programa

Una vez declarada una variable se reserva un espacio en la memoria que almacenará el valor que se asigna en un momento dado a esa variable durante la ejecución del programa:

Ej.:
```
begin                  (* Comienzo del programa *)
NumeroAlumnos:=40;     (* Se le asigna el entero 40 *)
```

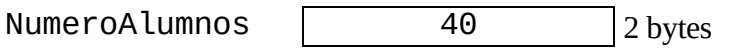

Figura 12. Asignación de un valor a una variable durante la ejecución de un programa

```
Writeln(NumeroAlumnos);  (* Se visualiza su valor
                                por pantalla *)
NumeroAlumnos:=50;    (* Se le asigna otro valor *)
```

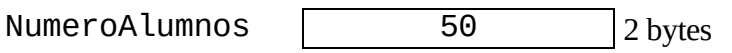

Figura 13. Asignación de otro valor a una variable durante la ejecución de un programa

```
{ Continuación del programa ... }
```

En la declaración de variables puede utilizarse cualquier tipo de dato predefinido de TurboPascal (que se verán en el siguiente apartado del capítulo) o cualquiera definido ya por el

usuario, e incluso puede definirse a la vez un nuevo tipo junto con la variable correspondiente, pero esto no se recomienda.

Ej.:
```
Var  a : 0..9;
     b : string[30];
     c : array[1..10] of real;
```

Lo recomendable es hacer primero la declaración de tipos de dato y luego la de variables, como se muestra a continuación:

```
Type digito = 0..9;
     nombre = string[30];
     vector = array[1..10] of real;
Var  a : digito;
     b : nombre;
     c : vector;
```

Estas declaraciones son equivalentes a las anteriores pero son más correctas técnicamente.

3.4. TIPOS DE DATOS EN TURBOPASCAL

Un tipo es un conjunto de valores de datos. Anteriormente se ha realizado una clasificación de los datos dependiendo de si modifican o no su valor durante la ejecución del programa. Por otro lado, puede realizarse, además, otra clasificación según la naturaleza o *tipo* de dato empleado: datos numéricos, caracteres, cadenas de caracteres, etcétera, independientemente de que sean datos constantes o variables.

La clasificación de los tipos es muy importante porque, en primer lugar, el programa debe conocer el tipo de dato que está manejando con el fin de reservar espacio en memoria para almacenarlo y cómo lo va almacenar (su representación o codificación binaria). En segundo lugar, es importante determinar los tipos de datos que se van a utilizar para prevenir un emparejamiento incorrecto de datos durante su manipulación. De esta manera es posible detectar errores en las asignaciones durante la compilación. Y en tercer lugar, el tipo de dato, además, determina qué operaciones se pueden realizar con ellos y la forma de ejecución de las operaciones. Los tipos de dato pueden clasificarse en cuatro grupos: simples, estructurados, *procedurales* o *procedimentales* y punteros.

3.5. TIPOS SIMPLES Ó ELEMENTALES

Los tipos simples o elementales **no** están definidos como combinación de otros. En TurboPascal, prácticamente todo tipo de dato ha de reducirse a una combinación de estos tipos elementales.

3.5.1. Tipos Ordinales (o Escalares)

Representan un conjunto finito y linealmente ordenado de valores. Un ejemplo sencillo sería el intervalo del conjunto de números enteros entre el 0 y el 100. Cada valor que puede asignarse a un dato tiene una posición en una serie ordenada. Por lo tanto, existe un primer y un último valor y cada valor tiene uno que lo precede y otro que lo sigue. La función estándar de TurboPascal `Ord` devuelve el orden de cada valor en la serie (0, 1, 2, 3, 4,...), excepto para los valores de tipo entero que devuelve el propio nominal.

3.5.1.1 TIPOS PREDEFINIDOS

Estos tipos no hay que mencionarlos en la declaración de tipos ya que están predefinidos en el lenguaje TurboPascal.

Enteros

Los tipos enteros predefinidos son tipos escalares, es decir, los valores correspondientes están ordenados. Hay varios tipos enteros que se diferencian en dos características: el espacio que ocupan en memoria y en el formato de codificación. Estas dos características influyen, a su vez, en el intervalo de representación. Los distintos tipos de dato entero predefinidos en TurboPascal se muestran en la Tabla 9:

Tabla 9. Tipos de dato enteros predefinidos en TurboPascal

TIPO		INTERVALO DE REPRESENTACIÓN (límite inferior..límite superior)	TAMAÑO
Byte	sin signo[3]	0 .. 255	8-bit / 1 byte
Word		0 .. 65535	16-bit / 2 bytes
Shortint	con signo[4] (codificados en complemento a 2)	-128 .. 127	8-bit / 1 byte
Integer		-32768 .. 32767	16-bit / 2 bytes
Longint		-2147483648 .. 2147483647	32-bit / 4 bytes

Ej.:
```
var n_alum : integer; n : byte;
begin
{...}
n_alum := 45;
n := 123;
...
end.
```

Las dos sentencias del cuerpo del programa son *sentencias de asignación* que sirven para dar valores a las variables durante la ejecución del programa. Los valores o constantes enteras también pueden darse en formato entero hexadecimal que emplea los dígitos 0, 1, 2, ...9, A, ..y F. Este formato se indica utilizando el signo `$` como prefijo del valor numérico.

Ej.:
```
n := $7B; (* equivalente a la anterior n:=123 *)
```

También pueden emplearse datos de cualquiera de los tipos enteros con los procedimientos estándar de entrada y salida de datos `Read`/`ReadLn`[5] y `Write`/`WriteLn` para asignar valores a variables enteras y visualizar datos de tipo entero por la pantalla.

Ej.:
```
readln(n);      (* Detiene la ejecucion hasta que *)
                (* se introduzca un valor por *)
                (* teclado para n  *)
```

[3] Se codifican mediante su expresión como suma de potencias en base 2. Por ejemplo, el valor entero 166 como dato de tipo `byte` se representaría como $166 = 128 + 32 + 4 + 2 = 1\cdot2^7 + 1\cdot2^5 + 1\cdot2^2 + 1\cdot2^1 = 10100110$, empleando 1 byte.

[4] Se codifican mediante su expresión binaria en complemento a 2. Por ejemplo, el valor entero –91 como dato de tipo `shortint` se representaría como $-91 = -128 + 32 + 4 + 2 = (-1)\cdot2^7 + 1\cdot2^5 + 1\cdot2^2 + 1\cdot2^1 = 10100110$, empleando 1 byte.

[5] Como se verá más adelante, los procedimientos estándar `Write`/`WriteLn` sirven para visualizar datos en pantalla o escribir datos en archivos de disco, mientras que `Read`/`ReadLn` se emplean para leer datos del teclado o de archivos.

```
writeln(n);     (* Se visualiza el valor *)
                (* de n por pantalla  *)
```

La elección de uno u otro tipo de dato entero para una variable, dependerá de los valores que *a priori* pueda tomar dicha variable durante la ejecución del programa. Por ejemplo, trabajar con una variable entera de tipo `byte` permitirá un intervalo de representación menor que la de un tipo `integer` pero el espacio ocupado en memoria será menor y la velocidad de proceso mayor.

El tipo de dato entero específico asociado a una constante numérica entera es el tipo entero predefinido con el menor intervalo de representación que incluya el valor de la constante. Por ejemplo, para el valor `–1526` el tipo de dato será `integer`, mientras que para el valor `78956` el tipo de dato será `longint`.

Para cualquiera de los tipos de dato enteros, la posición u ordinal asociada a cada constante o valor coincide con su propio valor entero. La constante o valor entero `23` tendrá la posición u ordinal asociado 23; la constante `–5` tendrá la posición –5, etc...

Tipo **Boolean**

El tipo de dato `Boolean` (booleano o lógico) puede tomar únicamente los valores lógicos `False` y `True`. Tiene un tamaño de 1 byte[6].

Ej.:
```
var respuesta : boolean;
begin
respuesta := true;
```

Como se verá más adelante, en una expresión los operadores lógicos (`Not`, `And`, `Or` y `Xor`) y los operadores relacionales (`=`, `<>`, `>`, `<`, `>=`, `<=` e `In`) producen resultados de tipo `Boolean` y, con la excepción del último, pueden tener operandos de tipo `Boolean`. Asimismo, pueden emplearse datos de tipo `Boolean` con los procedimientos estándar `Write`/`WriteLn` pero no con los `Read`/`ReadLn`. Este tipo de datos tiene una gran importancia en la evaluación de expresiones, especialmente en algunas sentencias de control de la lógica del programa.

Los valores `False` y `True` tienen asociados los ordinales 0 y 1, respectivamente, dentro del tipo de dato `Boolean`.

Tipo **Char**

Un dato de este tipo representa un valor del conjunto de los caracteres del código estándar ASCII: caracteres alfabéticos, dígitos numéricos, signos de puntuación y caracteres de control. Ocupan 1 byte (8 bits: 256 combinaciones diferentes = 256 caracteres en total). Una constante o valor literal de un tipo `Char` debe encerrarse entre comillas simples o bien puede representarse con el carácter # precediendo al ordinal correspondiente. Al final de este capítulo se facilita la *Tabla de caracteres ASCII* con todos los caracteres y sus ordinales correspondientes.

Ej.:
```
var caracter : char;
Begin
caracter := '9';
caracter := 'A';
caracter := #65; { el mismo efecto que la anterior }
```

[6] Para facilitar la compatibilidad con el entorno Windows, en la versión 7.0 de TurboPascal existen hasta cuatro tipos de dato booleanos: `Boolean`, `WordBool`, `LongBool` y `ByteBool`. Estos tipos tienen, respectivamente, los siguientes tamaños: 8, 16, 32 y 8 bits.

También pueden emplearse datos de tipo `Char` con los procedimientos estándar de entrada y salida de datos `Read`/`ReadLn` y `Write`/`WriteLn` para asignar valores a variables de tipo `Char` y visualizar datos de tipo `Char` por la pantalla.

La función estándar `Chr` convierte un valor entero (el ordinal) en el carácter ASCII correspondiente. La llamada a la función `Chr(65)` devuelve el carácter ASCII `'A'`. La función estándar `Ord` devuelve el ordinal del carácter ASCII indicado. La llamada a la función `Ord('A')` devuelve el valor entero `65`.

3.5.1.2 TIPOS DEFINIDOS POR EL USUARIO

Son tipos de dato que no están predefinidos por TurboPascal, siendo el propio programador el que se encarga de definirlos en la sección de declaraciones de tipos de dato del programa. Se clasifican en enumerados y subrangos.

Enumerados

Al declararlo, se introduce un nuevo identificador para un tipo de dato, a la vez que, se define los valores o constantes del nuevo tipo dentro de un número reducido de alternativas (256 valores diferentes como máximo). Cada uno de estos posibles valores se convierte en un *identificador* de una constante literal del tipo enumerado. Para estos valores, no pueden utilizarse constantes que tengan otro uso o sean identificadores ya empleados. Como una de las consecuencias de esta norma, dos tipos enumerados diferentes no pueden compartir un mismo valor.

Sintaxis: `TYPE` *`id_tipo`* `= (`*`valor_1,valor_2...,valor_n`*`);`

Ej.:
```
TYPE dia_sem = (lun,mar,mie,jue,vie,sab,dom);
     estaciones = (primavera,verano,otono,invierno);
VAR  visita : dia_sem;
     temporada : estaciones;
begin
visita:=sab;
estaciones:=verano;
```

Al ser un tipo ordinal, el orden de los valores se determina por la disposición de los elementos en la declaración. El primer elemento o valor ocupa la posición cero (0). Por ejemplo, la llamada a la función `Ord(mar)` devuelve el valor entero 1 mientras que la llamada a la función `Ord(primavera)` devuelve el valor entero `0`.

Las variables de tipo enumerado ocupan siempre 1 byte (8 bits, lo que permitiría, como máximo, 256 valores diferentes), independientemente del número de posibles distintos valores que puedan tomar.

Con los tipos de dato enumerados sólo pueden utilizarse los operadores de asignación y los de relación (igualdad `=` y desigualdad `<>`), no pueden emplearse con los procedimientos de entrada y salida de datos: `Read/ReadLn` o `Write/WriteLn`, pero pueden ser subíndices de datos de tipo `Array` como se verá más adelante.

Subrangos

Un tipo *subrango* es un subconjunto de un tipo ordinal ya declarado por el programador previamente o predefinido en el lenguaje TurboPascal. Es decir, toman valores de cualquier subconjunto de valores definidos antes en un tipo ordinal (¡no vale, por ejemplo, un intervalo numérico real!) indicando su límite inferior y su límite superior. La única limitación a estos límites la da el conjunto de valores correspondiente al tipo de dato en el que está incluido.

Sintaxis: `TYPE` *`identificador = lim_inferior..lim_superior`*`;`

El límite inferior y superior deben ser del mismo tipo ordinal, siendo el primero inferior o igual al segundo.

Ej.:
```
type anno = 1900 .. 1999;
     minuscula = 'a' .. 'z';
     dia_labora = lun .. vie;
var  caducidad : anno;
     letra : minuscula;
     inspeccion : dia_labora;
```

Pueden definirse los tipos de dato subrangos en la definición de variables pero esta práctica no es aconsejable. Es decir, la declaración de variables que se muestra a continuación

Ej.:
```
var  x : 0 .. 9;
     c : '0'..'9';
```

en la que no hay una declaración previa de identificadores para los tipos de dato subrango, no es incorrecta para el compilador; pero un buen programador debería realizar, previa a la declaración de variables, la correspondiente declaración de tipos:

Ej.:
```
type numero = 0 .. 9;
     digito = '0'..'9';
var  x : numero;
     c : digito;
begin
x:=1;
c:='1';
```

Los operadores que pueden emplear estos tipos de dato coinciden con los del tipo de dato a partir del cual se definen.

3.5.2. Tipos REALES

En TurboPascal un tipo real tiene un conjunto de valores que es un subconjunto de los números reales. También son predefinidos, pero no ordinales. TurboPascal 7.0 predefine cinco tipos reales con un tamaño de almacenamiento en memoria, intervalo de representación y precisión distinto para cada uno de ellos.

Tabla 10. Tipos de dato reales predefinidos en TurboPascal

TIPO	INTERVALO DE REPRESENTACIÓN (en valores absolutos)	DIGITOS SIGNIFICATIVOS	TAMAÑO (bytes)
Single	1.5e-45..3.4e38	7-8	4
Real	2.9e-39..1.7e38	11-12	6
Double	5.0e-324..1.7e308	15-16	8
Extended	3.4e-4932..1.1e4932	19-20	10
Comp	-9.2e18..9.2e18	19-20	8

El tamaño ocupado y el formato de almacenamiento[7] en memoria determina el intervalo de representación y la precisión de cada tipo.

[7] Los tipos de dato `Real`, `Single`, `Double` y `Extended` utilizan distintos formatos de mantisa y exponente para la codificación de los valores reales que representan. Por ejemplo, el tipo de dato `Real` emplea un total de 48 bits (6 bytes): el primer bit para representar el signo (s), los 39 siguientes para la representación binaria de la mantisa (m) como suma de potencias negativas de base 2 y los 8 bits restantes para el exponente (e) como suma de potencias positivas de base 2. De tal forma que si $0 < e \leq 255$ entonces el valor real representado, $v = (-1)^s * 2^{(e-129)} * (1.m)$ y si $e = 0$ entonces el valor representado $v = 0$

Ej.:
```
var  a,b : real;
begin
a := 3.123;
b := 2.6*a – 9.87;
...
```

Los valores reales también pueden darse en notación decimal científica o exponencial. Esta notación utiliza la letra `E` ó `e` seguida de un exponente como se muestra en el siguiente ejemplo:

```
a := 31.25E-1;     (* equivale a la anterior:31.25x10-1 *)
```

Es decir, la letra `E` se lee como *por diez elevado a* .

También pueden emplearse datos de cualquiera de los tipos reales con los procedimientos estándar de entrada y salida de datos `Read`/`ReadLn`[8] y `Write`/`WriteLn` para asignar valores a variables reales y visualizar datos de tipo real por la pantalla.

Aunque la mayoría de los ordenadores actuales dispone de uno, es interesante indicar que necesita un coprocesador matemático 8087 para trabajar con los tipos `Single`, `Double`, `Extended` y `Comp` o bien, si no se tiene instalado dicho coprocesador en el ordenador de trabajo, ha de *emularse* mediante la orden al compilador o *directiva* correspondiente (`$N`, `$E`).

Por otro lado, aunque se incluya genéricamente dentro de los tipos reales, en la práctica, el tipo `Comp` no almacena datos reales sino que toma valores enteros de 64 bits (8 bytes). El tipo `Comp` es un híbrido entre entero y real, ya que se almacena como un entero con representación binaria en complemento a 2 y se visualiza en pantalla como un real en notación científica. Aunque represente un entero no se considera un ordinal y no se le puede aplicar las operaciones de estos tipos de dato. El tipo `Comp` se emplea en programas donde se necesiten números muy grandes con precisión entera.

Como se verá más adelante, no pueden usarse datos reales como índices de un `Array` o sentencias `Case` o `For`, ni para crear tipos `Set` o subrangos.

3.6. TIPOS ESTRUCTURADOS

Los datos estructurados se verán en el capítulo 6.

3.7. DATOS ESTÁTICOS Y DINÁMICOS

Dentro de una tercera clasificación y dependiendo de cómo y en qué lugar de la memoria se almacenan, los datos podrán ser *estáticos* o *dinámicos*

3.7.1. DATOS ESTÁTICOS

Los datos estáticos tienen las siguientes características comunes:

a) Se declaran explícitamente en la parte de declaraciones, ya sea en la sección de constantes o en la de variables. No pueden crearse o emplearse otros datos estáticos en el programa que no hayan sido declarados previamente en esa zona.
b) Al declararse de esta manera, se define su tipo y tamaño en tiempo de compilación: antes de la ejecución de la primera sentencia del programa.
c) Tienen un único nombre o identificador.

Todos los datos vistos hasta ahora en los ejemplos son datos estáticos.

[8] Como se verá más adelante, los procedimientos estándar Write/WriteLn sirven para visualizar datos en pantalla o escribir datos en archivos de disco, mientras que Read/ReadLn se emplean para leer datos del teclado o de archivos.

3.7.2. DATOS DINÁMICOS

Los datos dinámicos tienen como características generales que:

a) No se declaran en la zona de declaraciones del programa, es decir, no se realiza una reserva de memoria antes de empezar a ejecutar la primera sentencia del programa.
b) Pueden crearse y destruirse durante la ejecución del programa. De esta manera, pueden formarse estructuras de datos dinámicos cuyo tamaño puede variar durante la ejecución del programa.
c) Se almacenan en una sección de la memoria distinta a la que se emplea para los datos estático.
d) A diferencia de los datos estáticos, los datos dinámicos pueden tener varios identificadores.

Los datos o variables dinámicas se verán detenidamente más adelante en el capítulo *Punteros y Variables Dinámicas*.

3.8. COMPATIBILIDAD ENTRE TIPOS DE DATOS

Entre los tipos de dato que se emplean en un programa se pueden establecer varios tipos de relaciones: de identidad (si son el mismo tipo de dato), de compatibilidad o de asignación compatible. Es importante tener en cuenta este tipo de relaciones al realizar determinadas operaciones y, en especial, en las asignaciones.

3.8.1. TIPOS DE DATO IDÉNTICOS

Dada la declaración de tipo de dato

```
TYPE T1, T2    = byte;
     T3        = byte;
     T4        = T2;
     T5, T6    = Set of byte;
     T7        = Set of byte;
```

Los tipos de dato `T1`, `T2`, `T3 y T4` son tipos de dato *idénticos*. Por otra parte, `T5 y T6` también son tipos de dato idénticos pero `T7` no lo es. Esto último se debe a que `Set of byte` no es un identificador de un tipo de dato.

Como se verá en el Capítulo de Procedimientos y Funciones, la identidad entre tipos de datos es condición necesaria únicamente entre los parámetros variables reales y formales en las llamadas a los procedimientos y funciones.

3.8.2. TIPOS DE DATO COMPATIBLES

Dos tipos de dato son compatibles si es cierta, al menos, una de las siguientes condiciones:

a) Ambos tipos de dato son idénticos
b) Ambos tipos son reales
c) Ambos tipos son enteros
d) Un tipo de dato es subrango de otro
e) Ambos son subrango del mismo tipo
f) Ambos son de tipo conjunto con elementos correspondientes a tipos de dato compatibles
g) Uno es de tipo cadena y el otro es de tipo cadena o carácter
h) Uno es de tipo `Pointer` y el otro es de cualquier tipo puntero
i) Ambos son de tipo puntero y apuntan a tipos de dato idénticos

j) Ambos son de tipo procedural con idéntico número y tipo de parámetros (uno a uno) e idéntico resultado en el caso de ser una función.

La compatibilidad entre tipos de datos es necesaria para una correcta compilación del código fuente en asignaciones, expresiones o en operaciones de relación pero en las asignaciones no es condición suficiente.

3.8.3. COMPATIBILIDAD EN LAS ASIGNACIONES

La compatibidad en la asignación es necesaria cuando se pasa o se asigna un valor a algo, a través de una sentencia de asignación o mediante un parámetro por valor en la llamada a un procedimiento o función. El compilador de TurboPascal realiza siempre una verificación de la compatibilidad entre tipos de dato en las asignaciones, pero además, puede verificar que los valores asignados a una variable (por ejemplo, en una sentencia de asignación) están dentro del intervalo de representación válido para el tipo de dato correspondiente a la variable[9].

En general, para evitar un error de compilación o una incorrecta asignación de valores a una variable (si el compilador no realiza la verificación), es necesario tener en cuenta que un valor de un tipo `T1` es compatible a nivel de asignación con un tipo `T2` (por ejemplo, en la sentencia de asignación `b:=a;`, si a es de tipo `T1` y b es de tipo `T2`) si se cumple, al menos, una de las siguientes premisas:

a) `T1` y `T2` son tipos idénticos
b) `T1` y `T2` son tipos ordinales compatibles y los valores de `T1` están dentro del intervalo de valores de `T2`
c) `T1` y `T2` son tipos reales compatibles y los valores de `T1` están dentro del intervalo de valores de `T2`
d) `T2` es un tipo real y `T1` es un tipo entero
e) `T1` y `T2` son tipos cadena
f) `T2` es un tipo cadena y `T1` es un tipo carácter
g) `T1` y `T2` son tipos conjunto compatibles y los valores de los elementos de `T1` están dentro del intervalo de valores de los elementos de `T2`
h) `T1` y `T2` son tipos puntero compatibles
i) `T1` y `T2` son tipos procedurales compatibles
j) `T1` es un tipo procedural y `T2` es un procedimiento o función con idéntico número y tipo de parámetros e idéntico tipo de resultado si es una función
k) `T2` es un tipo objeto (de una determinada clase) y `T1` es de idéntico tipo o descendiente

3.9. DECLARACIÓN DE NUEVOS TIPOS DE DATOS

Como ya se ha indicado anteriormente, en la parte de declaración de tipos de un programa pueden crearse nuevos tipos por parte del programador, pudiéndose definir tipos de dato simples y tipos de dato estructurados. Además, estos *nuevos* tipos pueden ser tanto tipos predefinidos como tipos definidos por el usuario.

Ej.:
```
type  entero     = integer;
      digito     = 0..9;
      logico     = boolean;
      caracter   = char;
      vector     = array[1..10] of entero;
```

[9] La inclusión del *metacomando* o *directiva del compilador* {$R+} en el programa fuente activa esta verificación mientras que la directiva {$R-} lo desactiva. Esta activación/desactivación puede realizarse también a través del menú Options/Compiler del entorno de TurboPascal (Range Checking)

```
        nombre     = string[20];
        fichero    = file of integer;
var     a     : entero;
        b     : logico;
        c     : caracter; ...
```

En el ejemplo, `entero` e `integer` son tipos de dato idénticos. El tipo de dato `digito` es compatible a nivel de asignación, pero no idéntico, con los tipos de dato `entero` e `integer`. Los tipo de dato `logico` y `boolean` son tipos de dato idénticos. También son idénticos los tipos de dato `caracter` y `char`. Los tipos de dato `nombre` y `string` son tipos de dato compatibles.

Tabla 11. Tabla de caracteres ASCII

0	*nul*	32	*spa*	64	@	96	`	128		160		192	À	224	à
1	*soh*	33	!	65	A	97	a	129		161	¡	193	Á	225	á
2	*stx*	34	"	66	B	98	b	130	‚	162	¢	194	Â	226	â
3	*etx*	35	#	67	C	99	c	131	*ƒ*	163	£	195	Ã	227	ã
4	*eot*	36	$	68	D	100	d	132	„	164	¤	196	Ä	228	ä
5	*eng*	37	%	69	E	101	e	133	…	165	¥	197	Å	229	å
6	*ack*	38	&	70	F	102	f	134	†	166	¦	198	Æ	230	æ
7	*bel*	39	'	71	G	103	g	135	‡	167	§	199	Ç	231	ç
8	*bs*	40	(	72	H	104	h	136	ˆ	168	¨	200	È	232	è
9	*tab*	41	)	73	I	105	i	137	‰	169	©	201	É	233	é
10	*lf*	42	*	74	J	106	j	138	Š	170	ª	202	Ê	234	ê
11	*vt*	43	+	75	K	107	k	139	‹	171	«	203	Ë	235	ë
12	*ff*	44	,	76	L	108	l	140	Œ	172	¬	204	Ì	236	ì
13	*cr*	45	-	77	M	109	m	141		173	-	205	Í	237	í
14	*so*	46	.	78	N	110	n	142		174	®	206	Î	238	î
15	*si*	47	/	79	O	111	o	143		175	¯	207	Ï	239	ï
16	*dle*	48	0	80	P	112	p	144		176	°	208	Đ	240	ð
17	*dc1*	49	1	81	Q	113	q	145	‘	177	±	209	Ñ	241	ñ
18	*dc2*	50	2	82	R	114	r	146	’	178	²	210	Ò	242	ò
19	*dc3*	51	3	83	S	115	s	147	“	179	³	211	Ó	243	ó
20	*dc4*	52	4	84	T	116	t	148	”	180	´	212	Ô	244	ô
21	*nak*	53	5	85	U	117	u	149	•	181	µ	213	Õ	245	õ
22	*syn*	54	6	86	V	118	v	150	–	182	¶	214	Ö	246	ö
23	*etb*	55	7	87	W	119	w	151	—	183	·	215	×	247	÷
24	*can*	56	8	88	X	120	x	152	˜	184	¸	216	Ø	248	ø
25	*en*	57	9	89	Y	121	y	153	™	185	¹	217	Ù	249	ù
26	*sub*	58	:	90	Z	122	z	154	š	186	º	218	Ú	250	ú
27	*esc*	59	;	91	[	123	{	155	›	187	»	219	Û	251	û
28	*fs*	60	<	92	\	124	|	156	œ	188	¼	220	Ü	252	ü
29	*gs*	61	=	93	]	125	}	157		189	½	221	Ý	253	ý
30	*rs*	62	>	94	^	126	~	158		190	¾	222	Þ	254	þ
31	*us*	63	?	95	_	127	*del*	159	Ÿ	191	¿	223	ß	255	ÿ

CONTROL (no *imprimibles*)

ASCII EXTENDIDO

Ejemplo:

Carácter	Representación binaria (Secuencia de bits)	Equivalente en decimal (ordinal)	Equivalente en hexadecimal
A	01000001	65	41

Bibliografía básica

- **García-Beltrán, A., Martínez, R.** y **Jaén, J.A.** *Métodos Informáticos en TurboPascal*, Ed. Bellisco, 2ª edición, Madrid, 2002
- **Joyanes, L.** *Fundamentos de programación, Algoritmos y Estructuras de Datos*, McGraw-Hill, Segunda edición, 1996
- **Duntemann, J.** *La Biblia de TurboPascal*, Anaya Multimedia, Madrid, 1991