6. DATOS ESTRUCTURADOS

Conceptos: *Estructura de datos, Array, Vector, Matriz, Índice, String, Cadena, Record, Campo, Set, Conjunto.*

Resumen: A diferencia de los datos de tipo simple que sólo pueden almacenar un valor, los datos estructurados o estructuras de datos pueden recolectar varios valores simultáneamente. Se hace una primera introducción a los datos estructurados destacando en primer lugar que se les asigna una cantidad fija de memoria durante la ejecución del programa cuando se declara una variable de un determinado tipo estructurado. El primer tipo estructurado es el tipo *array* que permite agrupar otros datos más simples de igual tipo bajo un mismo identificador. Este tipo de estructuras permiten definir vectores, matrices, tablas y estructuras multidimensionales. TurboPascal incorpora un tipo especial de *array*: el tipo string. Se define como una secuencia de caracteres cuya longitud puede variar entre 1 y 255. El tipo *record* está compuesto de elementos de diferentes tipos a cada uno de los cuales se les asocia un identificador. Finalmente se analiza el tipo estructurado *set* equivalente al concepto de conjunto matemático y otros tipos de datos no simples.

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

a) Describir los tipos de datos estructurados en el lenguaje de programación Turbopascal, su formato de representación y las operaciones más características que pueden realizarse con ellos (*Conocimiento*)
b) Escribir la declaración de variables de cualquiera de los tipos de datos estructurados (*Comprensión*)
c) Escribir el código necesario para acceder a un elemento o conjunto de elementos de una estructura de datos (*Comprensión*)
d) Seleccionar la estructura de datos más adecuada para una aplicación determinada (*Aplicación*)
e) Codificar una tarea sencilla convenientemente especificada, utilizando datos estructurados (*Aplicación*)

6.1. INTRODUCCIÓN

Los tipos estructurados de datos se componen de otros tipos de datos más simples previamente declarados o predefinidos en el lenguaje TurboPascal. Los tipos de datos estructurados en TurboPascal son los siguientes:

a) Array
b) String
c) Record
d) Set
e) File
f) Text
g) Object

Existen otros dos tipos de datos que, aunque no son estrictamente una composición de otros datos más simples también se van a describir en este capítulo: el tipo Pointer y el tipo Procedimiental.

6.2. Tipo Array

Un dato de tipo *array* es, en realidad, un conjunto o estructura de datos que engloba una colección de datos del mismo tipo. Pueden ser unidimensionales, denominados también vectores o listas, o multidimensionales, denominados matrices o tablas. Los números o valores que identifican a cada **elemento** particular del `Array` se llaman **índices**.

Sintaxis: `Type` *`ident`* `=` **`Array`** `[`*`TSub1,...,TSubn`*`]` **`of`** *`Tipo`*`;`

donde `TSub1,...,TSubn` es una sucesión de **tipos de dato** ordinales (¡no pueden ser variables y no valen tipos de dato reales!: sólo enteros, lógico, carácter, enumerado o subrangos de los anteriores) separados por comas y que especifican, según su producto cartesiano, el número de elementos de la estructura. `TSub`*`i`* es un identificador de un tipo de dato ordinal o un subrango de éste: *`lim_inf_i..lim_sup_i`*

Ej.:
```
TYPE  vector1 = Array [1..4] of Char;
      matriz1 = Array [1..10, 1..10] of Integer;
      matriz2 = Array [Boolean, 1..10] of Boolean;
      color = (blanco, amarillo, negro, rojo);
      ciudad = (Al,Ca,Co,Gr,Ja,Hu,Ma,Se);
      estacion = (prim,ver,oto,inv);
      cestacion = array[estacion] of string[9];
CONST est : cestacion =
            ('primavera','verano','otoño','invierno');
VAR   v : Integer;
      vect1,vect2 : vector1;
      matriz : matriz1;
      qt : matriz2;
      raza  : Array [1..40] of color;
      grados : Array [ciudad] of real;
begin
vect1[1] := 't';
vect1 := vect2;
matriz[2,5] := 6;
matriz[1,9] := matriz[2,5];
qt[true,4] := false;
raza[15] := amarillo;
```

```
grados[Al] := 18.5;
...
```

En el ejemplo anterior, se observa como puede accederse a cualquier elemento de la estructura `Array` referenciando el/los subíndice/s entre corchetes. Asimismo, pueden realizarse asignaciones entre datos `Array` del mismo tipo.

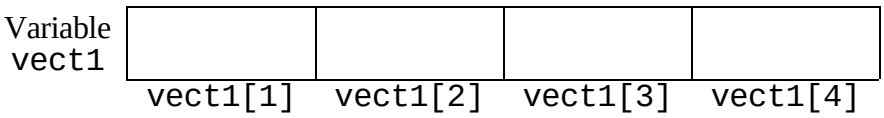

Figura 19. Espacio de memoria reservado para una variable array `vect1`de tipo `vector1`

El tamaño reservado en memoria para una variable de tipo `Array` es igual al número total de elementos por el tamaño del elemento, en bytes. Así, mientras la variable `vect1` del ejemplo anterior ocupa 4 elementos x 1 byte = 4 bytes, la variable `matriz` ocupa 10 x 10 elementos x 2 bytes = 200 bytes en la memoria durante la ejecución del programa.

Cuando se trabaja con datos de tipo `Array` (especialmente si son multidimensionales) hay que tener cuidado con la cantidad de memoria que hay que reservar ya que se podría sobrepasar la memoria disponible. En principio, TurboPascal sólo permite tipos de dato estructurados con un tamaño máximo de 65520 bytes. Las dos siguientes declaraciones de tipos de dato son, por lo tanto, incorrectas:

```
type vector = array[1..65536] of byte;
     vector2 = array[1..32800] of integer;
```

Con los tipos de dato `Array` sólo pueden utilizarse los operadores de asignación y no pueden emplearse, como estructura completa con los procedimientos de entrada y salida de datos: `Read/ReadLn` o `Write/WriteLn`. Esto es independiente de las operaciones que puedan realizarse con cada uno de los elementos que componen la variable *array*, si lo permite el tipo de dato correspondiente.

6.3. Tipo String

Este tipo de dato predefinido en el lenguaje TurboPascal permite representar una secuencia o cadena de caracteres correspondientes al código ASCII de un tamaño máximo de 255 (por defecto). Si se desea especificar un tamaño menor de 255 se utilizarán corchetes para delimitar un entero que especifica el tamaño máximo de la cadena de caracteres.

Una variable de este tipo ocupa en memoria tantos bytes como caracteres tenga más uno; en este byte se guarda la longitud real de la cadena almacenada en la variable. A esta longitud se le denomina *tamaño* ó *longitud lógica*. Puede accederse a cada uno de los caracteres que forman la secuencia de caracteres como si fueran datos de tipo `Array` unidimensionales de caracteres.

Sintaxis: `TYPE` *`identificador`* `: String[`*`J`*`]; { donde 1≤J≤255 }`

Ej.:
```
CONST LineLin = 79;
TYPE Nombre = String [10];
     Linea = String [LineLin];
VAR  n : nombre;
     comentario : linea;
```

Para la variable `n` se reserva un espacio de 11 bytes en la memoria durante la ejecución del programa.

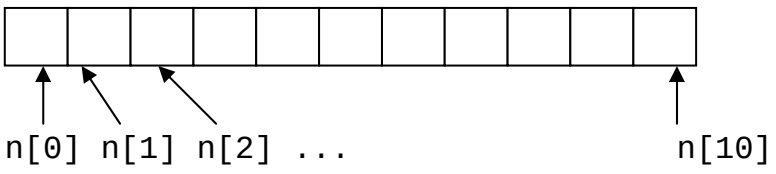

Figura 20. Espacio de memoria reservado para una variable `n` tipo `nombre`

En las expresiones y sentencias que manipulan datos de tipo `String`, el valor o la constante literal correspondiente va encerrado entre comillas simples. Pueden manipularse datos de tipo `String` con operaciones de asignación (`:=`), comparaciones (operadores de relación) y concatenaciones (`+`).

Ej.: `n := 'mario';`

Tiene como resultado...

#5	'm'	'a'	'r'	'i'	'o'					

Figura 21. Asignacuión de valores a la variable `n` tipo `nombre`

La variable `n`, en este caso, almacena una cadena de cinco caracteres. La función estándar `Length` devuelve la longitud de la cadena almacenada en una variable de tipo `String`. Tras la asignación anterior, la llamada a la función `Length(n)` devolvería el valor entero `5`. Como puede accederse a cada carácter de forma independiente, el mismo valor también podría obtenerse del espacio en memoria que se emplea para guardar el tamaño de la cadena que se almacena en la variable, con la llamada a la función `Ord(n[0])`.

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

A diferencia del tipo de dato `Char`, NO es un tipo de dato ordinal, ya que no es un conjunto "finito" de datos. Aunque sí se puede establecer el orden entre dos valores de tipo cadena. Éste se obtiene por el orden entre los valores de tipo carácter que componen las cadenas según las posiciones respectivas de los caracteres[11]:

```
'a' < 'anterior' < 'antes' < 'despues' < 'fuego' < 'luego'
```

Por otro lado, `'casas'` se considera mayor que `'casa'`, ya que en dos cadenas de distinto tamaño, cada carácter en la cadena de mayor tamaño sin el correspondiente carácter en la menor supone un valor superior.

También puede asignarse a una variable de tipo cadena una constante cadena de caracteres vacía.

Ej.: `n := '';`

Según muestra la Tabla 18 el operador suma es el único de este tipo. Es un operador binario que actúa sobre dos operandos de tipo `Char` o `String`. Da como resultado un valor de tipo `String`.

[11] Es importante tener en cuenta que, en la tabla de caracteres ASCII, los caracteres alfabéticos mayúsculas van antes que las minúsculas

Tabla 18. Operadores de cadena

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Suma de cadenas	'coche' + 'azul'	'cocheazul'

6.4. Tipo Record

Un tipo *record* o registro permite definir una estructura que almacena un conjunto de datos del mismo o de distintos tipos (excepto `File`). Los datos individuales se conocen como **campos** del registro y se declaran como variables cuando se define el tipo de registro. A cada uno de los campos se le asigna un identificador al realizar la declaración, no pudiendo existir dos identificadores de campo iguales dentro del mismo registro.

Sintaxis:
```
TYPE Tiporegistro = Record
       lista ident1 : tipo1;
       lista ident2 : tipo2;
       { ... }
       lista identn : tipon
     end;
```

Ej.:
```
Type meses = (En,Fb,Mr,Ab,My,Jn,Jl,Ag,Sp,Oc,Nv,Dc);
     fecha = record
       dia : 1..31;
       mes : meses;
       anno : 1900..2000
       end;
```

Un campo de un tipo registro puede ser de otro tipo registro (registros anidados). En el siguiente ejemplo, el tipo `ficha_personal` incluye un campo de tipo `fecha`.

```
Type ficha_personal = record
          nombre, ape1, apel2  : string [20];
          fecha_nacimiento     : fecha;
          profesion            : string [40];
          telefono             : integer
          end;
Var  cumple    : fecha;
     individuo : ficha_personal;
```

El acceso, para entradas o salidas, a los campos de registro, se realiza con el identificador del registro, un punto y el identificador del campo.

Sintaxis: *IdentificadorRegistro.IdentificadorCampo*

Ej.:
```
cumple.dia := 15;
write(cumple.dia);
individuo.fecha_nacimiento.mes := Fb;
```

O pueden manipularse los campos de un dato tipo `Record` determinado con la estructura `With`:

Ej.:
```
with cumple do
     begin
     dia:=27;
     mes:=En;
     anno:=1993;
     write(dia)
     end;
```

```
with individuo.fecha_nacimiento do
     begin
     dia:=2;
     mes:=My;
     anno:=1953
     end;
```

El tamaño de una variable de tipo `Record` es la suma de los tamaños de sus campos. Así, la variable `cumple` del ejemplo anterior ocupa 1 + 1 + 2 = 4 bytes en memoria durante la ejecución del programa.

La estructura tipo `Record` permite la introducción de campos variantes, que aparecen o no en una variable de ese tipo, en función del valor de un cierto campo. En general, los registros variantes tendrán una parte fija, que se declara en primer lugar, y otra variante. La principal ventaja de este tipo de estructura es el ahorro de memoria, ya que el espacio ocupado por una variable de este tipo es la suma del tamaño de la parte fija y el tamaño de la parte variante más grande. El campo de selección puede ser cualquier variable de tipo ordinal.

Sintaxis:
```
TYPE Tiporegistro = Record
               lista ident1 : tipo1;
               { ... }
               lista identn : tipon;
               case campo_selector:tipo of
                    valor1 : (lista ident1b);
                    { ... }
                    valorn : (lista identnb)
                    end;
```

Ej.:
```
TYPE ficha = Record
               nombre: String[20];
               dni  : String[8];
               CASE alumno    : Boolean OF
                         False: (prof  : Boolean;
                                 dpt   : String[20]);
                         True : (mat   : String[5];
                                 curso : 1..6)
               END;
```

6.5. Tipo Set

Un dato de tipo `Set` corresponde a la definición matemática de conjunto. Es una parte de un conjunto universal, de un tipo de dato base ordinal ya definido y tiene un máximo de 256 elementos. Aunque sus elementos deben pertenecer a un mismo tipo ordinal, dentro del conjunto no están ordenados. Los valores ordinales de todos los elementos deben estar dentro del intervalo [0-255]. La definición del tipo `Set` se realiza de la siguiente manera:

Sintaxis:
```
TYPE TipoSet = Set of tipo;
```

Ej.:
```
type dia = (lu,ma,mi,ju,vi,sa,dm);
     Frutas = (limon,naranja,uva,pera,platano);
     conj_caract = Set of Char;
     digitos = Set of 0..9;
     dias = Set of dia;
     clase_fruta = Set of frutas;
```

A continuación pueden declararse variables de tipo `Set`:

Ej.:
```
var  laborable : dias;
```

```
letras    : conj_caract;
conj_num  : digitos;
```

La sintaxis de asignación de datos de tipo `Set` es:

```
identificador := [valor_i, .., valor_j];
```

Ej.: `laborable := [lu, ma, mi, ju, vi];`

o bien, de forma más condensada:

```
laborable := [lu..vi];
letras :=['A','C','T','m'];
conj_num := [];   { se le asigna el conjunto vacio }
conj_num := conj_num + [2, 3];
```

El tamaño reservado en memoria para una variable de tipo `Set` es, en bytes, el cociente entero mas uno del número máximo de elementos posibles del conjunto menos uno dividido entre ocho. Por ejemplo, una variable de tipo conjunto que pueda albergar, como máximo, entre 1 y 8 elementos, ocupará 1 byte en memoria; entre 9 y 16 elementos, 2 bytes,... entre 249 y 256 elementos, 32 bytes.

No se pueden utilizar estos tipos de dato, los conjuntos o los elementos de un conjunto, con los procedimientos de entrada y salida de datos: `Read/ReadLn` o `Write/WriteLn`.

Las operaciones que pueden realizarse con datos de tipo `Set` pertenecen al álgebra de conjuntos. Los operadores de conjuntos definidos en TurboPascal son binarios y se resumen en la Tabla 19.

Tabla 19. Operadores de conjuntos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Unión	`[2,3] + [3,6]`	`[2,3,6]`
*****	Intersección (conjunto de elementos que estén a la vez en dos conjuntos)	`[2,3] * [3,6]`	`[3]`
-	Diferencia (conjunto de elementos pertenecientes a un primero que no están en un segundo conjunto)	`[2,3] - [3,6]`	`[2]`
=	Igualdad	`[2,3] = [3,6]`	`false`
<>	Desigualdad	`[2,3] <> [3,6]`	`true`
<=	Inclusión (de un conjunto en otro)	`[2,3] <= [3,6]`	`false`
=>	Inclusión inversa (de un segundo conjunto en un primero)	`[2,3] => [3]`	`true`
in	Pertenencia (Nota: el primer operando es del tipo de dato correspondiente al elemento del conjunto)	`3 in [3,6]`	`true`

6.6. Tipo File

El tipo predefinido `file` permite utilizar una estructura de datos que se emplea cuando es necesario manipular grandes cantidades de datos y deben almacenarse en un sistema de almacenamiento masivo (habitualmente, como archivo o fichero en el disco duro del ordenador). Un archivo es una secuencia lineal de valores de datos de un cierto tipo. Esta

secuencia no tiene longitud fija, ni predefinida. Si no se especifica el tipo de componentes será un fichero sin tipo (indefinido).

Ej.:
```
TYPE Fich_numeros = File of Integer;
     Fichero = File of Ficha;
     Archivo = File;
```

Este tipo de dato se verá con más detenimiento en el capítulo *Archivos*.

6.7. Tipo Text

El tipo predefinido `text` permite utilizar una estructura de datos de tipo archivo que contiene caracteres (datos tipo `Char`) organizados por líneas o filas.

Ej.:
```
VAR  fichero_texto : Text;
```

Este tipo de dato se verá con más detenimiento en el capítulo *Archivos*.

6.8. Tipo Pointer

Los punteros representan o almacenan direcciones de memoria en las que se almacenan datos de tipo dinámico. Los punteros no tienen porqué ser datos de tipo dinámico pueden ser datos estáticos que apuntan a datos dinámicos. Este tipo de dato se verá con más detenimiento en el capítulo *Punteros y Variables Dinámicas*.

6.9. Tipo Procedural o Procedimental

Los procedimientos y funciones, también llamados genéricamente rutinas, son módulos o conjuntos independientes de sentencias de un programa que pueden ejecutarse a través de una llamada. Admiten parámetros en función de los cuales pueden ejecutarse. Estos parámetros pueden ser de cualquiera de los tipos vistos anteriormente pero, incluso, pueden ser otros procedimientos o funciones. Para permitir esto, deben declararse tipo procedurales o procedimentales que definan un tipo de procedimiento o función. La sintaxis de definición del tipo procedural o procedimental es el siguiente:

En el caso de un tipo procedimiento:
```
Type TipoProc = Procedure(Parametros);
```
o bien en el caso de un tipo función:
```
Type TipoFunc = Function(Parametros):id_tipo;
```
`id_tipo` hace referencia al tipo de dato devuelto por la función.

Ej.:
```
Type Proced = Procedure;
     AsignaP = Procedure(var a:integer);
     FuncionUni = Function(x:real):real;
     FuncionBi = Function(x,y:real):real;
     MaxFun = Function(a,b:real; f:FuncionUni):real;
```

6.10. Tipo Object

Este tipo de dato, que no entra dentro del alcance de este curso, permite trabajar con la metodología de *Programación Orientada a Objetos* en TurboPascal.

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
- **Aho, A.H., Hopcroft, J.E. y Ullman, J.D.** *Estructuras de Datos y Algoritmos*, Addison-Wesley Iberoamericana, 1988
- **Kruse, R.** *Estructuras de Datos y Diseño de Programas*, Prentice-Hall, 1988