Tema 5
Arrays

Grado de Ingeniería Informática

Introducción a la Programación

Arrays

- 5.1. Descripción
- 5.2. Operaciones
- 5.3. Arrays multidimensionales
- 5.4. Tipo *string*
- 5.5. Algoritmos con arrays

Objetivos

- Conocer el tipo de dato array, cómo se define y las condiciones de su aplicación.
- Conocer el tipo de dato string, cómo se define, las condiciones de su aplicación y las principales funciones y procedimientos predefinidos.
- Presentar algoritmos fundamentales de búsqueda y ordenación de arrays.

5.1. Descripción

- **Primera visión:** un ***array*** es un tipo de dato ***estructurado*** que permite almacenar la información de forma compacta y manejable.

ElTablero

‘c’	‘x’	‘c’
‘x’	‘x’	
	‘c’	

1	-2	4

ElVector

5.1. Descripción

- Un **array** es una ***colección estructurada*** de componentes del ***mismo tipo*** a las que se puede acceder de forma individual por su posición dentro de la colección.
- Toda la colección de datos se almacena en un área de memoria contigua bajo ***un solo nombre***.
- Para ***acceder*** a cada ***componente individual*** se utilizan ***índices*** que indican la posición de la componente dentro de la colección.

5.1. Descripción

- **Ejemplo:**

El array se denomina: ElVector

1	-2	4

El elemento que ocupa la segunda posición del array ElVector

El array se denomina: ElTablero

‘c’	‘x’	‘c’
‘x’	‘x’	
	‘c’	

Este es el elemento que ocupa la tercera fila y primera columna del array ElTablero

5.1. Descripción

- Un **array** es **unidimensional** si a cada componente se accede mediante un único índice.

  Ejemplo:

1	-2	4

[posición]

- Un **array** es **bidimensional** si a cada componente se accede mediante 2 índices.

  Ejemplo:

‘c’	‘x’	‘c’
‘x’	‘x’	
	‘c’	

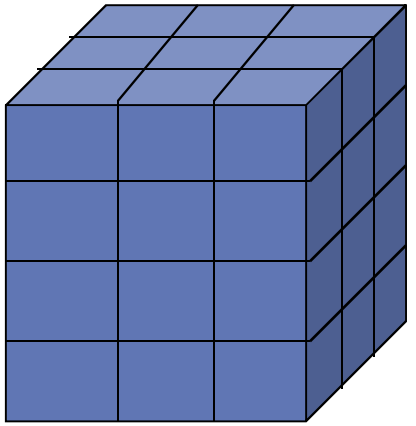
[posFila, posColumna]

5.1. Descripción

- Un **array** es **multidimensional** si a cada componente se accede mediante más de un índice.

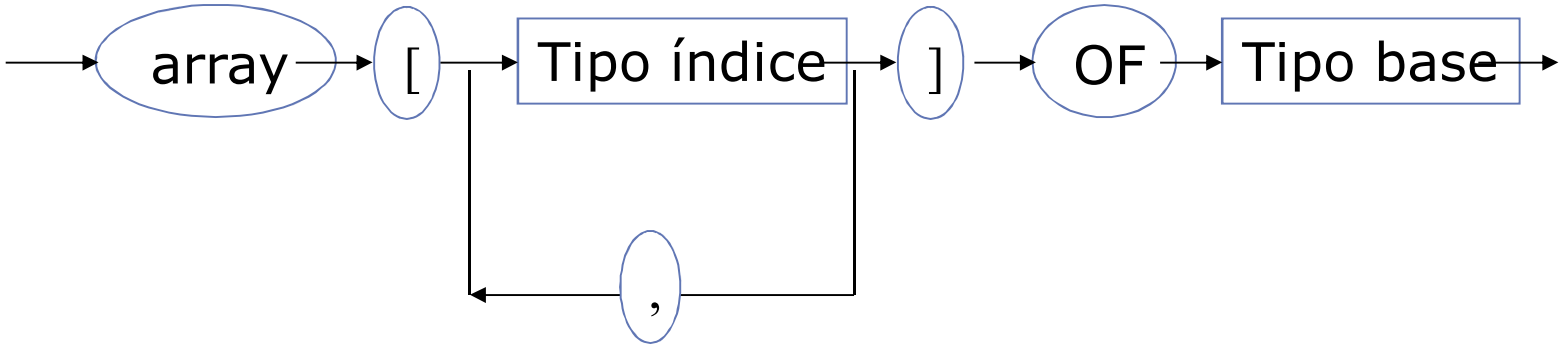
Ejemplo:

[pos1D, pos2D, pos3D]



5.1. Descripción

- Diagrama Sintáctico



Tipo índice: expresa el rango de los valores de los índices, debe ser un **ordinal** simple.
Tipo base: tipo de las componentes. Puede ser simple o compuesto.

5.1. Descripción

- Declaración de arrays:

Unidimensional:

array [TipoIndice1] OF Tipobase;

Multidimensionales ≈ Matrices y Tensores N-Dim.

array [TipoIndice1,TipoIndice2,…,TipoIndice**N**] OF Tipobase;

array [TipoIndice1] OF array [TipoIndice2] OF ...

array [TipoIndice**N**] OF Tipobase;

5.1. Descripción

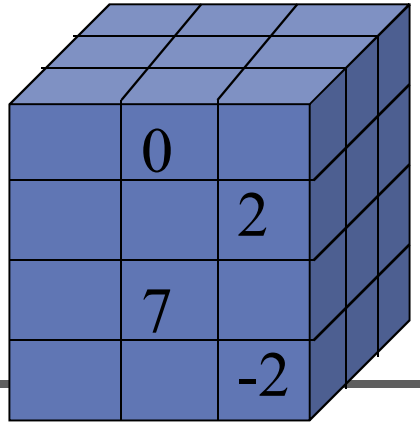
- Declaración de arrays, ejemplos:

1	-2	4

array [1..3] OF integer;

‘c’	‘x’	‘c’
‘x’	‘x’	
	‘c’	

array [1..3,1..3] OF char;



array [1..4,1..3,1..3] OF integer;

5.1. Descripción

✓ Un tipo de dato array se define en la sección de declaración de tipos **TYPE.**

```
TYPE
  TSubrango1 = 1..3;
  TSubrango2 = 1..4;
  TTipo1 = array [TSubrango1] OF integer;
  TTipo2 = array [TSubrango1,TSubrango1] OF
 char;
  TTipo3 = array
 [TSubrango2,TSubrango1,TSubrango1] OF
 integer;
VAR
  var1:  TTipo1;
  var2:  TTipo2;
  var3:  TTipo3;
```

Ejemplo 5.1

- Ejemplo:

$$u \cdot v = u_1v_1 + u_2v_2 + u_3v_3$$

```
CONST
  INI = 1;
  FIN = 3;
TYPE
   TSubrango1 = INI..FIN;
   TVector3Elem = array [TSubrango1] OF
  real;
VAR
    u,v: TVector3Elem; {con array}
    u1,u2,u3,v1,v2,v3: real;
```

Dos alternativas

5.1. Descripción

- Se pueden declarar **arrays anónimos**

```
VAR
   tresReales: array [1..3] OF real;
```

(Se ven afectados por las restricciones de definición de tipos anónimos vistas en el tema anterior)

- En el **acceso a un elemento** del array se utilizan tantos índices como dimensiones tenga dicho array.

- Para acceder a un elemento: se expresa el nombre de la variable array y entre corchetes el o los valores de los índices del elemento dentro del array.

5.1. Descripción

- Acceso a elementos, ejemplos:

1	-2	4

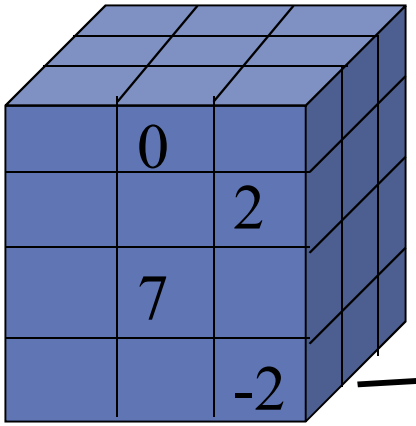
array [1..3] OF integer;

var1[3]

‘c’	‘x’	‘c’
‘x’	‘x’	
	‘c’	

array [1..3,1..3] OF char;

var2[3,2]



array [1..4,1..3,1..3] OF integer;

var3[4,3,2]

5.1. Descripción

- Los arrays son **estructuras de acceso directo**, ya que permiten almacenar y recuperar directamente los datos, especificando su posición dentro de la estructura.
- Los arrays son **estructuras de datos homogéneas**: sus elementos son TODOS del MISMO TIPO.
- El **tamaño** de un array se establece de forma FIJA, en un programa, cuando se define una variable de este tipo.

5.1. Descripción

- Ejemplos de definición de tipos arrays:

```
TYPE
 TSubrango1 = 1..3;
 TSubrango2 = 1..7;
 TSubrango3 = 'A'..'E';
 TVector = array [TSubrango1] OF real;
 TMatriz = array [TSubrango1, TSubrango2] OF char;
 TCubo = array [TSubrango1, TSubrango1, TSubrango1]
  OF BOOLEAN;
 TViviendas = array [TSubrango1, TSubrango1,
  TSubrango3]OF BOOLEAN;
 TEnorme = array [integer] OF real;
                {Erróneo fuera de rango}
                {Dimensión máxima Maxint-1}
```

Ejemplo 5.2

El tipo de los índices puede ser cualquier ordinal. Por ejemplo, un índice de tipo enumerado:

```
TYPE
  TDiasSemana = (Lun, Mar, Mie, Jue, Vie, Sab,
  Dom);
  TArrayDeDias = array [TDiasSemana] OF
                                    integer;
VAR
  visitasMuseo: TArrayDeDias;
BEGIN
  visitasMuseo[Lun]:= 58;
  visitasMuseo[Vie]:= 10;
```

Ejemplo 5.3

Un índice puede ser el resultado de una expresión compatible con el tipo de dicho índice. Ejemplo:

```
TYPE
  TDiasSemana = (Lun, Mar, Mie, Jue, Vie, Sab, Dom);
  TArrayDeDias = array [TDiasSemana] OF integer;
VAR
  dia: TDiasSemana;
  visitasMuseo, ingresoMuseo: TArrayDeDias;

BEGIN
  dia := Lun;
  visitasMuseo[dia] := 58;
  visitasMuseo [succ(dia)] := 10;
END.
```

5.1. Descripción

- **Almacenamiento de un array**
  - **Definición estática de variables**

Se reserva espacio de memoria, en posiciones contiguas, para el máximo de componentes que puede acoger el array.

Ejemplo: Memoria necesaria para **tresReales**

```
TYPE
   TSubrango1 = 1..3;
   T3Reales = array [TSubrango1] OF real;
VAR
   tresReales: T3Reales;
```

Si para un real se necesitan 4 bytes, para tresReales se reservan 4*3 = 12 bytes, en posiciones contiguas de memoria.

5.2 Operaciones

- **Proceso de todas las componentes** de un array.
- Operación de **entrada** de las componentes de un array.
- Operación de **salida** de las componentes de un array.
- Como ocurre con todos los ***tipos compuestos,*** **NO** se pueden realizar operaciones de entrada/salida con arrays completos.

Ejemplo 5.4

Proceso de **todas** las componentes de un array:

```
TYPE
  TDiasSemana = (Lun, Mar, Mie, Jue, Vie, Sab,
  Dom);
  TArrayDeDias = array [TDiasSemana] OF integer;
VAR
  dia: TDiasSemana;
  visitasMuseo, ingresoMuseo: TArrayDeDias;
BEGIN
  ...
  FOR dia := Lun TO Dom DO
    visitasMuseo[dia] := 0;
{Inicializamos a 0 todas las componentes del
  array}
  ...
END.
```

Ejemplo 5.5

Operación de **entrada** de las componentes de un array:

```
TYPE
  TDiasSemana = (Lun, Mar, Mie, Jue, Vie, Sab,
  Dom);
  TArrayDeDias = array [TDiasSemana] OF integer;

VAR
  dia: TDiasSemana;
  visitasMuseo, ingresoMuseo: TArrayDeDias;
BEGIN
  ...
  FOR dia := Lun TO Dom DO
      readln(visitasMuseo[dia]);
  readln(ingresoMuseo[Lun]);
  ...
END.
```

Ejemplo 5.6

Operación de **salida** de las componentes de un array:

```
TYPE
  TDiasSemana = (Lun, Mar,Mie, Jue,Vie, Sab,
  Dom);
  TArrayDeDias = array [TDiasSemana] OF integer;

VAR
  dia: TDiasSemana;
  visitasMuseo, ingresoMuseo: TArrayDeDias;
BEGIN
  ...
  FOR dia := Lun TO Dom DO
      writeln(visitasMuseo[dia]);
  ...
END.
```

Ejemplo 5.7

NO se pueden realizar operaciones de entrada/salida con arrays completos:

```
TYPE
  TDiasSemana = (Lun, Mar, Mie, Jue, Vie,
  Sab, Dom);
  TArrayDeDias = array [TDiasSemana] OF
  integer;
VAR
  dia: TDiasSemana;
  visitasMuseo, ingresoMuseo: TArrayDeDias;
BEGIN
  ...
    writeln(visitasMuseo); {ERROR}
  ...
END.
```

5.2 Operaciones

- **Asignación a arrays completos**

  - ✓ Se puede asignar un array a otro si los tipos son idénticos.

  - ✓ Dependiendo de las versiones de Pascal, la asignación de un array completo se puede realizar entre arrays de tipos equivalentes aunque no sean idénticos.

Ejemplo 5.8

```
CONST
   INI = 1;
   LIMITE = 100;
TYPE
   TRango = INI..LIMITE;
   TNota = array[TRango] OF real;
VAR
   notas_a, notas_aa: TNota;
BEGIN
      ...
      notas_a := notas_aa;
      ...
END.
```

5.2 Operaciones

- **Recomendaciones**
  - Evitar errores de intervalo:

    Cuando un subíndice toma valores fuera de su rango definido.

  - Para activar la verificación de intervalos:

    Options/Compiler Chequear Rango

5.2 Operaciones

- **Arrays como parámetros de subprogramas**
  - Se definen como parámetro empleando un tipo definido, nunca un tipo **anónimo**.
  - Respetar las reglas de compatibilidad de tipos entre parámetros reales y formales:
    - Parámetros formales **por referencia**, que sean arrays, deben ser **exactamente** del mismo tipo que sus parámetros reales (tipos idénticos).
    - Parámetros formales **por valor**, que sean arrays, deben ser **del mismo tipo** que sus correspondientes parámetros reales, **a menos que sean cadenas** (más adelante). En este caso **basta** con que sean de la **misma longitud**.

5.2 Operaciones

- **Arrays en funciones**
  - Las funciones **NO** pueden devolver valores de tipo array.
  - Si un subprograma necesita devolver un array, este se deberá pasar como **parámetro por referencia** de un procedimiento.
- **Paso de arrays por valor o por referencia**
  - El paso por valor puede conllevar el uso poco eficiente de la memoria (copia del array en el parámetro formal), especialmente si el array es muy grande.
  - Es muy común que se pasen por referencia aunque no se vaya a cambiar el valor de sus componentes (cuidado con efectos laterales). A no ser que sea necesario, no lo haremos.

5.2 Operaciones

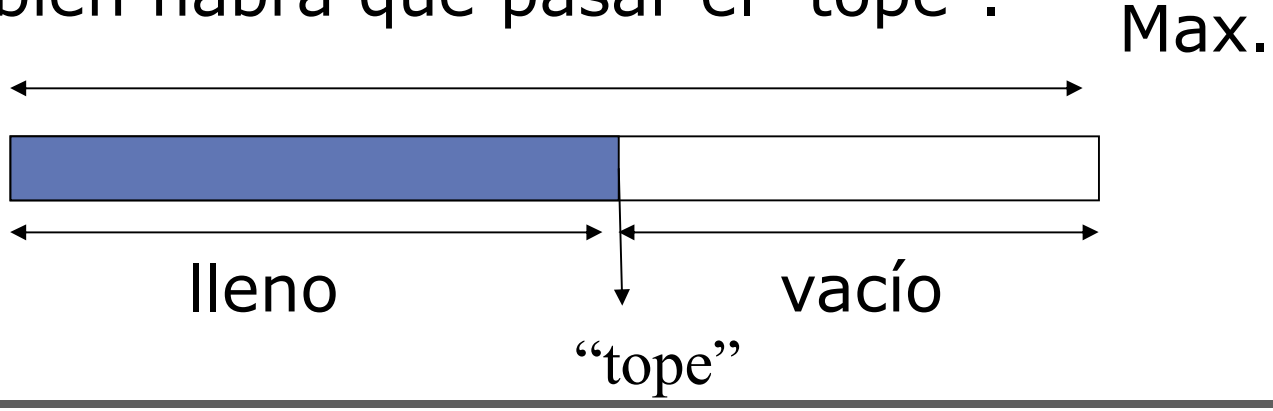
✓ Arrays parcialmente llenos

- En Pascal, la memoria correspondiente a un array se declara estáticamente (un array tiene un número fijo de componentes).

- Si este número puede variar de una ejecución a otra, habrá que hacer una estimación del número máximo de elementos que el array puede contener y llevar un registro de los límites de la parte ocupada.

5.2 Operaciones

- ✓ **Arrays parcialmente llenos** (continuación)
- Se puede utilizar una variable que vaya marcando el extremo superior de la parte ocupada.
  - Definir el array con el máximo y utilizar una variable indicadora ("tope") del número de componentes del array.
  - Cuando se pasa el array como parámetro también habrá que pasar el "tope".



5.2 Operaciones

Ejemplo: Leer por teclado y mostrar por pantalla los salarios de los empleados de una empresa. (Máximo 30)

Entrada:

Empleado Nº 1
    Num. Identificación: 1111
    Salario: 234.50
¿Más empleados (S/N) ? S
Empleado Nº 2
    Num. Identificación: 2222
    Salario: 345.50
¿Más empleados (S/N) ? n

Salida:

Empleado	Salario
1111	234.50
2222	345.50

Salario Medio: 290 ∈

Ejemplo 5.9

```
CONST
   INI = 1;
   MAX = 30;
TYPE
   TIndice = INI..MAX;
   TArrayEmpleados = array [TIndice] OF integer;
   TArraySueldos = array [TIndice] OF real;
…
PROCEDURE Registra(VAR empleados: TArrayEmpleados;
          VAR sueldos: TArraySueldos; VAR tope: TIndice);
VAR
  opcion: char;
BEGIN
  write('Quieres Introducir Empleados (S/N): ':50);
  readln(opcion);
```

Ejemplo 5.9

```
WHILE ((opcion = 'S') OR (opcion = 's')) AND
(tope < MAX) DO
BEGIN
    tope := tope + 1;
    writeln('Empleado N°: ',Tope,': ');
    write('Num. Identificacion: ');
    readln(empleados[tope]);
    write('Salario: ');
    readln(sueldos[tope]);
    write('Quieres Introducir Empleados (S/N):
':50);
    readln(opcion);
END {while}
END;{de Registra }
```

5.3 Arrays multidimensionales

- En algunos problemas los datos que se procesan se pueden organizar de forma natural como una tabla (2 dimensiones) o con un array de más de 2 índices.

  Ejemplo: las temperaturas máximas de varias ciudades a lo largo de los días de un mes.

Día\Ciudad	Soria	Madrid	Ávila	...
1	**15.3**	**17.2**	**12.4**	**...**
2	**...**			
...	**...**			
31	**...**			**...**

5.3 Arrays multidimensionales

- El acceso a cada elemento de un array de *N* dimensiones se realiza a través de *N* índices.
- Para acceder a todas las componentes de un array multidimensional serán necesarios tantos bucles anidados como dimensiones tenga el array (un bucle para cada dimensión).

Ejemplo 5.10

Acceso a un elemento de un array de 2 dimensiones:

```
TYPE
  Ciudades = (Soria, Madrid, Avila, ...,
  Zamora);
  tDias = 1..31;
  tTabla = array[tDias,ciudades] of real;
VAR
  tabla: tTabla;
  ciu:Ciudades;
  dia:tDias;
BEGIN
  tabla[1,Soria]:=15.3;
```

Ejemplo 5.11

Ejemplo: recorrido por filas para leer las componentes

```
For dia:= 1 to 31 do
    For ciu:= Soria to Zamora do
        Readln(tabla[dia,ciu]);
```

Ejemplo: recorrido por columnas para leer las componentes

```
For ciu:= Soria to Zamora do
    For dia:= 1 to 31 do
        Readln(tabla[dia,ciu]);
```

5.3 Arrays multidimensionales

- ✓ Los arrays bidimensionales nos permiten representar matrices y por lo tanto operar con ellas.
- ✓ Una matriz de 2 dimensiones *n* x *m* se podrá representar:

  array [1..n, 1..m] OF TipoBase

Ejemplo 5.12

```
PROCEDURE EscribeMatriz(matriz:tArray2dimensiones;
                                n:integer ;m:integer);
VAR
  f,c:integer;
BEGIN
  FOR f:= 1 TO n DO BEGIN
      FOR c:= 1 TO m DO
            WRITE (matriz[f,c]);
      WRITELN;
   END;{FOR}
END;{EscribeMatriz}
```

5.4. Tipo *string*

- String = Cadenas de caracteres = Array de caracteres

- PASCAL estándar no tiene tipo string.
- TURBO PASCAL incorpora el tipo string y las funciones y procedimientos para manejarlo.
- **Declaración:**
  - Lista de variables: string[*MAX*];

MAX es una constante entera que especifica el número máximo de caracteres que puede llegar a tener la cadena (por defecto 255).

5.4.1.Tipo *string*

- Declaración de variables. Ejemplos:

```
CONST
   LIMITE = 10;
VAR
   nombre: string[LIMITE];
   departamento: string[20];
   mensaje: string;{Como no se ha
   limitado explícitamente el tamaño,
   mensaje puede almacenar hasta 255
   caracteres}
```

5.4.1.Tipo *string*

- Se puede acceder a los caracteres de una cadena mediante sus índices, como en los arrays.
- La longitud de la cadena (n) es el número de caracteres que tiene.
- El string tiene además una posición 0 en la que se almacena el chr (n).

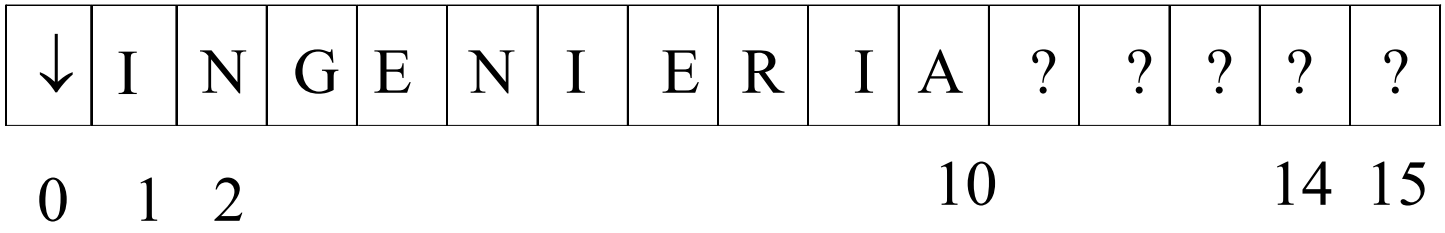
```
Longitud := ord(Cadena[0]);
```

5.4.1.Tipo *string*

```
CONST
  LIMITE = 8;
VAR
  nombre: string[LIMITE];
  departamento:
  string[15];
  descripcion: string;
```

```
INGENIERIA tiene 10
chr(10) =
ord(departamento[0]) ↓=
10
```

departamento



5.4.1.Tipo *string*

- **Asignación**

cadena := *expresión*

- *Expresión* puede ser una constante, una variable o una expresión cualquiera del tipo cadena o de tipo char.

```
VAR
    cadena1: string;
    cadena2: string[3];
BEGIN
  cadena1 := 'A';
  cadena1 := 'HOLA';
  cadena1 := cadena1[3];
  cadena2 := 'Maria';
  writeln(ord(cadena2[0]));
```

```
Cadena1    A
Cadena1    HOLA
Cadena1    L
Cadena2    Mar
             3
```

5.4.1.Tipo *string*

- **Operaciones**

- Se pueden **leer** y **escribir**:

  readln(cadena1);
  writeln(cadena1);

- **Comparación**:<, >, =, <=, >= y <>
  - Se comparan carácter a carácter.
  - Si una es más corta que la otra, se compara como si la más corta se rellenara con blancos.

5.4.1.Tipo *string*

Operaciones

- ✓ **Concatenación:** Dos formas de hacerlo
  - Operador **+**
    - string + string -> string
  - Función **Concat**
    - Concat(string, string)->string

```
nombre := 'Antonio';
apellidos := ' Banderas';
actor := nombre + apellidos
```

```
actor=Antonio Banderas
```

```
nombre := 'Antonio';
apellidos := ' Banderas';
actor := Concat(nombre, apellidos);
```

5.4.1.Tipo *string*

- **Funciones predefinidas**
  - ✓ **Longitud:** length(cadena)

    length:(string) -> int

    Halla la longitud lógica o real (valor entero) de la cadena argumento.

    lenght(cadena) ⇔ ord(cadena[0])

5.4.1.Tipo *string*

- **Posición de una cadena en otra:**

  pos(cadena1, cadena2)

  pos: (string, string)->int

  Devuelve:

  - La posición (valor entero) en la que comienza la cadena1 dentro de cadena2, o
  - El valor 0 cuando cadena1 no está incluida en cadena2.

Ejemplo 5.13

```
VAR
  cadena1, cadena2: string;
BEGIN
  ...
  cadena1 := 'Metodologia de
  programacion';
  cadena2 := 'programa';
  writeln(pos(cadena1, cadena2));
  ...
END.
```

Salida pantalla: 0

5.4.1.Tipo *string*

- **Extraer una subcadena:**

  copy(cadena, posición, tamaño)

  copy:(string, int, int)-> cadena

  Devuelve una subcadena de *cadena*, formada por *tamaño* caracteres a partir de *posición*.

Ejemplo 5.14

```
VAR
  cadena1: string;
BEGIN
  ...
  cadena1:= 'Metodologia de
  Programacion';
  writeln(copy(cadena1,3,4));
  ...
END.
```

Salida pantalla: todo

5.4.1.Tipo *string*

- **Procedimientos predefinidos**

- ✓ **Borrado:**

  delete (cadena, posición, tamaño)

  Borra *tamaño* caracteres de *cadena* a partir de *posición*. La cadena reduce su longitud en el número de caracteres eliminados.

Ejemplo 5.15

```
VAR
  cadena1: string;
BEGIN
  ...
 cadena1 := 'Metodologia de la
  Programacion';
 delete(cadena1,7,16);
 writeln(cadena1);
  ...
END.
```

Salida pantalla:

```
Metodoramacion
```

5.4.1.Tipo *string*

- **Inserción:**

  insert(cadena1, cadena2, posición)

  Inserta *cadena1* en *cadena2* a partir de *posición*. La cadena2 aumenta su longitud con el número de caracteres igual a la longitud de la cadena1.

Ejemplo 5.16

```
VAR
  cadena1, cadena2: string;
BEGIN
  ...
  cadena1 := 'Metodologia ';
  cadena2 := 'Programacion: ';
  insert(cadena1, cadena2, 1);
  writeln(cadena2);
  ...
END.
```

Salida pantalla: `Metodologia Programacion:`

5.4.1.Tipo *string*

- **Conversión:**

  str(expresión-numérica, cadena)

  Convierte el valor de *expresión-numérica* en *cadena* (su representación como cadena de caracteres).

Ejemplo 5.17

```
VAR
  i: integer;
  cadena: string;
BEGIN
  ...
  i := 1200;
  str(i, cadena);
  writeln(cadena);
  ...
END.
```

Pantalla: ‘1200’

```
VAR
  r: real;
  cadena: string;
BEGIN
  ...
  r := 5.5;
  str(r,cadena);
  writeln(cadena);
  ...
END.
```

Pantalla: 5.5000000000E+0.0

5.4.1.Tipo *string*

- **Conversión:**

  val(cadena, valorNumérico, código-error)

  Convierte *cadena* en su valor numérico real *valorNumérico* (también puede ser un entero) y devuelve un *código-error* (integer) cuyo valor podrá ser:

  - 0 si se puede hacer la conversión,
  - la posición del primer carácter que produjo el error en caso contrario.

Ejemplo 5.18

```
CONST
   CADENA = '345';
VAR
   error: integer;
   valor: real;
BEGIN
   ...
   val(CADENA,valor,error)
   ;
   ...
END.
```

valor:	3.4500000000E+2
error:	0

```
VAR
   cadena: string[10];
   error: integer;
   valor: real;
BEGIN
      cadena := 'hola';
      val(cadena,valor,error);
         ...
END.
```

valor:	
error:	1

5.5. Algoritmos con arrays

- Dos de las operaciones más usuales con los arrays son:

  - Búsqueda de un dato en un array.
  - Ordenación de las componentes de un array.

5.5.1. Algoritmos de búsqueda

- ✓ Determinan si un dato concreto se encuentra en una colección de datos del mismo tipo.
- ✓ En caso de que se encuentre el dato, permiten conocer la posición que ocupa.
- ✓ Los algoritmos de búsqueda más usuales son:

  - Búsqueda secuencial o lineal
  - Búsqueda binaria o dicotómica

5.5.1. Algoritmos de búsqueda

- Precondiciones:
  - La búsqueda se hará en un array **unidimensional** (vector), tratando de encontrar un elemento, ***elemBuscado*** **.**
  - El array ***vector*** es de ***n*** elementos.
  - El índice del array está formado por el intervalo **[*Primero..Ultimo*].**

5.5.1. Algoritmos de búsqueda

- Objetivo:

  Definir una función ***Búsqueda*** que encuentre el primer valor del índice ***posición***, de tal manera que se cumpla que:

  vector*[*posicion*] = *elemBuscado

5.5.1. Algoritmos de búsqueda

1. Búsqueda secuencial o lineal:

- ✓ Se realiza un recorrido del array comparando el elemento buscado con los valores contenidos en las componentes del array.
- ✓ Se comienza con la primera componente y se va avanzando de forma secuencial hasta que:
  - Se encuentra el valor buscado, o
  - Se llega al final del array (o de su parte ocupada) sin encontrarlo.

Ejemplo 5.19

```
CONST
  PRIMERO = 1 ;
  ULTIMO = 100 ;
TYPE
  TIntervalo = PRIMERO..ULTIMO;
  TElem = integer;
  TVector = array [TIntervalo] OF
  TElem;

VAR
  vector : TVector;
  elem : TElem;
```

Ejemplo 5.19 (Cont.)

```
FUNCTION
  BusquedaSecuencial1(v:TVector;elemBuscado:TElem):integer;
VAR
  posicion : TIntervalo;
BEGIN
  posicion := PRIMERO;
   WHILE (posicion < ULTIMO) AND (v[posicion] <>
  elemBuscado) DO
        posicion := succ(posicion);
   IF (v[posicion] <> elemBuscado) THEN
          BusquedaSecuencial1 := pred(PRIMERO)
  ELSE
          BusquedaSecuencial1 := posicion ;
END; {BusquedaSecuencial1}
```

Ejemplo 5.19 (Cont.)

```
FUNCTION
  BusquedaSecuencial2(v:TVector;elemBuscado:TElem):integer;
VAR
  posicion: integer;
BEGIN
  posicion:= pred(PRIMERO);
  REPEAT
      posicion := succ(posicion);
  UNTIL (posicion = ULTIMO) OR (v[posicion] = elemBuscado);
  IF  v[posicion] = elemBuscado THEN
       BusquedaSecuencial2 := posicion
  ELSE
       BusquedaSecuencial2 := pred(PRIMERO);
END; {BusquedaSecuencial2}
```

Ejemplo 5.19 (Cont.)

```
FUNCTION
   BusquedaSecuencial3(v:TVector;elemBuscado:TElem)
   :integer;
VAR
        posicion: pred(PRIMERO)..ULTIMO;{integer}
        encontrado: boolean; {centinela}
BEGIN
   posicion := pred(PRIMERO);
   encontrado := FALSE;
   BusquedaSecuencial3 := posicion;
   REPEAT
        posicion := succ(posicion);
        IF  v[posicion] = elemBuscado THEN
        BEGIN
             encontrado := TRUE;
             BusquedaSecuencial3 := posicion
        END
   UNTIL (posicion = ULTIMO) OR (encontrado);
END; {BusquedaSecuencial3}
```

5.5.1. Algoritmos de búsqueda

2. Búsqueda secuencial o lineal en un array ordenado:

- ✓ Precondición: los elementos del array están ordenados de forma creciente (decreciente).

  - Se comienza con la primera componente y se va avanzando de forma secuencial hasta que:

    - Se encuentra el valor buscado, o
    - Se llega a una componente con un valor mayor que el buscado (orden ascendente), o
    - Se llega al final del array sin encontrarlo.

Ejemplo 5.20

```
FUNCTION
   BusquedaSecOrdenada(v:TVector;elemBuscado:TEle
   m):integer;
VAR
   posicion: integer ;
BEGIN
   posicion := pred(PRIMERO);
   REPEAT
       posicion := succ(posicion);
   UNTIL (posicion = ULTIMO) OR (v[posicion] >=
   elemBuscado);
   IF  v[posicion] = elemBuscado   THEN
        BusquedaSecOrdenada := posicion
   ELSE
        BusquedaSecOrdenada := pred(PRIMERO)
END; {BusquedaSecOrdenada}
```

5.5.1. Algoritmos de búsqueda

3. Búsqueda binaria o dicotómica

- Precondición: los valores de los elementos del array están ordenados (Creciente/Decreciente).
- La búsqueda binaria divide el array en dos mitades y determina si ***elemBuscado*** está en la primera o en la segunda mitad.
- Continúa dividiendo el espacio de búsqueda en mitades hasta encontrar el elemento o determinar que no está.
- La búsqueda binaria es el método utilizado para buscar en un diccionario, listín telefónico, etc.

5.5.1. Algoritmos de búsqueda

Fases:

1. Comparar ***elemBuscado*** con el elemento ***central*** del array.
2. Si es igual, la búsqueda ha terminado.
3. Si no es igual, se busca en la mitad adecuada del array (descartando la búsqueda en la otra mitad):
   1. La primera mitad, si ***elemBuscado < vector*[*central*]**
   2. La segunda mitad, si ***elemBuscado > vector*[*central*]**
4. Se vuelve al paso 1. la mitad elegida.

Este proceso se repite hasta que se encuentra ***elemBuscado*** o la mitad elegida está vacía o tiene un solo elemento.

Ejemplo 5.21

Implementación:

```
FUNCTION BusquedaBinaria(v: TVector; elemBuscado:
  TElem):integer;
VAR
  extInf, extSup, central: integer;
  encontrado: boolean;
BEGIN
  extInf := PRIMERO;
  extSup := ULTIMO;
  encontrado := FALSE;
```

Ejemplo 5.21 (Cont.)

```
WHILE (NOT encontrado) AND (extSup >= extInf) DO
BEGIN  {WHILE}
      central := (extSup + extInf) DIV 2;
       IF  v[central] = elemBuscado THEN
           encontrado:= TRUE
       ELSE
            IF  v[central] < elemBuscado THEN
                  extInf := succ(central)
            ELSE
                  extSup := pred(central);
END; {WHILE}
IF encontrado THEN
      BusquedaBinaria := central
ELSE
      BusquedaBinaria := pred(PRIMERO)
END; {BusquedaBinaria}
```

5.5.2. Algoritmos de ordenación

- Mecanismo que permite ordenar los elementos de un array.

- Existen numerosos algoritmos de ordenación. Clasificándoles por la forma en que la realizan. Tendríamos algoritmos de:

  - Intercambio
  - Fusión

5.5.2. Algoritmos de ordenación

1. Algoritmos de ordenación por intercambio

- Los algoritmos de intercambio se caracterizan por intercambiar pares de elementos del array hasta conseguir su ordenación.

- Los más conocidos son:

  - **Selección directa**
  - **Inserción directa**
  - **Intercambio directo (burbuja)**

5.5.2. Algoritmos de ordenación

A. Selección directa

- ✓ NO es el más aconsejable para ordenar ARRAYS GRANDES, pero se comporta razonablemente bien en arrays pequeños.
- ✓ Se recorre el array seleccionando un elemento (menor/mayor) en cada recorrido y se coloca en la posición correcta. Tras n-1 recorridos, el array está ordenado.

5.5.2. Algoritmos de ordenación

Selección directa

67 33 **21** 84 49 50 75

21 **33** 67 84 49 50 75

21 33 67 84 **49** 50 75

21 33 49 84 67 **50** 75

21 33 49 50 **67** 84 75

21 33 49 50 67 84 **75**

21 33 49 50 67 75 84

5.5.2. Algoritmos de ordenación

Selección directa

Paso 1

Se selecciona el elemento de valor menor (mayor), intercambiando su posición con la del primer elemento del array.

Paso 2

Se busca el valor menor (mayor) en las posiciones restantes del array (2,...,n), intercambiando su posición con la del segundo elemento del vector.

5.5.2. Algoritmos de ordenación

Selección directa

Paso j-1

Se sitúa en la posición j-1 del array el valor menor (mayor) entre v[j-1] y v[n].

Paso n-1

Se sitúa en la posición n-1 del array el valor menor (mayor) entre v[n-1] y v[n].

5.5.2. Algoritmos de ordenación

Selección directa

Necesitamos 3 índices: - 2 para recorrer el array (i,j)
- uno que apunta al menor elemento (posMenor)

Necesitamos 1 variable auxiliar para el intercambio, valMenor

- Pasos, para i desde 1 a n-1:
  1. Asignar a posMenor el valor i
  2. Asignar a valMenor el valor v[posMenor]

  } Inicialización

  3. Para j desde i+1 a n:
     Si v[j]<valMenor, entonces:
     - Asignar a posMenor el valor j
     - Asignar a valMenor el valor v[posMenor]

  } Búsqueda

  4. Asignar a v[posMenor] el valor de v[i]
  5. Asignar a v[i] el valor valMenor

  } Intercambio

Ejemplo 5.22

```
PROCEDURE SeleccionDirecta (VAR v: TVector);
VAR    i, j, posMenor : integer ;
   valMenor: integer;
BEGIN
   FOR i := PRIMERO TO pred(ULTIMO) DO
   BEGIN
      valMenor := v[i];
      posMenor := i;
      FOR j := succ(i) TO ULTIMO DO
          IF  v[j] < valMenor THEN
          BEGIN
             valMenor := v[j];
             posMenor := j
          END; {IF}
      IF  posMenor <> i THEN
      BEGIN
          v[posMenor] := v[i] ;
          v[i] := valMenor;
      END; {IF}
   END; {FOR i}
END; {SeleccionDirecta}
```

Inicialización

Búsqueda

Intercambio

5.5.2. Algoritmos de ordenación

B. Inserción directa:

- ✓ Método eficiente de ordenación para conjuntos pequeños de datos.
- ✓ Consiste en recorrer el array v, insertando cada elemento v[i] en el lugar correcto entre los elementos ya ordenados v[PRIMERO] ,...,v[i-1].

5.5.2. Algoritmos de ordenación

Inserción directa:

Paso 1: Se considera v[1] como primer elemento.

Paso 2: Se inserta v[2] en la posición correspondiente en relación a v[1] y v[2] .

Paso i: Se inserta v[i] en la posición correspondiente en relación a v[1],...,v[i] .

Paso n: Se inserta v[n] en la posición correspondiente en relación a v[1],...,v[n] .

Resumen

- ✓Para cada i entre 2 y n, situar v[i] en su posición ordenada respecto de v[1],...,v[i-1] .

Ejemplo 5.23

```
PROCEDURE InsercionDirecta (VAR v: TVector) ;
VAR
   i, j: integer;
   aux: TElem;
BEGIN
   FOR i := succ(PRIMERO) TO ULTIMO DO
   BEGIN
       aux := v[i] ;
       j := pred(i) ;
       WHILE (j >= PRIMERO) AND (v[j] > aux) DO
       BEGIN
            v[j+1] := v[j] ;
            j := pred(j)
       END; {WHILE}
       v[j+1] := aux
   END; {FOR}
END; {InsercionDirecta}
```

5.5.2. Algoritmos de ordenación

C. Intercambio directo:

- ✓ También llamado **método** de la **burbuja**.
- ✓ Eficiente para conjuntos pequeños de datos.

Resumen

- ✓ Recorrer el array, buscando el menor elemento, desde la última posición hasta la actual. Una vez encontrado, se sitúa en la posición actual.
- ✓ Para ello, se intercambian valores adyacentes siempre que estén colocados en orden decreciente.

5.5.2. Algoritmos de ordenación

Intercambio directo:

Paso 1. Situar el elemento mayor en la última posición del array.

Paso 2. Situar el segundo elemento mayor en la penúltima posición del array.

Paso i. Se repite el proceso para las posiciones intermedias.

Paso n-1. Se comparan los valores de las dos primeras posiciones, situando el n-1 mayor en la segunda posición.

Ejemplo 5.24

```
PROCEDURE IntercambioDirecto(VAR v: TVector);
VAR
   i, j: TIntervalo ;
   aux: TElem;

BEGIN
   FOR i := PRIMERO TO pred(ULTIMO) DO
       FOR j := PRIMERO TO ULTIMO-i DO
          IF v[j] > v[j+1] THEN
          BEGIN
             aux := v[j] ;
             v[j] := v[j+1] ;
             v[j+1] := aux
          END; {IF}
END; {IntercambioDirecto}
```

5.5.2. Algoritmos de ordenación

2. Algoritmo de ordenación por fusión

Se basan en la operación de fusión de 2 arrays ordenados.

Dados 2 arrays ordenados A[INI..FA] y B[INI..FB], la fusión da como resultado un array C [INI..FA+FB] con todos los elementos de A y B y que también estará ordenado.

Vamos a ver el método:

- ***Mergesort*** o por mezcla.

5.5.2. Algoritmos de ordenación

En primer lugar vamos a ver cómo se fusionan 2 arrays ordenados (realmente es uno dividido en sus dos mitades: v [iz..ce] y **v** [ce+1..de], **el resultado queda en el array w):**

```
PROCEDURE Fusion(v:TVector; iz,de,ce: integer,
VAR w: TVector);
VAR i, j, k: integer;
BEGIN {Fusion}
     i := iz; j := succ(ce); k := iz;
     WHILE (i <= ce) AND (j <= de) DO
     BEGIN
          IF v[i] < v[j] THEN
          BEGIN
               w[k] := v[i];
               i := succ(i)
          END
```

5.5.2. Algoritmos de ordenación

```
        ELSE
          BEGIN
               w[k] := v[j];
               j := succ(j)
           END;

    k := succ(k);

END;{while}

FOR k := j TO de DO

    w[k] := v[k];

FOR k := i TO ce DO

    w[k+de-ce] := v[k];

END;{Fusion}
```

5.5.2. Algoritmos de ordenación

Fases del *mergesort*:

- Se divide el array a ordenar en dos partes iguales.
- Cada parte se ordena mediante llamadas recursivas hasta que se llega a un determinado tamaño umbral del problema.
- Se fusionan las soluciones generando el array inicial ordenado.

5.5.2. Algoritmos de ordenación

Mergesort:

- El tamaño umbral del problema corresponde a subarrays de tamaño 1 (un array de un elemento se considera ordenado).
- También se puede considerar un tamaño umbral distinto de 1, y aplicar un algoritmo de ordenación que se comporte bien para arrays de pequeño tamaño.

Ejemplo 5.25

```
PROCEDURE Mergesort(VAR v: TVector; izq, der:
  TIntervalo);
VAR
  centro: TIntervalo;
BEGIN {Mergesort}
  centro := (izq + der) DIV 2;
  IF  izq < centro THEN
      Mergesort(v, izq, centro);
  IF centro + 1 < der THEN
      mergesort(v, centro+1, der);
  Fusion(v, izq, der, centro, v)
END;{mergesort}
```

ascension.lovillo@urjc.es