

Tema 11

Formaciones

Necesidad de las formaciones

Sirven para permitir la generalización de la declaración, referencia y manipulación de datos del mismo tipo.

Vectores

Un vector está constituido por una serie de valores, todos ellos del mismo tipo, a los que se les da un nombre común que identifica a toda la estructura globalmente. Cada valor concreto dentro de la estructura se distingue por su índice o número de orden que ocupa en la serie.

- **Declaración de vectores:** Una estructura de tipo vector se declara de la siguiente forma:

```
[ TpoVec = ] ARRAY TpoInd OF TpoEle
```

En donde:

- *TpoVec*: Es el nombre del nuevo tipo de vector que se declara.
- *TpoInd*: Es el tipo al que pertenece el índice y puede ser un escalar, enumerable, un subrango o cualquiera de los tipos predefinidos.
- *TpoEle*: Puede ser cualquier tipo y corresponde al tipo de dato que constituye cada uno de los elementos del vector.
- **Nota:** Si no se define el tipo de vector (*TpoVec*) se dice que el tipo es anónimo, pero hay que tener cuidado, ya que se producen incompatibilidades de tipo.
- **Operaciones con vectores:** Al manejar datos estructurados de tipo vector se dispone de dos posibilidades: operar con el dato completo, o bien operar con cada elemento por separado.
 - **Asignación:**

```
vector1 := vector2
```
 - **Referencia a un elemento:**

```
vector[ índice ]
```

Formaciones anidadas. Matrices

Es posible definir y utilizar vectores cuyos elementos sean a su vez otros vectores anidados dentro del primero.

- **Declaración de matrices:** Una estructura de tipo matriz se declara de la siguiente forma:

```
[ TpoMtz = ] ARRAY TpoInd1
  { OF ARRAY TpoIndN } OF TpoEle
```

o bien usar este otro formato:

```
[ TpoMtz = ] ARRAY TpoInd1, ... ,
  TpoIndN OF TpoEle
```

En donde:

- *TpoMtz*: Es el nombre del nuevo tipo de matriz que se declara.
- *TpoIndX*: Es el tipo al que pertenece cada uno de los índices y pueden ser un escalar, enumerable, un subrango o cualquiera de los tipos predefinidos.
- *TpoEle*: Puede ser cualquier tipo y corresponde al tipo de dato que constituye cada uno de los elementos de la matriz.
- **Nota:** Si no se define el tipo de matriz (*TpoMtz*) se dice que el tipo es anónimo, pero hay que tener cuidado, ya que se producen incompatibilidades de tipo.
- **Operaciones con matrices:** Al manejar datos estructurados de tipo matriz se dispone de dos posibilidades: operar con el dato completo, o bien operar con cada elemento por separado.
 - **Asignación:**

```
matriz1 := matriz2
```
 - **Referencia a un elemento:**

```
matriz[ índice1 ][ índiceN ]
```

 o bien, si se ha usado el 2º tipo de definición:

```
matriz[ índice1, índiceN ]
```

Esquemas típicos de operación

En la utilización habitual de las formaciones se presentan de forma reiterada ciertas operaciones que responden a un mismo esquema.

- **Recorrido:** Consiste en realizar la misma operación con todos y cada uno de los elementos de una formación.

```
FOR i := 1 TO fin DO
  (* Operaciones *)
  vector[ i ] := ...;
END;
```

- **Búsqueda secuencial:** Para buscar un elemento dentro de una formación es necesario recorrerla, pero no siempre en su totalidad. El recorrido se debe detener en cuanto se encuentre el elemento buscado, y por tanto sólo será completo cuando no se encuentre el elemento buscado dentro de la formación.

```
Inicial la búsqueda;
LOOP
  IF fin THEN EXIT END;
  Verificar encuentro;
  IF Encontrado THEN EXIT END;
  Pasar al siguiente elemento;
END;
```

- **Inserción:** Cuando los elementos de un vector se van incorporando al mismo de una forma paulatina, uno detrás de otro, se puede aprovechar esta circunstancia para insertar el nuevo elemento en la posición que le corresponda.

```
Iniciar inserción;
WHILE (NOT HayHueco) AND (NOT fin)
  DO
    Desplazar elemento;
    Pasar al siguiente elemento;
  END;
  Insertar nuevo elemento;
```

- **Ordenación por inserción directa:** El primer elemento de supone ordenado. A continuación, se extrae el segundo elemento y se crea un hueco, que se utiliza para ampliar el vector ya ordenado. Después consiste en insertar el elemento extraído en su lugar correspondiente entre los elementos ya ordenados. El proceso se repite con el resto de elementos.

```
FOR i := 2 TO fin DO
  item := vector[i];
  j := i-1;
  WHILE (item<vector[j]) AND (j>=1)
    DO
      vector[j+1] := vector[j];
      j := j-1;
    END;
  vector[j+1] := item
END;
```

- **Búsqueda por dicotomía:** En un vector ordenado, si comparamos el elemento a buscar con el que está justo en la mitad de los datos, podemos decidir si ése es el elemento que buscamos o debemos continuar buscando, pero sólo en la mitad derecha o sólo en la mitad izquierda.

El mismo proceso se puede repetir con la mitad elegida, comparando con el elemento que está en el centro de dicha mitad. En cada comparación, la búsqueda se reduce a comprobar si el dato buscado está entre la mitad de los anteriores. La búsqueda finaliza cuando se encuentra el elemento o no es ninguno de los dos datos consecutivos a los que queda reducida la comparación después de sucesivas divisiones en mitades.

```
pos := 0;
izq := 1;
dch := fin;
WHILE (pos=0) AND (izq<=dch) DO
  mit := (izq+dch) DIV 2;
  IF item=vector[mit] THEN
    pos := mit;
  ELSEIF item<vector[mit] THEN
    dch := mit - 1;
  ELSE
    izq := mit + 1;
  END;
END;
```

- **Simplificación de las condiciones de contorno:** La programación de operaciones con vectores, realizadas elemento a elemento, exige con frecuencia realizar un tratamiento especial de los elementos extremos del vector o, en general, de los elementos del *contorno* de una formación.

- **Centinelas:** Consiste en ampliar el vector con un elemento más de índice 0 para evitar tener que estar comprobando si llegamos al límite.

Búsqueda: Copiamos el dato a buscar en la posición del centinela (0), buscamos de atrás hacia delante y siempre encontrará el dato, si está (pos<>0) o no (pos=0).

```
vector[0] := item;
pos := fin;
WHILE item<>vector[pos] DO
  pos := pos -1;
END;
```

Inserción:

```
FOR i := 2 TO fin DO
  item := vector[i];
  vector[0] := item;
  j := i-1;
  WHILE item<vector[j] DO
    vector[j+1] := vector[j];
    j := j-1;
  END;
  vector[j+1] := item
END;
```

- **Matrices orladas:** Estas se dimensionan con dos filas y dos columnas más de las que se necesitan (bordes). En ellas se inicializa todo su contorno (filas y columnas extras) con un valor tal que permita identificar dicho contorno, con lo que cualquier operación que se realice con la matriz no necesita tener en cuenta sus límites señalados.

Vector de caracteres: Ristra (String)

Cualquier vector cuya declaración sea de la forma siguiente se considera una ristra o String con independencia de su longitud particular (N):

```
ARRAY [ 0 .. N ] OF CHAR
```

Características:

- El primer elemento de una ristra siempre es el de índice igual a cero.
- En un vector de esta clase se pueden almacenar ristas de diferentes longitudes (si caben). Para distinguir la longitud útil en cada momento se reserva siempre espacio para un carácter más, y se hace que toda ristra termine con un carácter nulo (0C) situado al final.

Argumentos de tipo vector abierto

Un vector abierto es aquel del que sólo se define el tipo de sus elementos, pero no el rango del índice. El tipo implícito de su índice es siempre un subrango comprendido entre 0 y un valor máximo. Se declaran de la siguiente forma:

```
PROCEDURE proc(X: ARRAY OF TpoEle)
```

Y para hacer referencia dentro de la función o procedimiento al tamaño del vector, se usa la función predefinida:

```
HIGH( X )
```

Se pueden utilizar vectores abiertos como argumentos en la definición de procedimientos o funciones. Por ejemplo, la función `WriteString`:

```
PROCEDURE WriteString(s: ARRAY OF
                                CHAR) ;

    CONST Nulo = 0C;
    VAR k: INTEGER;
BEGIN
    K := 0;
    LOOP
        IF k > HIGH(s) THEN EXIT END;
        IF s[k] = Nulo THEN EXIT END;
        Write(s[k]);
        INC(k)
    END;
END WriteString;
```