TEMA 4

ARRAYS

ARRAYS Y ESTRUCTURAS

- ÍNDICE
  - ESTRUCTURAS DE DATOS
  - LENGUAJE FUERTEMENTE TIPADO
  - ARRAYS UNIDIMENSIONALES O VECTORES
  - ARRAYS MULTIDIMENSIONALES
  - OPERACIONES CON VECTORES
  - ALMACENAMIENTO DE ARRAYS EN MEMORIA

TIPOS DE DATOS

- ***Tipo de datos:*** Conjunto específico de **valores de datos** y un conjunto de **operaciones** que actúan sobre esos datos.
- Tipos de datos:

  - ***Básicos, incorporados o integrados (*estándar)**, que se incluyen en los lenguajes de programación;

  - ***Definidos por el programador o por el usuario* del lenguaje.**

- Tipos de datos:

  ***- Simples* (sin estructura):** Numérico, booleano y carácter. Cada variable (identificador) representa un elemento.

  ***- Compuestos* (estructurados):** conjuntos de partidas de datos simples con relaciones definidas entre ellos. Un identificador representa múltiples datos individuales.

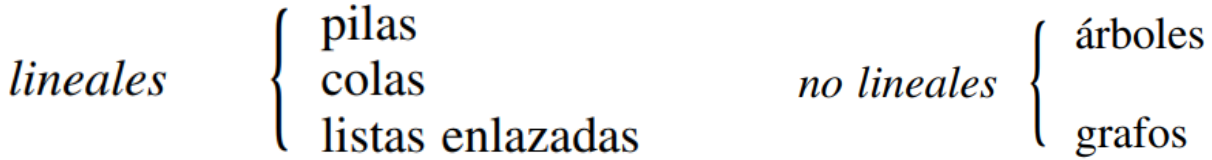
ESTRUCTURAS DE DATOS

- **Estructuras de datos estáticas:** Aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa.

  Ejemplo: arrays, registros, ficheros o archivos.

- **Estructuras de datos dinámicas:** No tienen limitaciones o restricciones en el tamaño de memoria ocupada. Crecen o decrece a medida que se ejecuta el programa.

  Ejemplo: listas, árboles y grafos.



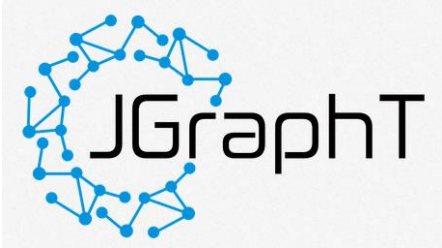
ESTRUCTURA DE DATOS

- Elección del tipo de estructura de datos idónea a cada aplicación: Dependerá esencialmente del **tipo de aplicación** y, en menor medida, del lenguaje.

- Se buscará la **existencia de librerías (módulos)** en las que se trabaje con esas estructuras de datos. Comprobar qué operaciones permite.

- Ejemplo: Librería Java para trabajar con grafos:

  https://jgrapht.org/



- **Si no está implementada la estructura** que necesitemos**, se simulará con algoritmos (y código fuente)**, dependiendo del algoritmo y lo que permita el lenguaje de programación.

LENGUAJE FUERTEMENTE TIPADO

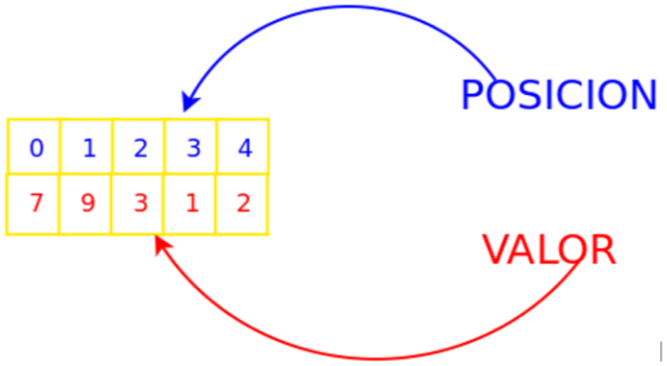
- Lenguaje de programación que impone restricciones estrictas sobre la **mezcla de valores con diferentes tipos de datos**. Cuando se violan tales restricciones se produce un error.
- El comportamiento de las operaciones es más predecible en comparación con las de los lenguajes débilmente tipados.
- El programador debe explícitamente convertir un tipo de datos en otro. Esto se conoce como **conversión o moldeado de tipos (type casting)** y debe estar soportado por el compilador.
- Un lenguaje de programación fuertemente tipado **no es necesariamente mejor** que uno débilmente tipado. Los escenarios en los que la flexibilidad será mucho más útil que la rigidez, y viceversa, son perfectamente aceptables.
- La diferenciación entre lenguajes fuertemente tipados y lenguajes débilmente tipados es algo borrosa.

ARRAYS

- Un **array** es una **colección estructurada** de ***elementos*** ***del mismo tipo,*** agrupadas bajo un mismo nombre (identificador) y a los 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 elemento individual se utilizan **índices**, que indican la posición del elemento dentro de la colección.
  - Los elementos están **ordenados**, esto es, se pueden identificar por su posición.
  - Cada elemento contiene un **valor** o dato.

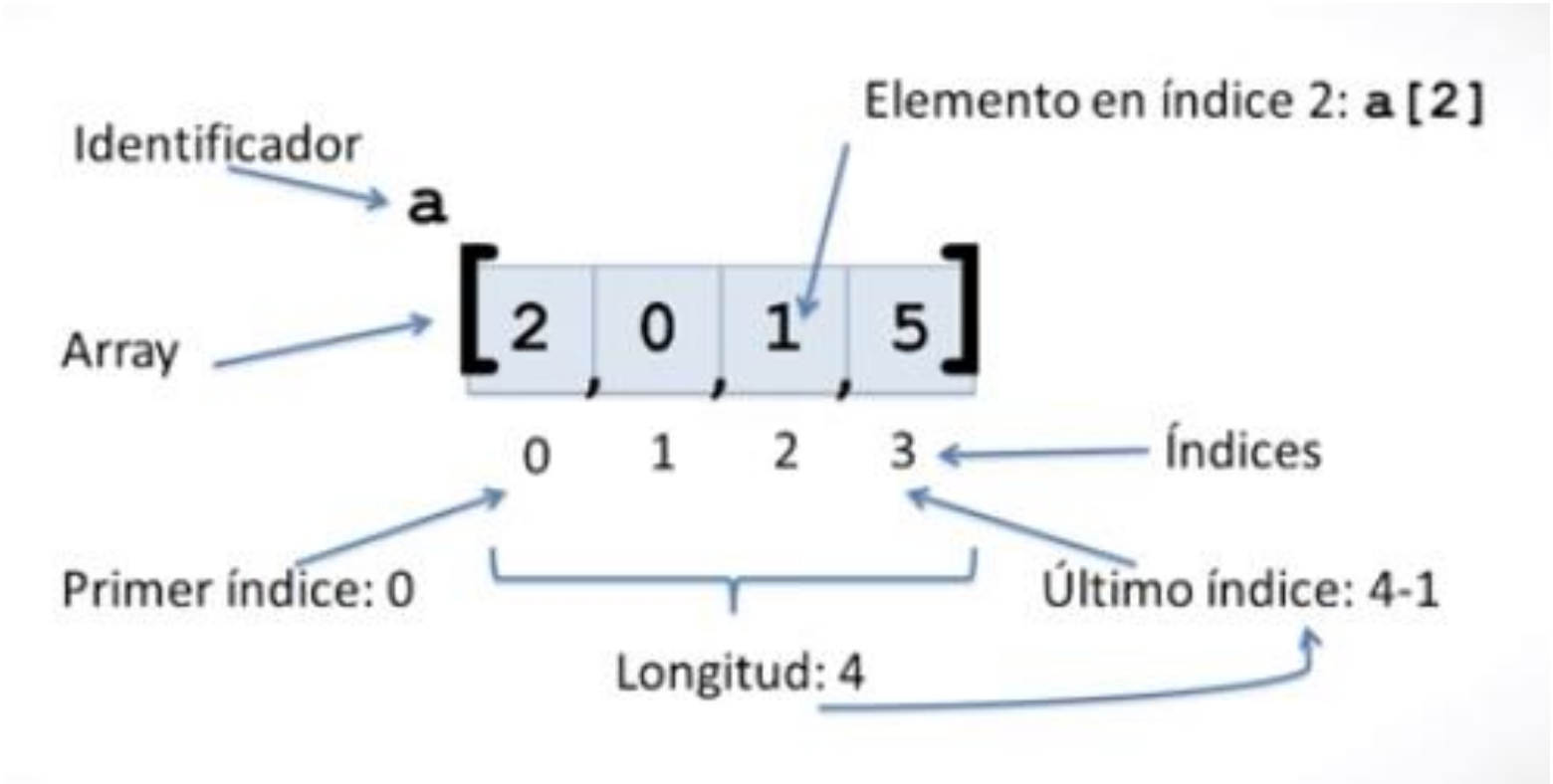
ARRAYS

- Los arrays son estructuras de **acceso directo**, ya que permiten almacenar y recuperar directamente los datos, especificando su **posición** dentro de la estructura.



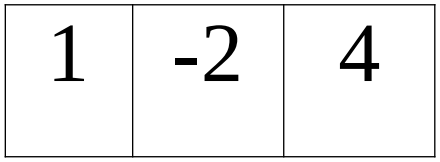
- Los arrays son estructuras de datos homogéneas: sus elementos son **TODOS** del **MISMO TIPO**.
- El **tamaño** de un array se establece de forma **FIJA**, en un programa, cuando se define una variable de este tipo.
- Es un estructura estática.

ARRAYS


Identificador
a
Elemento en índice 2: a [2]
Array
[2, 0, 1, 5]
0 1 2 3
Índices
Primer índice: 0
Último índice: 4-1
Longitud: 4

ARRAYS

- **Ejemplos:**



elVector [1]

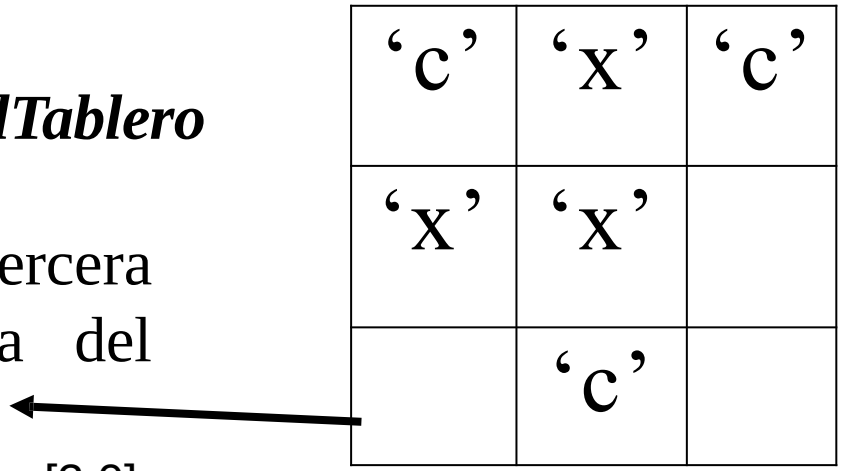
el array se denomina: ***elVector***

el elemento que ocupa la **segunda posición** del array ***elVector***

---

el array se denomina: ***elTablero***

elemento que ocupa la tercera fila y primera columna del array ***elTablero***



elTablero [2,0]

ARRAYS UNIDIMENSIONALES O VECTORES

- Un array es **unidimensional** si a cada elemento se accede mediante un único índice. Es una secuencia ordenada.
  - Ejemplo: vector

2	-5	4

Posición o número de orden

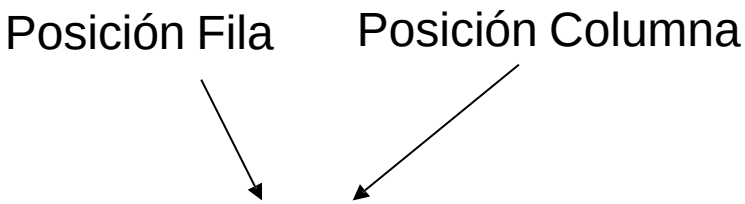
vector[0] contiene el 2
vector[1] contiene el -5
vector[2] contiene el 4

vector fila **vector columna**

$F = \{ 9 \quad -3 \quad 4.5 \quad 10 \quad -190 \}$ $V = \begin{Bmatrix} 3 \\ 4 \\ 9.5 \\ -12 \\ 89 \\ -222 \end{Bmatrix}$

ARRAYS BIDIMENSIONALES o MATRIZ

- Un array es **bidimensional** si a cada elemento se accede mediante 2 índices.
  - Ejemplo: `Dimension matriz [ 3 , 3]`



matriz[0,0] contiene ‘c’

matriz[0,1] contiene ‘x’

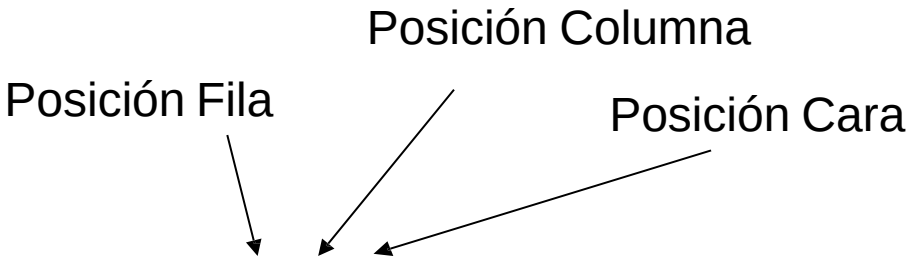
…

matriz[2,1] contiene ‘c’

‘c’	‘x’	‘c’
‘x’	‘x’	
	‘c’	

ARRAYS MULTIDIMENSIONALES

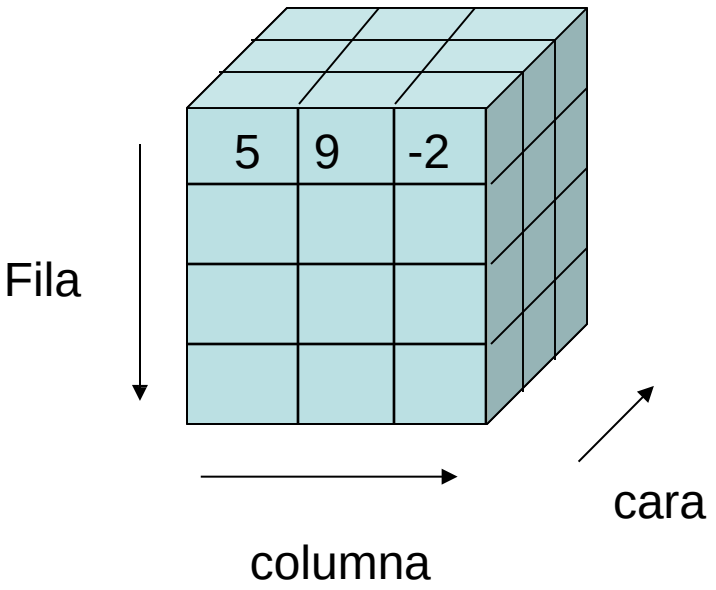
- Un array es **multidimensional** si a cada elemento se accede mediante 3 o más índices.
  - Ejemplo: cubo



cubo[0,0,0] contiene 5

cubo[0,1,0] contiene 9

cubo[0,2,0] contiene -2



DECLARACIÓN

ARRAY MULTIDIMENSIONAL

- La notación usada es:

```
Definir <identificador> Como <tipo>;
Dimension <identificador> [<maxl>,...,<maxN>];
```

  – Donde:

  <*identificador*> es el identificador del tipo array.

  Los *N* parámetros indican la cantidad de *dimensiones* y el *valor máximo de cada una de ellas*. La cantidad de dimensiones puede ser una o más, y la **máxima cantidad de elementos** debe ser una **expresión numérica positiva**.

  En perfil estricto (PseInt) no se permite utilizar variables para dimensionar arrays.

  <*tipo*> es el tipo de datos de los elementos del array.

DECLARACIÓN

- Ejemplo:

El índice del ejemplo comienza en 1 (base 1) , pero en la asignatura emplearemos como primer índice 0 (base 0).

En algunos lenguajes de programación comienza en 1.

```
Proceso Prueba
        Definir num como entero;
        Dimension num[5];
        num[1]=5;
        num[2]=10;
        num[3]=15;
        num[4]=20;
        num[5]=25;
        Para i<-1 Hasta 5 Con Paso 1 Hacer
           escribir num[i];
        Fin Para
FinProceso
```

PSeInt permite que el índice comience en 1, en lugar de 0, tal y como se muestra en el ejemplo.

Pero, **para la asignatura, se empleará índice 0, que es el que se emplea en el perfil estricto definido en PSeInt.**

OPERACIONES CON ARRAYS

- **Operaciones sobre elementos:**
  - Asignación.
  - Lectura
  - Escritura

- **Operaciones sobre el array completo:**
  - Recorrido
  - Actualización (añadir, borrar, insertar)
  - Ordenación
  - Búsqueda

OPERACIONES CON ARRAYS

- ASIGNACIÓN

```
vectorA[0] <- 12;
vectorB[2] <- a + 3;
vectorC[4] <- VectorD[2] + 7;
```

- ACCESO A UN ELEMENTO
  - Para acceder a un elemento hay que especificar el nombre de la variable array seguida de la posición (índices) entre corchetes.

```
vectorA[posicion]
matriz[fila, columna]
```

OPERACIONES CON ARRAYS

- ENTRADA:
  - Mediante asignación:

```
vectorA[1] ← 100  |  matriz[2,0] ← 3.14
```

  - Mediante lectura:

```
Leer vectorA[5]   |   Leer matriz[5,2]
```

  - Todos los elementos mediante lectura:

```
Para i<-0 hasta 9 Hacer  | Para i <- 0 hasta 4 Hacer
   Leer vectorA[i];      |   Para j <-0 Hasta 1 Hacer
FinPara                  |      Leer matriz[i,j];
                         |   FinPara
                         | FinPara
```

  - Todos los elementos mediante asignación:

```
Para i<-0 Hasta 9 Hacer  | Para i<- 0 hasta 4 Hacer
   vectorA[i] ← 0;       |   Para j <- 0 Hasta 1 Hacer
FinPara                  |      matriz[i,j] ← 0.0;
                         |   FinPara
                         | FinPara
```

OPERACIONES CON ARRAYS

- SALIDA:
  - De un elemento:

```
Escribir 'Componente 1=', vectorA[1];
Escribir 'Componente 1,4=', matriz[1, 4];
```

  - De todos los elementos:

```
Para i<-0 Hasta 9 Hacer
   Escribir 'componente ',i,'=',vectorA[i];
FinPara

Para i<-0 Hasta 4 Hacer
   Para j<-0 Hasta 1 Hacer
      Escribir 'matriz[', i,',', j,']', matriz[i,j];
   FinPara
FinPara
```

- Como ocurre con todos los tipos compuestos, **NO** se pueden realizar operaciones de entrada/salida **con arrays completos.**

OPERACIONES CON ARRAYS

- RECORRIDO: Consiste en realizar una **acción general en todos** los elementos del array. La acción puede ser: inicialización, consulta o actualización del valor.
  - Ej: Lectura/escritura

```
Para i<-limInf Hasta limSup Hacer
   Leer vectorA[i]; // Escribir vectorA[i];
FinPara
//INICIALIZACIÓN O ACTUALIZACIÓN
Para i<-limInf Hasta limSup Hacer
   vectorA[i] ← valorNuevo;
FinPara
```

OPERACIONES CON ARRAYS

- RECORRIDO PARA MATRIZ:
  - Ej: Lectura/escritura

```
Para i<-limInf1 Hasta limSup1 Hacer
   Para j<-limInf2 Hasta limSup2 Hacer
     Leer matriz[i,j]; //Escribir matriz[i,j];
   FinPara
FinPara
 //INICIALIZACIÓN O ACTUALIZACIÓN
 Para i<-limInf1 Hasta limSup1 Hacer
    Para j<-limInf2 Hasta limSup2 Hacer
       matriz[i,j] ← valorNuevo;
    FinPara
 FinPara
```

Ejercicios nº 12, 13 y 14

Ejercicio nº 15 para casa

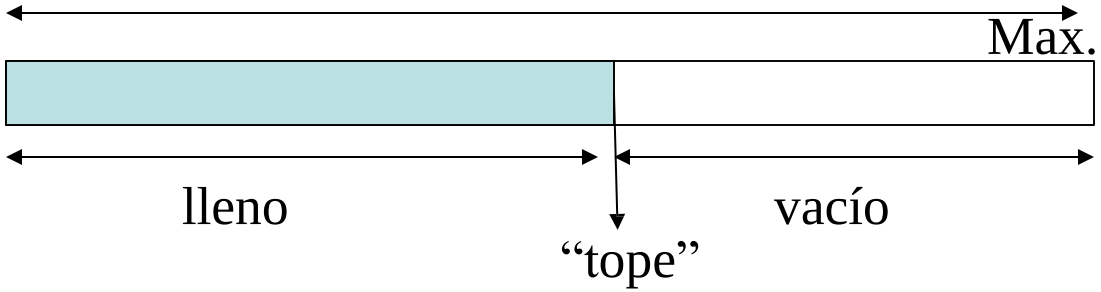
ARRAYS PARCIALMENTE LLENOS

- La memoria correspondiente a un array se declara estáticamente (tiene un número fijo de elementos).
  - 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.**

- **Array dinámico:** Asignación de espacio de tamaño fijo dividido en dos partes: la primera es la parte ocupada de del array y la segunda está libre para su crecimiento.
- **NO hay que olvidar NUNCA** que es una **estructura estática** y tiene un **tamaño fijo.**

ARRAYS PARCIALMENTE LLENOS

- 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 elementos del array con datos. Es la capacidad dinámica.



  - O bien rellenar con datos de fácil identificación, para conocer de esta forma si el elemento está vacío. Por ejemplo: En un vector de enteros positivos, rellenar con valor “-1” los elementos vacíos.

OPERACIONES CON VECTORES

- AÑADIR DATOS. Consiste en adicionar un nuevo elemento a partir del último.
  - Requisito previo: Que el array no esté lleno.
    - **Ejemplo:** Sea el vectorA que puede albergar 10 elementos y solo tiene 7 rellenos (tope=7); vectorA[0], vectorA[2], … , vectorA[6]. Podremos **añadir hasta 3 elementos nuevos**.

      Bastaría darles valor, mediante asignaciones o lectura e incrementar tope hasta 10:

```
vectorA[7] ← 15          tope ← 10

vectorA[8] ← 5
vectorA[9] ←  -9
```

OPERACIONES CON VECTORES

- INSERTAR DATOS. Consiste en introducir un nuevo elemento.
  - Requisito previo: Que el array no esté lleno.
    - **Ejemplo:** Sea el vectorA con 7 elementos con datos (tope=7); vectorA[0], vectorA[2], … , vectorA[6]. Podremos **insertar un elemento (-3) en la posición 5 (índice 4).**

      Habría que:

      - Desplazar los elementos vectorA[4], vectorA[5], vectorA[6] una posición hacia arriba.
      - Sobreescribir vectorA[4] ← -3
      - Incrementar tope en una unidad

Diferencia posición e índice

OPERACIONES CON VECTORES

- INSERTAR DATOS.

  Ejemplo:

```
posicion ← 5    // índice = 4
i ← tope
Mientras i >= posicion Hacer
    // desplaza los elementos superiores
    vectorA[i] ←   vectorA[i-1]
    i ← i-1 //decrementa el contador
FinMientras
// sobrescribe (inserta) el elemento
vectorA[posicion-1] ← -3
tope ←  tope + 1
```

OPERACIONES CON VECTORES

- BORRAR DATOS. Consiste en suprimir un elemento existente.
  - **Ejemplo:** Sea el vectorA con 7 elementos (tope=7); vectorA[0], vectorA[2], … , vectorA[6]. Podremos **borrar el elemento de la posición 2 (índice 1)**.

    Habría que:

    - Desplazar los elementos vectorA[2], vectorA[3], .., vectorA[6] una posición hacía abajo.
    - Decrementar tope en una unidad.

OPERACIONES CON VECTORES

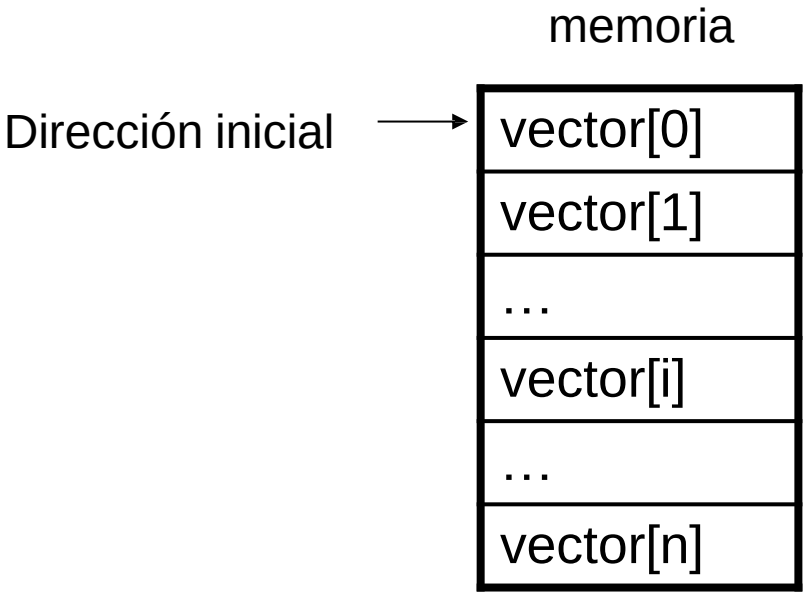
- BORRAR DATOS.

  Ejemplo:

```
posicion ← 2    // índice = 1
i ← posicion
Mientras i < tope Hacer
   // desplaza los elementos superiores
   vectorA[i-1] ← vectorA[i]
   i ← i+1 //incrementa el contador
FinMientras
// decrementa el tope
tope ← tope - 1
```

ALMACENAMIENTO DE ARRAYS EN MEMORIA

- **Array unidimensional**
  - Se almacenan, a partir de una dirección de memoria, todos los elementos seguidos.



vector[0] se encuentra en la **dirección_inicial**.

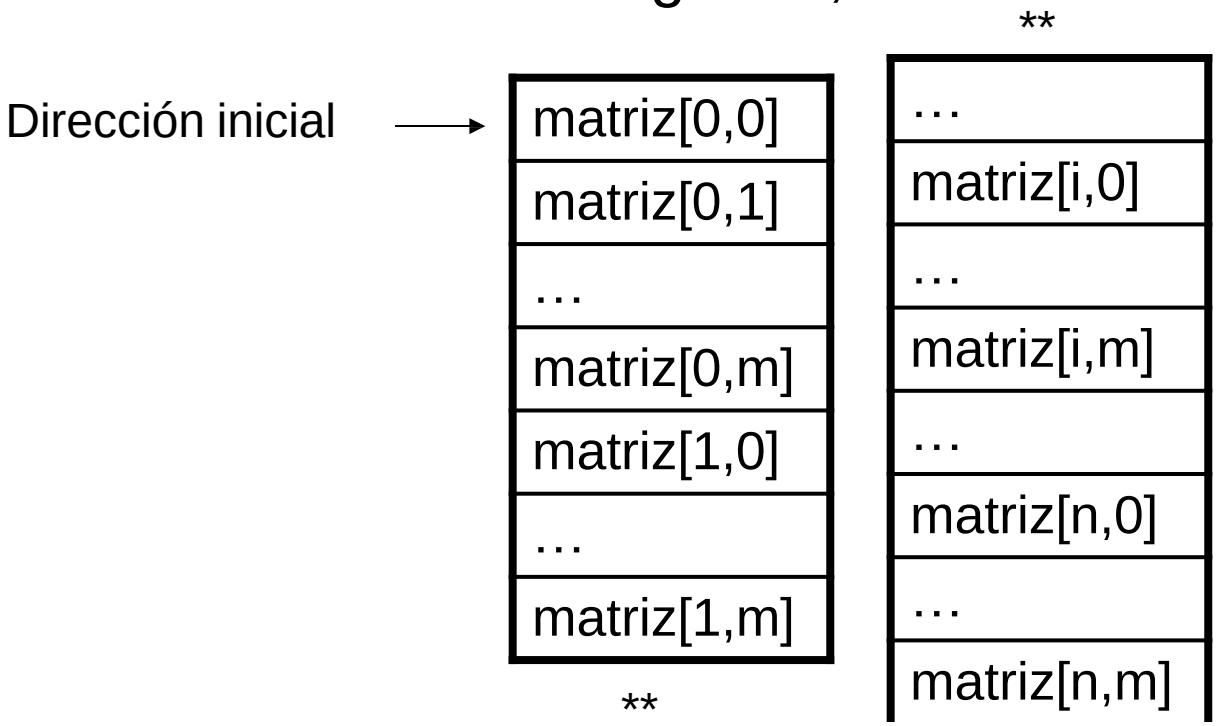
vector[1] se encuentra en la dirección dada por: **direccion_inicial + S** siendo S el tamaño en bytes de un elemento.

vector[i] se encuentra en: **dirección_inicial + i * S**

ALMACENAMIENTO DE ARRAYS EN MEMORIA

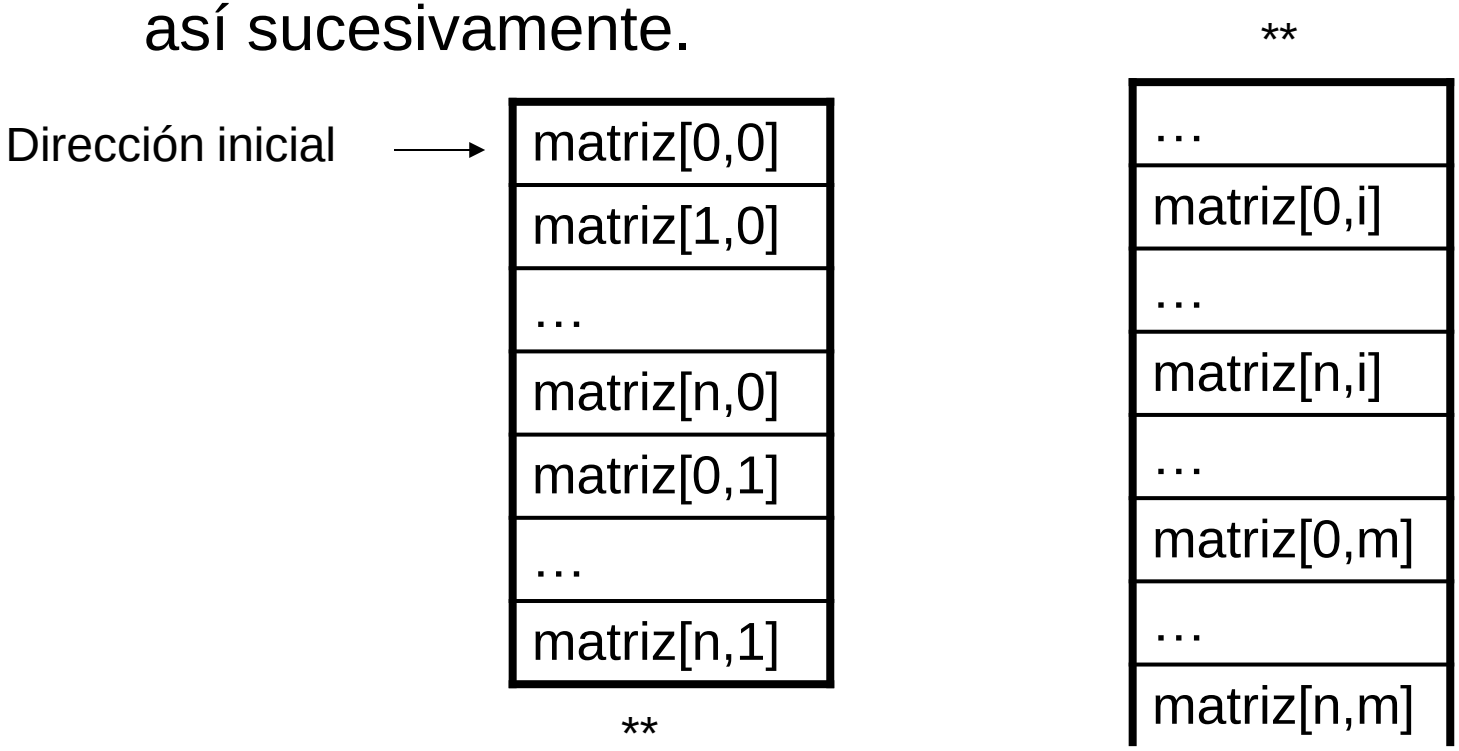
- **Array bidimensional (N filas y M columnas)**

  – ALMACENAMIENTO POR FILA. Se almacenan todos los elementos seguidos de la primera fila, a continuación los de la segunda, así sucesivamente.



ALMACENAMIENTO DE ARRAYS EN MEMORIA

- **Array bidimensional (N filas y M columnas)**
  - ALMACENAMIENTO POR COLUMNA. Se almacena todos los elementos seguidos de la primera columna, a continuación los de la segunda, así sucesivamente.



Ejercicio de operaciones con vectores parcialmente llenos