Estructuras de datos: Arrays

Unidimensionales: **Vectores**
Multidimensionales: **Matrices y Cubos**

1. Introducción.

Los tipos de datos primitivos, implican unidades simples disponibles en todos los lenguajes de programación, estos son: enteros, reales, carácter y lógicos. Los tipos de datos compuestos por **contraste**, están construidos sobre la base de los tipos de datos primitivos, un ejemplo representativo es el tipo String o cadena de caracteres.

2. Clasificación:

Los tipos de datos primitivos están organizados en estructuras de datos. Estas a su vez están clasificadas como: Estáticas y Dinámicas

- **Datos Primitivos o simples**
  - Estándar
    - Integer (entero o punto fijo)
    - Real (real o punto flotante)
    - Character (carácter)
    - Boolean (lógico)
  - Definido por el programador (no estándar)
    - Enumerated (enumerativo)
- **Datos estructurados**
  - Estáticos
    - Arrays (vector / matriz /cubos)
    - String (cadena)
    - Record (registro)
    - File (archivo)
  - Dinámicos
    - Listas enlazadas: Pilas, Colas
    - Arboles y Grafos.

2.1. Las Estructuras de datos estáticas.

Se denominan estructuras estáticas por el hecho de **definir el tamaño ocupado en memoria antes que el programa se ejecute**, y en consecuencia no se puede redimensionar o modificar dicho tamaño en tiempo de ejecución.

Estas estructuras, están implementadas en los lenguajes como: **arrays (**vectores, matrices y cubos**), registros y ficheros.**

2.2. Las Estructuras de datos dinámicas.

No presentan las limitaciones del tamaño de memoria ocupada o reservada para manipular conjunto de datos. Estas implementaciones se basan en el uso de un tipo de dato especial denominado: **puntero**.

Las estructuras dinámicas soportadas por la mayoría de los lenguajes son: las Listas enlazadas (como las pilas y colas), los árboles y grafos.

Una característica importante que diferencia a **los tipos de datos** es el siguiente:

Los **tipos primitivos** o intrínsecos definen variables simples y representan a un único elemento. Son proporcionados por el sistema (**Entero, Real, carácter y lógico).**

Los **tipos de datos estructurados** son conjuntos de variables que representan mediante un identificador (nombre) a *múltiples datos* o *elementos*, pudiendo cada uno de estos elementos, ser referenciado independientemente

3. Los Arrays.

3.1. Concepto.

IMPORTANTE

En matemática como en computación a los **arrays** se les conoce como **vectores, matrices**, pero no son equivalentes, sin embargo guardan algunas similitudes de estructura

Un array (arreglo) es un **"conjunto finito y ordenado de elementos homogéneos".**

La propiedad **finito** se deduce por ser *estático*, mientras que la propiedad **ordenado** implica que sus elementos, primero, segundo, enésimo del array, pueden ser *identificados* mediante un índice. Por ultimo, son **homogéneos** por estar compuestos por elementos del mismo *tipo*. (Elementos de tipo cadena, o todos de tipo entero, etc).

3.2. Clasificaron de Arrays. Los array pueden agruparse según su dimensión en:

- Unidimensionales o **Vectores**
- Multidimensionales o **Matrices**
  - Bidimensionales o tablas
  - Tridimensionales o cubos, etc.

4. Array Unidimensional: **Los Vectores.**

Son matrices de una sola dimensión. Un array unidimensional es denominado **vector**, y es el tipo más simple de array.

4.1. Notaciones**.** Los vectores, tienen las siguientes notaciones posibles.

```
a[1], a[2], a[3], …,  a[i], ..., a[n]
```

Observe que el vector tiene un nombre o identificador único y el índice de un elemento: [1], [2], [3], , [i], [n] determina el orden o posición en el vector. Este índice permite el acceso al valor del elemento**.**

IMPORTANTE: Los array en la mayoría de los lenguajes modernos, los primeros elementos se inican en uno (1) o en cero (0). Razón por la cual se les denomina arrays base 0 y arrays base 1 respectivamente. Aunque técnicamente el primer elemento puede iniciarse en cualquier valor.
En la notación algorítmica, preferentemente emplearemos base 1 por ser más intuitiva y natural.

4.2. Notación algorítmica.

Definición de tipo

tipo
array*[dimensiones]* ***de*** *<tipo_dato>: <identificador_tipo_array>*

Declaración de tipo

<idenfificador_tipo_array>: lista_de_variables

VECTORES

Ejemplos de definición y declaración de arrays unidimensionales.

Definicion 1. Definir estructura tipo array para almacenar las notas de 20 alumnos.

```
tipo
      array[1..20] de entero: arrNotas
var
      arrNotas:  notas
```

	[1]	[2]	[3]	…	[i]	…	…	[18]	[19]	[20]
nota	15	17	14					13	18	16

Significa que notas de tipo `arrNotas` es un vector de 20 elementos base uno (1 a 20) de tipo entero. El acceso a cada elemento se denota: nota[1], nota[2], nota[3], nota[4], …, nota[20]

Definicion 2. Array o vector para almacenar las temperaturas medias de la semana.

```
tipo
      array[0..6] de real: vTemp
var
      vTemp:  t
```

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]
22.4	22.7	22.3	21.8	21.9	22.6	23.0

Significa que t de tipo `vTemp` es un vector base cero de 7 elementos (0 a 6) de tipo real, donde:

```
t[i]  esta en el rango  0 <=  i  < = 6
```

4.3. Rango del vector.

El número de elementos de un vector se denomina rango del vector. El valor mínimo permitido de un vector se denomina ***limite inferior*** del vector (lower: L) y el valor máximo permitido se denomina ***limite superior*** (upper: U). Así,

El rango del vector **a[L : U]** es **(U – L) + 1** (Elementos de una sucesión)

Así, el vector ventaMes[1:30] tiene rango: 30 – 1 + 1 (o también 30 elementos)
y el vector *clientesVisa*[0:50] tiene rango: 50 – 0 + 1 (o también 51 elementos).

En general:

- El rango del vector en **X[1 : n]** es **n** (array unidimensional base 1)
- El rango del vector en **X[0 : n]** es **n +1** (array unidimensional base 0)

Conclusiones:

- Un vector es una secuencia ordenada de elementos como:

  `x[L]  x[L+1]  x[L+2]  …  …  x[i]  …  …  x[U-1]  x[U]`

  Donde cada elemento x[i], L <= i <= U y es de tipo de datos T

- El limite inferior no tiene por que comenzar en uno. El índice del primer elemento de un vector puede ser 1 ò 0 según el lenguaje de programación, siendo el mas común el segundo (array base cero).

  `Así: el vector a[0:6] donde el rango es: 6 – 0 + 1 = 7 elementos`

  `a[0]  a[1]  a[2]  a[3]  a[4]  a[5]  a[6]`

- Un array es una estructura ordenada, y el acceso a sus elementos se realiza siempre mediante el subíndice, definido como un entero (positivo / negativo).

  `Así: el vector m[-3:3] donde el rango es: 3 – (-3) + 1 = 7 elementos`

  `m[-3]  m[-2]  m[-1]  m[0]  m[1]  m[2]  m[3]`

- Los subíndices de un vector pueden ser enteros, variables o expresiones enteras.
  Asi: sea el vector v base cero de rango 5 (cinco elementos) y tipo T: real

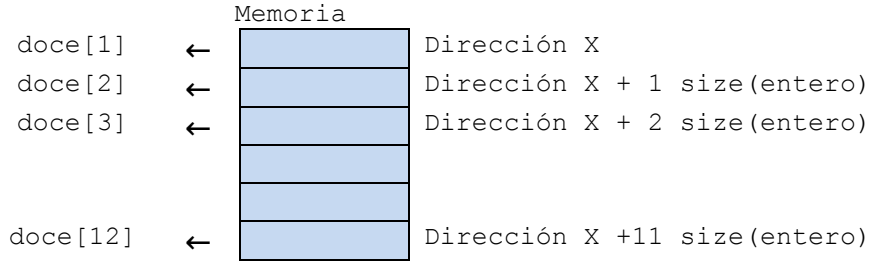
v[0]	v[1]	v[2]	v[3]	v[4]
13.4	-7	15.0	0.9	22

Si entero: i ← 2

`v[i+1]` representa el elemento v[3] de valor 0.9
`v[i+2]` representa el elemento v[4] de valor 22
`v[i-2]` representa el elemento v[0] de valor 13.4

- Los vectores se almacenan en memoria RAM en un orden adyacente. Según esto, un vector de 12 elementos denominado vDocena, estará representado por 12 posiciones de memoria consecutiva.

```
tipo
      array[1..12] de entero: vDocena
var
      vDocena: doce
```



- **CADENAS.** Las cadenas de caracteres son un tipo especial de vector en los que cada elemento es un carácter (ASCII / UNICODE). Este tipo de estructura se suele emplear para almacenar cualquier tipo de texto. En algunos lenguajes de programación (C++ y C ANSI) emplean – internamente – el carácter especial ASCII (0) o null para marcar el fin de la cadena de caracteres.
  OBSERVACION: A diferencia de los array numéricos, las cadenas o arrays de caracteres, suelen ser administradas mediante funciones de tratamiento de cadenas, proporcionados por los lenguajes (librerías).

5. Operaciones con vectores.

Las operaciones a las que pueden ser sometidos los arrays durante el proceso de solución de algoritmos, son:

- Asignación/ Inicializacion
- Acceso secuencial (recorrido):
  - Operación de Lectura
  - Operación de Escritura
- Operaciones de Actualización:
  - Añadir
  - Borrar
  - Insertar
- Operaciones de Ordenación.
- Operaciones de Búsqueda.

IMPORTANTE

Operaciones con Arrays.

En general las operaciones implican el tratamiento de los elementos uno a uno del array.

5.1 Asignación - Inicializacion.

La asignación de valores a los elementos de un vector se realiza con la instrucción de asignación. Asi:

Sean los vectores a y b de tipo array real y entero, se pueden inicializar respectivamente:

a[1:6] ← { I_1, I_2 , I_3, …, I_6 }, con 6 elementos I de tipo real
b[]← { I_1, I_2, I_3, I_4, I_5}, con 5 elementos I de tipo entero

Se puede realizar las operaciones de asignación:

```
a[4] ← 16.2 asigna el valor 16.2 al elemento 4 del vector A.
a[6] ← 18.5 asigna el valor 18.5 al elemento 6 del vector A.
```

O También:

```
b[1] ←  33
b[2] ←  B[1] + 16
b[5] ←  B[2] – B[1]
```

IMPORTANTE

Operaciones de recorrido de un array. Si necesita asignar valores a todos los elementos de un vector de rango amplio, implemente algoritmos con estructuras de control: **repetitivas** o **iterativas** (operaciones de recorrido).

5.2 Acceso secuencial al vector (recorrido).

El acceso a todos los elementos de un vector para lectura/escritura de datos en el, se implementa mediante un proceso denominado *recorrido del vector,* y para ello se usa estructuras repetitivas o bucles (cuyas contadores se usan como subíndices del vector para acceder a sus elementos). El incremento del contador del bucle producirá el tratamiento sucesivo de los elementos del vector.

5.2.1 Lectura / Escritura de datos en arrays.

Las operaciones de entrada / salida de datos en arrays normalmente se efectúan con iteraciones o loops, e incluso estructuras de control selectivas. Estas operaciones se representan mediante las instrucciones:

Operación	Sintaxis	Descripción
Recorrido del vector	**leerArray**(A) **escribirArray**(A)	lectura del vector A escritura del vector A
Lectura de un elemento Escritura de un elemento	**leer**(V[4]) **escribir**(V[4])	leer el elemento v[4] del vector V escribir el elemento v[4] del vector V

a.) Operación de Lectura: El Algoritmo de entrada de datos en arrays se representa Asi.

```
Sea el vector a[1:5] = { a[i] }

La operación de lectura se denota:
      leerArray(a)

Y la asignación de elementos:
      leer(a[i])
```

```
(Técnica iterativa: bucle for)
desde( i  ← 1 hasta 5 )hacer
      //asignar desde el teclado
      leer(a[i])
fin_desde
```

O también

```
(Técnica iterativa: bucle while)
i ← 1
mientras( i <= 5 )hacer
      leer(a[i])
      i ← i + 1 //incremento
fin_mientras
```

b.) Operación de Escritura: El Algoritmo de salida de datos desde arrays se representa Asi.

```
Sea el vector b[1:5] = { b[i] }

La operación de escritura se denota:
      escribirArray(b)

y la escritura de ítems:
      escribir(b[i])
```

```
(Técnica iterativa: bucle for)
desde( i ← 1 hasta 5 )hacer
      //escribir elementos
      escribir(b[i])
fin_desde
```

O también

```
(Técnica iterativa: bucle while)
i ← 1
mientras ( i <= 5 ) hacer
//escribir elementos I
      escribir(b[i])
      i ← i + 1 //incremento
fin mientras
```