

# Programación II

Vectores y matrices

# Estructura de datos

Así como podemos clasificar a las variables por su tipo de dato. También podemos clasificarlas por su dimensión. Un ejemplo de esta clasificación podría ser:



## Variables simples

Variables que aceptan un valor a la vez. También conocidas como escalares.



## Objetos

Instancias de clases. Pueden almacenar varios atributos y además poseen comportamiento.



## Arrays

Variables que aceptan varios valores a la vez. Se conocen como **vectores** a los arrays unidimensionales y **matrices** a los arrays multidimensionales

# Vectores

Una estructura de datos que admite varios valores a la vez dentro de una misma variable. Las reglas a seguir para utilizar vectores son las siguientes:

- **Tamaño conocido:** Para poder crear un vector es necesario poder saber su tamaño de antemano.
- **Indexación base-0:** El primer elemento del vector se encuentra en la posición 0 del mismo. El índice debe ser un número natural ( $N_0$ ).
- **Elementos homogéneos:** Todos los elementos del vector deben pertenecer al mismo tipo de dato. Es decir, podemos tener un vector de enteros, de flotantes, de booleanos, etc.

# Declaración

```
int mi_vector[10];
```

Para **declarar un vector**, debemos indicar el tipo de dato al que pertenecerán los elementos del mismo, luego el nombre del vector y entre corchetes el tamaño del vector.

El **tamaño** del vector debe ser un número natural mayor a cero y constante. Es decir, no puede ser a partir de una variable.

# Declaración

```
const int TAM = 40;  
float mi_vector[TAM];
```

En esta declaración, el **vector** *mi\_vector* de tipo float tiene un tamaño de *TAM* elementos. Al compilar el programa, el vector se creará de 40 elementos que es el valor que contiene la **constante** *TAM*.

# Acceso a elementos del vector

```
int pos = 6;  
int vec[8];  
cout << vec[2];  
cin >> vec[4]  
cout << vec[pos];
```

Código C/C++

Para acceder a un elemento del vector, basta con indicar la posición del elemento entre corchetes. Recordar que el primer elemento comienza en 0 y, por lo tanto, el último será la posición *tamaño-1*

0	
1	
2	
3	
4	
5	
6	
7	

Representación del vector `vec[8]` que almacena 8 números enteros.

# Acceso a elementos del vector

→

```
int pos = 6;  
int vec[8];  
cout << vec[2];  
cin >> vec[4]  
vec[pos] = 100;
```

Código C/C++

→

0	
1	
2	
3	
4	
5	
6	
7	

Representación del vector `vec[8]` que  
almacena 8 números enteros.

# Acceso a elementos del vector

```
→ int pos = 6;  
   int vec[8];  
   cout << vec[2];  
   cin >> vec[4]  
   vec[pos] = 100;
```

Código C/C++

→

0	
1	
2	
3	
4	
5	
6	
7	

Representación del vector `vec[8]` que  
almacena 8 números enteros.



# Acceso a elementos del vector

```
→ int pos = 6;  
   int vec[8];  
   cout << vec[2];  
   cin >> vec[4]  
   vec[pos] = 100;
```

Código C/C++

0	
1	
2	
3	
4	
5	
6	100
7	

→ Representación del vector `vec[8]` que almacena 8 números enteros.

# Acceso a elementos del vector

```
int valor, valor_1, valor_2;  
int vec[4] = {10, 20, 30, 40};  
  
valor = vec[1];  
valor_1 = vec[0] * vec[3];  
if (vec[0] > 0){  
    valor_2 = vec[0];  
}  
vec[1]++;
```

Código C/C++

0	
1	
2	
3	

vec[4]

valor	
valor_1	
valor_2	

# Acceso a elementos del vector

```
int valor, valor_1, valor_2;  
int vec[4] = {10, 20, 30, 40};  
  
valor = vec[1];  
valor_1 = vec[0] * vec[3];  
if (vec[0] > 0){  
    valor_2 = vec[0];  
}  
vec[1]++;
```

Código C/C++

0	10
1	21
2	30
3	40

vec[4]

valor	20
valor_1	400
valor_2	10

# Inicializar elementos

Al declarar un vector, todos sus elementos contendrán basura. Si necesitamos que el mismo contenga algún valor en todos sus elementos. Se puede hacer de las siguientes maneras:

```
int i;  
int vec[100];  
  
for (i=0; i<100; i++){  
    vec[i] = 0;  
}
```

Código C/C++ que asigna cero a todos los elementos del vector vec.

```
int vec[100] = {};
```

Código C/C++ que inicializa el vector vec con todos sus elementos en cero.

# Matrices

Una estructura de datos que admite varios valores a la vez dentro de una misma variable pero organizados en varias dimensiones. Siguen las mismas reglas a los vectores ya que, por definición, lo son.

- **Tamaño conocido:** Para poder crear un vector es necesario poder saber su tamaño de antemano.
- **Indexación base-0:** El primer elemento del vector se encuentra en la posición 0 del mismo. El índice debe ser un número natural ( $N_0$ ).
- **Elementos homogéneos:** Todos los elementos del vector deben pertenecer al mismo tipo de dato. Es decir, podemos tener un vector de enteros, de flotantes, de booleanos, etc.

# Declaración

```
int mi_matriz[12][31];
```

```
float mi_otra_matriz[12][5][10];
```

Para **declarar una matriz**, debemos indicar el tipo de dato al que pertenecerán los elementos del mismo, luego el nombre de la matriz y entre corchetes el tamaño de cada dimensión de la misma.

El **tamaño** de la dimensión debe ser un número natural mayor a cero y constante. Es decir, no puede ser a partir de una variable.

# Acceso a elementos de una matriz

```
int mat[7][3] = {};  
mat[5][2] = 10;  
cin >> mat[2][1];  
cout << mat[6][0];
```

Código C/C++

Para acceder a un elemento de la matriz, basta con indicar la posición de las dimensiones entre corchetes. Recordar que el primer elemento en cada dimensión comienza en 0 y, por lo tanto, el último será la posición *tamaño-1*

	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	10
6	0	0	0

Representación de la matriz[7][3] que almacena 21 números enteros dispuestos en 7 filas y 3 columnas.

# Ejercicio

Un comercio dispone de 10 artículos y desea llevar un control de las ventas realizadas en sus 4 sucursales.

Cada venta se registra con los siguientes datos:

- Número de artículo (1 a 10)
- Sucursal (1 a 4)
- Monto de la venta

El proceso de carga finaliza cuando se ingresa un número de artículo igual a 0.

Se pide:

- 1) Calcular la recaudación total de cada artículo en cada sucursal.