

# **Cap. IV Arreglos**

**DEPARTAMENTO DE PROGRAMACIÓN DE  
COMPUTADORAS  
DESARROLLO DE SOFTWARE I**

**PROF. MITZI MURILLO DE VELÁSQUEZ Msc.**

# Arreglos



Hasta el momento sólo hemos visto variables simples que contienen solamente un valor en memoria.

Qué es un arreglo?

Conjunto de variables con un mismo nombre y el mismo tipo de datos pero con índices diferentes.

Ejemplo :

```
int pob;
```

```
int pob[10];
```

**Qué necesitamos para calcular la población de las 10 provincias de Panamá?**

# DEFINICIÓN DE ARREGLO



Es una estructura de datos **homogénea** formada por una colección **finita** de elementos, **todos** del mismo **tipo**, los cuales están almacenados **consecutivamente** en memoria y se referencia con un **nombre común**.

**Homogénea**, quiere decir que todos los datos son **del mismo tipo**.  
es decir, **si se define un arreglo de tipo entero, todos sus datos son de tipo entero**

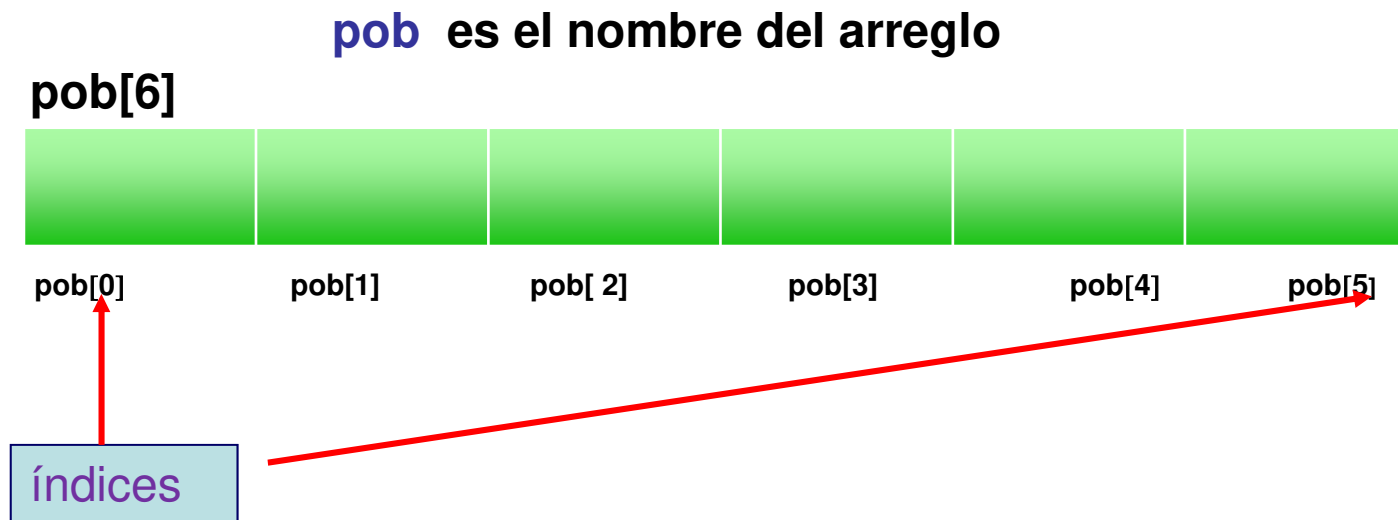
**finita**, significa que tienen un tamaño definido. Cuando se crea el arreglo se define el número de elementos.

**Consecutivamente**, indica que hay un primer elemento, un segundo elemento y así sucesivamente.

# REPRESENTACIÓN GRÁFICA



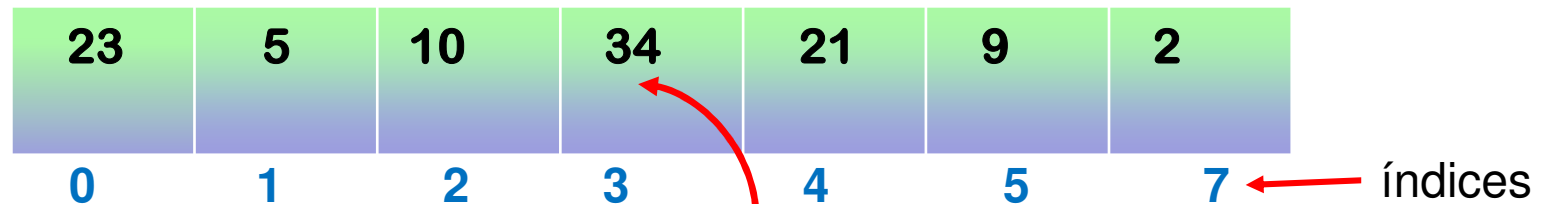
Dado que sus elementos se almacenan en forma consecutiva o secuencial, éste se puede representar como una tabla de n casillas.



# Cómo se accede a un arreglo

- Cada dato de la variable está en una **posición** determinada de la memoria, la cual está **numerada**.
- Para obtener el contenido de un elemento del arreglo se utiliza su posición relativa conocida como **índice**, el cual nos permite obtener el dato o contenido.

**Nombre del arreglo : tab [7];**



Para obtener el contenido **34** del arreglo **tab** , entonces colocamos :

**tab[3]**

nombre

índice

# Tipos de Arreglos

El número de índices determina la dimensionalidad del arreglo.

entero bat[4]; ← Arreglo unidimensional



entero mat[3][3]; Arreglo bidimensional

	columns		
filas	10	-3	4
	6	7	-2
	14	48	-33

entero cubo[3][3][3]; arreglo Tridimensional



# Tipos de Arreglos

## Arreglos Unidimensionales

- Es el arreglo de información que hace referencia a la posición lineal de los datos. Es decir, que un vector puede ser conceptualizado como un conjunto de datos que forman una lista, o una fila, o una columna.

**declaración del arreglo:**

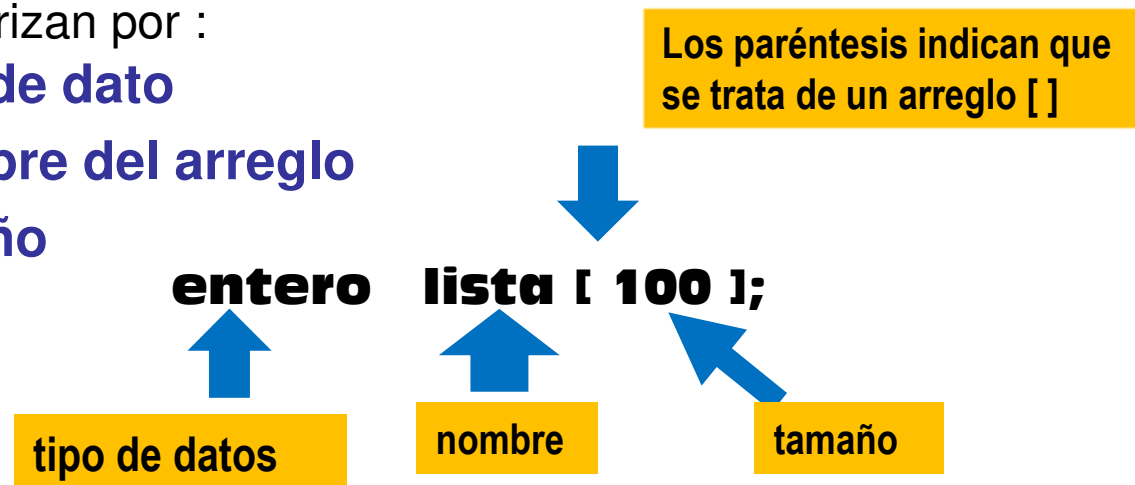
**[ tipo de datos nombre del arreglo (**tamaño**) ]**

# DECLARACIÓN DE UN ARREGLO

Los arreglos se declaran como las variables ordinarias, excepto que deben tener una especificación de tamaño para indicarle al compilador el número de elementos que va a tener.

Se Caracterizan por :

1. **tipo de dato**
2. **nombre del arreglo**
3. **tamaño**





# DECLARACIÓN DE UN ARREGLO

## Ejemplos:

**entero lista[100];**

- Indica al compilador la cantidad de almacenamiento de memoria necesario para guardar **100 elementos** de **tipo entero** para la variable **lista**.

**flotante salarios [40];**

- Reservamos espacio para **40 elementos** para la variable **salarios** de **tipo flotante**.

**Los arreglos se declaran para reservar espacio en memoria.**

# Operaciones Básicas con Arreglos

**ASIGNACIÓN:** La asignación de valores a un elemento del vector se realizará con el operador de asignación ( = ).

**entero tab[ 5];**

Se almacena el valor 45

en la primera posición (0) :

**tab[0] = 45;**

**tab**

0	0	0	0	0
0	1	2	3	4

**tab**

45	0	0	0	0
0	1	2	3	4

# Arreglos Unidimensionales

## Operaciones Básicas :

**entero tab[ 5];**

- Se almacena el valor 5  
en la segunda posición :

**tab [ 1] = 5;**

tab	45	5	0	0	0
	0	1	2	3	4

- Se almacena el valor de 12  
en la tercera posición

**tab[2] = 12;**

tab	45	5	12	0	0
	0	1	2	3	4

# Arreglos Unidimensionales

**ACCEDER:** El acceder al contenido de un elemento del vector se realizará con el operador de asignación. ( = )

`entero tab[ 5];`  
Se considera el valor 45  
en la primera posición (0)



`var = tab[0];`  
ó  
`var = tab[i];`

`tab`

45	5	12		
0	1	2	3	4

`entero var;`



Donde **i** debe tener el valor de la posición 0.

# Arreglos Unidimensionales

## Cómo hacemos para acceder al contenido de un arreglo ?

Cada posición del arreglo se toma como una variable, de esta manera se coloca el nombre del arreglo y entre paréntesis cuadrados la posición del elemento que queremos acceder.

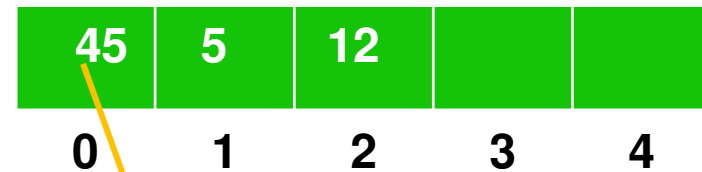
Si queremos acceder al contenido del primer elemento del vector

**var = tab[0];**

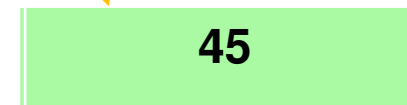
se usa el Operador =

**entero tab[ 5];**

**tab**



**entero var;**



**var**

# Arreglos Unidimensionales

**Operaciones aritméticas :** Se realizan igual que con las variables corrientes, sólo que se deben manejar con índices.

45	5	12	0	0
0	1	2	3	4

**vec**

```
entero vec[ 5];
```

```
entero suma;
```

- 1) suma = vec[0] + vec[1];
- 2) vec [2] = vec [4] + 2 ;
- 3) vec[3] = vec [1] + vec [4];
- 4) suma = suma + vec [3] + vec[0];
- 5) vec[0] = vec[3] \* 4;

# Arreglos Unidimensionales

Entre las operaciones más frecuentes con los arreglos tenemos:

- **Recorrer un arreglo:** no es más que utilizar un índice que se mueve desde el primero hasta el último elemento o dependiendo, del último al primer elemento.

- Las operaciones de asignación, de comparación se realizan elemento por elemento, para ello se utilizan ciclos.

**Ejemplo:** `para (i = 1 = 0; i < 9; i += 1)`  
    `{ si ( tab[i] > tab2[i] )`  
        ....procesos

El uso de ciclos de repetición facilita el recorrido de los arreglos . La forma más adecuada de manejar los arreglos es usando el **ciclo para**.

# Arreglos Unidimensionales

**Recorrer un arreglo** : no es más que utilizar un índice que se mueve desde el primero hasta el último elemento o dependiendo del último al primer elemento.

```
entero i;  
entero tabla[5];  
para ( i = 0; i<=4;i=i+1)  
{  
    tabla[i] = i * i;  
}
```

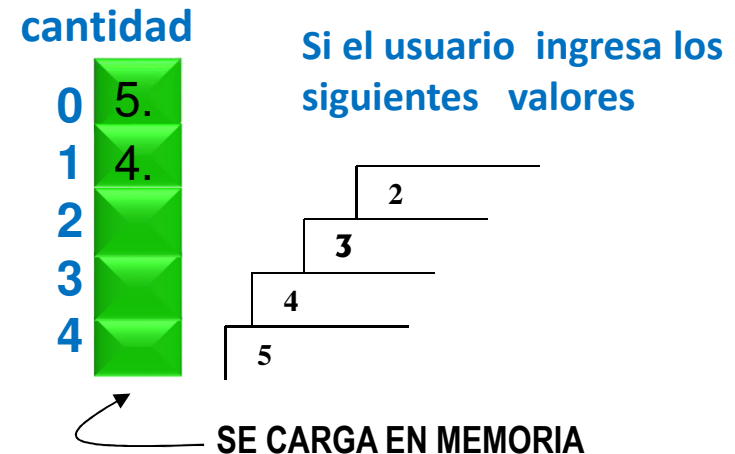


# Arreglos Unidimensionales

## 2. Cargar un arreglo o Lectura :

Permite introducir datos desde el teclado, leyendo el dato y accediendo a cada elemento del vector usando su nombre y el índice. También se **conoce** como **cargar** un vector desde el teclado.

```
para ( i = 0; i<=4;i=i+1) {  
    leer(dato);  
    cantidad[i] = dato;  
}
```



# Arreglos Unidimensionales

- **Imprimir el vector:** Significa que se va a acceder a los elementos de un vector para visualizar o mostrar su contenido.

## Imprimiendo el arreglo

```
para ( i = 0; i<=4;i=i+1) {  
    imprimir (cantidad[i] );  
}
```

**Imprime:** 5.0  
4.0  
3.0  
2.0  
1.0

cantidad	
0	5.0
1	4.0
2	3.0
3	2.0
4	1.0

Si nuestro arreglo está cargado con los datos:

La variable de control del ciclo se usa para identificar la posición de cada elemento del vector

El ciclo para facilita el el recorrido de los arreglos.

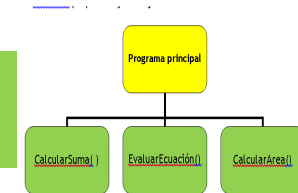
## Ejemplo no. 1

Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector.

### 1. ANÁLISIS Y DISEÑO

Entrada	5 números ( num), vect[5]
Proceso	Repetir 5 veces ( i ) leer y cargar vector ( <b>vect</b> )
Salida	vector cargado ( vect), no hay salida

# Ejemplo 1 . Cargar vector o tabla



Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector.

## 2. Algoritmo CargarVector {

```
// declarativas
entero num1, i;
entero vect[5];
```

*// Cargar vector*

```
para (i = 0; i<=4; i=i+1) {
    imprimir("Ingrese un número");
    leer(num1);
    vect[i] = num1;
}
}
```



## 3. PRUEBA DE ESCRITORIO

vect



variables de memoria			
i	num1	vect[i]	PANTALLA

## Ejemplo no. 1

Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector.

### Código en C

```
/* Cargar un vector de 5 elementos */
#include<iostream.h>
using namespace std;
int main()
{
    //Declarativas
    int vect[10]={0}, i,num=0;

    //Cargar vector
    for (i=0; i<=4; i+=1)
    {
        cout<<"\nIngrese un numero: ";
        cin>>num;
        vect[i] = num;
    }
    return 0;
}
```

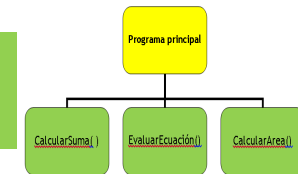
## Ejemplo no. 2

Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector. Luego imprime cada uno de los datos almacenados.

### 1. ANÁLISIS Y DISEÑO

Entrada	5 números ( num)
Proceso	Repetir 5 veces ( i ) leer y cargar vector ( <b>vect</b> ) Repetir 5 veces ( i ) mostrar datos vector ( <b>vect</b> )
Salida	vect[0 ] hasta vect[4]

## Ejemplo 2 . Imprimir vector o tabla



Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector. Luego imprime cada uno de los datos almacenados.

### 2.Algoritmo MostrarValor {

// declarativas

entero num1, i;

entero vect[5] ;

*// Cargar vector*

para (i = 0; i<=4; i=i+1) {

imprimir("Ingrese un numero");

leer(num1);

vect[i] = num1;

}

para (i = 0; i<=4; i=i+1) {

imprimir("Elemento: " , vect[i] );

}

}



### 3. PRUEBA DE ESCRITORIO

vect



variables de memoria			
i	num1	vect[i]	PANTALLA

## Ejemplo no. 2

Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector. Luego imprime cada uno de los datos almacenados.

### Código en C

```
/* Cargar un vector de 5 elementos e imprime cada uno de sus
elementos*/
#include<iostream.h>
using namespace std;
int main()
{
//Declarativas
int vect[10]={0}, i,num=0;
//Cargar vector
for (i=0; i<=4; i+=1){
    cout<<"\nIngrese un numero: ";
    cin>>num;
    vect[i] = num;
}
for (i=0; i<=4; i+=1){ // Imprimir contenido del vector
    cout<<"\n Elemento es: ";
    cout<<vect[i];
}
return 0;
}
```



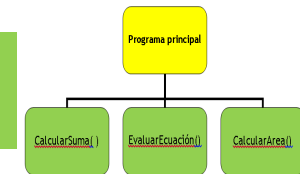
## Ejemplo no. 3 Sumar elementos de un vector

Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector. Posteriormente calcula la suma de los elementos e imprime el resultado.

### 1. ANÁLISIS Y DISEÑO

Entrada	5 números ( num) , vect [5]
Proceso	Repetir 5 veces ( i ) leer y cargar vector ( <b>vect</b> ) Repetir 5 veces ( i ) Acumular en (sum) = sum + vect[ i ]
Salida	sum

# Ejemplo 3. Sumar elementos del vector



Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector. Posteriormente calcula la suma de los elementos e imprime el resultado.

## 2. Algoritmo SumaV {

// declarativas

entero num1, sum, i;

entero vect[5];

sum = 0;

// llamada a la función

para (i = 0; i <= 4; i = i + 1) {

imprimir("Ingrese un numero:");

leer(num1);

vect[i] = num1;

}

para (i = 0; i <= 4; i = i + 1) {

sum = sum + vect[i];

}

imprimir ("La suma de los 5  
números es", sum);

}



## 3. PRUEBA DE ESCRITORIO

vect

0	1	2	3	4

variables de memoria				
i	num1	vect[i]	sum	PANTALLA

## Ejemplo no. 3

Elabore una algoritmo que solicita 5 números enteros y los almacena en un vector. Posteriormente calcula la suma de los elementos e imprime el resultado.

### Código en C

```
/* Cargar un vector de 5 elementos e imprime cada uno de sus
elementos*/
#include<iostream.h>
using namespace std;
int main()
{
//Declarativas
int vect[10]={0}, i,num=0;
//Cargar vector
for (i=0; i<=4; i+=1){
    cout<<"\nIngrese un numero: ";
    cin>>num;
    vect[i] = num;
}
for (i=0; i<=4; i+=1){ // Imprimir contenido del vector
    cout<<"\n Elemento es: ";
    cout<<vect[i];
}
return 0;
}
```

*Escribir un algoritmo que realice las siguientes operaciones:* • Lea 10 números enteros y almacénelos en un vector llamado vec.

*Lea un número entero cualquiera desde el teclado.*

- *Calcule e imprima cuántas veces se encuentra almacenado el número leído en el punto anterior, dentro del vector vec.*

*Nota: Optimizar el uso de variables para almacenar elementos en el vector.*

# Reglas para Manejo de Índices

1. Un índice debe ser un número entero, pero también pueden ser el valor de una variable de tipo entero.

Ejemplo:

**entero i**

**i = 2**

**tab ( 3 ) , tab ( i )**

2. Un índice debe tomar solamente valores positivos, enteros.

**kk ( 10 )            aa(1000)**

**temp(-5) incorrecto X**

3. Un índice no puede ser una variable con subíndice.

**entero j ,s**

**kk( j (s) ) incorrecto**

# Reglas para Manejo de Índices

1. Un índice debe ser un número entero, pero también pueden ser el valor de una variable de tipo entero.

Ejemplo:

**entero i**

**i = 2**

**tab ( 3 ) , tab ( i )**

2. Un índice debe tomar solamente valores positivos, enteros.

**kk ( 10 )          aa(1000)**

**temp(-5) incorrecto X**

3. Un índice no puede ser una variable con subíndice.

**entero j ,s**

**kk( j (s) ) incorrecto**

# Reglas para Manejo de Índices

4. Una variable que representa un arreglo no debe usarse sin índice.

Ejemplo:

```
entero tab (10);
```

```
entero edad;
```

```
-----
```

```
-----
```

```
edad = tab; X INCORRECTO
```

# Reglas para Manejo de Índices

6. El nombre del vector tal como aparece en el enunciado debe tener exactamente el mismo número de índices que en cualquier otra parte del programa.

Ejemplo:

```
entero tab(10);
```

✓ **X** dato = **tab( i, j);** /\* es un vector, solo usa un índice\*/



# Paso de Arreglos Unidimensionales a Funciones

## 3. Se puede realizar de dos formas

- ☐ Un elemento del arreglo (**Paso por valor**)
- ☐ El arreglo completo (**Paso por Referencia**)

.


# Paso de Arreglos Unidimensionales a Funciones

## a . Un elemento del Arreglo

Cuando se pasa un elemento del arreglo a una función , el parámetro será una copia del valor transferido.

```
r = Funcion_por_Valor(tab[2]);
```

```
entero Funcion_por_Valor(entero x) {  
    imprimir("valor de x:", x);  
    x = x + 20;  
    retornar();  
}
```



# Paso de Arreglos Unidimensionales a Funciones

El nombre del arreglo se puede usar como argumento de una función, *permitiendo que el arreglo completo sea pasado a la función.*

Para hacer esto, en la llamada a la función, se pone el nombre del arreglo sin índice. Esto pasa **la dirección del primer elemento** del arreglo a la función. Lo cual significa que los arreglos **se pasan por referencia**, en realidad se pasa la dirección del primer elemento del arreglo.

# Paso de Arreglos Unidimensionales a Funciones

## A. Paso de un arreglo a funciones ( **es paso por Referencia** )

El **paso por referencia** es enviar la **dirección de memoria** de una variable a un parámetro. Esto ocasiona que cualquier cambio al parámetro, en la función, afecta a la variable original.

**flotante** tab[10];

-

**Modificar\_Vector(tab)**

-----

```
Modificar_Vector(flotante vec[ ]) {  
  para ( i = 0; i<=9;i=i+1 ) {  
    vec[i] = vec[i] * 5;  
  }  
  retornar(); }
```

Cuando se pasa el nombre del vector a una función se pasa la **dirección de memoria** del primer elemento del mismo.