

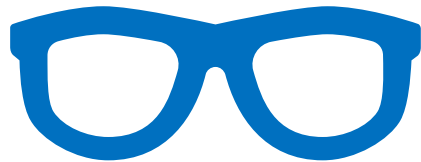
# PROGRAMACIÓN

**Unidad 5:** Tipos de datos derivados: arreglos. Arreglos unidimensionales. Arreglos numéricos. Arreglos de caracteres: Cadenas. Uso de funciones de la biblioteca estándar. Arreglos y funciones.

# Repasemos lo visto



# Tener en cuenta



Todo lo que leerá a partir de este momento es solo una guía para la clase de teoría.

Usted como estudiante debe tomar apuntes de lo tratado en clase de manera de enriquecer sus conocimientos.



# Tipos de datos derivados



*“Existe una cantidad conceptualmente infinita de tipos de datos derivados, formados a partir de los tipos de datos simples”.*

Brian Kernighan y Dennis Ritchie

# Estructuras de Datos Clasificación

---

Según el tipo de información que contienen

Estructuras de datos  
homogéneas

Estructuras de datos  
heterogéneas

# Estructuras de Datos Clasificación



Según la asignación de memoria

Estructuras de datos  
estáticas

Estructuras de datos  
dinámicas

# En el lenguaje C se predefinen los siguientes tipos de datos derivados:

- Arreglos <<<
- Punteros
- Estructuras
- Uniones

# Arreglos



- “Un arreglo es una colección de variables ordenadas e indexadas, todas de idéntico tipo que se referencian usando un nombre común”.
- “El array (también conocido como arreglo, vector o matriz) permite trabajar con una gran cantidad de datos del mismo tipo bajo un mismo nombre o identificador.”



# Características de los arreglos

- ❑ Es un tipo de dato homogéneo.
- ❑ Es una estructura indexada.
- ❑ Es un tipo de dato estático.
- ❑ Los arreglos usan un espacio de almacenamiento contiguo.
- ❑ El nombre del arreglo es una referencia a la dirección de la 1° componente

# Un arreglo puede ser:

- De una dimensión (un índice)

```
int vector[3];
```

- De varias dimensiones (varios índices)

```
int matriz[3][3];
```

- Las componentes del arreglo pueden ser de cualquier tipo: caracteres, enteros, reales, punteros , estructuras.

# ¿Cómo se declara un arreglo de una dimensión?

Forma General: **tipo** **nombre**[**tamaño**];

**nombre**: cualquier identificador válido.

**tipo**: puede ser un tipo aritmético (*int*, *float*, *double*), *char*, puntero, estructura, etc.

**tamaño** : expresión constante entre corchetes que define cuantos elementos guardará el arreglo.

A tener en cuenta...

Un arreglo siempre se declara incluyendo entre los corchetes el máximo número de elementos, salvo que inicialice el arreglo al mismo tiempo.

# Arreglos de una dimensión

*“ Un vector o arreglo lineal es un tipo de dato arreglo con un índice, es decir, con una dimensión”.*

- El acceso a las componentes de un arreglo se puede hacer en forma directa, a través del nombre del arreglo y la notación de subíndice que indica la posición .

# Acceso al contenido de un arreglo

Para acceder a uno de los elementos del arreglo en particular, basta con invocar el nombre del arreglo y especificar entre corchetes del elemento.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int arre3[] = {2,4,6,8,10};
6      printf("arre3[3]= %d", arre3[3]);
7
8      return 0;
9  }
```

Por ejemplo, si se quiere acceder al cuarto elemento del arreglo `arre3`, se invocaría de la siguiente manera: `arre[3]`.

Tenga en cuenta que el arreglo almacena desde la casilla 0. Por tanto, en un arreglo de 20 casillas, éstas están numeradas del 0 al 19.

¿Qué numero me mostrara por pantalla?

`arre3[3] = 8`

# Operaciones

Los elementos de un vector se utilizan en las expresiones de C como cualquier otra variable.

Ejemplos:

$$a[5] = 0.8;$$

$$a[9] = 30. * a[5];$$

$$a[0] = 3. * a[9] - a[5] / a[9];$$

$$a[3] = (a[0] + a[9]) / a[3];$$

# ¿Cómo se trabaja con el tipo arreglo en diseño?

Ejercicio: Realizar un algoritmo que cuenta la cantidad de cada dígito que hay en una frase.

Algoritmo contadorDigito

ENTRADA: frase: arreglo de caracteres. MF= '\0'

SALIDA: cont\_digito: entero >0

Vble aux: i: entero >=0

A0. Inicializar

A1. LEER(frase)

A2. Mientras (frase<sub>i</sub> <> MF)

    Si (frase<sub>i</sub> es dígito) // en c usamos la función **isdigit()**

        cont\_digito ← cont\_digito + 1

    fin\_si

    i ← i+1

fin\_mientras

A3. ESCRIBIR(cont\_digito)

A4. PARAR

# Del tamaño del arreglo



- ❑ La dimensión del arreglo debe ser establecida a priori.
- ❑ El tamaño o dimensión del arreglo debe ser una expresión constante.
- ❑ La declaración de un arreglo, es una reserva de memoria, por lo tanto, el tamaño no puede ser una variable ni una expresión variable



# Ejemplos de Arreglos

```
1  #include <stdio.h>
2  #define TAMA 3
3
4  int main() {
5
6      int arre1[5];
7      int arre2[TAMA];
8      int arre3[] = {0,1,2,3,4};
9
10     return 0;
11 }
12
```

Un arreglo siempre se declara incluyendo entre los corchetes el máximo número de elementos, salvo que inicialice el arreglo al mismo tiempo.

La declaración que sigue..... ¿es correcta?

*int arreglo[];*

# Acceso al contenido de un arreglo

```
1  #include <stdio.h>
2  #define TAMA 20
3
4  int main() {
5
6      int arre2[TAMA], j=0;
7
8      printf("Ingresa un 20 numeros enteros:");
9
10     for(int i=0; i<5; i++)
11     {
12         printf("\n arre2[%d]:", i);
13         scanf("%d", &arre2[i]);
14     }
15
16     printf("El arreglo ingresado es: ");
17     while(j<TAMA)
18     {
19         printf("\n arre2[%d]=%d", j, arre2[j]);
20         j++;
21     }
22     return 0;
23 }
```

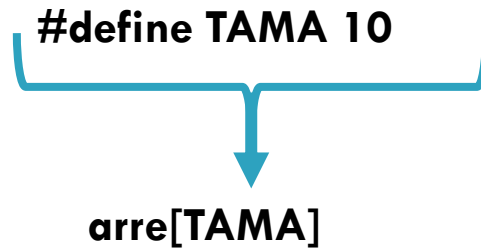
¿Qué error encuentra en el código?

El arreglo ingresado es:

```
arre2[0]=1
arre2[1]=2
arre2[2]=3
arre2[3]=4
arre2[4]=5
arre2[5]=0
arre2[6]=16
arre2[7]=0
arre2[8]=0
arre2[9]=0
arre2[10]=0
arre2[11]=0
arre2[12]=8
arre2[13]=0
arre2[14]=4199705
arre2[15]=0
arre2[16]=8
arre2[17]=0
arre2[18]=22
arre2[19]=0
```

# Tamaño de un Arreglo

`#define TAMA 10`



`arre[TAMA]`

2	4	6	8	10	#\$"%	12548	/&\$#"	985	!°#&%
arre[0]	arre[1]	arre[2]	arre[3]	arre[4]	arre[5]	arre[6]	arre[7]	arre[8]	arre[9]

# Tamaño de un Arreglo

Se debe ser responsable en la administración del espacio de memoria cuando se trabaja con arreglos.

```
1  #include <stdio.h>
2  #define LIMITE 10
3  #define PROBLEMA 500
4
5  int main(void)
6  {   int indice, arreglo[LIMITE];
7
8      for(indice=0; indice < PROBLEMA; indice++)
9      {
10         arreglo[indice]=indice;
11     }
12
13     return 0;
14 }
```

# Recomendación

- Usar constantes definidas con `#define` para indicar el tamaño.
  - ▣ Facilita , ordena y organiza los datos.
  - ▣ Asegura que las referencias posteriores al arreglo ( por ejemplo en un lazo de lectura) no podrán superar el tamaño especificado.
  - ▣ Superar el tamaño máximo declarado puede generar situaciones irregulares con resultados perjudiciales, como lo vimos en ejemplos anteriores.

# Uso de constantes enteras

Definición de  
una constante  
TAMA

```
1  #include <stdio.h>
2  #define TAMA 20
3
4  int main()
5  {
6      int arreglo[TAMA];
7      ...
```

Declaración de  
un arreglo



# Importante!



- ❑ En C no se puede operar (comparar, sumar, restar, etc) con todo un vector o toda una matriz como una única entidad
- ❑ Hay que tratar sus elementos uno a uno por medio de bucles for o while

# Arreglos

```
1  #include <stdio.h>
2  #define TAMA 20
3
4  int main() {
5
6      int arre2[TAMA], j=0;
7
8      printf("Ingrese un 20 numeros enteros:");
9
10     for(int i=0; i<TAMA; i++)
11     {
12         printf("\n arre2[%d]:", i);
13         scanf("%d", &arre2[i]);
14     }
15
16     printf("El arreglo ingresado es: ");
17     while(j<TAMA)
18     {
19         printf("\n arre2[%d]=%d", j, arre2[j]);
20         j++;
21     }
22     return 0;
23 }
```

Hay que tratar sus elementos uno a uno mediante estructuras de iteración.



# Inicialización de Arreglos



Hay 3 formas de inicializar un arreglo:

- ❑ Por omisión.
- ❑ Por inicialización explícita.
- ❑ En tiempo de ejecución.

# Por omisión

```
1  #include <stdio.h>
2  #define TAMA 5
3  int arre2[TAMA];
4
5  int main()
6  {
7      int arre1[TAMA];
8
9      printf("El arreglo declarado de manera global es: ");
10     for(int j = 0; j<TAMA; j++)
11     {
12         printf("\n arre2[%d]=%d", j, arre2[j]);
13     }
14
15     printf("\nEl arreglo declarado local a main(): ");
16     for(int j = 0; j<TAMA; j++)
17     {
18         printf("\n arre1[%d]=%d", j, arre1[j]);
19     }
20     return 0;
21 }
```

Un arreglo declarado en forma global, se inicializa por omisión (por default), en ceros binarios, a menos que se indique lo contrario.

```
El arreglo declarado de manera global es:
arre2[0]=0
arre2[1]=0
arre2[2]=0
arre2[3]=0
arre2[4]=0
El arreglo declarado local a main():
arre1[0]=8
arre1[1]=0
arre1[2]=22
arre1[3]=0
arre1[4]=9708464
```

# Por inicialización explícita

```
1  #include <stdio.h>
2  #define TAMA 3
3
4  int main()
5  {
6      float arre1[] = {78.6, 98.9, 45.7, 34.5, 56.7 } ;
7      int arre3[] = {2,4,6};
8
9      printf("El arreglo 3 es: ");
10     for(int j = 0; j<TAMA; j++)
11     {
12         printf("\n arre3[%d]=%d", j, arre3[j]);
13     }
14
15     printf("\nEl arreglo 1: ");
16     for(int j = 0; j<TAMA; j++)
17     {
18         printf("\n arre1[%d]=%f", j, arre1[j]);
19     }
20
21     return 0;
22 }
```

En la declaración, se asignan valores, según la siguiente norma: los valores a ser asignados a los elementos del arreglo deben estar encerrados entre llaves y separados por comas.

El arreglo 3 es:

arre3[0]=2

arre3[1]=4

arre3[2]=6

El arreglo 1:

arre1[0]=78.599998

arre1[1]=98.900002

arre1[2]=45.700001

# En tiempo de Ejecución

```
1  #include <stdio.h>
2  #define TAMA 5
3
4  int main()
5  {
6      float arre1[TAMA];
7
8      printf("Ingresar 5 numeros reales: ");
9      for(int j = 0; j<TAMA; j++)
10     {
11         printf("\n arre1[%d]", j);
12         scanf("%f", &arre1[j]);
13     }
14
15     printf("\nEl arreglo 1: ");
16     for(int j = 0; j<TAMA; j++)
17     {
18         printf("\n arre1[%d]=%.2f", j, arre1[j]);
19     }
20
21     return 0;
22 }
```

El caso mas común. Las componentes del arreglo tomarán valores que son leídos desde algún dispositivo de entrada ó sino valores generados por el mismo programa.

```
...
printf("Ingresar 5 numeros reales: ");
for(int j = 0; j<TAMA; j++)
{
    arre1[j] = j * 2;
}

printf("\nEl arreglo 1: ");
for(int j = 0; j<TAMA; j++)
{
    printf("\n arre1[%d]=%d", j, arre1[j]);
}

return 0;
}
```

# Arreglo de más de una dimensión

Como se declara un arreglo de más de una dimensión?

Forma General: **tipo** *nombre*[tam1][tam2]...[tamN];

La diferencia principal es que se necesitarán múltiples índices para acceder a los elementos en las diferentes dimensiones del arreglo.

# Ejemplo

Veamos un ejemplo de un arreglo de dos dimensiones



```
#include <stdio.h>
#define FILAS 5
#define COLUMNAS 4

int main()
{
    int matriz[FILAS][COLUMNAS];

    // Llena la matriz con algunos valores
    for (int i = 0; i < FILAS; i++) {
        for (int j = 0; j < COLUMNAS; j++) {
            matriz[i][j] = (i * FILAS) + j + 1;
        }
    }

    // Muestra la matriz
    printf("Matriz:\n");
    for (int i = 0; i < FILAS; i++) {
        for (int j = 0; j < COLUMNAS; j++) {
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

# Ejemplo

Como sería el código si trabajamos con un arreglo de 3 dimensiones?

`arreglo[indice1][indice2][indice3]`

Debemos tener en cuenta que:

1° índice: sería la profundidad.

2° índice: la cantidad de filas.

3° índice: la cantidad de columnas.



**Tarea: cargar y mostrar como queda un arreglo de 3 dimensiones**

# Cadenas –Arreglos de caracteres

- Este es el tipo de arreglo mas popular en código C.
- La ultima celda del vector de caracteres se reserva para el carácter que marca el fin de la cadena, que es el carácter nulo: “\0”.



# Inicialización explícita de una cadena

Los valores a asignar deben estar encerrados entre llaves y separados por comas.

```
1  #include <stdio.h>
2  #define MAX 45
3
4  int main(void)
5  {
6      char apellido[MAX] = {'P','e','r','e','z','\0'};
7      // agrega al final el correspondiente cero de terminación al final de la cadena
8      char nombre[MAX] = "Marcelo";
9      char unt[12]= {'U','n','i','v','e','r','s','i','d','a','d'};
10
11      ...
12
```

# Funciones de biblioteca y arreglos

## Función de entrada para cadena de caracteres:

**gets(arre):** almacena datos ingresados desde stdin a la cadena denominada arre. Un carácter ingresado `\n` de nueva línea se convierte en un cero de terminación (`\0`)

## Función de salida para cadena de caracteres:

**puts(arre):** encamina la cadena arre hacia stdout. Un cero de terminación (`\0`) al final de la cadena se convierte en un carácter de nueva línea (`\n`).

# Ejemplos

```
1  #include <stdio.h>
2  #define TAMA 50
3  int main()
4  {
5      char nombre[TAMA];
6
7      printf("Ingrese su nombre: ");
8      gets(nombre);
9
10     printf("El nombre ingresado es:");
11     puts(nombre);
12
13     return 0;
14 }
```

Ingrese su nombre: Ana Sofia  
El nombre ingresado es: Ana Sofia

# Ejemplos

Cargar cadenas de caracteres en un arreglo de dos dimensiones.

```
#include <stdio.h>

int main() {
    char nombres[3][20];

    // Solicitar al usuario que ingrese nombres
    for (int i = 0; i < 3; i++) {
        printf("Ingrese el nombre %d: ", i + 1);
        gets(nombres[i]);
    }

    // Imprimir los nombres ingresados
    for (int i = 0; i < 3; i++) {
        printf("Nombre %d: %s\n", i, nombres[i]);
    }

    return 0;
}
```

# Funciones de biblioteca <string.h>

Mencionaremos solo algunas funciones.

Prototipo	Descripción
<code>int strlen(cad1)</code>	Retorna la longitud de cad1
<code>int strcmp(cad1, cad2)</code>	Compara cad1 con cad2, carácter a carácter
<code>char *strcat(cad1, cad2)</code>	Concatena la cad2 al final de la cad1. Retorna cad1.

# Funciones y Arreglos

```
1  #include <stdio.h>
2  #define TAMA 10
3
4  void Inicializar(int arre[TAMA] , int cant);
5  void CargarArreglo(int arre[TAMA], int cant);
6
7  int main()
8  {
9      int arre[TAMA];
10     int cant;
11
12     printf("Ingrese la cantidad de Elementos (<10):");
13     scanf("%d", &cant);
14
15     Inicializar(arre, cant);
16     CargarArreglo(arre, cant);
17
18     for(int i=0; i< cant; i++)
19     {
20         printf("\n arre[%d] = %d", i, arre[i]);
21     }
22
23     return 0;
24 }
```

```
void CargarArreglo(int arre[TAMA], int cant)
{
    int i;
    for(i=0; i<cant; i++)
    {   arre[i]= i * 3; }
}

void Inicializar(int arre[TAMA] , int cant)
{
    int i;
    for(i=0; i<cant; i++)
    {   arre[i]=0;      }
}
```

# A modo de resumen



Los arreglos, como una unidad, no admiten:

- ❑ Asignación directa entre ellos
- ❑ Operaciones aritméticas directas
- ❑ Comparaciones directas
- ❑ Devolución como valor de devolución de una función

# A pedido de ustedes...





# Tarea para la casa



1. ¿Qué es un arreglo en C?
2. ¿Cuál es la diferencia entre una variable y un arreglo en C?
3. ¿Cómo se declara un arreglo en C?
4. ¿Cuál es la importancia de la longitud de un arreglo?
5. ¿Cuál es la forma de acceder a los elementos individuales de un arreglo?
6. ¿Qué es el índice de un arreglo y cómo se utiliza?
7. ¿Cómo se inicializa un arreglo en C?
8. ¿Cuál es la diferencia entre un arreglo unidimensional y un arreglo multidimensional?
9. ¿Cómo se realiza la copia de un arreglo en otro en C?
10. ¿Qué es una cadena de caracteres en C?
11. ¿Cómo se puede determinar la longitud de una cadena de caracteres en C?
12. ¿Cómo se puede recorrer un arreglo utilizando bucles (for, while)? ¿se puede recorrer con do-while?
13. ¿Cómo se pasa un arreglo a una función como argumento?
14. ¿Una función puede retornar un arreglo?

**Lo revisamos la próxima clase.**

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("Hasta la próxima clase!!\n");
```

```
    return 0;
```

```
}
```