

# Arrays

# Disclaimer



Los códigos fuente utilizados para la explicación del funcionamiento del “stack” son solo a finés didácticos y no representa una óptima solución al problema a resolver.

# Variables

- Primitive Types in C
- Pointers
- Arrays

# Tipos de datos (repaso)

- Enteros
  - `char`, `int`
- Decimales
  - `float`, `double`
- Modificadores
  - `short` [`int`]
  - `long` [`int`, `double`]
  - `signed` [`char`, `int`]
  - `unsigned` [`char`, `int`]

32 ó 64 bit

```
$ arch
$ echo $MACHTYPE
$ lscpu
```

C Data Type	32-bit	64-bit	printf
<code>char</code>	1	1	<code>%c</code>
<code>short int</code>	2	2	<code>%hd</code>
<code>unsigned short int</code>	2	2	<code>%hu</code>
<code>int</code>	4	4	<code>%d / %i</code>
<code>unsigned int</code>	4	4	<code>%u</code>
<code>long int</code>	4	8	<code>%ld</code>
<code>long long int</code>	8	8	<code>%lld</code>
<code>float</code>	4	4	<code>%f</code>
<code>double</code>	8	8	<code>%lf</code>
<code>long double</code>	12	16	<code>%Lf</code>
<code>pointer</code>	4	8	<code>%p</code>

# ¿ es posible resolver ?

>\_

Ingrese 5 valores enteros:

Valor 1: 10

Valor 2: 12

Valor 3: 19

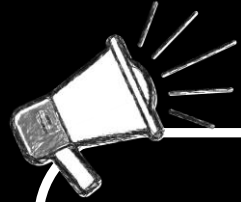
Valor 4: 1

Valor 5: 4

Promedio = 9.2

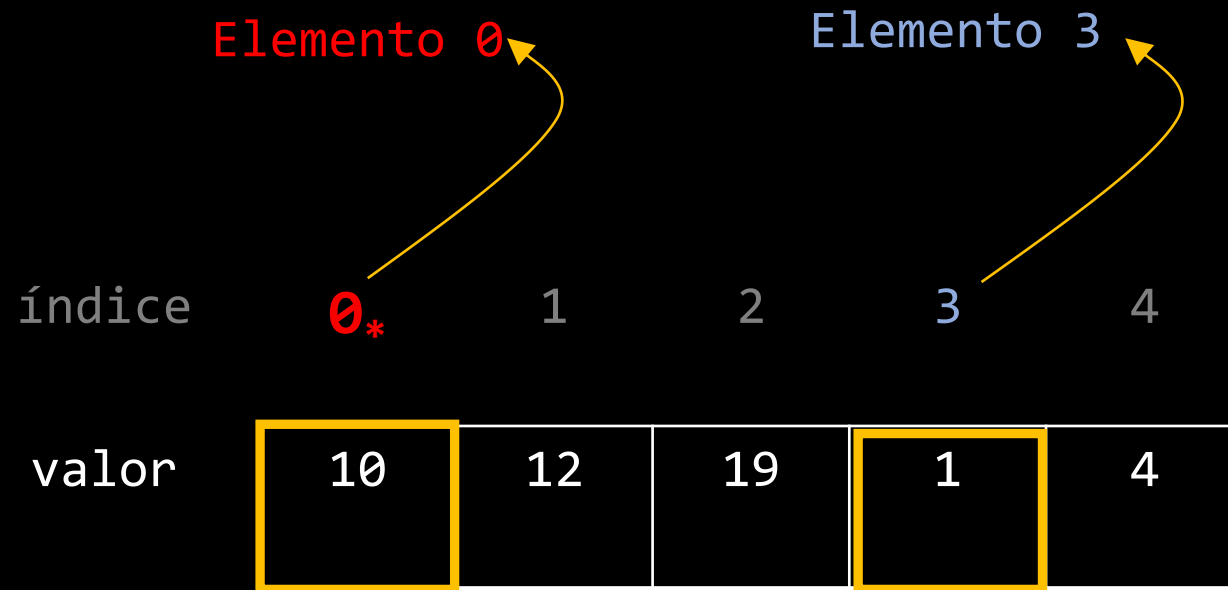
Valores superiores al promedio = 3

XX: entrada por teclado



## array@definición

- Variable con un conjunto de datos del “mismo tipo”
- Se ubican en direcciones “consecutivas de memoria”



\* índice “0-based int”

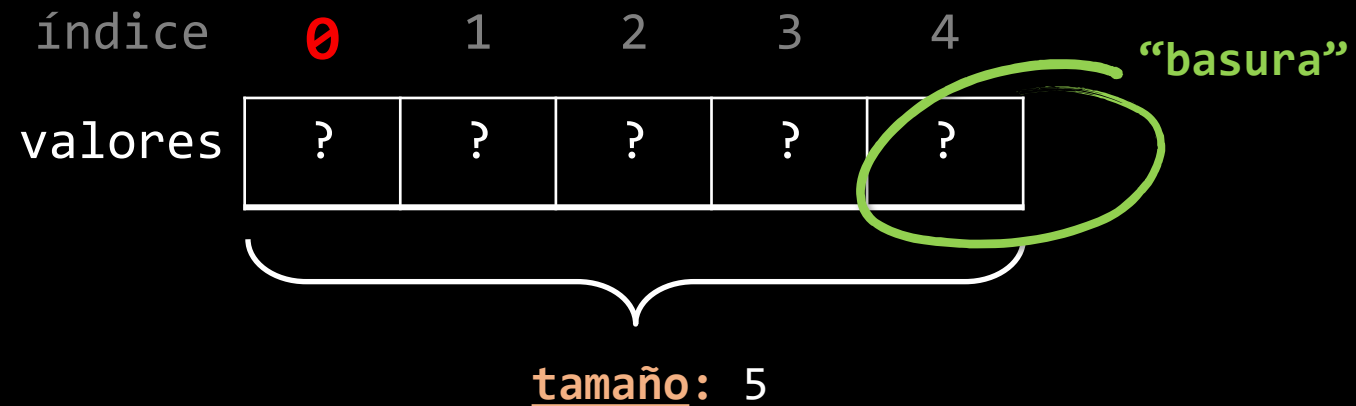
# array@declaración

**tipo nombre[tamaño];**

- El tamaño en cantidad de elementos de “tipo”
- El tamaño se define en tiempo de compilación (\*excepción C99, no recomendada)

- Ejemplo:

```
int valores[5];
```

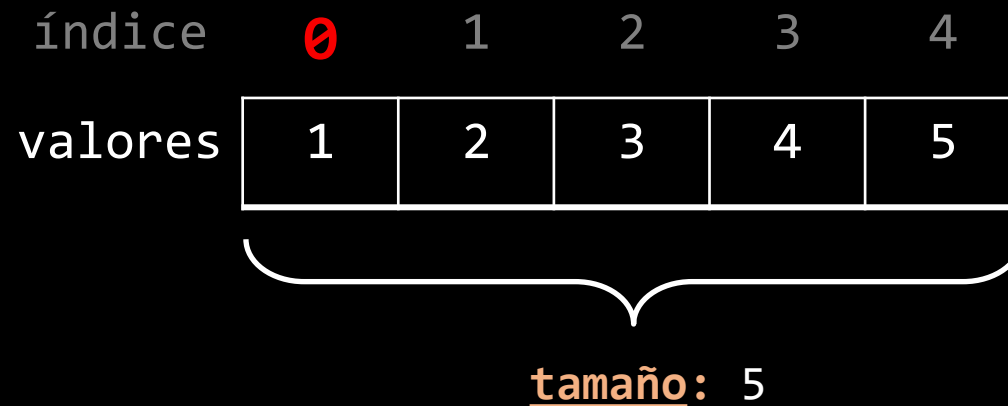


# array@declaración e inicialización

```
tipo nombre[tamaño] = { valor, valor, ...} ;
```

- Ejemplo:

```
int valores[5] = { 1, 2, 3, 4, 5};
```



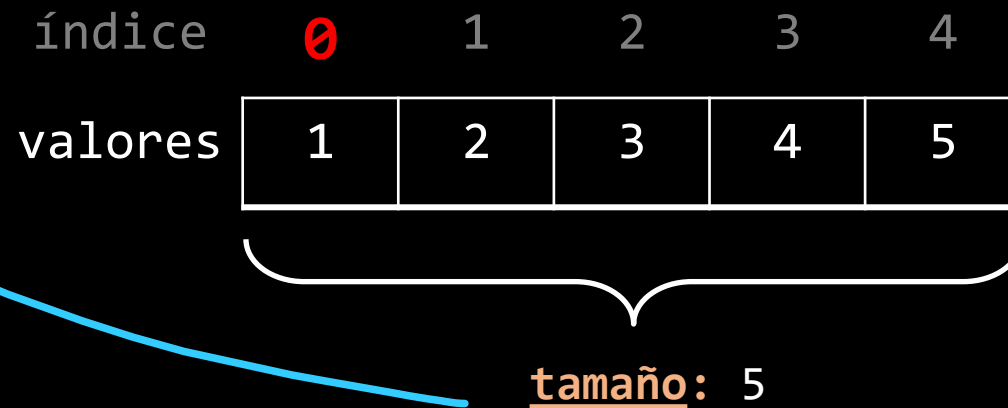


# array@declaración e inicialización

```
tipo nombre[] = { valor, valor, ...} ;
```

- Ejemplo:

```
int valores[] = { 1, 2, 3, 4, 5};
```



# array@acceso a un elemento

- nombre[indice] // acceso
- nombre[indice] = valor // modificación

• Ejemplo:

```
int x, valores[5];
```

```
valores[0] = 21;
```

```
valores[3] = -7;
```

```
x = valores[0];
```

índice

0

1

2

3

4

valores

21

?

?

-7

?

“basura”

tamaño: 5

x

21

# array@limites

- `Limites`: entre 0 y el tamaño del array - 1.

Leer o Escribir fuera de los limites es un error del programador, que no es verificado por el compilador (algunos compiladores emiten warning pero obviamente es una verificación estática)

- Ejemplo

```
int x, valores[5];
```

```
valores[1] = 21;    // Correcto
```

```
valores[5] = -7;    // Error, fuera de limite (0...4)
```

¿Qué sucede en este caso de error?

- Leer o Escribir en una posición de memoria no deseada
- “Segmentation fault”

# completar...

5'



int x[5]

1	3	7	14	19
---	---	---	----	----

20

int y[5]

23	25	28	12	0
----	----	----	----	---

40

int z[5]

8	31	-10	14	98
---	----	-----	----	----

60

Referencia	Dirección	Valor
y[3]		
y[6]		
y[-1]		
x[15]		

# array@inicializar



Buena práctica de programación el uso de `#define` para el tamaño del array

```
#define MAX 5
```

```
int main(){  
    int valores[MAX];  
    int i;
```

```
    for(i=0;i<MAX;i++){  
        valores[i] = 0;  
    }
```

índice	0	1	2	3	4	
valores	?	?	?	?	?	i ?

valores	0	?	?	?	?	i	0
valores	0	0	?	?	?	i	1
valores	0	0	0	?	?	i	2
valores	0	0	0	0	?	i	3
valores	0	0	0	0	0	i	4

# array@sizeof()



sizeof() valido únicamente dentro del scope del array

```
#define MAX 5
```

```
int main(){  
    int valores[MAX];  
    int i;  
  
    for(i=0;i<MAX;i++){  
        valores[i] = 0;  
    }
```

índice	0	1	2	3	4
valores	0	0	0	0	0
i	0				

4 bytes

```
printf("Tamaño del array en bytes es: %ld\n", sizeof(valores));
```

```
printf("Tamaño del array en elementos es: %ld\n", sizeof(valores)/sizeof(int));
```

a programar...

15' 

>\_

Ingrese 5 valores enteros:

Valor 1: 10

Valor 2: 12

Valor 3: 19

Valor 4: 1

Valor 5: 4

Promedio = 9.2

Valores superiores al promedio = 3

XX: entrada por teclado


# array@punteros

- ¿Por qué entre cada dirección hay 4 bytes? Si es el array fuese del tipo char, cuanto habría?
- Porque es necesario que sean del mismo tipo?

```
#define MAX 5
```

```
int main(){  
    int valores[MAX] = { 1, 3, 9, 12, 15};  
    int i;  
  
    for(i=0;i<MAX;i++){  
        printf("%d %d %p\n", i, valores[i], &valores[i]);  
    }  
}
```

índice	valor	dirección
0	1	0x7ffe64feffe0
1	3	0x7ffe64feffe4
2	9	0x7ffe64feffe8
3	12	0x7ffe64feffec
4	15	0x7ffe64fefff0

 4 bytes



# array@punteros

```
int x[] = { 1, 3, 9, 12, 15};
```

x equivalente &x equivalente &x[0]

índice    valor    Dirección

0	1	0x7ffe64feffe0	x
1	3	0x7ffe64feffe4	x + 4
2	9	0x7ffe64feffe8	x + 8
3	12	0x7ffe64feffec	x + 12
4	15	0x7ffe64fefff0	x + 16



El nombre de la variable (identificador) representa la dirección de inicio del array (**read-only**)

# array@punteros



```
int x[] = { 1, 3, 9, 12, 15};
```

```
printf("%d %d %p\n", i, x[i], &x[i]);  
printf("%d %d %p\n", i, *(x+i), (x+i));
```

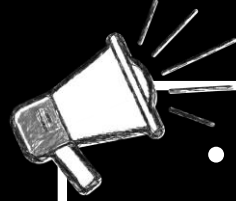
índice    Valor    Dirección

0	1	0x7ffe64feffe0	x
1	3	0x7ffe64feffe4	x + 4
2	9	0x7ffe64feffe8	x + 8
3	12	0x7ffe64feffec	x + 12
4	15	0x7ffe64fefff0	x + 16

Notación array	Notación puntero
x[0]	*(x)
x[1]	*(x+1)
x[2]	*(x+2)
x[3]	*(x+2)
x[4]	*(x+3)

4 bytes  
Artemetica  
de punteros

# array@funciones



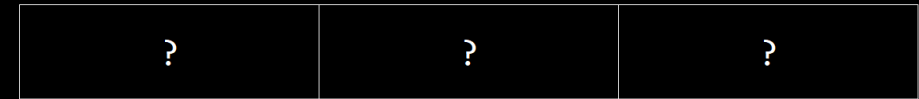
- Un array únicamente puede pasarse por referencia.

```
int arr[3];  
init_int_array(arr);
```

```
void init_int_array(int *p){  
    int i;  
  
    for(i=0; i<sizeof(p); i++){  
        p[i] = 0;  
    }  
}
```

arr  
0x7ffd58978d40

p  
0x7ffd58978d18



0x7ffd58978d40



sizeof() valido únicamente dentro  
del scope del array

# array@funciones

```
// array
#define ARRAY_INT_SIZE 5

void init_int_array(int *p, int size);

int main(){
    int arr[ARRAY_INT_SIZE];

    init_int_array(arr, ARRAY_INT_SIZE);
    return 0;
}

void init_int_array(int *p, int size){
    int i;
    for(i=0; i<size; i++){
        p[i] = 0;
    }
}
```



- Un array únicamente puede pasarse por referencia.
- Debe indicarse el tamaño o algún valor que indique cual es el ultimo elemento

arr  
0x7ffd58978d40



p  
0x7ffd58978d18

<u>0x7ffd58978d40</u>
-----------------------



# Resumen

- Memoria continua
- No se chequea limites
- No se inicializa
- Usualmente lo utilizamos como puntero al primer elemento
- Con funciones solo por referencia
- Solo se puede utilizar sizeof() dentro del scope del propio array
- Si bien las ultimas versiones de C permiten arrays de longitud variable, no es una buena practica.

# a programar...

30' 

>\_

```
int array_find(int *, int, int)
```

Devuelve el índice del primer element encontrado. -1 si no lo encuentra

```
int array_equal(int *, int, int *, int)
```

Devuelve 1 si son iguales. -1 si son distintos

```
void array_fill(int *, int, int)
```

Inicializa el array con el valor dado

```
int array_copy( int *, int , int*, int)
```

Copia un array en otro. -1 en caso de que los tamaños no sean iguales

```
float array_average(int *, int)
```

Devuelve el promedio

# array@multidimensional

```
int p[3][5] = {  
    {0, 1, 2, 3, 4},  
    {0, 2, 4, 6, 8},  
    {1, 3, 5, 7, 9}  
};
```

tipo nombre[filas][columnas];

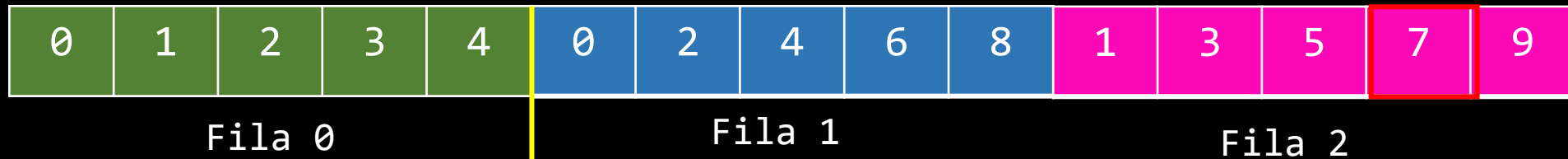
- Ejemplo:

```
int p[3][5];
```

# array@multidimensional

```
int p[3][5] = {  
    {0, 1, 2, 3, 4},  
    {0, 2, 4, 6, 8},  
    {1, 3, 5, 7, 9}  
};
```

p[0][0]	p[0][1]	p[0][2]	p[0][3]	p[0][4]
p[1][0]	p[1][1]	p[1][2]	p[1][3]	p[1][4]
p[2][0]	p[2][1]	p[2][2]	p[2][3]	p[2][4]



¿Cómo calcula el  
desplazamiento el  
Compilador?

Fila "i" =  $p + i * (\text{Columnas} * \text{sizeof(int)})$

Fila "1" =  $p + 1 * (5 * 4) = p + 20$



# completar...

5'



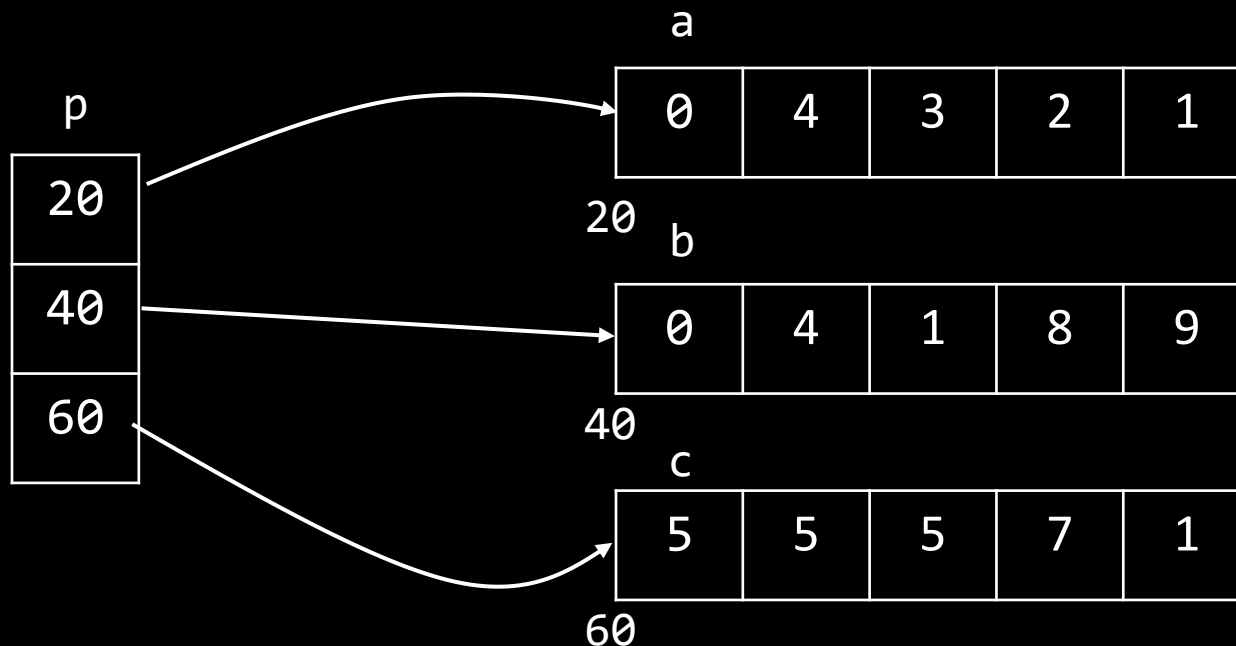
Referencia	Dirección	Valor
p[3][3]		
p[2][5]		
p[2][-1]		
p[0][-1]		

# array@multi-nivel o array de punteros

```
int a[5] = {0, 4, 3, 1, 1};  
int b[5] = {0, 4, 1, 9, 8};  
int c[5] = {5, 5, 5, 7, 1};
```

```
int *p[3] = {a,b,c};
```

- La variable “p” es un array de 3 elementos
- Cada elemento de “p” es un puntero
- Cada puntero apunta a un array de 3 elementos



- Los arrays a,b,c podrían tener tamaños diferentes

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

```
int main(){
```

```
    int a[5] = {0, 1, 2, 3, 4};
```

```
    int b[5] = {0, 2, 4, 6, 8};
```

```
    int c[5] = {1, 3, 5, 7, 9};
```

```
    int *p[3] = {a,b,c};
```

```
    // acceso a un elemento
```

```
    printf("p[0][0]=%d\n", p[0][0]);    //0
```

```
    printf("p[0][4]=%d\n", p[0][3]);    //3
```

```
    printf("p[1][4]=%d\n", p[1][4]);    //8
```

```
    printf("p[2][3]=%d\n", p[2][3]);    //7
```

```
    return 0;
```

```
}
```

$*(*(p+i)+j)$

¿Como calcula el desplazamiento el compilador?

$\text{Mem}[\text{Mem}[p+8*i]+4*j]$

$8 == \text{sizeof(int *)}$

$4 == \text{sizeof(int)}$

# array@multi-(dimensional vs nivel)

## multi-dimensional

```
int p[3][5] = {  
    { 0, 1, 2, 3, 4 },  
    { 0, 2, 4, 6, 8 },  
    { 1, 3, 5, 7, 9 }  
};
```

$p[i][j]$

$*(*(p+i) + j)$

$\text{Mem}[ p + 20*i + 4*j ]$   
(columns\*sizeof(int))\*i

## multi-nivel

```
int a[5] = { 0, 1, 2, 3, 4 };  
int b[5] = { 0, 2, 4, 6, 8 };  
int c[5] = { 1, 3, 5, 7, 9 };  
int *p[3] = { a, b, c };
```

$p[i][j]$

$*(*(p+i) + j)$

$\text{Mem}[ \text{Mem}[p+8*i] + 4*j ]$

$==$

$==$

$!=$