



UNRN

Universidad Nacional
de Río Negro



| unrionegro

Punteros

UNRN

Universidad Nacional
de Río Negro





¿Preguntas?



**Recuerden que se
viene el parcial**

(11 y 12 septiembre)



Algunas cuestiones corregidas



Sobre arreglos


tipo identificador[CAPACIDAD];

**Recuerden:
el tamaño **solo**
puede ser definido
al compilar***

***más adelante vamos a ver como liberarnos de esta restricción**

Pero entonces, ¿que es arreglo?

```
#define SIZE 3  
int arreglo[SIZE];
```



```
int arreglo[SIZE];
```

```
sizeof(arreglo) == sizeof(int)*SIZE;
```



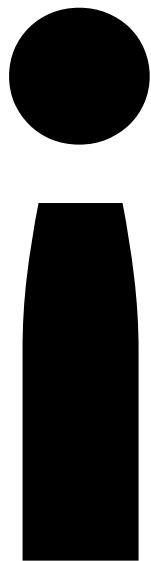
**Pero con una
variable funciona**

¡Funciona!, pero no es correcto

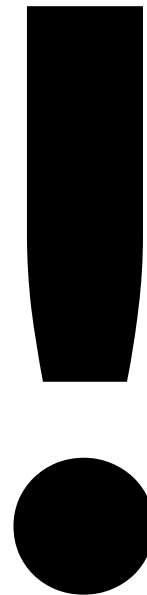
```
int tamano;  
scanf("%d", &tamano);  
int arreglo[tamano];
```



Ya vamos a ver
por qué no
es correcto



El tamaño **solo
puede ser
definido al
compilar**



Y algunas cuestiones sobre

Documentación de arreglos

Descripción de la función con arreglos

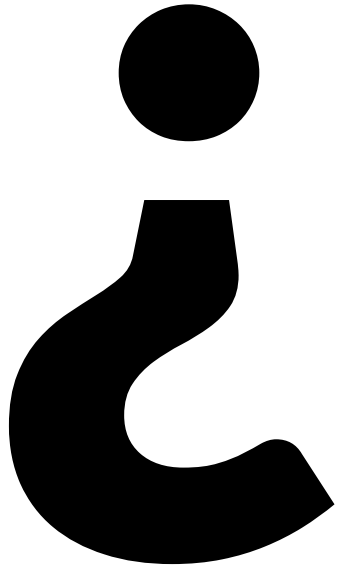
```
/*  
 * Este procedimiento se encarga de ordenar el arreglo  
 * pasado como argumento 'en el lugar'.  
 * @param arreglo los valores a ordenar  
 *     pre: el arreglo es válido y posee elementos  
 *     post: el arreglo es modificado, quedando ordenado  
 * @param capacidad es la cantidad de elementos en el arreglo  
 *     pre: el valor es consecuente con el arreglo  
 *     post: el valor de la capacidad no cambia  
 * @returns no posee  
 */  
void ordena(int arreglo[], int capacidad);
```

**Recuerden que las
postcondiciones
'*generales*' son
correctas también.**

**Es muy necesario
documentar que los
argumentos pueden
cambiar**

**Para que quien llama lo
sepa**

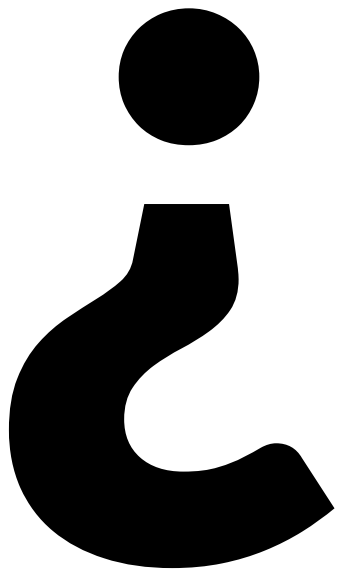
Pero...



**por qué cambian
en la función las
variables afuera**

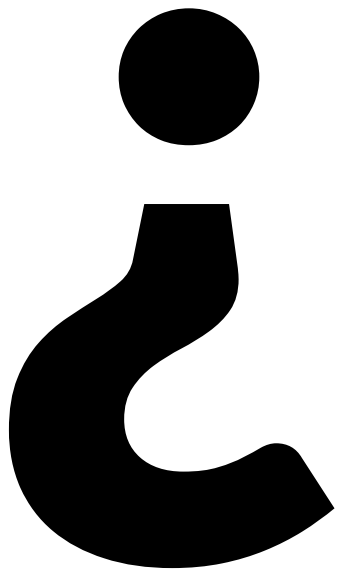


punteros



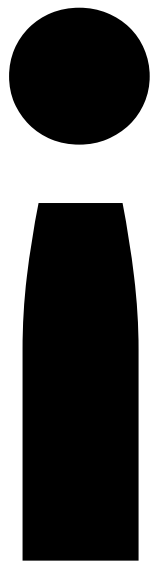
qué son



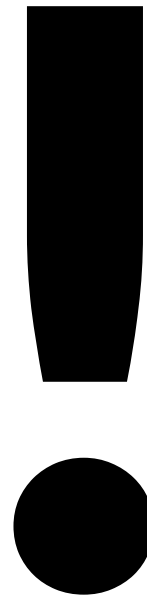


**Lo vimos
indirectamente la
*clase pasada***





**Lo vimos
indirectamente la
*clase pasada***



Con los arreglos

tipo*

Puntero a

Declaración de una variable de tipo puntero a

`<tipo-dato>* <nombre-variable>;`



```
int* ptr;
```

Una variable 'puntero a' int llamada ptr

```
int* ptr;  
// como le decimos a donde apuntar
```

Ojo que, un puntero como r-value **puede** recibir números

```
ptr = 10;
```



¡Pero no debe! Porque el contenido de la variable **no es** un número

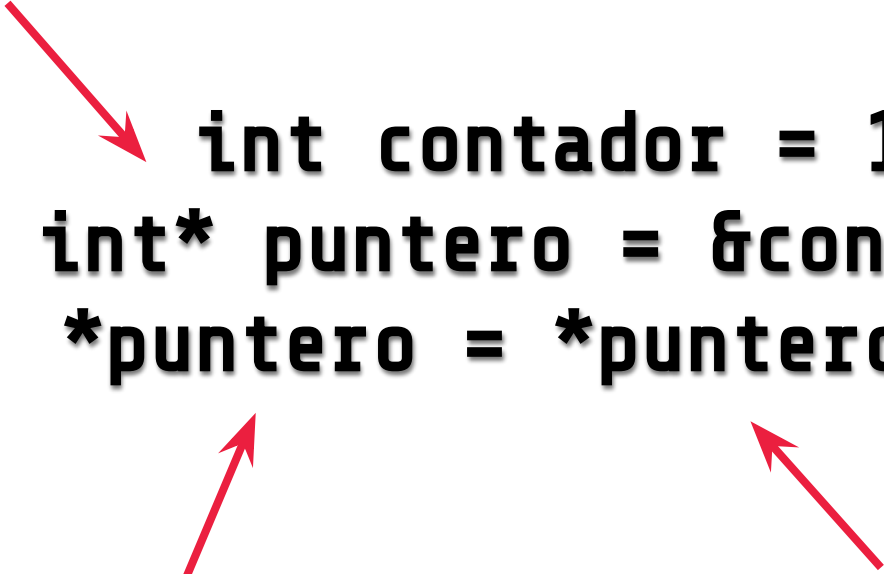
**ptr contiene
una dirección de
memoria**

```
int variable = 10;  
int* ptr = &variable;  
*ptr = 30;
```

***puntero**
Contenido apuntado

&variable **Dirección de**

Un ejemplo multiple



The diagram illustrates pointer manipulation with three red arrows. One arrow points from the top-left towards the variable 'contador' in the first line of code. Two other arrows point from the bottom towards the dereferenced pointer '*puntero' in the third line of code.

```
int contador = 10;  
int* puntero = &contador;  
*puntero = *puntero + 1;
```

Inicialización en un valor neutro

```
int* ptr = NULL;
```

NULL es el valor de puntero para cuando *no apunta* a alguna variable.

Inspeccionando punteros

```
int numero = 10;  
int *puntero = &numero;
```

```
printf("direccion de numero %p\n", &numero);  
printf("contenido de puntero %p\n", puntero);  
printf("direccion de puntero %p\n", &puntero);  
printf("apuntado por puntero %d\n", *puntero);
```

De paso vemos **%p** de printf

¡A la terminal!



El tamaño de los punteros

Es siempre el mismo

```
sizeof(int*) == sizeof(char*)
```

Ojo con el *

```
int contador = 10;  
int* puntero = &contador;  
*puntero = *puntero * 2;
```



¿Preguntas?

Efectos en el pasaje de argumentos

Siempre y solo por valor

**¿Pero si pasamos
un puntero?**

**Y por referencia si
pasamos un puntero**

De-referenciación en funciones

```
void intercambio(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```


Documentando pasaje por referencia

```
/**
 * Este procedimiento toma dos punteros a enteros y intercambia los valores a los que apuntan.
 *
 * @param izquierdo - Puntero al primer entero.
 * @param derecho - Puntero al segundo entero.
 *
 * @pre Los punteros a 'izquierdo' y 'derecho' deben apuntar a
 *      direcciones de memoria válidas que contengan enteros.
 *
 * @post Después de llamar a esta función, los valores apuntados
 *      por 'izquierdo' y 'derecho' habrán sido intercambiados.
 */
void intercambio(int* izquierdo, int* derecho);
```



**La misma situación
que con los arreglos**



¿Preguntas?

Arreglos II

Parecidos pero no iguales

char[] \neq char*

Libertad de asignación

Un puntero puede reasignarse para apuntar a diferentes ubicaciones en la memoria.

Pero un arreglo está asociado con una ubicación específica en la memoria y no puede reasignarse.

¡Esto no funciona!

```
char[40] cadena = "Hola Mundo";  
cadena = "chau mundo";
```

Tamaño

Un arreglo tiene un tamaño fijo que se determina en el momento de la declaración.

Un puntero tiene un tamaño fijo asociado, el necesario para almacenar la dirección de memoria.

Degradación

Cuando pasas un arreglo como argumento a una función, se *degrada* a un puntero automáticamente.

Esto significa que la función recibirá una dirección de inicio del arreglo y no la longitud real del arreglo.

**Razón por la cual
hace falta
capacidad
en el TP3**

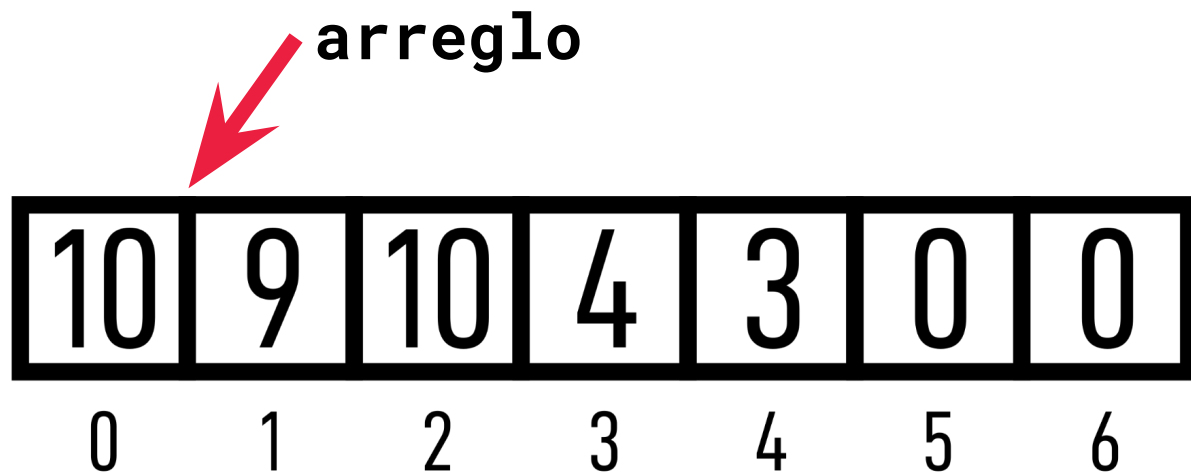
Aritmética de punteros

¡Con cuidado!

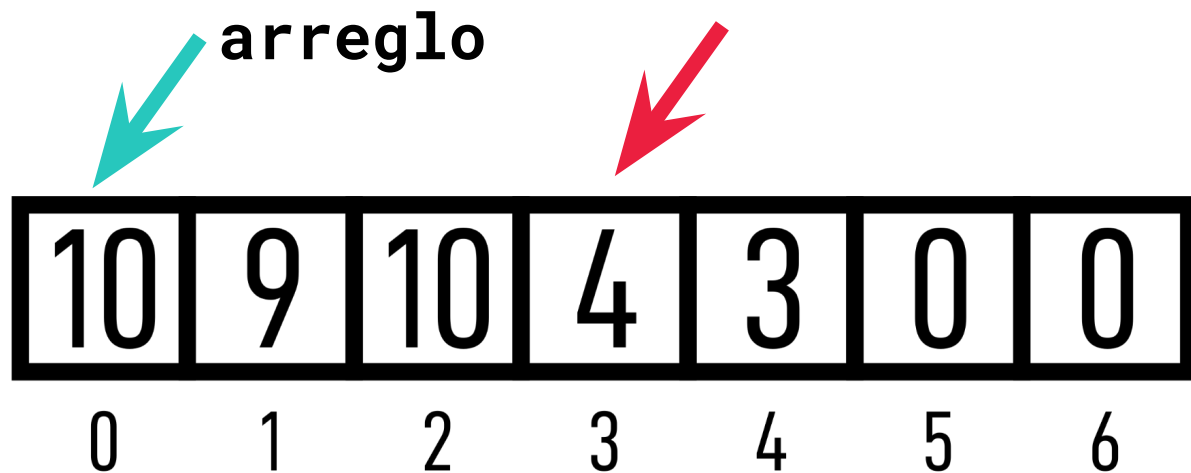
Que sí nos pasamos...

con suerte nos enteramos

```
int arreglo[7] = {10, 9, 10, 4, 3, 0, 0};
```



arreglo+3



Pero...

Arreglos a la fuerza

```
char cadena[] = "Hola Mundo";  
char* puntero = cadena; // lo 'degradamos'  
  
printf("%s\n", puntero+2);
```


¿arreglo+3 es igual a arreglo[3]?



¿Preguntas?

unrn.edu.ar

UNRN

Universidad Nacional
de **Río Negro**



| **unrionegro**