
UNRN
Universidad Nacional
de Río Negro
| unrionegro

Arreglos y cadenas

UNRN
Universidad Nacional
de Río Negro

5
2023

P1

Comentarios de documentación II

En sus tres componentes

Descripción
Precondición
Postcondición

Descripción

Explicación breve y precisa de lo que hace el programa/función.

Su objetivo es conocer rápidamente qué hace el programa .

Pre-condición

Descripción de lo que entra al programa y función.

*Su objetivo es detallar de forma precisa y **no-ambigua** las características de la entrada*

Post-condición

Descripción de los datos de salida del programa y función.

*Su objetivo es detallar de forma precisa y **no-ambigua** las características de la salida.*

{ Lo revisamos para ver que entendieron de la consigna }

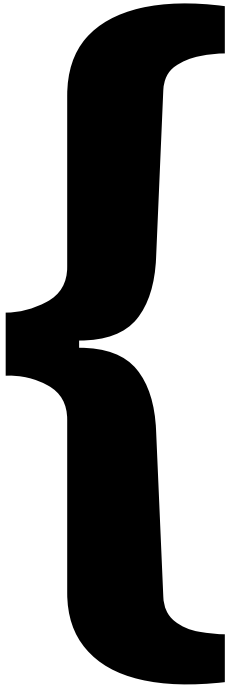
{ ejemplos }

Descripción de la función - ejemplo 1

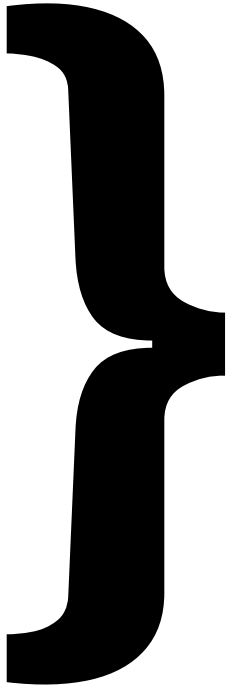
```
/*
 * Toda función debe tener un comentario que
 * describa que es lo que hace.
 * Indicando que rol cumple y hace cada argumento.
 * Esta función se encarga de sumar dos numeros enteros
 * @param termino1 es el primer termino de la suma
 * @param termino2 es el segundo termino a sumar
 * @returns la suma de ambos terminos
 * PRE-CONDICION: numero enteros en el rango de +/-2^8.
 * POST-CONDICION: numero entero que es suma de los
 *                          términos de entrada. La salida tendra
 *                          un rango entre +/- 2^9
 */
int suma(int termino1, int termino2);
```

Descripción de la función - ejemplo 2

```
/*
 * La descripción de la función.
 * ¿Cual es su objetivo?
 * @param termino1 es el primer término a operar
 *                 debe ser positivo y menor a 3000 (PRE)
 * @param termino2 es el segundo término a operar
 *                 debe ser negativo y mayor a 3000 (PRE)
 * @returns la operación de ambos términos
 *      el signo del resultado será el del número mayor
 *      el resultado no puede ser mayor a +/-6000 (POS)
 */
int operar_terminos(int termino1, int termino2);
```

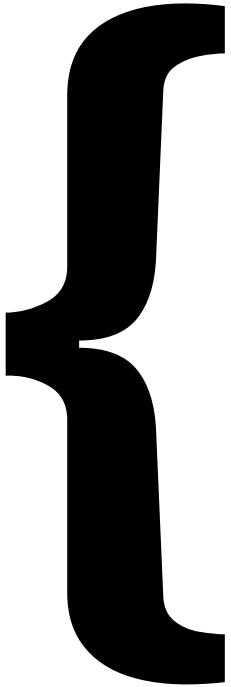


No hay un estilo correcto/incorrecto



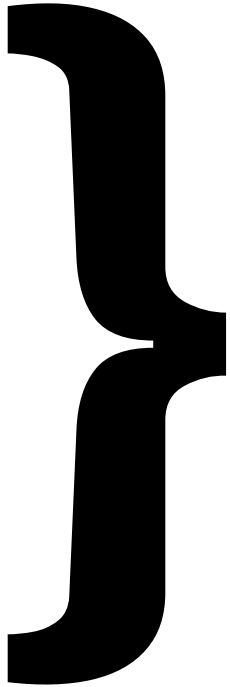
Tiene que estar

Por otro lado



En la pre/post condición, indicar el tipo no es tan importante

Esto ya está dado por el tipo del argumento/retorno



Céntrense en las características que tiene el valor para probar la función

Rango de valores

¿Que valores son válidos para la función?

de entrada y de salida

El divisor no puede ser cero

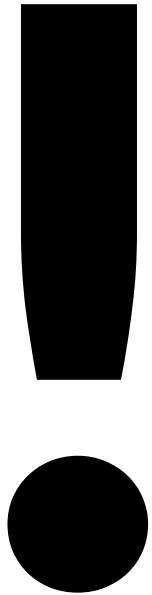
El factorial no puede ser negativo

Valores especiales

Si usan algún valor en particular con un significado diferente al que pueda ser

```
‘c’ para celcius

Retorno -1 cuando no puedo calcular
```

Son una invitación a pasarles esos valores



Hace más fácil asegurarnos de que hace lo que supuestamente debe


¡testing!

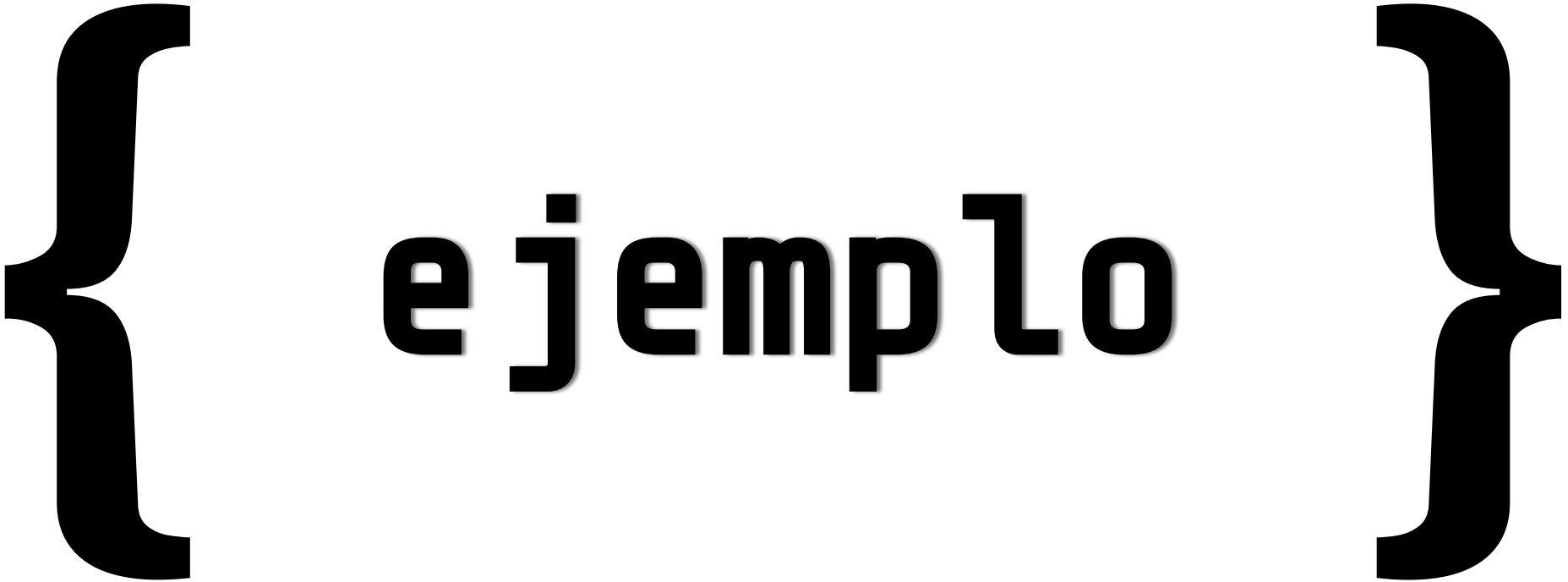
Ejercicio - documentación

```
int incremento(int valor)
{
    valor = valor + 1;
    return valor;
}
```


¿Preguntas?

funciones II

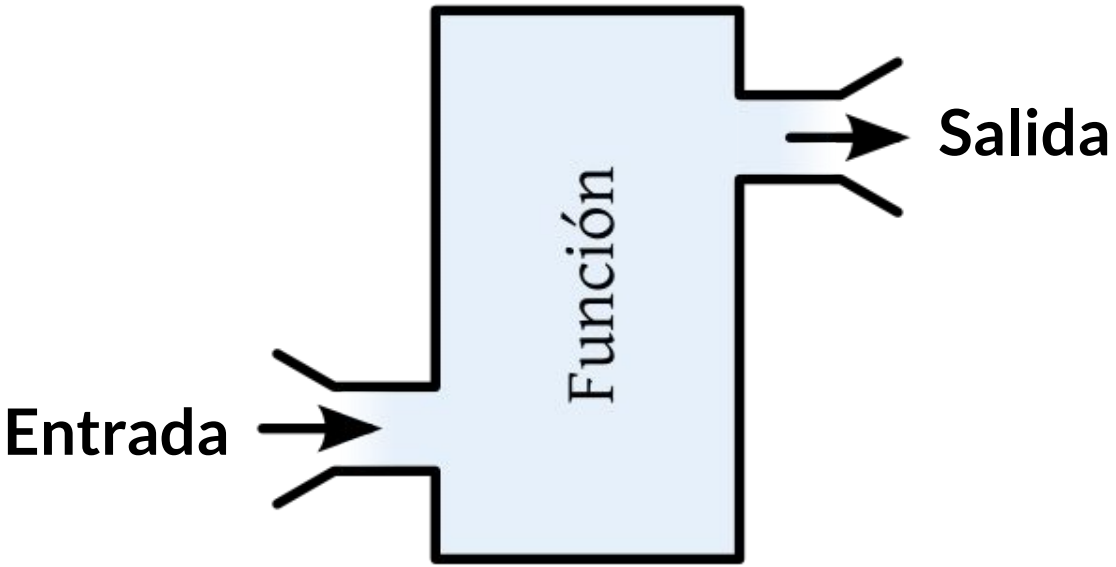
Cómo viaja la información en los argumentos


ejemplo

Ejemplo I - Consola en vivo

```
/**
 * Ejemplo pasaje de argumentos
 */

#include <stdio.h>

int incremento(int valor)
{
    valor = valor + 1;
}

int main()
{
    int variable = 10;
    incremento(variable);
    printf("variable: %d\n", variable);
    return 0;
}
```

—



Esto se “mantiene”, si no hay salida, no hay cambios afuera.

¡no hay cambios!

¡es lo importante!

Lo que ‘*entra*’ es una copia del valor de la variable

Lo que pasa en la función queda en la función

```
void incremento(int valor)
{
    valor = valor + 1;
}
```

Si la función tiene/usa un `printf`

¿Hay cambios?

¿Pero ahora?

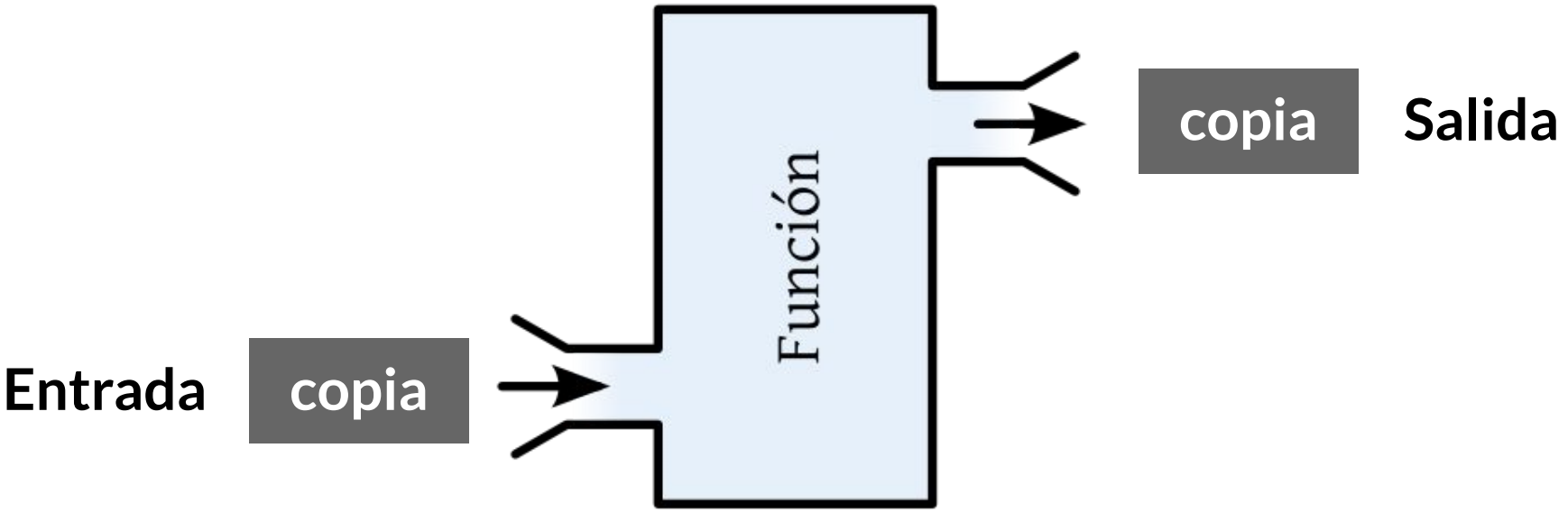
```
int incremento(int valor)
{
    valor = valor + 1;
    return valor;
}
```

Ejemplo II - Consola en vivo

```
/**
 * Ejemplo pasaje de argumentos
 */

#include <stdio.h>

int incremento(int valor)
{
    valor = valor + 1;
}

int main()
{
    int variable = 10;
    incremento(variable);
    printf("variable: %d\n", variable);
    return 0;
}
```

¿Y ahora?

—



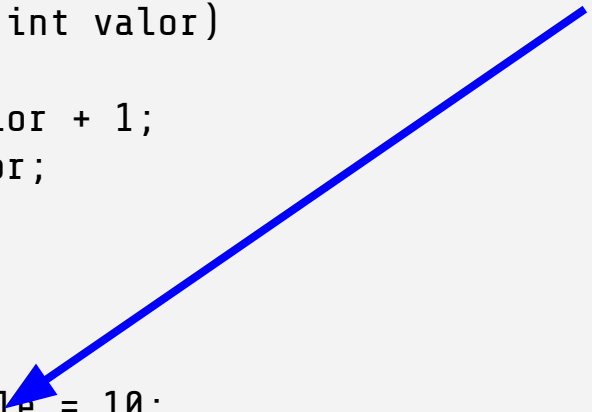
Si no guardamos la salida, no hay cambios

La llamada a la función es un r-value

Que es necesario guardar para que ‘tenga sentido’

Ejemplo III - Consola en vivo

```c
/**
 * Ejemplo pasaje de argumentos
 */

#include <stdio.h>

int incremento(int valor)
{
    valor = valor + 1;
    return valor;
}

int main()
{
    int variable = 10;
    variable = incremento(variable);
    printf("variable: %d\n", variable);
    return 0;
}
```



Hay que guardar la salida


¿Preguntas?

El preprocesador I

El preprocesador

Es como un ayudante que prepara el terreno antes de que el compilador real haga su trabajo

Para “constantes”

```
#define PALABRA valor
```

macros

Son substituciones textuales de `PALABRA` por el `valor`

(lo vamos a usar exclusivamente para constantes)

(por ejemplo)

```
#define TRUE 1
#define FALSE 0
#define DIVISION_POR_CERO -1
```

Usos

Substituir “números mágicos”

En particular, retornos de funciones.

Ayuda a que el código sea más claro

```
int resultado = division_positiva(dividendo, divisor);

if (resultado == DIVISION_POR_CERO)
{
    printf("¯\_(ツ)_/¯");
}
else if (resultado == DIVISION_NEGATIVA)
{
    printf("Error en los argumentos");
}
```


¿Preguntas?

Cadenas

secuencias de caracteres

Caracteres individuales

```
char caracter = 'A';
```

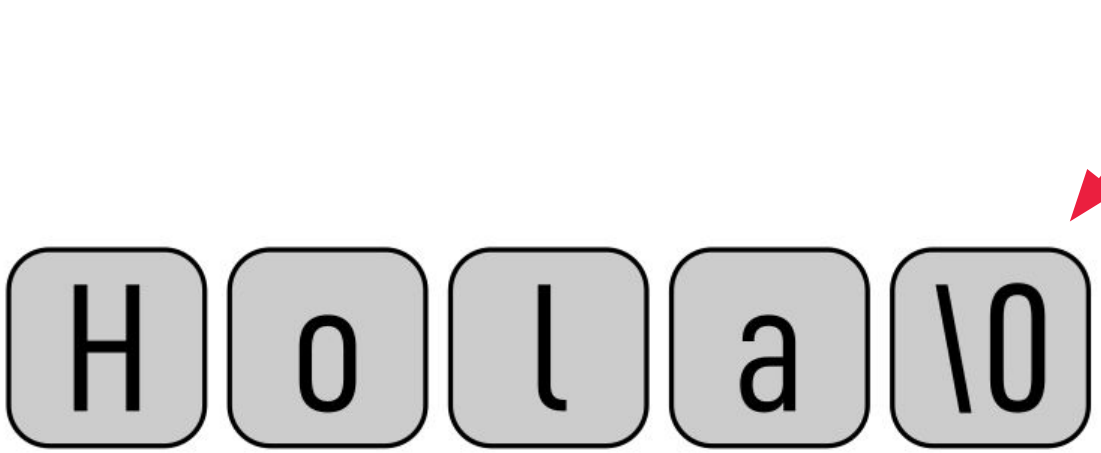
Cadenas

indica que es una secuencia de

```
char cadena[] = "hola";
```

¿cuántos caracteres tiene “Hola”?

Las cadenas en C tienen un carácter adicional



Qué indica donde termina

Cadenas

indica cuantos `char` guardar

```
char cadena[5] = "hola";
```

Podemos indicar nosotros el tamaño

{ Si no indicamos el tamaño, **va a lo justo y necesario** }

Consideraciones

```
char cadena[5] = "hola";
cadena[0] = cadena[3];
```

puede ser `l-value` y `r-value`

Consideraciones II

`cadena`

no puede ser `l-value` y `r-value`*

*¡Por ahora!

Lo que nos lleva a

Ejemplo cadenas I

```
#include <stdio.h>

int main()
{
    char cadena[] = "Hola";
    printf("cadena: %s\n", cadena);
    printf("posicion 0: %c\n", cadena[0]);
    printf("posicion 4: %c\n", cadena[4]);
    cadena[0] = cadena[3];
    printf("cadena: %s\n", cadena);
    return 0;
}
```

¿Cual es la salida?

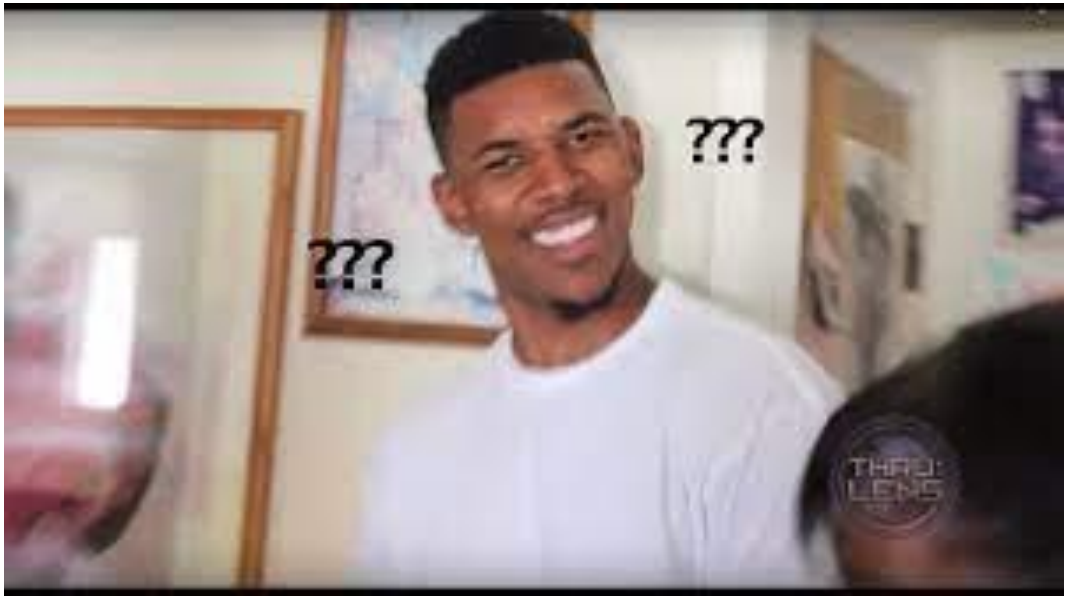
Hasta acá, todo mas o menos como uno espera

¿Y acá?

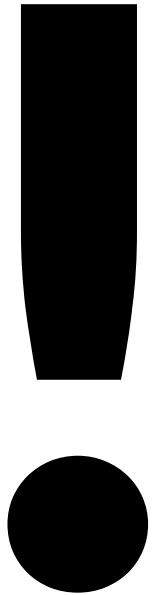
Ejemplo cadenas II

```c
#include <stdio.h>

int main()
{
    char antes[] = "XXXXX";
    char cadena[5] = "Hola";
    char despues[] = "YYYYY";
    printf("cadena: %s\n", cadena);
    printf("%c\n", cadena[-4]);
    printf("%c\n", cadena[6]);
    return 0;
}
```

¿Cual es la salida?



La memoria está mucho más cerca que en Python



Caractereristicas de una cadena

Largo de una cadena

Caracteres hasta el `\0`

Capacidad de una cadena

Cantidad de espacio reservado para la secuencia.

¿De qué largo son?

```
#include <stdio.h>

int main()
{
    char primera[] = "Hola Mundo";
    char segunda[6] = "Programacion";
    char tercera[] = "Adios Mundo\n";
    printf("primera: %s\n", primera);
    printf("segunda: %s\n", segunda);
    printf("tercera: %s\n", tercera);
    return 0;
}
```

Como podemos saber

```
#include <string.h>


int strlen ( char[] str );


Cuenta los caracteres hasta el \0
```

Operador `sizeof`

```
sizeof(variable)
```

nos da el tamaño en bytes de la variable

Pero usen

Macros de preprocesador

```
#define TAMANIO 5

char cadena[TAMANIO] = "hola";
```

Reserven *por lo menos* uno más de lo que escriban.

Recuerden:
el tamaño solo
puede ser definido
al compilar*

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

Cadenas y funciones

Declaren en el main

Salvo que sea una cadena temporaria, no declaren cadenas en las funciones

No retornen cadenas creadas en funciones

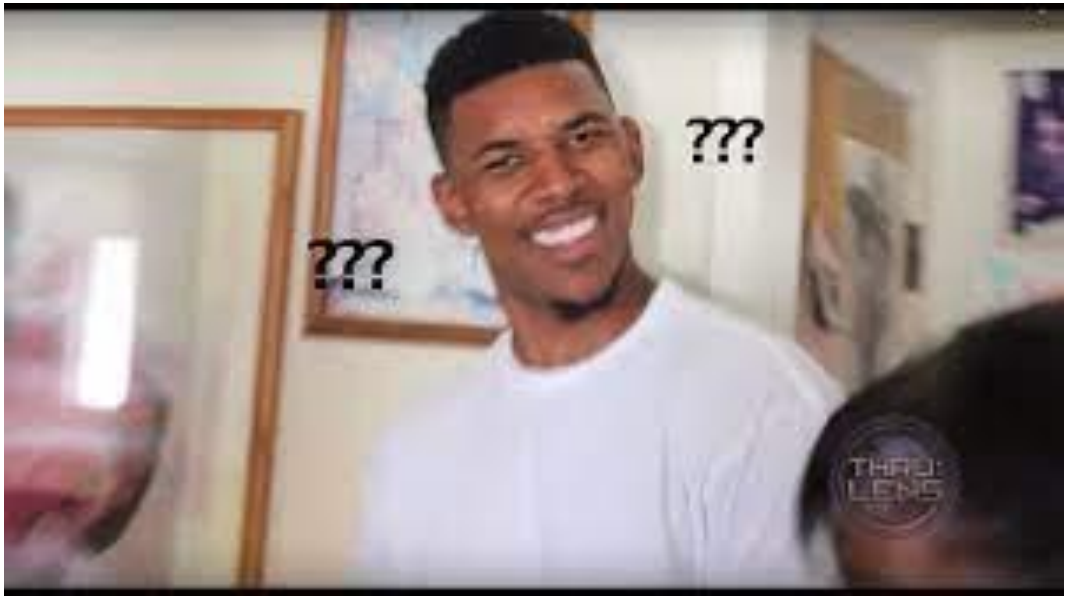
```
char[] generador()
{
    char lacadena[LARGO_CADENA] = “Hola mundo”;
    return lacadena;
}
```

La razón de esto la vamos a ver más adelante

¿Cuál es el resultado?

```
void generador(char lacadena[])
{
    lacadena[0] = “X”;
}

int main()
{
     char cadena[] = "Hola Mundo";
     printf("cadena: %s\n", cadena);
     generador(cadena);

     printf("cadena: %s\n", cadena);
     return 0;
}
```

¿queda igual?



¿Y el tamaño?

```
void generador(char lacadena[])
{
    lacadena[0] = “X”;
}

int main()
{
     char cadena[] = "Hola Mundo";
     printf("cadena: %s\n", cadena);
     generador(cadena);

     printf("cadena: %s\n", cadena);
     return 0;
}
```

¿queda igual?

Se lo podemos pasar como argumento

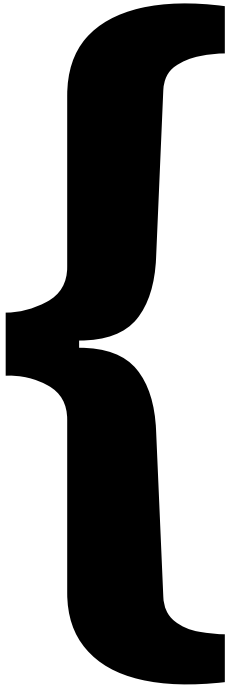
```
void generador(char lacadena[], int capacidad)
{
    lacadena[0] = “X”;
    //ahora sabemos la cadena mas larga (-1) que podemos alojar
}

int main()
{
     char cadena[] = "Hola Mundo";
     printf("cadena: %s\n", cadena);
     generador(cadena);

     printf("cadena: %s\n", cadena);
     return 0;
}
```

O usar el macro por todos lados

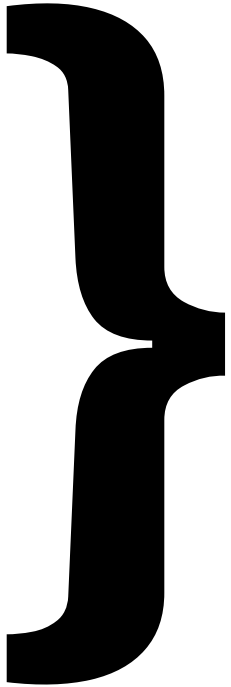
Podemos recorrer carácter a carácter.

```
void imprimidor(char lacadena[], int capacidad)
{
     int i = 0;
     int bandera = 1;
     while (i<capacidad && bandera)
     {
     printf("%d%c, ",i, lacadena[i]);
     if (lacadena[i] == '\0')
     {
          bandera = 0;
     }
     i++;
     }
     printf("\n");
}
```

Arreglos
`arrays`



Como las cadenas



```
tipo identificador[CAPACIDAD];
```

{ Pero sin ‘terminador’ (\0) }

Algunos ejemplos

```
#define ARREGLO_GRANDE 100
#define ARREGLO_CHICO 5

int arreglo[ARREGLO_GRANDE];
short arreglo_corto[ARREGLO_CHICO] = {1, 2, 3, 4, 5};
```

Si o si necesitamos indicar el tamaño

Aplican las mismas restricciones

No retornen un arreglo creado en una función.

```
int maximo(int identificador[], int tamano);
```


¿Preguntas?

unrn.edu.ar

UNRN Universidad Nacional de Río Negro

| unrionegro