

Funciones

UNRN

Universidad Nacional
de Río Negro

r20



Programación 1 es:

Martín René Vilugrón
mrvilugron@unrn.edu.ar

Teórico de los antiguos astronautas

Daniel Teira
deteira@unrn.edu.ar

Práctica comisión 1

Miguel Mariguín
mmariguin@unrn.edu.ar

Práctica comisión 2

Mauro Fermín
mafermin@unrn.edu.ar

Práctica comisión 3

Sobre las prácticas

A

**Es su
responsabilidad
completarla
completa**

B

**Se corregirá un
ejercicio de cada TP,
7 días después.
(asegúrense de hacer push)**

C

De necesitar
corrección, hay una
semana completa
luego de la revisión

D

**Los repositorios se
pueden crear hasta
una semana
después de su
emisión**

Por ejemplo - TP1

Lanzado el 7/8

Entrega el 14/8

Correcciones 21/8

**Es muy importante
que esté el
formulario completo
para registrar el
resultado del TP**

E

**La revisión incluye
las cuestiones de
estilo**

Estados posibles de un TP

-  TP OK
-  Correcciones
-  Entrega tarde
-  Rechazado
-  Sin Corregir
-  Sin entregar
-  Sin comenzar (sin commits)
-  Repositorio no creado

De la planilla de control y que recibirán por correo

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



Las cuestiones de estilo

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?

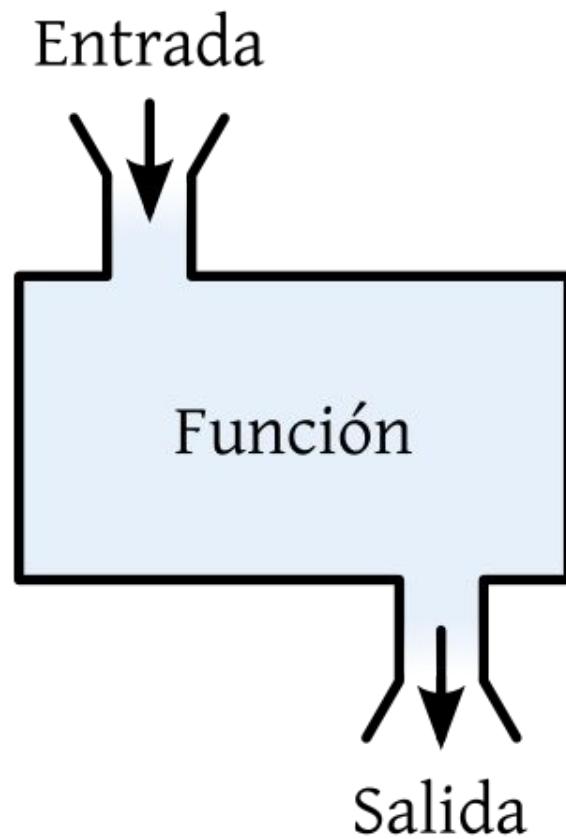


Funciones

parte 2

Concepto

Definición conceptual de una función



entrada

tipo identificador_funcion(tipo nombre, ...)

salida

**de entrada
tipo nombre
todos los necesarios**

**de salida
tipo
solo uno***

***mas adelante vamos a ver como nos podemos saltar esto**

Definición de funciones



tipo_retorno **identificador** (**lista_argumentos**)

Concretamente

**en donde cada ítem separado por coma de
lista_argumentos es**

tipo_argumento identificador

¿Y la salida?

```
return variable;
```

**funciona igual* que
Python**

Y la salida de la función

```
int suma(int op1, int op2)
{
    int suma = op1 + op2;
    return suma;
}
```

¿Puede no tener salida?

si, y es llamada
procedimiento

o

**Cuando no devuelve nada, se
indica con la palabra
reservada**

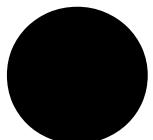
void

```
void mi_funcion()
{
    bloque;
}
```

Un ejemplo de procedimiento

Un ejemplo de procedimiento

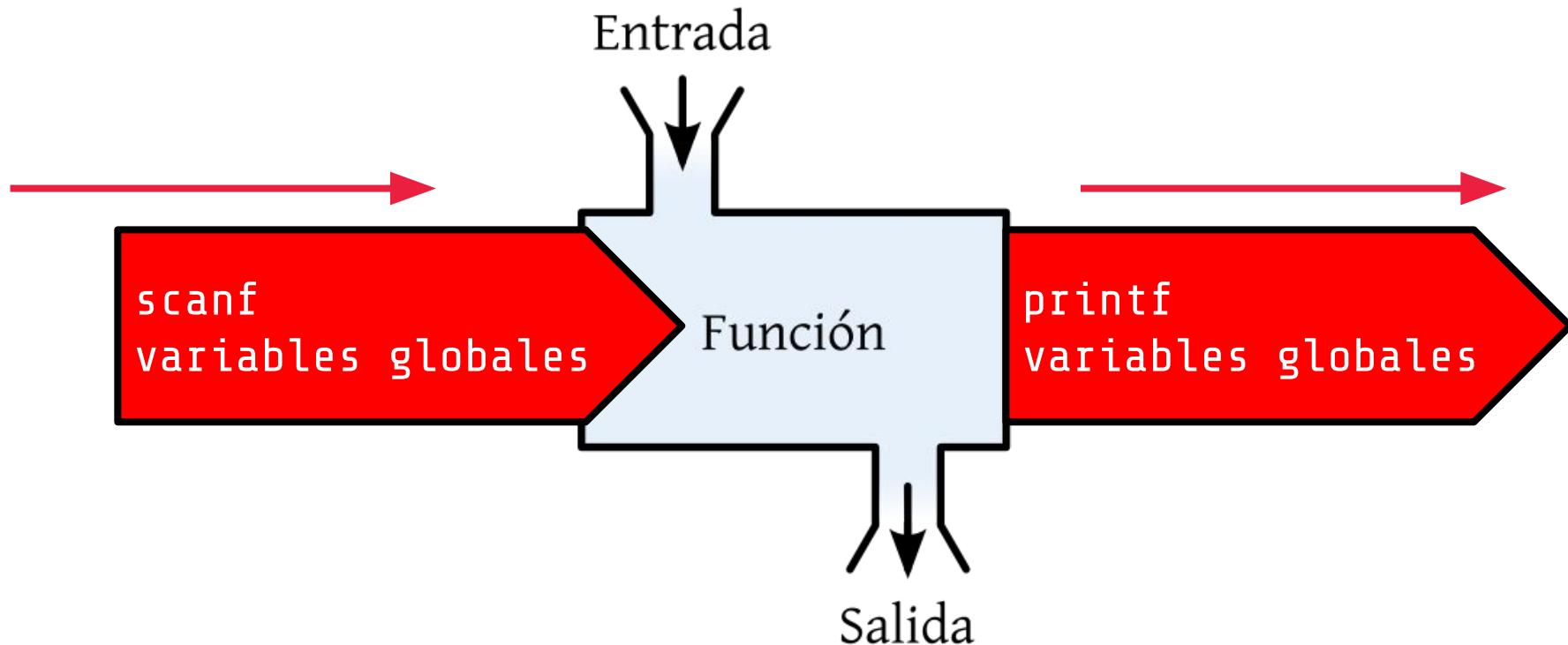
```
void muestra_mensaje()
{
    printf("Hola Mundo!\n");
}
```



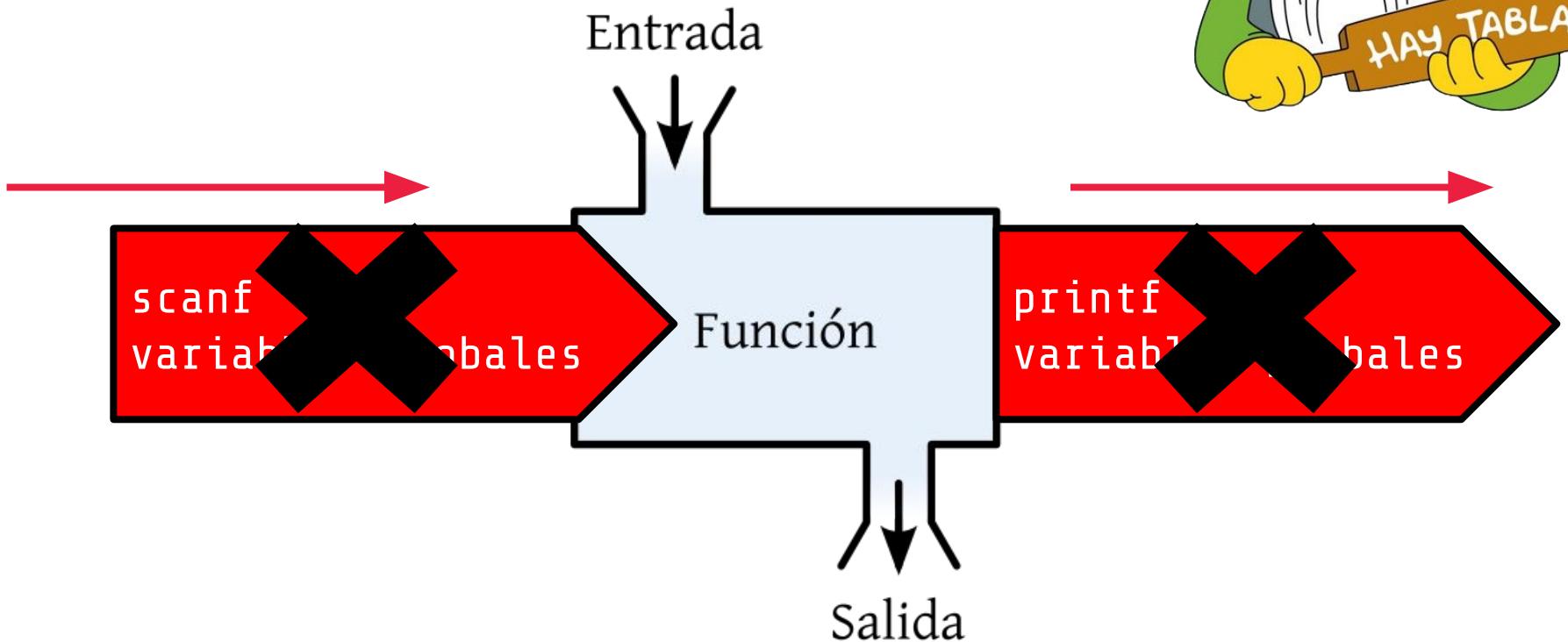
**Los argumentos
son la única
forma de pasar
información a
una función**

nope

Otras formas de pasar información



¡Pero no están permitidas!



Salvo para funciones como por ejemplo:

```
int leer_entero()
{
    int valor = 0;
    scanf("pasame un valor entero %d", &valor);
    return valor;
}
```

1

Sin usar variables globales

1



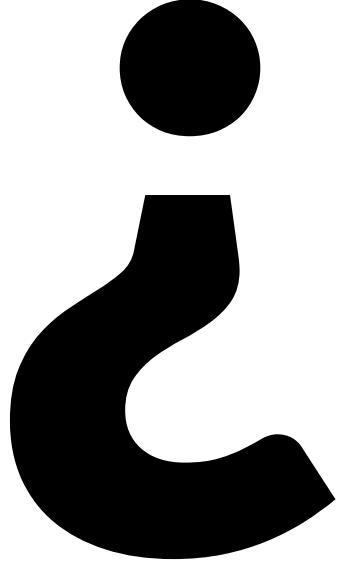
¿Preguntas?



—

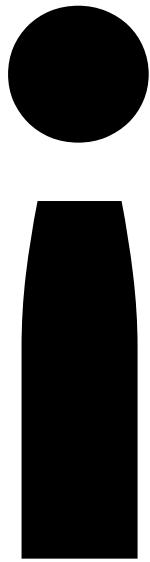
Encadenando funciones



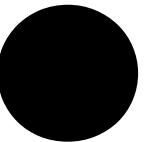
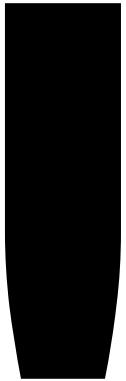


**una función
puede llamar
a otra**



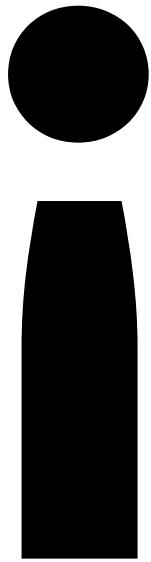


Sí

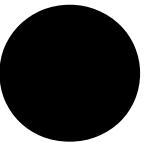
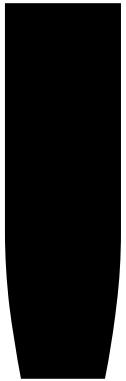


**¿El orden de las
funciones es
importante?**

***no**



Sí



```
void funcion_1()
{
    funcion_2();
}
```

```
void funcion_2()
{
    funcion_1();
}
```

**¡A la
terminal!**



Prototipos de función

**Es la función definida
pero sin cuerpo.**

void mi_funcion();

```
void funcion_1();  
void funcion_2();
```

```
void funcion_1()  
{  
    funcion_2();  
}  
void funcion_2()  
{  
    funcion_1();  
}
```

Un ejemplo más
concreto

Un ejemplo más completo

```
void saludo();
```



Es necesario

```
int main()
{
    saludo();
    return 0;
}
```



Porque se usa antes de
su implementación

```
void saludo()
{
    printf("Hola Mundo");
}
```

El ‘prototipo’ no es necesario en este caso

```
void saludo()
{
    printf("Hola Mundo");
}
```

```
int main()
{
    saludo();
    return 0;
}
```

Acá se implementa
antes de usarse

Se usa pero ya esta
implementada

Cuestiones generales

1

**Todas las funciones
con documentación
completa.**

0

Descripción de la función

```
/*
 * Toda función debe tener un comentario sobre que
 * describa que es lo que hace.
 * Indicando que hace cada argumento.
 * Esta función se encarga de sumar dos numeros enteros
 * @param termino1 es el primer término de la suma
 * @param termino2 es el segundo término a sumar
 * @returns la suma de ambos términos
 */
int suma(int termino1, int termino2);
```

**Por
qué**

Ayuda a entender las responsabilidades de lo que programamos

**Y aquellas funciones
que terminan
describiendo con un
“y”**

**probablemente
sean *dos* funciones**

Una función que probablemente sean dos

Esta función se encarga de algo y otra cosa

**Durante el
cuatrimestre
vamos a ir viendo
ejemplos**



escribiendo el código



documentando el código

1

**Siempre que sea
possible*, una
responsabilidad por
función.**

2



¿Preguntas?



Alcance de variables

Ámbitos/alcance/scope principales

```
#include <stdio.h>

int global = 10;

int suma(int n, int m)
{
    int local = n + m;
    static int compartida;
    return local;
}
```

Global

Las variables declaradas fuera de cualquier función tienen alcance global. Esto significa que son accesibles desde cualquier parte del programa.

¿Dónde se declaran?

```
int variable_global = 10;

int main() {
    printf("La variable global es %d\n", variable_global);

    return 0;
}

void procedimiento(){
    printf("La variable_global existe acá %d\n", variable_local);
}
```



"We don't do that here"

Local

Las que venimos utilizando;
Las variables declaradas dentro de una función tienen alcance local. Esto significa que solo son accesibles desde dentro de la función en la que se declaran.

```
int main() {
    int variable_local = 20;
    printf("La variable local vale %d\n", variable_local);
    return 0;
}

void procedimiento(){
    printf("La variable_local no existe acá %d\n", variable_local);
}
```

Bloques {}

Las variables declaradas dentro de un bloque tienen alcance de bloque. Esto significa que solo son accesibles desde dentro del bloque en el que se declaran.

Las llaves pueden ir ‘sueltas’ :-)

```
int main()
{
    {
        int variable_bloque = 30;

        // La variable variable_bloque solo es accesible desde dentro del bloque
        printf("El valor de la variable de bloque es %d\n", variable_bloque);
    }

    // La variable variable_bloque no es accesible fuera del bloque
    printf("El valor de la variable de bloque es %d\n", variable_bloque);

    return 0;
}
```

Valores compartidos

Un valor static sigue estando en el alcance de la función

```
int incremento(int numero)
{
    static int veces;
    veces = veces + 1;
    printf("llamamos %d veces \n", veces);
    return numero + 1;
}
```

¡Un uso piola!

```
{  
    static int veces;  
    veces = veces + 1;  
    printf("*** DEBUG - se llamo %d veces \n", veces);  
}
```

Como un bloque aislado dentro de una función.

1

No están permitidas las variables globales

3

Pasaje de argumentos

Cómo viaja la información en los argumentos

e j e m p l o

Ejemplo I - Consola en vivo

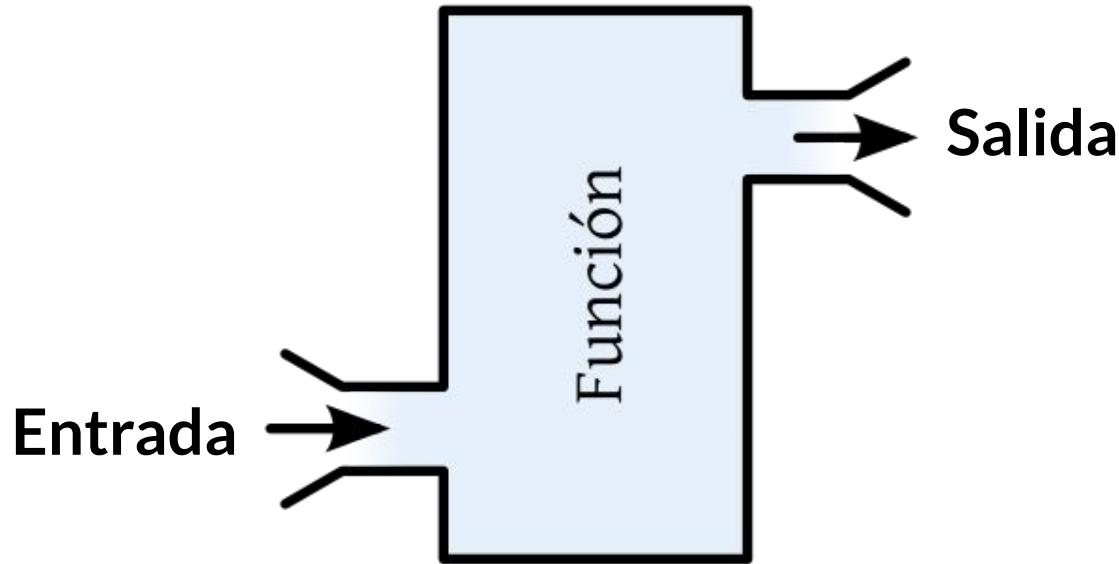
```
/**  
 * Ejemplo pasaje de argumentos  
 */  
  
#include <stdio.h>  
  
int incremento(int variable)  
{  
    variable = variable+ 1;  
}  
  
int main()  
{  
    int variable = 10;  
    incremento(variable);  
    printf("variable: %d\n", variable);  
    return 0;  
}
```

Ejemplo II - Consola en vivo

```
* Ejemplo pasaje de argumentos
*/
#include <stdio.h>

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

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?

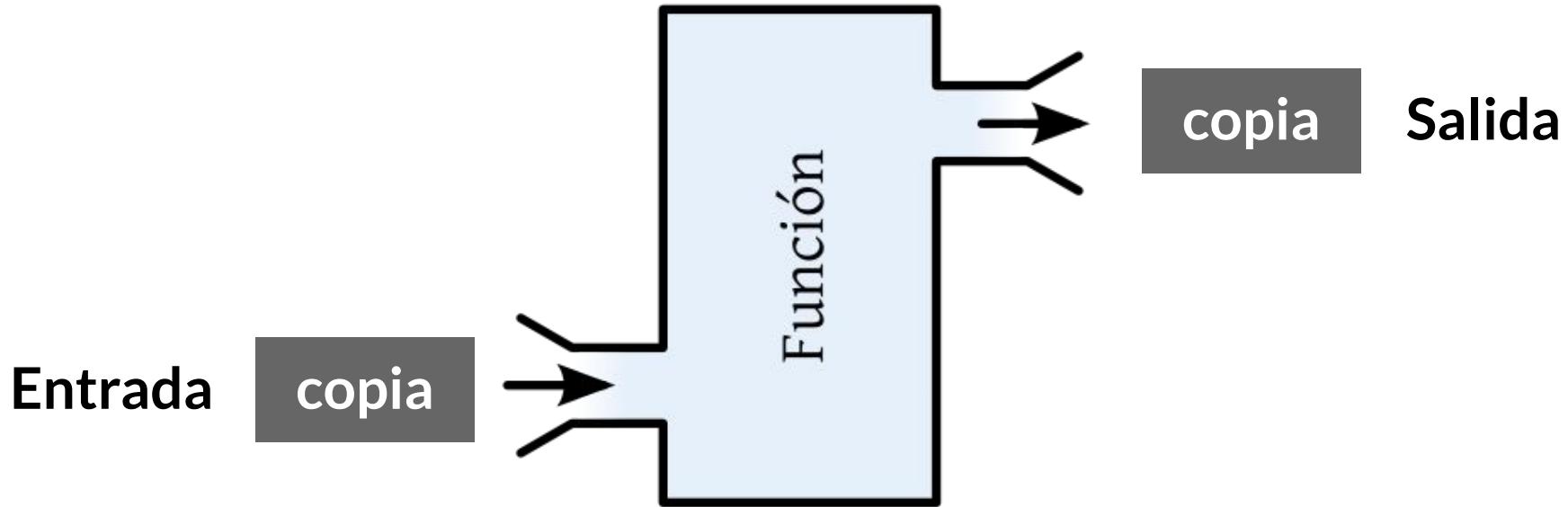
```
int incremento(int valor)
{
    valor = valor + 1;
    return valor;
}
```

Ejemplo III - 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 IV - Consola en vivo

```
* 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

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



preprocesador

parte 1



El preprocesador

Es un programa separado, que prepara el terreno antes de que el compilador haga su trabajo

Para “constantes”

```
#define PALABRA valor
```

“Macros”

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

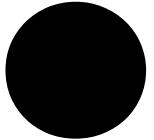
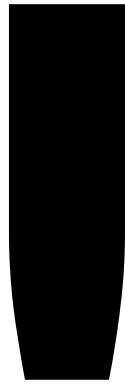
```
int resultado = division_positiva(dividendo, divisor);

if (resultado == DIVISION POR CERO)
{
    printf("-\u263a/-");
}
else if (resultado == DIVISION NEGATIVA)
{
    printf("Error en los argumentos");
}
```

```
#include <libreria>
#include "archivo"
```



**Puede ser
cualquier
archivo**



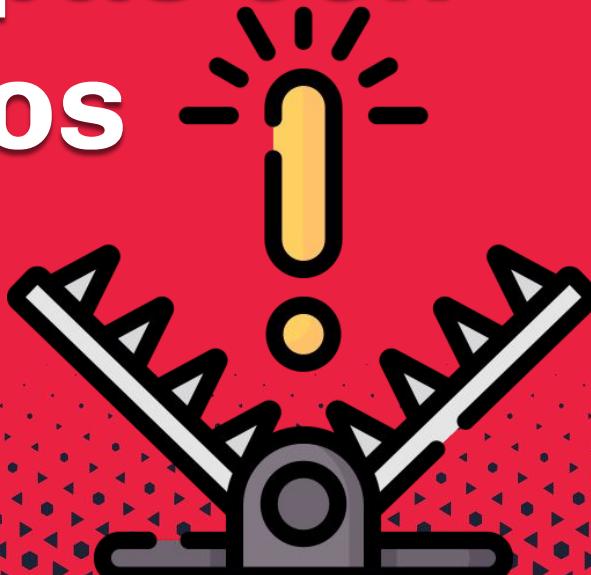
**Vamos a volver a
este tema más
adelante**

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



Trampas con macros



Cadenas strings



—

¿Que son?

—

Cadenas

secuencias de caracteres

caracteres individuales

char caracter = 'A';

Caracteres ASCII de control			Caracteres ASCII imprimibles		
00	NULL	(carácter nulo)	32	espacio	64
01	SOH	(inicio encabezado)	33	!	65
02	STX	(inicio texto)	34	"	66
03	ETX	(fin de texto)	35	#	67
04	EOT	(fin transmisión)	36	\$	68
05	ENQ	(consulta)	37	%	69
06	ACK	(reconocimiento)	38	&	70
07	BEL	(timbre)	39	'	71
08	BS	(retroceso)	40	(72
09	HT	(tab horizontal)	41)	73
10	LF	(nueva línea)	42	*	74
11	VT	(tab vertical)	43	+	75
12	FF	(nueva página)	44	,	76
13	CR	(retorno de carro)	45	-	77
14	SO	(desplaza afuera)	46	.	78
15	SI	(desplaza adentro)	47	/	79
16	DLE	(esc.vínculo datos)	48	0	80
17	DC1	(control disp. 1)	49	1	81
18	DC2	(control disp. 2)	50	2	82
19	DC3	(control disp. 3)	51	3	83
20	DC4	(control disp. 4)	52	4	84
21	NAK	(conf. negativa)	53	5	85
22	SYN	(inactividad sínc)	54	6	86
23	ETB	(fin bloque trans)	55	7	87
24	CAN	(cancelar)	56	8	88
25	EM	(fin del medio)	57	9	89
26	SUB	(sustitución)	58	:	90
27	ESC	(escape)	59	;	91
28	FS	(sep. archivos)	60	<	92
29	GS	(sep. grupos)	61	=	93
30	RS	(sep. registros)	62	>	94
31	US	(sep. unidades)	63	?	95
127	DFI	(suirimir)		-	

¡Secuencias de escape!

\n - salto de línea

\t - tabulador

\\ - la barra invertida misma

\ " - la comilla doble

\ ' - la comilla simple

Como darle un significado alternativo a los mismos caracteres

Cadenas

indica que es una secuencia de



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

—

¿cuántos caracteres tiene “Hola”?

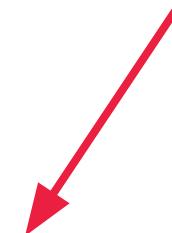
Las cadenas en C tienen un carácter adicional

H o l a \0

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;
}
```

¿Cuál es la salida?

**Hasta acá, todo mas
o menos como uno
espera**

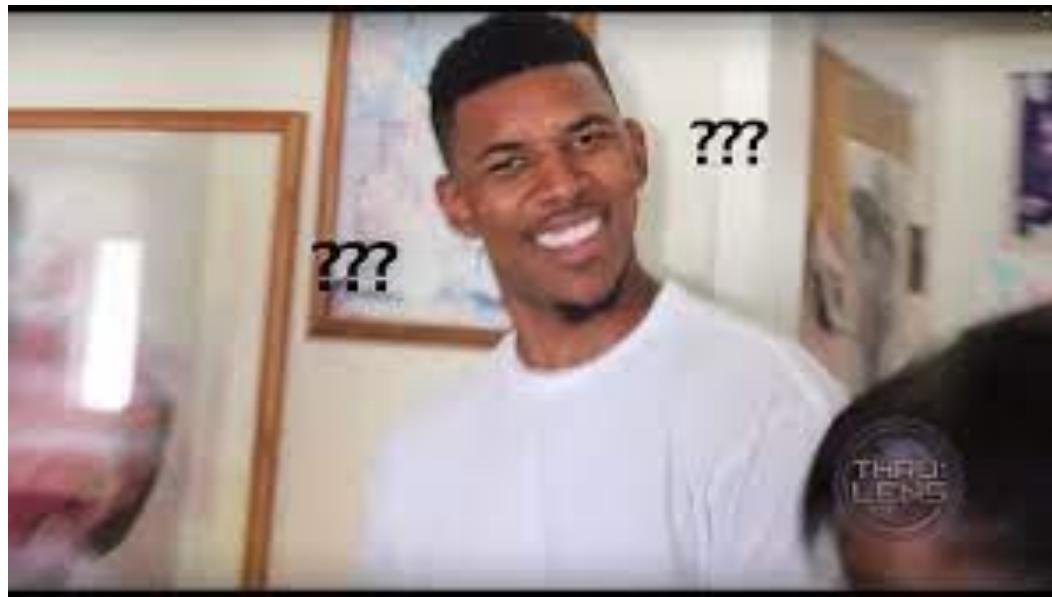
¿Y acá?

Ejemplo cadenas II

```
#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;
}
```

¿Cuál es la salida?





**La memoria está
mucho más cerca
que en Python**

Características de una cadena

Largo de una cadena

Caracteres hasta el \0

Capacidad de una cadena

Cantidad de espacio reservado para la secuencia.

¿Cuál es el largo y la capacidad de cada variable?

```
#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

`size_t sizeof(variable)`

nos da el tamaño en bytes de la variable

Valor tipo `size_t`

Un valor entero sin signo de por lo menos 16-bits que representa un tamaño en memoria



¿Preguntas?



Al momento de
definir una cadena

Macros de preprocesador

```
#define TAMANIO 15
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
restrictión**



¿Preguntas?



Arreglos

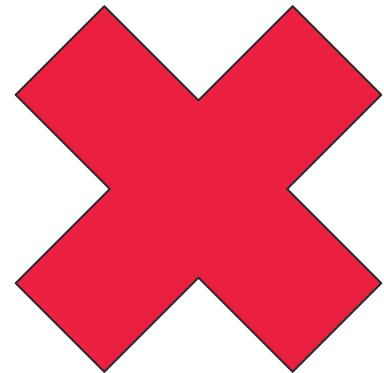
Cadenas y funciones

Declarén en el main

Salvo que sea una cadena temporaria, no declarén 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

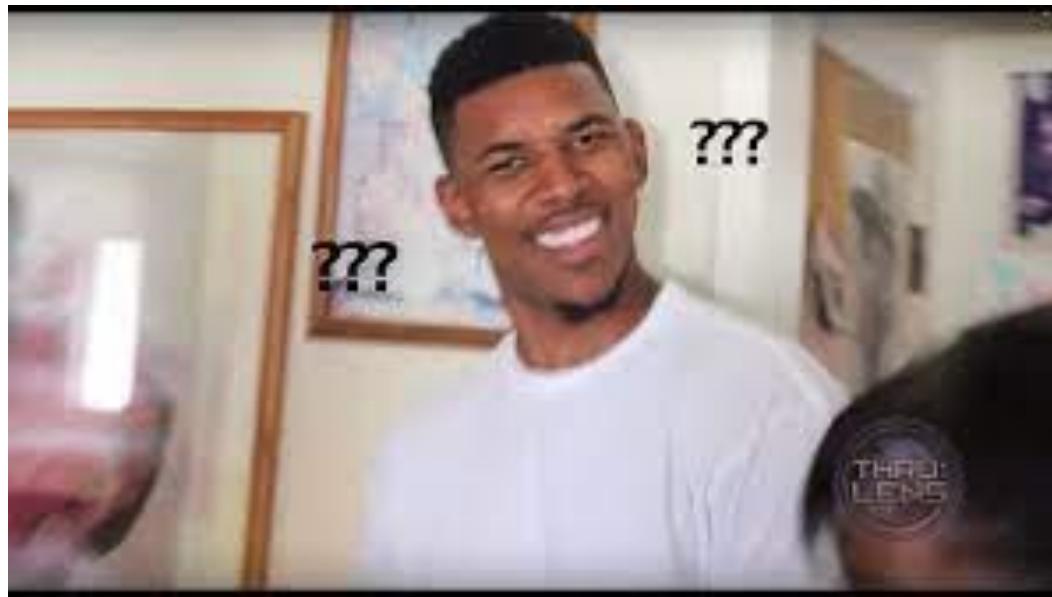
¿Cuál es el resultado?

```
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;
}
```

¿quedó igual?



**¡A la
terminal!**



¿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(int capacidad, char lacadena[])
{
    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(int capacidad, char lacadena[])
{
    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");
}
```



¿Preguntas?



¿Qué pasa con sizeof en un argumento así?

```
void tamanio(char lacadena[100])
{
    printf("el tamaño de lacadena es %d\n", sizeof(lacadena));
}
```

¿Que muestra este printf ?

**¡A la
terminal!**



**No se fíen de sizeof en
arreglos como argumentos
de funciones***

***Ya vamosa
ver por qué**

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



Arreglos arrays

Como las
cadenas

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};
```

En funciones
si o si
necesitamos indicar el
tamaño

Aplican las mismas restricciones

No retornen un arreglo creado en una función.



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

Preferentemente en este orden

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



Una observación sobre el tamaño de los arreglos y cadenas

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

***más adelante vamos a ver como liberarnos de esta
no atracción**

Pero entonces, ¿que es arreglo?

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

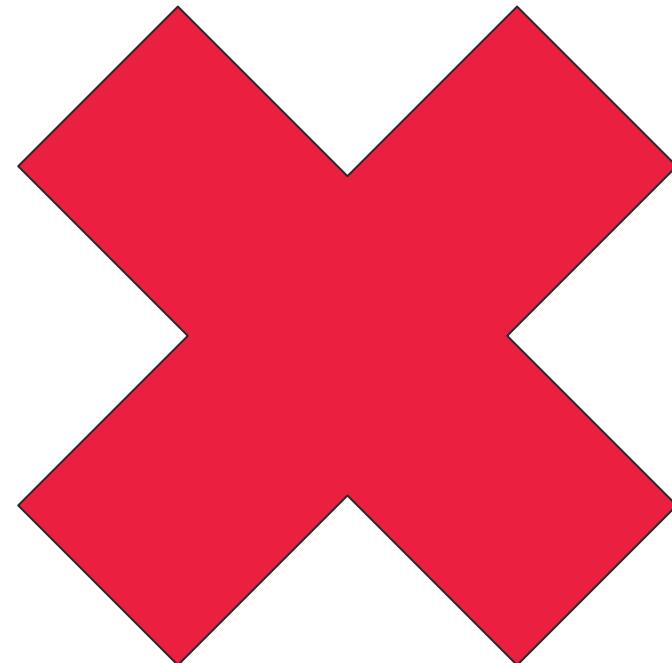
```
int arreglo[SIZE];
```

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

Pero con una
variable !
funciona

¡Funciona!, pero no es correcto

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



Ya vamos a ver

por qué no

es correcto

1

**Los arreglos solo
pueden ser creados
de un tamaño fijo al
compilar**

4



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

TP2

Funciones

Hasta la próxima



unrn.edu.ar

UNRN

Universidad Nacional
de Río Negro



| unrnionegro