

Funciones en C

¿Qué es una función?

Una **función** es un bloque de código con un **nombre** que realiza una tarea específica. Se puede **invocar (llamar)** desde otras partes del programa.

Sirve para:

- Reutilizar código
Organizar un programa en partes más pequeñas
- Evitar duplicación

```
int sumar(int a, int b) {  
    return a + b;  
}
```



Estructura de una función

```
tipo_de_retorno nombre_funcion(tipo_param1 nombre1, tipo_param2 nombre2) {
```

```
    // cuerpo de la función
```

```
}
```

```
float promedio(int suma, int cantidad) {
```

```
    return (float)suma / cantidad;
```

```
}
```



Tipos de retorno

El **tipo de retorno** indica el tipo de dato que devuelve la función.

Si no devuelve nada, se usa void.

```
int obtenerEdad();      // devuelve un entero
```

```
char obtenerLetra();   // devuelve un carácter
```

```
void mostrarMenu();    // no devuelve nada
```



¿Qué pasa si una función no retorna nada?

- Si una función está declarada como void, **no puede usar return con un valor.**
- Se usa para **acciones** (mostrar mensajes, modificar variables externas, etc.)
- Si no se retorna nada en una función que debería, el comportamiento es indefinido.

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

Paso de parámetros: por valor vs por referencia

En C, todos los argumentos se pasan por valor. Pero hay una excepción importante con arrays y punteros.

En C, los parámetros **se pasan normalmente por valor a funciones** y subrutinas internas. Sin embargo, se puede pedir que se pasen por referencia utilizando & u punteros.

Los arreglos de estructura son automáticamente pasados en llamadas por referencia.

Los **cambios realizados** a los argumentos pasados por **referencia** en la función llamada **tienen efecto** en la función que llama.

Generar una **copia implica** un **costo** adicional en términos de memoria



Paso de parámetros: por valor vs por referencia

En C, todos los argumentos se pasan por valor. Pero hay una excepción importante con arrays y punteros.

En C, los parámetros **se pasan normalmente por valor a funciones** y subrutinas internas. Sin embargo, se puede pedir que se pasen por referencia utilizando & u punteros.

Los arreglos de estructura son automáticamente pasados en llamadas por referencia.

Los **cambios realizados** a los argumentos pasados por **referencia** en la función llamada **tienen efecto** en la función que llama.

Generar una **copia implica** un **costo** adicional en términos de memoria



Paso por valor

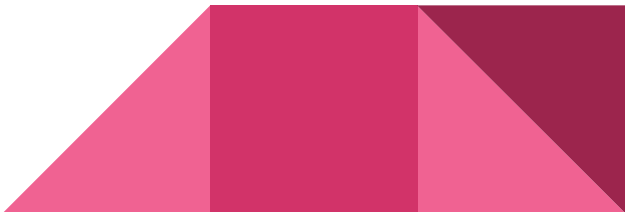
Se pasa **una copia** del valor. La función **no puede modificar la variable original**.

```
void doblar(int x) {  
    x = x * 2;  
}
```

llamada:

```
int a = 5;
```

```
doblar(a); // 'a' sigue siendo 5
```



Paso por referencia

Se pasa la dirección de memoria de una variable. Permite modificar la original.

```
void doblar(int * x) {  
    *x = *x * 2;  
}
```

llamada:

```
int a = 5;
```

```
doblar(&a); // 'a' ahora va a ser 10
```



Variables locales y globales

Variables locales:

- Se declaran dentro de una función.
- Solo existen mientras la función se ejecuta
- No se puede acceder a ellas desde fuera.

```
void ejemplo() {  
    int x = 10; // local  
}
```



Variables locales y globales

Variables globales:

- Se declaran fuera de cualquier función.
- Se puede acceder desde cualquier parte del programa.
- **Se debe usar con cuidado**, porque cualquier función puede modificarla.

```
int contador = 0; // global
```


```
void aumentar() {  
    contador++;  
}
```



El caso especial de los arrays

Cuando se pasa un array a una función, **se pasa la dirección del primer elemento** (como si fuera un puntero). Entonces, **la función puede modificar el contenido del array original**.

```
void imprimirArray(int arr[ ], int n) {  
    for (int i = 0; i < n; i++) {  
        printf("%d ", arr[i]);  
    }  
}  
void ponerEnCero(int arr[ ], int n) {  
    for (int i = 0; i < n; i++) {  
        arr[i] = 0; // modifica el array original  
    }  
}
```



Prototipos de las funciones

En C, una función debe estar **declarada antes de ser usada**, o al menos su **prototipo** debe estar declarado arriba.

```
int sumar(int, int); // prototipo
```

```
int main() {  
    int r = sumar(3, 4);  
}
```



¿Para qué usamos funciones void?

Mostrar mensajes (printf)

Cambiar variables globales

Modificar arrays o estructuras pasadas por referencia

Ejecutar tareas sin devolver resultado



Resumen

Concepto	Explicación
Función	Bloque de código con un nombre
Parámetros	Datos que recibe la función
Retorno	Dato que devuelve la función
void	Función sin retorno
Local vs Global	Ámbito de la variable
Paso por valor	Copia del dato
Paso por referencia	Dirección de memoria (puntero)
Array como parámetro	Se pasa la dirección (se puede modificar)