

# Ayudantía N° 1 Metodología de Programación

Eduardo Toloza C.

## REPASO

### ¿Que es una función?

Una función es un tipo subalgoritmo dentro de un algoritmo, es el término para describir una secuencia de órdenes que hacen una tarea específica de una aplicación más grande. Las funciones pueden recibir datos de entrada y retornar datos de salida.

#### Implementación:

**Tipo\_de\_Dato Identificador\_Funcion** (parámetros)

```
{
  Conjunto de Instrucción, Operaciones y/o Tareas
}
```

#### > Ejemplo de una función cualquiera en C (sin retorno):

```
void mostrarVector (int V[20], int d)
{
  for (int i = 1 ; i <= d ; i++)
  {
    printf ("%d",V [i]);
  }
}
```

Podemos concluir que:

El Tipo de dato de la función es:	<b>void</b>
El Nombre o Identificador de la función es:	<b>mostrarVector</b>
Los parámetros formales :	El vector <b>v[20]</b> y el entero <b>d</b>
El valor que retorna esta función es:	<b>Ninguno</b>

Como la función *mostrarVector* es de tipo void, quiere decir que no retornara ningún valor de salida. ¿Pero que sucede cuando el tipo de dato de la función a void? Continuación veremos un ejemplo de una **función con retorno**.

> **Ejemplo de una función cualquiera en C** (con retorno):

```
float areaC(float radio)
{
    float pi=3.14;
    float area=pi*radio*radio;
    return area;
}
```

Podemos concluir que:

El Tipo de dato de la función es:

**float (Real)**

El Nombre o Identificador de la función es:

**areaC**

Los parámetros formales son:

La variable flotante radio

El valor que retorna esta función es:

**área (pi\*radio\*radio ; tipo Real)**

**¿Como se hace una llamada a una función?**

Para el *primer ejemplo*, donde la función **NO** retornaba ningún valor el método de llamada es por ejemplo de la siguiente manera:

```
mostrarVector (vector, 10) ;
```

Como podemos apreciar nuestros dos parámetros reales son el vector de nombre **vector** y el entero **10**.

Para el *segundo ejemplo*, donde la función **SI** retornaba valor, el método de llamada es el siguiente:

```
float area=areaC(20.5) ;
```

Acá creamos la variable que se **encargara de almacenar el valor que retorne la función areaC**, por ultimo como asignación hacemos el llamado a la función y pasamos un numero decimal cualquiera como parámetro, en este caso el entero **20.5 sería nuestros parámetro reale**, entonces el valor de la **variable área** debiera ser **1319.585083**.

No solo es necesario pasar los valores directamente como parámetros, también podemos pasarlos como variables, como se hace continuación:

```
float radio=20.5;
float area=areaC (radio) ;
```

Lo que sería exactamente lo mismo que hicimos anteriormente.

## Paso de Parámetros

Los parámetros formales son una lista de tipos e identificadores que se sustituirán por los parámetros reales y se usarán como variables dentro de la función.

Los parámetros se pasan normalmente por **valor**, pero también se pueden pasar por **referencia**. El paso de parámetros por **referencia** admite dos sintaxis ligeramente diferentes en C: anteponiendo el operador \* (asterisco) al nombre del parámetro o anteponiendo el operador & (este último solo válido para C++).

### >> Paso de Parámetros por Valor

```
int funcion (int x, int y)
```

En este caso la función recibirá únicamente el valor de los dos parámetros, **x** e **y**. La función podrá **utilizar** esos valores a lo largo de su código, e incluso podrá **cambiarlos**. Pero cualquier cambio en **x** e **y** **no afectará a los parámetros reales**.

### >> Paso de Parámetros por Referencia

```
int funcion (int x, int *y)
```

En esta función, el paso del parámetro “x” es por valor y el del parámetro “y”, por referencia. El paso de parámetros por referencia recibe la dirección de memoria del parámetro real. Entonces al hacer algún en este valor, el cambio **afectará el valor de los parámetros reales**.

**Importante:** Cada vez que se vaya a usar el parámetro pasado por referencia *dentro del código de la función*, será necesario acompañarlo del asterisco. Por ejemplo:

```
int funcion (int x, int *y)
{
    *y = 5;
    x = 17 + *y;
}
```

Por último, *también en la llamada a la función* hay que indicar explícitamente si alguno de los parámetros se está pasando por referencia, utilizando el operador &. Por lo tanto, **para llamar a la función del ejemplo anterior con los parámetros reales va1 y va2** habrá que escribir:

```
resultado = funcion(va1, &va2);
```

**Ejercicio en C con ambos casos:**

```
#include <stdio.h>

/* Paso de parámetros por valor.
   En este ejemplo, esta función no tendrá el efecto deseado, porque las variables
   del programa principal no se verán afectadas.
*/
void intercambia_valor(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

/* Paso de parámetros por referencia.
   Esta función sí que consigue intercambiar los valores de las variables
   del programa principal.
*/
void intercambia_referencia(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

// Programa principal
int main()
{
    int var1 = 30, var2 = 90;
    printf("Datos antes de llamar a la funcion: var1 = %i, var2 = %i\n", var1, var2);
    intercambia_valor(var1, var2);
    printf("Datos despues de llamar intercambia_valor: var1 = %i, var2 = %i\n", var1, var2);
    intercambia_referencia(&var1, &var2);
    printf("Datos despues de llamar intercambia_referencia: var1 = %i, var2 = %i\n", var1, var2);
    getchar();
    return 0;
}
```