

# FUNCIONES EN C

Este documento es una extracción del libro Programación en C, metodología, estructura de datos y objetos, de Luis Joyanes Aguilar e Ignacio Zahonero Martínez. Fue recopilado por la MSc. Ana María Salgado G. y complementado con ejercicios obtenidos de otros textos e Internet.

## FUNCIONES

Las funciones en C desempeñan el papel de las subrutinas o procedimientos en otros lenguajes, esto es, permiten agrupar una serie de operaciones de tal manera que se puedan utilizar más tarde sin tener que preocuparnos por cómo están implementadas, simplemente sabiendo lo que harán.

El uso de funciones es una práctica común y recomendable ya que permite modularizar nuestro código, simplificando así el desarrollo y la depuración del mismo. Para utilizar funciones en un programa es necesario declararlas previamente al igual que las variables (en este caso indicaremos los argumentos de entrada y su tipo, y el tipo del valor que devolverá) y definir las operaciones que contiene.

Cada función tiene un nombre y una lista de argumentos que recibe la función. De manera general se puede dar a la función el nombre que se quiera con la excepción de main que se reserva en C para indicar el inicio de un programa, los nombres para las funciones utilizan las mismas reglas de los identificadores.

En C la declaración de una función tiene la siguiente estructura:  
 tipo\_devuelto nombre\_funcion (argumentos);

**Y su definición:**

```
tipo_devuelto nombre_funcion (argumentos)
{
    sentencias;
}
```



**Nota:** A la hora de definir una función que **no** acepte argumentos escribiremos void en vez de los argumentos.

**tipo\_devuelto:** Se refiere al tipo de dato que retorna la función, el mismo que puede ser int, char, o float, o un puntero a cualquier tipo C, o un tipo struct. Ej:

```
int calculo_kilometraje(int litros, int kilómetros);
char mayusculas(char car);
float DesvEst(void);
struct InfoPersona BuscarRegistro(int num_registro);
```

En caso de que la función no retorne ningún valor se utiliza el especificador de tipo void. Ej:  
 void VisualizarResultados(float Total, int num\_elementos);

Si se omite el tipo de retorno C asume que se trata de un int.  
 verResultados(float Total, int longitud);

Aunque el uso de int es opcional, por razones de claridad y consistencia se recomienda su uso. Así, la función anterior se puede declarar también:  
 int VerResultados(float Total, int longitud);

**Parámetros o argumentos:** La sintaxis utilizada es en base a declaraciones de prototipos de funciones. En ella se incluye información acerca de los tipos de datos que la función espera como argumento. La declaración de la función se hace antes de la función main, esto es lo recomendable con el fin de poder visualizar todas las declaraciones de las funciones a utilizar; antes de main aunque es posible ubicarla después de main. Ejemplo:

```
int centrar(. . . );    main()
main()                 {
{                       int centrar(. . . );
    . . .
    . . .
}
```

Si la función no tiene parámetros formales se debe poner void ejemplo:

```
void líneas(void)
```

La palabra void al inicio de la función indica que no retorna ningún valor.

La sintaxis es de acuerdo al prototipo de funciones ejemplo:

**Definición:**

```
double división (double valor1, int valor)
```

Esta definición es idéntica a la declaración pero no lleva punto y coma al final ( ; ) de la misma. Los identificadores de los parámetros y sus tipos se incluyen en la cabecera de la función. No hay declaración separada para los parámetros.

```
double división1 (double valor1, int valor2);    /*Declaración del prototipo */

main()
{
    /* Cuerpo del programa principal */
}

double división1 (double valor1, int valor2);
{
    /* Cuerpo de la función */           /* Función con prototipo */
    . . .
}
```

**Resultados de una función:**

Una función puede devolver un único valor. El resultado se muestra con una sentencia **return** cuya sintaxis es:

```
return(expresión);
return;
```

Esta concluye la ejecución de la función y devuelve el control al medio que lo llamó, si la proposición return contiene una expresión el valor de ésta también regresa al medio que hizo la llamada; además el valor retornado se convertirá al tipo dado por el especificador de tipo de la función.

El valor devuelto(**expresión**) puede ser cualquier tipo de dato excepto una función o un array. Se pueden devolver valores múltiples devolviendo un puntero o una estructura.

En una función puede haber cero o más proposiciones `return`; si no hay ninguna, el control vuelve al medio que hizo la llamada al llegar a la llave que delimita el cuerpo de la función.

```
double valor_abs(double x)
{
    if (x>0.0)
        return(x);
    else
        return(-x);
}
```

Pongamos un ejemplo, un programa en el que queremos incluir una función que devuelva el factorial de un número:

La declaración: `int factorial(int a);`

debe coincidir con la definición de la función factorial que aparece posteriormente, si no coincide obtendremos un error en la compilación del programa. El valor que calcula la función `factorial()` se devuelve por medio de la sentencia `return`, ésta puede estar seguida de cualquier expresión o aparecer sola. En tal caso la función no devuelve ningún valor y al llegar a ese punto simplemente se devuelve el control a la función desde la que se invocó.

Si el tipo de retorno es ***void***, la sentencia ***return*** se puede escribir como ***return***; sin ninguna expresión de retorno, o bien de modo alternativo se puede omitir la sentencia ***return***.

```
void func1(void)
{
    puts("Esta función no devuelve valores");
}
```

## LLAMANDO A UNA FUNCION

Para llamar a una función se escribe en sitio adecuado el nombre de la función seguido de paréntesis abierto y cerrado. Si se pasan parámetros, estos deben colocarse dentro de los paréntesis separados por coma.

Normalmente la llamada a una función se realizará desde la función principal ***main()***, aunque naturalmente también podrá ser desde otra función.

### Ejemplo #1:

```
#include <stdio.h>
int suma(int, int); //funcion prototipo
void main(void)
{
    int a = 10, b = 20, c;
    c = suma(a, b);
    printf("suma de a + b = %d\n", c);
}
```

```
int suma(int x, int y)
{
    return(x + y);
}
```

**Ejemplo #2:**

```
//func1.c

#include <stdio.h>

void func1(void)
{
    puts("Segunda funcion");
    return;
}

void func2(void)
{
    puts("Tercera funcion");
    return;
}

void main()
{
    puts("Primera función llamada main()");
    func1();
    func2();
    puts("main se termina");
}
```

**Ejemplo #3:**

```
//maximo2.c
#include <stdio.h>

int max(int, int); //funcion prototipo

void main()
{
    int a = 10, b = 20;
    printf("%d\n", max(a, b));
}

int max(int x, int y)
{
    int z = 0;
    if (x > y)
        z = x ;
    else
        z = y;
    return z;
}
```

**Ejemplo #4:**

```
//media.c

#include <stdio.h>

double media(double x1, double x2)
{
    return((x1 + x2)/2);
}

void main()
{
    double num1, num2, med;
    printf("Introducir dos numeros reales:");
    scanf("%lf %lf",&num1, &num2);
    med = media(num1, num2);
    printf("El valor medio es %g \n",med);
}
```

**Prototipo de las funciones**

La **declaración** de una función se denomina **prototipo**. Los prototipos de una función contienen la cabecera de la función, con la diferencia de que los prototipos terminan con un punto y coma. Específicamente un prototipo consta de los siguientes elementos: **nombre de la función, una lista de argumentos encerrados entre paréntesis y un punto y coma**. Los prototipos de las funciones llamadas en un programa se incluyen en la cabecera del programa para que así sean reconocidas en todo el programa.

**Sintaxis:**

**Tipo\_retorno nombre\_función(lista\_de\_declaración\_parámetros);**

**Tipo\_retorno** Tipo del valor devuelto por la función o palabra reservada void si no devuelve un valor.

**nombre\_función** Nombre de la función.

**lista\_de\_declaración\_parámetros** Lista de declaración de los parámetros de la función, separados por comas(los nombres de los parámetros son opcionales).

```
double FahrCelsius(double tempFahr);
int longitud(int h, int a);
struct persona entrada(void) ;
char *concatenar(char *c1, char *c2) ;
double intensidad(double, double);
```

Los prototipos se sitúan normalmente al principio de un programa, antes de la definición de la función main(). El compilador utiliza los prototipos para validar que el número y los tipos de datos de los argumentos reales de la llamada a la función son los

mismos que el número y tipo de argumentos formales en la función llamada. Si se detecta una inconsistencia, se visualiza un mensaje de error. Sin prototipos, un error puede ocurrir sin un argumento con un tipo de dato incorrecto se pasa a una función.

Una **declaración** de la función contiene sólo la cabecera de la función y una vez declarada la función, la **definición** completa de la función debe existir en algún lugar del programa, antes o después de **main()**.

**Ejemplo #5:**

```
//area_rectangulo.c

#include <stdio.h>

float area_rectangulo(float b, float a); //declaración o prototipo
float entrada();

void main()
{
    float b, h, area;
    printf("\n Base del rectangulo: ");
    b = entrada();
    printf("\n Altura del rectangulo: ");
    h = entrada();
    area = area_rectangulo(b,h);
    printf("\n Area del rectangulo: %.2f",area);
}

float entrada()
{
    float m;
    do
    {
        scanf("%f",&m);
    }while(m <= 0.0);

    return m;
}

float area_rectangulo(float b, float a)
{
    return(b*a);
}
```

***Paso de parámetros a funciones. Llamadas por valor***

Es importante destacar que en C todos los argumentos de una función se pasan por valor. Se le conoce como paso de parámetros por valor al hecho de enviar valores a una función, estos valores serán recibidos en variables temporales de tal manera que si se llevan a cabo modificaciones, los cambios no se reflejarán en las variables originales. Lo que significa que no podemos modificar directamente desde una función las variables de la función que la ha llamado.

**Ejemplo #6:**

```
//func1.c
#include <stdio.h>

int power(int , int ); //prototipo de funcion
void main(void)
{
    int r;
    r = power(4,2);
    printf("r= %d\n", r);
}

int power(int x, int n)
{
    int p;
    for(p=1; x>0; x--)
        p *= n;
    return(p);
}
```

Si quisiéramos modificar una variable llamando a una función tendríamos que pasarle como argumento a dicha función la dirección en memoria de esa variable (un puntero a la variable).

***Paso de parámetros por referencia (por dirección)***

Cuando a una función se le pasa un parámetro por referencia, lo que se hace es enviar a la función la dirección de memoria de la variable. La función puede modificar el valor de la variable pasada por referencia (se le pasa la dirección de memoria, se modifica el contenido y se almacena en la misma dirección, con lo que se trata de la misma variable). Al pasar una variable por referencia, la variable debe ir precedida por el operador &. Se pueden pasar punteros como argumentos, método empleado para la transferencia de arrays.

La declaración:

```
int funcion(char *car, int &p);
```

Indica una función que devuelve un valor entero y recibe dos argumentos: un puntero \*car y la dirección de memoria &p de una variable.

```
char *funcion(char *p, int y);
```

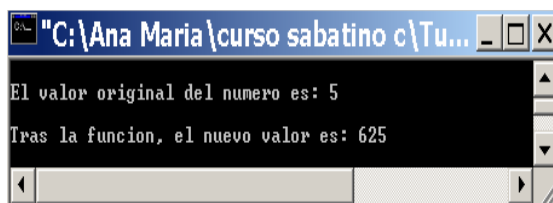
Función que recibe un puntero a carácter y un entero, devolviendo un puntero a carácter. El puntero que recibe y devuelve puede ser un simple carácter o una cadena de caracteres (en este caso puede requerir su inicialización como cadena o la realización de una reserva dinámica de memoria en función del tamaño deseado).



**Ejemplo #7:**

```
//por_referencia.c
/* Eleva un numero a la cuarta potencia pasando el argumento por referencia*/
#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
double por_referencia(double *x);
void main(void)
{
    double numero = 5.0;
    double res;
    printf("\nEl valor original del numero es: %g\n", numero);
    res = por_referencia(&numero);
    printf("\nTras la funcion, el nuevo valor es: %g\n", res);
}
```



```
double por_referencia(double *numero)
{
    double res;
    res = pow(*numero, 4);
    return (res);
}
```

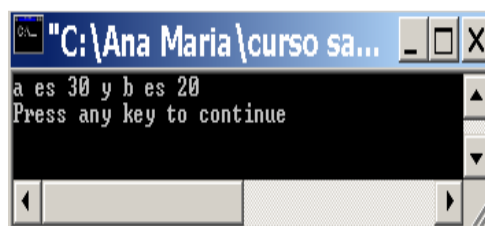
**Ejemplo #8:**

```
/* referen.c - Paso de parámetros por referencia */
#include <stdio.h>
```

```
void intercambio(int *, int *); /* prototipo de la función */
```

```
void main(void)
{
    int a = 20, b = 30;
    intercambio(&a, &b); /* a y b son pasados por referencia */
    printf("a es %d y b es %d\n", a, b);
}
```

```
void intercambio(int *x, int *y)
{
    int z = *x; /* z = contenido de la dirección x */
    *x = *y; /* contenido de x = contenido de y */
    *y = z; /* contenido de y = z */
}
```



## FUNCIONES DE TIPO void

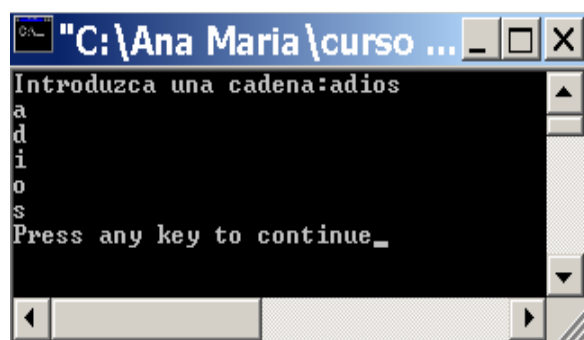
Una función se declara de tipo **void** para evitar que la misma devuelva algún valor.

### Ejemplo #9:

```
//void.cpp
#include <stdio.h>

void imprime_col(char s[60]);
void main(void)
{
    char cadena[60];
    printf("Introduzca una cadena:");
    gets(cadena);
    imprime_col(cadena);
}

void imprime_col(char s[60])
{
    int i;
    for(i=0; s[i]!='\0'; ++i)
        printf("%c\n", s[i]);
    return;
}
```



## Ejercicios resueltos:

1. Escriba un programa que utilice:
  - Una función llamada **par\_impar**, que determine si un número entero es par o impar.
  - La función **positivo\_negativo** con un parámetro de tipo int, que visualice “positivo” o “negativo” respecto del valor pasado como argumento.
  - La función **cuadrado**, que regrese el cuadrado del entero pasado como parámetro.
  - La función **cubo**, que regrese el cubo del entero pasado como argumento.
  - La función **contar**, que cuente cuántas veces se entra a dicha función.

La función main llamará a todas las funciones para un valor determinado.

```
/* funcionesvarias.c - Cómo es un número. Contar.
*/
#include <stdio.h>

void par_impar(int);
void positivo_negativo(int);
int cuadrado(int);
int cubo(int);
int contar(void);
```

```
void main(void)
{
    int n = 10;

    par_impar(n);
    positivo_negativo(n);
    printf("cuadrado de %d = %d\n", n, cuadrado(n));
    printf("cubo de %d = %d\n", n, cubo(n));
    printf("\nContar hasta tres: ");
    printf("%d ", contar());
    printf("%d ", contar());
    printf("%d\n", contar());
}

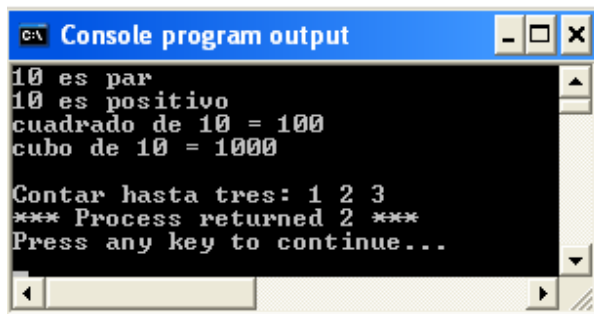
void par_impar(int n)
{
    if(n % 2 == 0)
        printf("%d es %s\n", n, "par");
    else
        printf("%d es %s\n", n, "impar");
    return;
}

void positivo_negativo(int n)
{
    if(n >= 0)
        printf("%d es %s\n", n, "positivo");
    else
        printf("%d es %s\n", n, "negativo");
    return;
}

int cuadrado(int n)
{
    return n * n;
}

int cubo(int n)
{
    return n * n * n;
}

int contar(void)
{
    int n = 1;
    return n++;
}
```



2. Realizar un programa que llame a las funciones sumar( ), restar( ), multiplicar( ) y dividir( ), pasándole como parámetros dos datos de tipo float y que cada una regrese un valor float correspondiente a la suma, resta, multiplicación y división respectivamente.

//operaciones.c

```
#include <stdio.h>
```

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

```
float restar(float a, float b)
{
    return a-b;
}
```

```
float multiplicar(float a, float b)
{
    return a*b;
}
```

```
float dividir(float a, float b)
{
    return a/b;
}
```

```
void main(void)
{
    float x,y,z;
    printf("x = ");
    scanf("%f",&x);
    printf("y = ");
    scanf("%f",&y);
    z = sumar(x,y);
    printf("%g + %g = %g \n",x,y,z);
    z = restar(x,y);
    printf("%g - %g = %g \n",x,y,z);
    z = multiplicar(x,y);
    printf("%g * %g = %g \n",x,y,z);
    z = dividir(x,y);
    printf("%g / %g = %g \n",x,y,z);
}
```

3. Se leen tres lados de un triángulo. Clasifíquelos en función de los valores de los lados en:
- Equilátero: tres lados iguales.
  - Isósceles: dos lados iguales.
  - Escaleno: tres lados diferentes.

En un programa utilice una función que determine el tipo de triángulo.

```
//tipotriangulo.c
#include <stdio.h>

void tipotriangulo(float lado1, float lado2, float lado3)
{
    if((lado1 == lado2)&&(lado2 == lado3))
        printf("Triangulo equilatero\n");

    if(((lado1 == lado2)&&(lado2 != lado3)) || ((lado2 == lado3)&&(lado3 != lado1)) ||
        ((lado3 == lado1)&&(lado1 != lado2)))
        printf("Triangulo Isosceles\n");

    if((lado1 != lado2) && (lado2 != lado3) && (lado3 != lado1))
        printf("Triangulo Escaleno\n");
    return;
}

void main(void)
{
    float l1,l2,l3;
    printf("Introduzca los tres lados del triangulo: ");
    scanf("%f %f %f",&l1,&l2,&l3);
    tipotriangulo(l1,l2,l3);
}
```

4. Realice una función que calcule la distancia entre dos puntos. Llame dicha función desde el cuerpo principal del programa.

```
//distanciapuntos.c
//Calcula la distancia entre dos puntos
#include <stdio.h>
#include <math.h>

double distancia(double x1,double y1,double x2, double y2);

void main(void)
{
    double d, x1,x2,y1, y2;
    printf("Introduzca las coordenadas del punto 1(x1,y1):");
    scanf("%lf,%lf",&x1,&y1);
    printf("Introduzca las coordenadas del punto 2(x2,y2):");
    fflush(stdin);
    scanf("%lf,%lf",&x2,&y2);
    d = distancia(x1,y1,x2,y2);
    printf("distancia entre dos puntos = %g\n",d);
}
```

---

```
double distancia(double x1,double y1,double x2, double y2)
{
    double dist;
    dist=sqrt( pow((x2-x1),2) + pow((y2-y1),2) );
    return(dist);
}
```

5. Escribir una función con dos parámetros, x n, que devuelva lo siguiente:

$$x + \frac{x^n}{n} - \frac{x^{n+2}}{n+2} \quad \text{Si } x \geq 0$$

$$x + \frac{x^{n+1}}{n+1} - \frac{x^{n-1}}{n-1} \quad \text{Si } x < 0$$

//expresion.c

```
#include <stdio.h>
```

```
#include <math.h>
```

```
float funcion(float x, int n)
```

```
{
    float y,z;
    if(x>=0)
    {
        y=x+(pow(x,n)/n)-(pow(x,n+2))/(n+2);
        return y;
    }
    else if(x<0)
    {
        z=x+(pow(x,n+1)/(n+1))-((pow(x,n-1))/(n-1));
        return z;
    }
}
```

```
void main(void)
```

```
{
    float x, r;
    int n;
    printf("Introduzca un numero real, x= ");
    scanf("%f",&x);
    printf("Introduzca un entero, n = ");
    scanf("%d",&n);
    r=funcion(x,n);
    printf("Resultado = %f\n",r);
}
```

6. Dado el siguiente código:

```
#include <stdio.h>
void poner_a_cero(double *);

main( )
{
    double a = 10;
    poner_a_cero(&a);
    printf("%g\n",a);
}

void poner_a_cero(double *a)
{
    *a = 0;
    return;
}
```

Al compilar y ejecutar este programa:

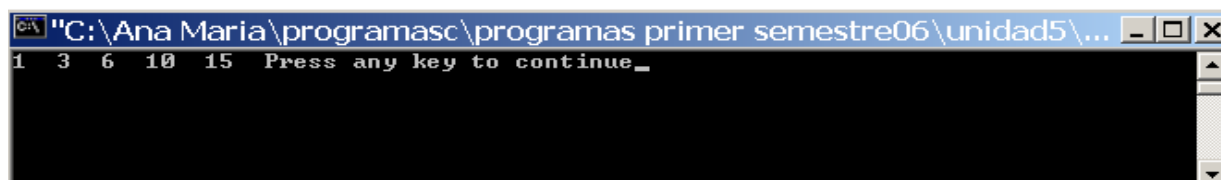
- a) Se visualizará 0
- b) Se visualizará 10
- c) Se obtendrá un error: sobra la sentencia **return**
- d) Se obtendrá un error: redefinición del parámetro formal a

7.Cuál es la salida del siguiente programa?

```
//funciones1.c
#include <stdio.h>
int func1(int cont);

void main(void)
{
    int a, cont;
    for(cont = 1; cont <= 5; cont++)
    {
        a=func1(cont);
        printf("%d ",a);
    }
}

int func1(int x)
{
    static int y=0;
    y+=x;
    return(y);
}
```



### Ejercicios Propuestos:

1. Realizar una función llamada `factorial`, que calculará el factorial del valor de tipo entero que será pasado como parámetro.
2. Realizar una función llamada `media3`, que toma tres números reales como parámetros, y no devuelve nada. Esa función debe calcular la media de los tres números pasados como parámetros y mostrar con un mensaje cuál es la media calculada.
3. Realizar una función llamada `ultima`, que toma una cadena de hasta 10 caracteres como parámetro, y devuelve el último carácter. Esa función debe devolver el último carácter si no es vacía (es decir, si tiene caracteres); si es vacía ("" ) debe devolver un carácter terminador ('\0') para indicar que era vacía.
4. Defina una función llamada **`hypotenuse`** que calcule la longitud de la hipotenusa de un triángulo rectángulo, cuando son conocidos los otros dos lados. La función debe tomar dos argumentos del tipo `double` y regresar la hipotenusa también como `double`.