

---

27 de agosto de 2009

## **Funciones en C++**

Para entender lo que es una función primero se conocerá en qué consiste una función:

Una función contiene dos partes principales:

- ❖ Encabezado de la función.
- ❖ Cuerpo de la función.

A su vez el encabezado de la función contiene lo siguiente:

- ❖ Especificación del tipo del valor a ser retomado por la función.
- ❖ Un nombre de la función (siguiendo las reglas).
- ❖ Un conjunto de argumentos (dato pasado como entrada a una función) que están separados por comas. El conjunto de argumentos es opcional. Si es de existir cada argumento debe estar precedido por su declaración de tipo.

Ejemplo tipo\_de\_dato nombre\_de\_la\_función (tipo arg1, tipo arg2,...);

Nota importante: cuando un argumento o también llamado parámetro es entera no es necesario colocarlo ya que el compilador asume de manera predeterminada que es de tipo entero es decir (int) pero para efecto de estilo de programación es necesario colocarlo porque sería un falta terrible de programación que a veces ni el compilador acepta. Por otra parte este tipo de argumento se denomina argumentos formales, ya que estos representan los identificadores de las entradas transferidas desde el programa que hace la llamada de la función llamada.

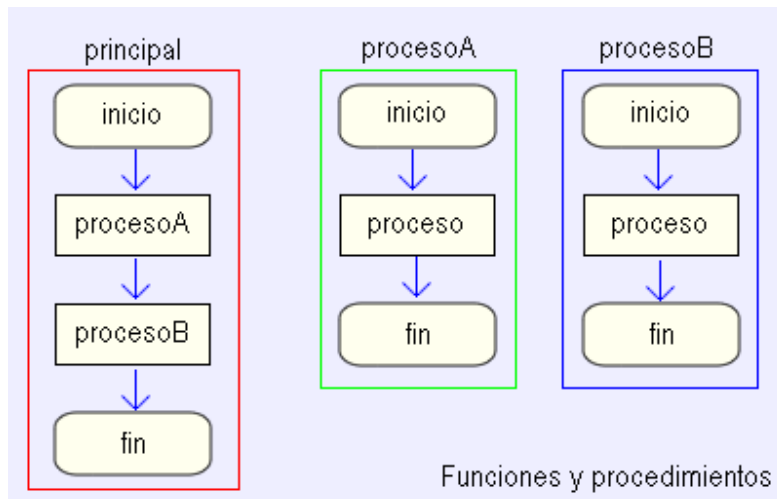
El cuerpo de la función: Es el conjunto de sentencia encerradas entre llaves. Este conjunto de sentencia define la tarea que lleva a cabo la función. El cuerpo puede contener cero o más sentencias return. Las sentencias return ayudan a pasar el valor del resultado único al programa que hizo la llamada.

Sentencia return:

Return(expresión);

En definitiva una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente

utilizadas durante un programa y de esta manera reducir la cantidad de código. Cuando una función es invocada se le pasa el control a la misma, una vez que esta finalizó con su tarea el control es devuelto al punto desde el cual la función fue llamada.



### Ejemplo de una función

Para comenzar, vamos a considerar el caso en el cual se desea crear la función **cuadrado()**, misma que deberá volver el cuadrado de un número real (de punto flotante), es decir, cuadrado() aceptará números de punto flotante y regresará una respuesta como número flotante.

**Nota:** aunque para la función que veremos el tipo de retorno coincide con el tipo de parámetro pasado, algunas veces las cosas pueden cambiar, es decir, no es obligatorio que una función reciba un parámetro de un tipo y que tenga que regresar una respuesta de dicho tipo.

```
// Regresar el cuadrado de un número
```

```
double cuadrado(double n)
```

```
{
    return n*n;
}
```

### Parámetros

Normalmente, las funciones operan sobre ciertos valores pasados a las mismas ya sea como constantes literales o como variables, aunque se pueden definir funciones que no reciban parámetros. Existen dos formas en C++ de pasar parámetros a una función; por referencia o por valor. El hecho es que si en una declaración de función se declaran parámetros por referencia, a los mismos no se les podrá pasar valores literales ya que las referencias apuntan a objetos (variables o funciones) residentes en la memoria; por otro lado, si un parámetro es declarado para ser pasado por valor, el mismo puede pasarse como una constante literal o como una variable. Los parámetros pasados por referencia pueden ser alterados por la función que los reciba, mientras que los parámetros pasados por valor o copia no pueden ser alterados por la función que los recibe, es decir, la función puede manipular a su antojo al parámetro, pero ningún cambio hecho sobre este se reflejará en el parámetro original.

### **Parámetros por valor**

La función `cuadrado()` (ver arriba) es un clásico ejemplo que muestra el paso de parámetros por valor, en ese sentido la función `cuadrado()` recibe una copia del parámetro `n`. En la misma función se puede observar que se realiza un cálculo ( $n*n$ ), sin embargo el parámetro original no sufrirá cambio alguno, esto seguirá siendo cierto aún cuando dentro de la función hubiera una instrucción parecida a `n = n * n;` o `n*=n;`.

### **Parámetros por referencia**

Para mostrar un ejemplo del paso de parámetros por referencia, vamos a retomar el caso de la función `cuadrado`, salvo que en esta ocasión cambiaremos ligeramente la sintaxis para definir la misma. Veamos:

```
// Regresar el cuadrado de un número
```

```
double cuadrado2(double &n)
```

```
{
```

```
    n *= n; return n;
```

```
}
```

Al poner a prueba las funciones `cuadrado()` y `cuadrado2()` se podrá verificar que la primera de estas no cambia el valor del parámetro original, mientras que la segunda sí lo hace.

---

27 de agosto de 2009

## **Función especial *main***

Para poder crear un programa en C, es necesario que haya una función especial definida. Esta función se llama *main* y es el punto de entrada o inicio de ejecución del programa, es decir, es el lugar por donde va a empezar la ejecución de nuestro programa. La declaración más sencilla (que alcanza para los programas que vamos a realizar) es la siguiente:

```
int max ()  
{  
sentencia1;  
sentencia2;  
}
```

Ejemplos:

### **Determinar el máximo de dos números enteros**

```
Int max(int x,int y){  
Int salida;  
If(x>=y)  
    Salida=x;  
Else  
    Salida=y;  
Return(salida);  
}
```

Explicación: se define una función llamada max que retorna un int como resultado.

El cuerpo de la función es bastante simple, dado que sólo verifica cuál de las variables es mayor y retorna el valor.

Ejemplo 2:

### **Determinar si un número es impar**

```
Int esimpar(int x){
```

---

27 de agosto de 2009

```
Int salida;  
If(x%2!=0)  
Salida=1;//verdadero si es impar  
Else  
Salida=0 //falso si no es impar  
Return(salida);
```

Explicación: este programa sólo verifica si el residuo de la división entre 2 es 0. Si el residuo no es 0, entonces éste es un número impar y se retorna un valor verdadero.

Ejemplo 3:

#### **Determinar el máximo común divisor de dos enteros**

```
Int mcd(int x,int y){  
Int salida;  
While(x!=0 && y!=0){  
If(x>=y)  
X=x%y;  
Else  
Y=y%x;  
}  
If(x==0)  
Salida=y; //y es mcd  
Else  
Salida=x; //x es mcd  
Return(salida);  
}
```

---

27 de agosto de 2009

Explicación: la función explota el hecho de que el mcd de dos enteros es el producto de los dos números enteros dividido entre su mcd.

### **Ejercicios propuestos**

Hacer un programa que genere números primos.

Análisis: el problema aquí es aceptar un número entero positivo grande NUM, y generar todos los números primos entre 2 y NUM. El siguiente algoritmo resuelve ese problema.

Paso 1: Definir una función primo(int x) que verifica si x es primo o no

Paso 2: Ingresar y validar NUM

Paso 3: Para (todo k desde 2 a NUM)

Si(primo(k) == verdadero)

Imprimir k;

Basado en este algoritmo se puede escribir el programa para generar los números primos entre 2 y NUM.

### **Funciones Recursivas**

Es una técnica algorítmica donde una función de manera de acometer una tarea, se llama a sí misma con valores modificados en sus argumentos.

¿Cuál es el objetivo de que una función se llame a sí misma? Bueno, déjeme decirle que habrá ocasiones en las cuales el uso de esta técnica tiene sus ventajas. Por ejemplo, las personas que manejan el lenguaje de las matemáticas saben que hay ciertos procesos que son repetitivos, tal es el caso del procedimiento para obtener el factorial de un número. Así, procederemos a escribir un programa y dentro del mismo la función Factorial, misma que servirá como ejemplo de una función recursiva. factorial tomará un solo parámetro tipo Long (entero largo) pasado por valor y regresará el factorial del número.

**! ADVERTENCIA!** Las funciones recursivas deben tener una forma para poder salir y así evitar que éstas se estén llamadas de manera infinita, ya que esto ocasionaría un error conocido como desbordamiento de pila (Stack Overflow).

Ejemplo de funciones recursivas

---

27 de agosto de 2009

Calcular el factorial de un entero

Factorial de 5=5x4x3x2x1=120

En general factorial de n como n! (donde n>0)=nx(n-1)x(n-2)x...x1

```
// programa funciones04.cpp
```

```
#include <stdlib.h>
```

```
#include <iostream.h>
```

```
// Factorial es una función recursiva.
```

```
// Nota: el máximo valor para el parámetro n es 16
```

```
long factorial(long n)
```

```
{
```

```
    if (n <=1) return(1);
```

```
        else return( n * factorial(n - 1) );
```

```
}
```

```
// factorial2 es una función no recursiva.
```

```
// Nota: el máximo valor para el parámetro n es 16
```

```
long factorial2(long n)
```

```
{
```

```
    long p, r;
```

```
    r = 1;
```

```
    if (n >1) {
```

---

27 de agosto de 2009

```
    for (p=1; p<=n; p++) r = r * p;
}
return r;
}

// Punto de prueba
int main()
{
    cout << "Demostración de una función recursiva" << endl << endl;

    // llamada a la función recursiva
    cout << "Factorial de 6 = " << factorial(6) << endl;

    // llamada a la función no recursiva
    cout << "Factorial de 6 = " << factorial2(6) << endl;

    system("pause");
    return 0;
}
```