Las **funciones** son una herramienta indispensable para el programador, tanto las funciones creadas por él mismo como las que le son proporcionadas por otras librerías, cualquiera que sea el caso, las funciones permiten automatizar tareas repetitivas, encapsular el código que utilizamos, e incluso mejorar la seguridad, confiabilidad y estabilidad de nuestros programas. Dominar el uso de funciones es de gran importancia, permiten modularizar nuestro código, separarlo según las tareas que requerimos, por ejemplo una función para abrir, otra para cerrar, otra para actualizar, etc. básicamente una función en nuestro código debe contener la implementación de una utilidad de nuestra aplicación, es decir que por cada utilidad básica (abrir, cerrar, cargar, mover, etc.) sería adecuado tener al menos una función asociada a ésta.

¿Funciones, métodos o procedimientos?

En el mundo de la programación, muchos acostumbramos hablar indistintamente de estos tres términos sin embargo poseen diferencias fundamentales.

Funciones:

Las funciones son un conjunto de procedimientos encapsulados en un bloque, usualmente reciben parámetros, cuyos valores se utilizan para efectuar operaciones y adicionalmente retornan un valor. Esta definición proviene de la definición de función matemática la cual posee un dominio y un rango, es decir un conjunto de valores que puede tomar y un conjunto de valores que puede retornar luego de cualquier operación.

Métodos:

Los métodos y las funciones son funcionalmente idénticos, pero su diferencia radica en el contexto en el que existen. Un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo en método está asociado a un objeto, básicamente un método es una función que pertenece a un objeto o clase, mientras que una función existe por sí sola, sin necesidad de un objeto para ser usada.

Procedimientos:

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo en la definición no hay nada que se lo impida. En el contexto de C++ un procedimiento es básicamente una función void que no nos obliga a utilizar una sentencia return.

Durante este artículo hablaremos sobre procedimientos y funciones, los métodos son parte de un tema diferente.

Declarando funciones en C++

La sintaxis para declarar una función es muy simple, veamos:

```
tipo nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento]...])
{
    /*
        * Bloque de instrucciones
    */

    return valor;
}
```

Recordemos que una **función siempre retorna algo**, por lo tanto es obligatorio declarar un tipo (el primer componente de la sintaxis anterior), luego debemos darle un nombre a dicha función, para poder identificarla y llamarla durante la ejecución, después al interior de paréntesis, podemos poner los argumentos o parámetros. Luego de la definición de la "firma" de la función, se define su funcionamiento entre llaves; todo lo que esté dentro de las llaves es parte del cuerpo de la función y éste se ejecuta hasta llegar a la instrucción return.

Acerca de los argumentos o parámetros

Hay algunos detalles respecto a los argumentos de una función, veamos:

Una función o procedimiento pueden tener una cantidad cualquier de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más parámetros. Aunque habitualmente no suelen tener más de 4 o 5.

Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.

Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo de la función.

Consejos acerca de return

Debes tener en cuenta dos cosas importantes con la sentencia return:

Cualquier instrucción que se encuentre después de la ejecución de return NO será ejecutada. Es común encontrar funciones con múltiples sentencias return al interior de condicionales, pero una vez que el código ejecuta una sentencia return lo que haya de allí hacia abajo no se ejecutará.

El tipo del valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara int, el valor retornado debe ser un número entero.

Veamos algunos ejemplos.

Ejemplos de funciones

Veamos algunos ejemplos prácticos de funciones en C++.

Ejemplo 1:

```
int funcionEntera()//Función sin parámetros
{
    int suma = 5+5;
    return suma; //Acá termina la ejecución de la función
    return 5+5;//Este return nunca se ejecutará
    //Intenta intercambiar la línea 3 con la 5
    int x = 10; //Esta línea nunca se ejecutará
}
```

Como puedes ver es un ejemplo sencillo, si ejecutas esto, la función te retornará el valor de suma que es 10 (5+5). Las líneas posteriores no se ejecutarán nunca, aunque no generen error alguno, no tienen utilidad. Puedes notar que para este caso es lo mismo haber escrito return suma que escribir return 5+5. Ambas líneas funcionan equivalentemente.

Ejemplo 2:

```
char funcionChar(int n)//Función con un parámetro
{
    if(n == 0)//Usamos el parámetro en la función
    {
        return 'a'; //Si n es cero retorna a
        //Notar que de aquí para abajo no se ejecuta nada más
    }
    return 'x';//Este return sólo se ejecuta cuando n NO es cero
}
```

Aquí hicimos uso se múltiples sentencia return y aprovechamos la característica de que al ser ejecutadas finaliza inmediatamente la ejecución de la parte restante de la función. De este modo podemos asegurar que la función retornará 'a' únicamente cuando el valor del parámetro n sea cero y retornará un 'x' cuando dicho valor no sea cero.

Ejemplo 3:

```
bool funcionBool(int n, string mensaje)//Función con dos parámetros
{
    if(n == 0)//Usamos el parámetro en la función
    {
        cout << mensaje;//Mostramos el mensaje
        return 1; //Si n es cero retorna 1
        return true;//Equivalente
    }
    return 0;//Este return sólo se ejecuta cuando n NO es cero
    return false;//Equivalente
}
```

Aquí ya tenemos una función que recibe dos parámetros, uno de ellos es usado en el condicional y el otro para mostrar su valor por pantalla con cout, esta vez retornamos valores booleanos 0 y 1, pudo ser true o false también.

Hablemos un poco de los procedimientos

Los procedimientos son similares a las funciones, aunque más resumidos. Debido a que los procedimientos no retornan valores, no hacen uso de la sentencia return para devolver valores y no tienen tipo específico, solo void. Veamos un ejemplo:

Ejemplo de procedimientos

```
void procedimiento(int n, string nombre)
{
    if(n == 0)
    {
        cout << "hola" << nombre;
        return;
    }
    cout << "adios" << nombre;
}
```

De este ejemplo podemos ver que ya no se usa un tipo sino que se pone void, indicando que no retorna valores, también podemos ver que un procedimiento también puede recibir parámetros o argumentos.

Atención: Los procedimientos también pueden usar la sentencia return, pero no con un valor. En los procedimientos el return sólo se utiliza para finalizar allí la ejecución de la función.

Invocando funciones y procedimientos en C++

Ya hemos visto cómo se crean y cómo se ejecutan las funciones en C++, ahora veamos cómo hacemos uso de ellas.

```
nombreFuncion([valor,[valor]...]);
```

Como puedes notar es bastante sencillo invocar o llamar funciones en C++ (de hecho en cualquier lenguaje actual), sólo necesitas el nombre de la función y enviarle el valor de los parámetros. Hay que hacer algunas salvedades respecto a esto.

Detalles para invocar funciones

El nombre de la función debe coincidir exactamente al momento de invocarla.

El orden de los parámetros y el tipo debe coincidir. Hay que ser cuidadosos al momento de enviar los parámetros, debemos hacerlo en el mismo orden en el que fueron declarados y deben ser del mismo tipo (número, texto u otros).

Cada parámetro enviado también va separado por comas.

Si una función no recibe parámetros, simplemente no ponemos nada al interior de los paréntesis, pero SIEMPRE debemos poner los paréntesis.

Invocar una función sigue siendo una sentencia habitual de C++, así que ésta debe finalizar con ';' como siempre.

El valor retornado por una función puede ser asignado a una variable del mismo tipo.

Una función puede llamar a otra dentro de sí misma o incluso puede ser enviada como parámetro a otra.

Ejemplos de uso de funciones

En el siguiente código vamos a hacer un llamado a algunas de las funciones y al procedimiento, que declaramos anteriormente.

```
int main()
{
    funcionEntera(); //Llamando a una función sin argumentos

     bool respuesta = funcionBool(1, "hola"); //Asignando el valor retornado a una variable

    procedimiento(0, "Juan");//Invocando el procedimiento

    //Usando una función como parámetro
    procedimiento(funcionBool(1, "hola"), "Juan");

    return 0;
}
```

En el código anterior podemos ver cómo todas las funciones han sido invocadas al interior de la función main (la función principal), esto nos demuestra que podemos hacer uso de funciones al interior de otras. También vemos cómo se asigna el valor retornado por la función a la variable 'respuesta' y finalmente, antes del return, vemos cómo hemos usado el valor retornado por **'funcionBool'** como parámetro del procedimiento.