**Complementario Funciones (C++)**

## ¿Funciones, métodos o procedimientos?

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

### Funciones:

Las funciones son un conjunto de procedimiento encapsulados en un bloque, usualmente reciben parámetros, cuyos valores 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 lo 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.

**Funciones (C++)**

Una función es un bloque de código que realiza alguna operación. Una función puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar argumentos a la función. Una función también puede devolver un valor como salida. Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función. La función siguiente acepta dos enteros de un llamador y devuelve su suma; *a* y *b* son *parámetros* de tipo **int** .

**C++**

int sum(int a, int b)

{

return a + b;

}

Se puede invocar la función o *llamarla* desde cualquier número de lugares del programa. Los valores que se pasan a la función son los *argumentos*, cuyos tipos deben ser compatibles con los tipos de parámetro de la definición de función.

**C++**

int main()

{

int i = sum(10, 32);

int j = sum(i, 66);

cout << "El valor de j es" << j << endl; // 108

}

No hay ningún límite práctico para la longitud de la función, pero un buen diseño tiene como objetivo funciones que realizan una sola tarea bien definida. Los algoritmos complejos deben dividirse en funciones más sencillas y fáciles de comprender siempre que sea posible.

En C++, a diferencia de otros lenguajes, una función también pueden definirse en el ámbito de espacio de nombres (incluido el espacio de nombres global implícito). Estas funciones se denominan funciones *libres* o *funciones no miembro*; se usan en gran medida en la biblioteca estándar.

**Elementos de una declaración de función**

Una *declaración* de función mínima está formada por el tipo de valor devuelto, el nombre de función y la lista de parámetros (que puede estar vacía), junto con palabras clave opcionales que proporcionan instrucciones adicionales para el compilador. El ejemplo siguiente es una declaración de función:

**C++**

int sum(int a, int b);

Una definición de función se compone de una declaración, más el *cuerpo*, que es todo el código entre las llaves:

C++Copiar

int sum(int a, int b)

{

return a + b;

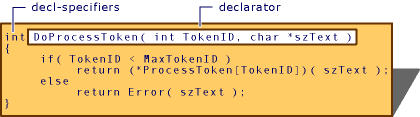
}

Una declaración de función seguida de un punto y coma puede aparecer en varios lugares de un programa. Debe aparecer antes de cualquier llamada a esa función en cada unidad de traducción. La definición de función debe aparecer solo una vez en el programa, según la regla de una definición (ODR).

Los elementos necesarios de una declaración de función son los siguientes:

1. El tipo de valor devuelto, que especifica el tipo del valor que devuelve la función, o **void** si no se devuelve ningún valor. En C++ 11, **auto** es un tipo de valor devuelto válido que indica al compilador que infiera el tipo a partir de la instrucción return. En C++ 14, decltype(auto) también se permite. Para obtener más información, consulte más adelante Deducción de tipos en tipos de valor devueltos.
2. El nombre de función, que debe comenzar con una letra o un carácter de subrayado y no puede contener espacios. En general, un carácter de subrayado inicial en los nombres de función de la biblioteca estándar indica funciones de miembro privado o funciones no miembro que no están pensadas para que las use el código.
3. La lista de parámetros, que es un conjunto delimitado por llaves y separado por comas de cero o más parámetros que especifican el tipo y, opcionalmente, un nombre local mediante el cual se puede acceder a los valores de dentro del cuerpo de la función.

La ilustración siguiente muestra las partes de una definición de función. El área sombreada es el cuerpo de la función.

  
Elementos de una definición de función

**Acerca de los argumentos o parámetros**

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

1. 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.
2. Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.
3. 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:

1. 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.
2. 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.

**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.

**Definiciones de función**

Una *definición de función* está formada por la declaración y el cuerpo de la función, entre llaves, que contiene declaraciones de variables, instrucciones y expresiones. En el ejemplo siguiente se muestra una definición de función completa:

**C++**

int foo(int i, std::string s)

{

int value {i};

MyClass mc;

if(strcmp(s, "default") != 0)

{

value = mc.do\_something(i);

}

return value;

}

Las variables declaradas dentro del cuerpo se denominan variables locales. Se salen del ámbito cuando finaliza la función; por lo tanto, una función nunca debe devolver una referencia a una variable local.

**C++**

MyClass& boom(int i, std::string s)

{

int value {i};

MyClass mc;

mc.Initialize(i,s);

return mc;

}

**Parámetros de función y argumentos**

Una función tiene una lista de parámetros separados por comas de cero o más tipos, cada uno de los cuales tiene un nombre mediante el cual se puede acceder a ellos dentro del cuerpo de la función. El llamador pasa argumentos, que son valores concretos cuyos tipos son compatibles con la lista de parámetros.

De forma predeterminada, los argumentos se pasan a la función por valor, lo que significa que la función recibe una copia de la variable que se pasa.

Para hacer que los argumentos se pasen por referencia (concretamente, referencia de valor l), agregue un cuantificador de referencia al parámetro (&) :

**C++**

void DoSomething(std::string& input){...}

Cuando una función modifica un argumento que se pasa por referencia, modifica el objeto original, no una copia local. Para evitar que una función modifique este tipo de argumento, califique el parámetro como const&:

**C++**

void DoSomething(const std::string& input){...}

**C++ 11:** Para controlar explícitamente los argumentos que se pasan por referencia rvalue o lvalue-Reference, use un signo de y comercial en el parámetro para indicar una referencia universal:

**C++**

void DoSomething(const std::string&& input){...}

Una función declarada con la palabra clave Single **void** en la lista de declaraciones de parámetros no toma ningún argumento, siempre y cuando la palabra clave **void** sea el primer y único miembro de la lista de declaraciones de argumentos. Los argumentos de tipo **void** en otro lugar de la lista producen errores. Por ejemplo:

C++

// OK same as GetTickCount()

long GetTickCount( void );

**Tipos de valor devuelto de función**

Una función no puede devolver otra función o una matriz integrada; sin embargo, puede devolver punteros a estos tipos, o una *expresión lambda*, que genera un objeto de función. Excepto en estos casos, una función puede devolver un valor de cualquier tipo que esté en el ámbito, o bien no puede devolver ningún valor, en cuyo caso el tipo de valor devuelto es **void** .

**Tipos de valor devueltos finales**

Un tipo de valor devuelto "normal" se encuentra en el lado izquierdo de la firma de función.

**Variables locales de función**

Una variable que se declara dentro de un cuerpo de función se denomina *variable local* o simplemente *local*. Las variables locales no estáticas solo son visibles dentro del cuerpo de función y, si se declaran en la pila, se salen del ámbito cuando finaliza la función. Cuando se crea una variable local y se devuelve por valor, el compilador normalmente puede realizar la *optimización del valor devuelto con nombre* para evitar operaciones de copia innecesarias. Si una variable local se devuelve por referencia, el compilador emitirá una advertencia, ya que cualquier intento por parte del llamador de usar esa referencia se producirá después de la destrucción de la variable local.