Resumen teórico de funciones en C

La mayoría de los lenguajes de programación permiten la división de un programa en módulos. De acuerdo con el lenguaje que se esté utilizando pueden ser denominados subprogramas, subrutinas, procedimientos, funciones, etc. En C los módulos se denomina funciones.

La división en módulos tiene varios objetivos:

1) **Facilita el diseño** y la construcción de un sistema, al proveer un mecanismo de abstracción: se divide lo complejo en partes simples que interactúan.
2) **Permite el reuso** de los componentes de software: un módulo pensado para un programa se puede utilizar en otro.
3) **Facilita la lectura** de un programa, y la búsqueda y solución de errores: al ser menos complejo y resolver una sola cosa es más sencillo encontrar los errores.
4) **Mejora la eficiencia** de los programas, en cuanto al uso de los recursos de hardware.

Definición de función

Una función es un conjunto de instrucciones que realiza una función específica.
Puede recibir un conjunto de valores de entrada (a los que se denomina parámetros), y proporcionar UN valor de retorno.

Programa C

Un programa correctamente codificado en C, cuenta con una función principal (main()), generalmente de pocas líneas, y un conjunto de funciones.

Las funciones pueden estar contenidas en el mismo archivo fuente del programa, o ser externas a éste, es decir, estar contenidas en otro archivo. Pueden ser funciones de librería generales (de la librería estándar de C u otras librerías existentes), o definidas por el programador.

Para usar las funciones de la librería de C, debe incluirse al inicio del archivo fuente una referencia al archivo de cabecera que la contiene (ejemplo: #include <stdio.h>, o #include <cstdio>)

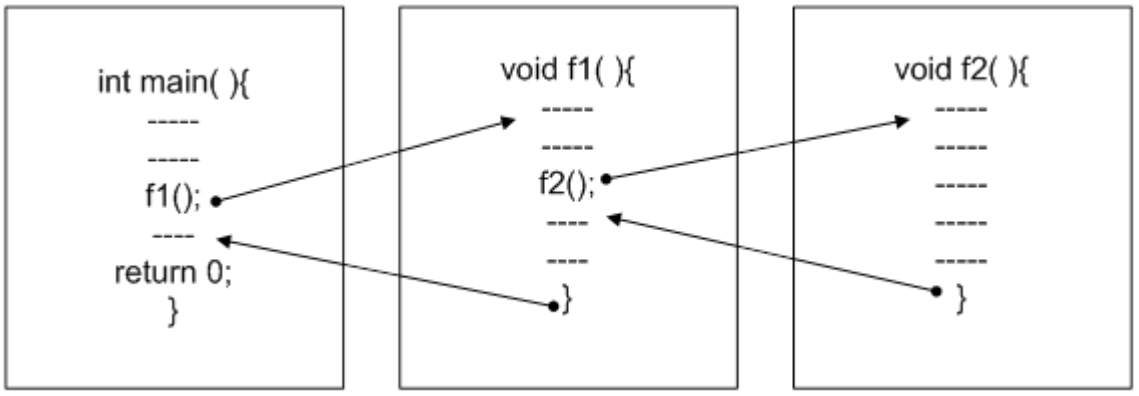
El programa principal (main()) se relaciona con las funciones llamándolas, y si es necesario pasándoles los parámetros (datos) que éstas necesitan.

La **llamada es una línea de código con el nombre de la función y entre paréntesis los parámetros**. Si la función devuelve un valor, éste debe ser asignado a una variable en la misma línea de la llamada.

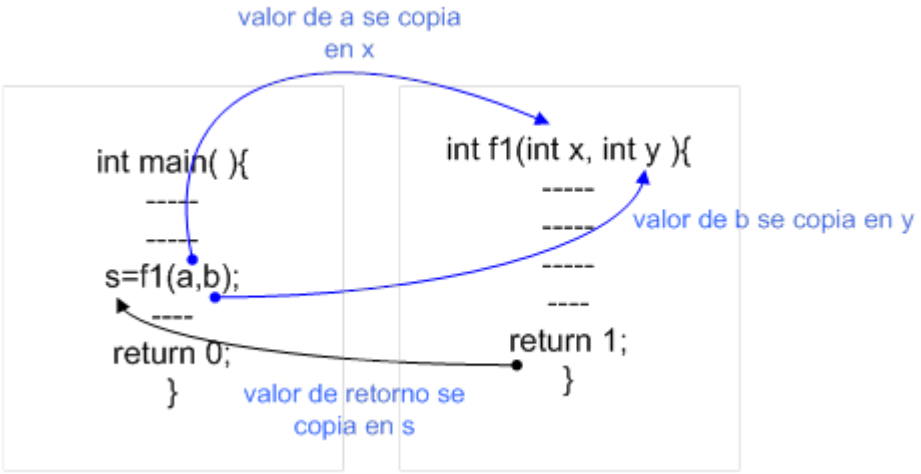
Al llamar a la función la ejecución del programa continúa dentro del cuerpo (el código) de la función llamada. Cuando esta termina, devuelve el control a main(), es decir, se ejecuta la línea de código inmediata siguiente.

En C, cualquier función puede llamar a otra función

El funcionamiento es el mencionado anteriormente: main() llama a una función f1(), la ejecución continúa dentro de f1(). Si f1() llama a otra, f2(), se ejecuta el código de f2(). Cuando termina f2() el programa vuelve a f1(), y cuando termina f1(), el control vuelve a main(), como se ilustra en la figura siguiente:



Si a la función llamada se le envían datos, éstos se copiarán en las variables definidas en la función como parámetros; si la función devuelve un valor, ese valor puede ser almacenado en una variable. Las acciones se muestran en la figura:



Nota: se puede terminar la ejecución de cualquier función, y en cualquier parte de ella con la sentencia return.

Declaración y uso de funciones

Para que un programa pueda utilizar una función es obligatorio que la “conozca” antes. Al igual que una variable la función debe estar declarada antes de usarse.

Las alternativas entonces son:

- La función se encuentra definida en un archivo de cabecera externo: se debe incluir el archivo mediante un #include nombre.h
- La función está escrita completa antes de main() (o de la función que la utilice)

- El prototipo de la función está escrito antes de main(). Al leer el prototipo, el compilador sabe que cuando se haga referencia a ese nombre de función, debe ejecutar el código asociado a ese nombre.

Un prototipo de función tiene el siguiente formato:

valor_devuelto nombre_funcion(tipo arg1,tipo arg_2,....tipo argumento_n)

El compilador necesita saber:

- el nombre de la función
- el tipo de dato que devuelve. Si no devuelve nada hay que poner void
- el tipo de dato de los argumentos, o sea, los tipos y la cantidad de datos que recibirá la función. Si no tiene argumentos, se pone void, o se deja vacío.

Por ejemplo:

void ponerCero(int *, int)

- El nombre de la función es ponerCero
- Como la función no devuelve valor, el tipo de dato devuelto es void.
- Los tipos de datos que la función recibirá son un puntero a entero, y un entero.

Paso de parámetros por valor y por dirección

De acuerdo con la operación que querramos que la función realice, o de los tipos de datos involucrados, el pasaje de parámetros que hace la función que llama a otra función puede realizarse por valor, o por dirección (también se suele denominar a este método por referencia).

En el primer caso, paso por valor, se hace una copia del valor en una variable local a la función. La función no tiene acceso a la variable original, y por lo tanto no la puede modificar.

Cuando el paso de parámetros se realiza por dirección, lo que se envía es la dirección física de la variable. La función puede a través de la dirección que recibe modificar el contenido de la variable original. Esta forma de pasaje es la única que se puede utilizar para enviar un vector o una matriz a una función.

Como norma, o criterio para seleccionar la forma en la que una función recibe los parámetros podemos decir:

- Si no queremos –ni necesitamos- que la función modifique el valor de la variable original debemos hacer el pasaje de parámetros por valor.
- Si necesitamos que la función modifique el valor de una o más variables que se envían como parámetros, o el parámetro es un vector o una matriz, debemos hacer el pasaje por dirección.

Por último, recordar que se debe respetar el orden en el cual fueron establecidos los parámetros en el prototipo, tanto en la llamada como en la definición de la función.