¿Qué es una función?

Una función es básicamente un subprograma.

Un subprograma se encuentra conectado nuestro programa principal, el llamado main, y nosotros lo podemos llamar cuando lo necesitemos.

Anteriormente hemos usado funciones, como trunc, que la invocábamos desde main, y al pasarle una serie de valores nos devolvía otros.

Una función puede o no devolver valor, es decir, viene anclada a un tipo de dato. Una función tipo void es del tipo que no retorna valor, y no necesitan tener un return al final como si pasaba con *int main*.

Este tipo de funciones son muy útiles para dividir el programa en módulos y hacerlo mucho más simple, además de permitirnos repartir el trabajo.

Una función consta de: un tipo de datos como indicamos anteriormente, así cuando la llamamos puede devolver algo o no.

Como por ejemplo aquí, donde ingresamos una fecha usando un string, y mediante una función somos capaces de extraer el mes de la fecha como un número tipo entero. Es importante aclarar la función devuelve el valor a través del return.

También consta de un nombre, con el cual la identificaremos y utilizaremos para llamarla.

Después del nombre, van los parámetros o argumentos de la función, donde indicaremos los datos que le llegarán a la función para trabajar, y que deberemos completar a la hora de llamarla.

La función debe declararse antes del main, pero si queremos podemos declarar un prototipo de **esta forma** para más adelante, debajo del main, definir la función y asi estructurar de manera distinta el programa. La **definición** de una función consta de abrir los corchetes (ósea el cuerpo) y escribir el código que contenga la misma.

A la hora de enviar datos a una función, existen 2 formas de pasarlo:

Por **valor**, donde enviaremos el dato directamente a la función, y en caso de que este sea modificado dentro de la misma, este cambio no se verá reflejado en el programa principal, es decir, en el momento que volvamos al main, la variable va a contener el mismo valor que tenía antes de pasar por la función.

También tenemos el pasaje por **referencia**, donde se utiliza el operador & para enviarle a la función la posición de memoria donde se encuentra el dato, y así en caso de modificarla, el cambio se realizará tanto en el programa principal como en la misma función, esto nos es muy útil para que una función sea capaz de devolver más de un valor.

También podremos establecer argumentos por defecto, ósea parámetros que en caso de no especificarse ya tengan un valor asignado. Estos argumentos se ponen siempre al final de la lista de parámetros formales.

Ahora hablemos del uso de arrays en funciones, estas tienen algunas características especiales q vale la pena mencionar:

Guion 2

Bien, ¿Qué es una función?

Podemos decir que una función es un subprograma, es decir, un programa que existe a la par de nuestro programa principal pero que no necesariamente comparta el mismo flujo de datos (a no ser que nosotros se lo indiquemos), es algo así como si cada uno estuviese en un mundo distinto.

Lo interesante es que podemos agarrar este subprograma, y llamarlo desde el main cuando lo necesitemos, en el momento que la llamemos esta se ejecutara como la programemos, y puede o no devolvernos algún dato a nuestro programa principal (recuerden que como dije anteriormente, las funciones tienen su propio flujo de datos separado del que está en main, y pueden a veces no modificar o no devolver ningún dato).

Es una herramienta muy útil de C++ que nos permite dividir nuestro programa en módulos, ósea en piezas distintas, y juntarlas para crear un solo programa, pero que este mucho mejor estructurado y hasta en algunos casos, sea más eficiente.

Veamos algunos ejemplos de porque son tan útiles las funciones:

1. **Automatización de procesos:** Algunas veces, repetimos algún algoritmo varias veces en nuestro programa, como por ejemplo la búsqueda binaria o la secuencial, que como vimos, sobre todo la binaria, la forma del algoritmo es medianamante “larga”, entonces si nos toca hacer algún programa donde la tengamos que usar 2 o más veces en distintas partes del código, quizás en vez de estar copiando y pegándola en otras partes, sea más útil convertirla en una función que la llamemos desde main cuando queramos, y nos devuelva automáticamente lo que estamos buscando. Incluso podemos incorporarla a un **if** para hacer un uso de ella más eficientemente, es por eso que yo en el anterior video explique que era medio ineficiente implementar este tipo de algoritmos sin conocer funciones.
2. **Modulización:** Nos permite dividir el programa por partes, es decir, un programa principal que vaya llamando una serie de subprogramas que en conjunto formen un solo programa. Es como si estuviera hecho de distintas piezas que simplemente las unimos y funciona a la perfección.

Esto puede parecer una boludez pero nos permite trabajar de una manera mucho más cómoda y organizada, ya que por ejemplo, si tengo un programa compuesto de un módulo A, B, y C, y necesito trabajar en el módulo B, no tengo porque irme al C ni al A, sino que busco la función B directo y trabajo ahí sin afectar a las demás. Esto es buenísimo a la hora de que varias personas trabajen en un mismo código, así te podes repartir distintas partes del programa en funciones, y que cada uno trabaje en la que le toque.

Retomando el ejemplo anterior, seria como si yo estuviera en un grupo con Pepito y Juanito, y decidimos repartirnos una vez tengamos el código main hecho, en que función trabaja cada uno, siendo Pepito en la A, Juanito en la B, y yo en el C; al final del día, cuando cada uno tenga listo sus modulos y deseemos implementarlos en el código principal, lo único que tendría que hacer cada uno es poner su función dentro del código, y si alguno tiene algún bug o problema es únicamente cuestión de fijarse en la función correspondiente.

Ahora bien, hemos visto varios ejemplos y más o menos como es que trabaja una función, pero, ¿Cómo la implementamos en nuestro código?

Lo primero es definir el tipo de función que estamos haciendo, es decir, que tipo de dato nos devuelve, esto se indica al principio de la estructura de la función. Si ponemos tipo int, nos devolverá un numero entero, si es tipo string, una cadena, y así con cualquier tipo de dato. La excepción aquí es **void**, que se utiliza cuando queremos que una función no devuelva ningún valor. Esto nos puede ser útil si queremos por ejemplo armar un menú como módulo, y este al no tener necesidad de devolvernos nada al main, simplemente se dejará como tipo void.

Después va el nombre de la función y seguido de este, entre paréntesis ponemos los datos que le llegarán a la función, es decir, datos que ingresaremos cuando la llamemos desde main, y que al invocar al módulo, le llegaran bajo el nombre que le pongamos a esas variables.

Una vez terminemos eso, simplemente van unos corchetes donde pondremos las instrucciones que incluya la función, y recuerden que en caso de que esta no sea tipo void, siempre debe llevar un return al final que nos devuelva algo.

Existen muchas formas de utilizar las funciones, y al final todo se resume en el uso que les des vos, además de lo que te pida el enunciado del examen que estés haciendo, así que sentite cómodo de usarlas cuanto quieras mientras tu código quede organizado.

Otras cosas que vale la pena mencionar de las funciones:

* **Prototipo de función**

Si bien las funciones se arman por arriba del main, podemos declararlas al principio y luego definirlas abajo.

* **Parámetros por defecto**

Podemos hacer que una función tenga algunos valores por defecto, es decir, que cuando la llamemos no sea necesario darles un valor, esto se hace poniendo las variables que tienen un valor por defecto a la derecha del todo.

* **Pasar valores por referencia**

Como mencione antes, cuando llamamos a una función, le mandamos los valores que necesita para trabajar pasando los parámetros dentro del paréntesis, pero estos datos no se modifican en el main sino que ES dentro de la misma función, al volver al main el valor de esa variable se quedará como estaba antes de llamar a la función.

Ahora, supongamos que queremos enviar una variable a la función y que esta sea modificada dentro de la misma, y que al volver al main, este valor se mantega, eso se hace mediante el uso del operador “&” y nos permite hacer cosas interesantes como que una función devuelta más de un valor.

* **Como trabajar arrays en funciones:**

Cuando queremos que nuestra función sea capaz de acceder a un array del programa principal, lo que se hace es declarar un vector y poner unos corchetes vacíos, de esta forma le estamos indicando que le llegará un vector, y al estar vacíos, este vector adoptará el tamaño del vector que le enviemos. Cabe aclarar que los arrays se manejan por referencia, es decir, que si envio un array a una función, y modifico algún dato de ese array, se modificará en el programa principal también.

Cuando queremos trabajar con matrices o arrays de más dimensiones, el modo de operar es bastante similar, solo que se debe especificar el número de los índices de las otras dimensiones.

Para evitar que un array sea modificado dentro de uan función, podemos aplicar la etiqueta const.

Me quedan en el tintero las funciones recursivas, pero normalmente no las usamos, así que estas quedaran para un video aparte para los que sean más curiosos. Si necesitan una explicación más detallada de este tema, les dejo en la descripción la clase de funciones donde explico esto de manera más extensa y completa, ya que al ser esto un video resumen trato de comprimir todo lo máximo posible.

Ideas para la clase:

* Como aplicar funciones, cuando usar que tipo de dato