Clase 5: Funciones

Mauricio Gómez García

Índice

Introducción

Las funciones son conjuntos de instrucciones que se pueden definir por separado del programa principal, y que se pueden llamar múltiples veces dentro de nuestro programa.

Entender cómo definir y utilizar las funciones es parte fundamental de crear programas en C y en muchos otros lenguajes, y conocer la manera de implementarlas entre si permite crear programas funcionales que son más fáciles de leer y escribir por las personas.

Definición

Las funciones dentro de los lenguajes de programación son una manera de estructurar y seccionar nuestro código para así dividir las tareas en múltiples módulos que realizan una acción. Como su nombre indica, cumplen una única función y se pueden utilizar más de una vez dentro de nuestro programa.

C tiene una implementación muy básica pero útil de las funciones, conocidas también como subrutinas.

Un código bien elaborado en C hace uso de las funciones para dividir las tareas a realizarse lo más posible. Es decir, cada subrutina debe de ejercer una única función que, en conjunto con otras funciones, llegan a completar tareas más complejas.

El punto de las funciones es reutilizar código, para así evitar reescribir instrucciones utilizadas más de una vez, lo que permite encontrar errores más fácilmente y arreglarlo en todas las instancias donde se utiliza dicha función errónea.

Por último, las funciones nos permiten implementar ciertos conceptos de diseño como el principio de abstracción[1]

[1]The Abstraction Principle: https://www.cs.sjsu.edu/~pearce/modules/lectures/ood/principles/Abstraction.htm

Sintaxis

Ya hemos utilizado funciones en ejemplos prévios, específicamente en nuestros programas debe de existir siempre una función `main()`. Esta función indica la función principal a ejecutarse dentro de nuestro programa, y como toda función puede admitir parámetros. Dado que hemos visto esta función, veamos más a profundidad cómo declarar una función:

```
<type> <nombre>([parametros]) {
    ...
    codigo
    ...
}
```

En donde:

- type: El tipo de dato que devolverá la función. Existe una excepción a esto, con funciones de tipo `void`, los cuales no devuelven ningún valor.
- nombre: El nombre con el cual se puede llamar a la función dentro de nuestro código.
- [parametros]: Una lista separada por comas de los parámetros a utilizarse dentro de nuestra función. Cada parámetro debe de declararse con el tipo de dato que será junto con un identificador o nombre.
- {codigo}: El bloque o cuerpo de la función, que contiene las estructuras a ejecutarse al llamarse la función.

Conociendo ya las partes, observemos el siguiente ejemplo básico de una función:

```
int cuadrado(int x) {
    return (x*x);
}
```

Observemos las partes de esta función:

- type: `int`, un valor entero
- nombre: `cuadrado()`
- [parametros]: `int x`, un único parámetro entero con nombre `x`
- {codigo}: `return (x*x);`, devuelve el resultado de `x*x`

Conociendo esto y analizando el código de esta función, podemos

conocer que `cuadrado()` es una función que nos devuelve el cuadrado de un número `x`. Una vez que hayamos declarado y definido nuestra función, la podemos utilizar en nuestro código cuantas veces sea necesario:

```c
#include <stdio.h>

// declaracion de funcion 'cuadrado'
int cuadrado(int x) {
    return (x*x);
}

int main() {
    int x = 10;
    int y = 4;

    // primer uso de funcion 'cuadrado'
    printf("10 al cuadrado es %d\n", cuadrado(x));

    // segundo uso de funcion 'cuadrado'
    printf("4 al cuadrado es %d\n", cuadrado(y));

    return 0;
}
```

El resultado de este código en la consola debe ser la siguiente:

```
10 al cuadrado es 100
4 al cuadrado es 16
```

Alcance (Scope)

El alcance o scope de una función hace referencia a qué variables y valores tiene disponible para su uso dentro de ella. Cada función únicamente puede acceder a valores que estén dentro de su alcance, por lo que entender este concepto es fundamental.

Una función siempre tendrá acceso a los valores que se definen dentro de el, pero nunca tendrá acceso a aquellos que se definen fuera de el.

De igual manera, algunas estructuras de control (particularmente los `if`) utilizan variables para llevar a cabo ciertas funciones. Dichas variables únicamente tienen un alcance dentro de la estructura de control relevante, por lo que su uso fuera de la estructura es imposible.

Veamos un ejemplo utilizando dos funciones:

```c
#include <stdio.h>

int funcion(int x) {
    int y = 5;
    return (x*y);
}

int main() {
    int x = 10;
    printf("%d\n", y);
    return 0;
}
```

En este programa podemos observar que definimos una función `funcion()`. Dentro de dicha función, se declara una variable `int y = 5`. Esta variable únicamente tiene alcance dentro de la función `funcion()`, y ya que se está intentando acceder a este valor fuera de su alcance obtendremos un error al momento de intentar compilar el programa:

```
test.c: In function ‘main’:
test.c:10:18: error: ‘y’ undeclared
(first use in this function)
   10 |     printf("%d", y);
      |                  ^
```

Temas importantes

Declaración inicial (Forward Declaration)

La declaración inicial, o forward declaration como se conoce en inglés, es la capacidad de declarar una función junto con sus parámetros, pero definir el cuerpo de la función después en el código fuente. Esta funcionalidad permite que, al editar y analizar código, podamos observar funciones o declaraciones más importantes o relevantes en líneas más cercanas al principio del archivo, y dejar las definiciones de otras funciones para después.

Existen lenguajes que no requieren declaraciones iniciales, como Python, pero en C es **obligatorio** declarar una función ántes de llamarlo dentro de otra. Es por esto que es importante conocer la declaración inicial.

Tomemos este ejemplo, en donde primero se declara la función de cuadrado, y se define después en el código:

```c
#include <stdio.h>

// Declaracion de funcion 'cuadrado'
int cuadrado(int x);

int main() {
    int x = 10;

    // Uso de 'cuadrado' en funcion principal
    printf("El cuadrado de 10 es %d\n", cuadrado(x));
    return 0;
}

// Definicion de funcion 'cuadrado'
int cuadrado(int x) {
    return (x*x);
}
```

Como se puede observar, en este ejemplo primero se realiza la declaración inicial de nuestra función `cuadrado()`, pero no definimos el cuerpo de la función. Seguido de esto declaramos y definimos

nuestra función principal, y al final definimos el cuerpo de la función `cuadrado`. Nótese que tanto en la declaración como definición es importante definir los parámetros de entrada, y estos deben de coincidir tanto en la declaración como en la definición.

La salida de este programa debe ser la siguiente:

```
El cuadrado de 10 es 100
```

Recursión

La recursión es la propiedad de una entidad para contenerse a si mismo [2]. En términos de programación, se refiere a programas, funciones o subrutinas que tienen la capacidad de llamarse a si mismos dentro de su cuerpo de código. Esto nos permite crear funciones que, al llamarse a si mismos, ejecuten las mismas instrucciones definidas en ella.

Es importante notar que, aunque la recursión es una propiedad importante y útil, también puede presentar un peligro de programación, ya que una mala definición de recursividad puede llevar a una ejecución infinita.

C permite la recursión de funciones sin necesidad de crear definiciones o configuraciones externas. Únicamente es importante tomar en cuenta la integración de una condición de salida.

Observemos un ejemplo donde se utiliza la recursión para devolver el factorial de un valor:

```
int factorial(int x) {
    if (x<=1) {
        return x;
    }
    return (x*factorial(x-1));
}
```

Observemos el flujo de esta función. En primera instancia, se checa si el valor de `x` es menor o igual a `1`. Si es el caso, únicamente se devuelve el valor de `x`. De otro modo, se devuelve el valor de `x` multiplicado por el factorial de `x-1`. De este modo, el valor de `x` va disminuyendo gradualmente, y una vez que se alcance el valor de salida, todas las multiplicaciones recursivas se realizan, terminando en devolver la factorial de nuestro valor `x`.

Tomando el siguiente código fuente:

[2] Recursión: https://es.wikipedia.org/wiki/RecursiÃşn

```c
#include <stdio.h>

int factorial(int x);

int main() {
    int x;
    printf("Introduzca un numero: ");
    scanf("%d", &x);
    printf("%d! es igual a %d\n", x, factorial(x));
    return 0;
}

int factorial(int x) {
    if (x<=1) {
        return x;
    }
    return (x*factorial(x-1));
}
```

El resultado al ingresar el valor 5 es el siguiente:

```
5! es 120
```

Conclusión

Las funciones son una estructura fundamental dentro de la programación en C, y entender cómo funcionan, cómo definirlos y cómo implementarlos, así como su comportamiento, es necesario al momento de escribir código.

Llevar a cabo las mejores prácticas al crear funciones también es fundamental para tener código legible y fácil de entender, seccionando las funcionalidades y responsabilidades del código en partes que se pueden desarrollar independientemente unos de otros, y así evitar también reescribir código en distintos lugares.

Tarea opcional
Se recomienda practicar el uso de funciones para cimentar mejor las ideas vistas en la sesión. Para ello, se solicita al interesado crear un programa que implemente una función llamada `fibonacci()`. Esta función deberá devolver el valor número `n` de la secuencia Fibonacci[3], utilizando recursión si es posible.

[3] Fibonacci Sequence: https://www.mathsisfun.com/numbers/fibonacci-sequence.html