Introducción a las funciones en C

UTN FRT
Algoritmos y estructuras de datos
(54) 000 0000

Empezamos..

26/03/23

Concepto

Las funciones son bloques de código que se pueden invocar desde cualquier parte de un programa. Su propósito es dividir un programa en tareas más pequeñas y manejables. Las funciones también permiten reutilizar el código, lo que significa que una función puede ser utilizada en varios lugares diferentes dentro de un programa.

Definición

En C, se define una función utilizando la siguiente sintaxis:

```
tipo_retorno nombre_funcion(parametros){
    // Código de la función
    return valor_retorno;
}
```

Especificaciones y llamado

Donde tipo_retorno es el tipo de valor que devuelve la función (por ejemplo, int, float, char, etc.), nombre_funcion es el nombre que se le dará a la función, y parametros es una lista separada por comas de los parámetros que la función puede recibir.

La sección de código dentro de las llaves {} es el cuerpo de la función, que contiene las instrucciones que la función realiza cuando se llama.

La función puede o no devolver un valor utilizando la instrucción return. Si la función no devuelve un valor, entonces el tipo de retorno debe ser void. Si la función devuelve un valor, el tipo de valor que devuelve debe coincidir con el tipo de retorno declarado en la definición de la función.

Para hacer un llamado se usa el siguiente código: *nombre_funcion(parametros);*

Donde nombre_funcion es el nombre de la función que se va a llamar y parametros son los valores que se le pasan a la función (si la función espera recibir parámetros). Es posible llamar a una función desde otra función. Para hacerlo, simplemente se llama a la función desde el cuerpo de la otra función.

Argumentos y retorno de funciones

Las funciones también pueden recibir argumentos o parámetros, que son valores que se le pasan a la función para que los utilice en su ejecución. Por ejemplo, podríamos tener una función que calcule la suma de dos números y que reciba como argumentos los dos números a sumar:

```
int sumar(int a, int b) {

    int resultado = a + b;

    return resultado;

}
```

En este caso, la función sumar recibe dos argumentos (a y b) y retorna el resultado de la suma. En el cuerpo de la función se define una variable resultado que se inicializa con la suma de a y b. Para llamar a esta función, simplemente debemos pasarle dos números como argumentos: `int resultado = sumar(3, 5);  // resultado = 8`

La función sumar se encarga de realizar la suma de los dos números que le pasamos y retorna el resultado, que es almacenado en la variable resultado.

Además de recibir argumentos, las funciones también pueden retornar valores, como en el ejemplo anterior. El tipo de dato que retorna la función se define en la declaración de la función, y se especifica antes del nombre de la función:

```
int sumar(int a, int b) {

int resultado = a + b; return resultado; }
```

Funciones y pasaje de parámetros

Algunas veces, es necesario modificar los valores de las variables que se le pasan a una función. Por ejemplo, podríamos tener una función que multiplique por dos un número y que modifique el valor de la variable original:

```
void duplicar(int* num) {
    *num *= 2;
}
```

En este caso, la función duplicar recibe como argumento un puntero a un entero (int*). En el cuerpo de la función, se modifica el valor del entero apuntado por el puntero utilizando la notación de puntero (*num). Para llamar a esta función, debemos pasarle la dirección de memoria de una variable de tipo entero:

```
int num = 5;
duplicar(&num);  // num = 10
```

En este caso, se le pasa a la función duplicar la dirección de memoria de la variable num, utilizando el operador de referencia (&num). La función modifica el valor de la variable original (num) y duplica su valor. También es posible definir una función que modifique varias variables a la vez utilizando punteros:

```
void sumar_dos_numeros(int a, int b, int* resultado) {
    *resultado = a + b;
}
```

En este caso, la función recibe dos números (a y b) y un puntero a entero (resultado). En el cuerpo de la función se define una variable suma que almacena la suma de a y b. Luego, se asigna el valor de suma a través del puntero resultado, utilizando la notación de puntero (*resultado).

Funciones con argumentos

Hasta ahora hemos visto cómo se puede crear una función que no requiere argumentos, pero en muchos casos necesitamos que una función tome valores de entrada para realizar su tarea. Para hacerlo, podemos definir los argumentos de la función entre paréntesis después del nombre de la función. La sintaxis de una función con argumentos es la siguiente:

```
tipo_de_retorno nombre_de_la_funcion(tipo_de_arg1 arg1, tipo_de_arg2 arg2, ..., tipo_de_argn argn) {
    // Código de la función  }
```

Donde tipo_de_arg1, tipo_de_arg2, ..., tipo_de_argn son los tipos de datos de los argumentos que la función espera recibir. En el siguiente ejemplo, creamos una función que toma dos enteros como argumentos y los suma:

```
int suma(int num1, int num2) {
    return num1 + num2;    }
```

Para llamar a esta función, simplemente la usamos con dos valores enteros como argumentos: *int resultado = suma(4, 5);* El valor de resultado será igual a 9.

Funciones con argumentos de referencia

En algunos casos, es necesario modificar el valor de una variable en una función y hacer que ese cambio sea visible fuera de la función. Para hacer esto, podemos utilizar argumentos por referencia. Los argumentos por referencia son aquellos que se pasan a la función utilizando su dirección de memoria, en lugar de su valor. Para indicar que un argumento es por referencia, usamos el operador **&** antes del nombre de la variable.

La sintaxis de una función con argumentos por referencia es la siguiente:

```
tipo_de_retorno nombre_de_la_funcion(tipo_de_arg1* arg1, tipo_de_arg2* arg2, ..., tipo_de_argn* argn) {
    // Código de la función }
```

Cuando pasamos un argumento por referencia, cualquier cambio que se haga en su valor dentro de la función afectará a la variable original fuera de la función. En el siguiente ejemplo, creamos una función que toma dos enteros por referencia y los suma.

```
void suma_por_referencia(int* num1, int* num2) {
    *num1 += *num2;   }
```

Para llamar a esta función, pasamos dos variables enteras por referencia:

```
int a = 4, b = 5;

suma_por_referencia(&a, &b);
```

Tipos de funciones

Existen varios tipos de funciones que podemos utilizar en C, a continuación se presentan algunos ejemplos:

- Funciones sin valor de retorno y sin parámetros: Son aquellas que no reciben ningún parámetro de entrada y no devuelven ningún valor de salida. Su sintaxis es la siguiente:

```
void nombre_funcion(void) { // Cuerpo de la función }
```

- Funciones con valor de retorno y sin parámetros: Son aquellas que no reciben ningún parámetro de entrada, pero devuelven un valor de salida. Su sintaxis es la siguiente: tipo_de_dato nombre_funcion(void)

```
{ // Cuerpo de la función

return valor_de_salida;

}
```

- Funciones sin valor de retorno y con parámetros: Son aquellas que reciben uno o más parámetros de entrada, pero no devuelven ningún valor de salida. Su sintaxis es la siguiente:

```
void nombre_funcion(tipo_de_dato parametro1, tipo_de_dato parametro2, ...)

{ // Cuerpo de la función }
```

- Funciones con valor de retorno y con parámetros: Son aquellas que reciben uno o más parámetros de entrada y devuelven un valor de salida. Su sintaxis es la siguiente:

```
tipo_de_dato nombre_funcion(tipo_de_dato parametro1, tipo_de_dato parametro2, ...)
{ // Cuerpo de la función
    return valor_de_salida;
}
```

Es importante tener en cuenta que el tipo de dato que se devuelve en una función con valor de retorno debe ser compatible con el tipo de dato declarado en la firma de la función. Además, los nombres de los parámetros declarados en la firma de la función son opcionales y se utilizan para referirse a los mismos dentro del cuerpo de la función.