Functions in C

Md. Aminul Islam Shazid

22 Nov 2025

Outline

Introduction

Introduction to Functions in C

- A function is a *reusable* block of code that performs a specific task
- Functions help organize programs into smaller and manageable sections
- The `main()` function is the entry point to every C program

Why Use Functions

- To avoid repeating the same code
- To make programs easier to understand and maintain
- To divide a large problem into smaller parts
- To allow reusability of code

Advantages of Using Functions

- Reduces code duplication
- Enhances readability
- Helps debugging and testing individual parts easily
- Supports modular program design

Syntax

Syntax of a Function

```
return_type function_name(parameter_list) {
    // body of the function
    return the_return_value;  // optional
}
```

- Function declaration tells the compiler about the function
- Function definition contains the actual code
- Function call transfers control to the function

Example: Function with No Parameters

```c
void greet() {
    printf("Hello, World!");
}
```

Example: Function with One Parameter

```c
void printNumber(int n) {
    printf("The number is %d", n);
}
```

Return Type and Return Value

```
int square(int n) {
    return n * n;
}
```

- The return type defines the type of value a function returns
- The return statement sends a value back to the calling code

Example: Function with Multiple Parameters

```c
int add(int a, int b) {
    return a + b;
}
```

Calling Functions

```
greet();                    // no parameter
printNumber(5);             // one parameter
sum = add(4, 6);            // multiple parameters
```

Types of Functions

- Library functions - predefined in header files like `printf()`, `scanf()`, `sqrt()`
- User-defined functions - created by the programmer

Examples

Example: Function with No Parameters and No Return Value

```c
#include <stdio.h>

void greet(){
    printf("Hello, World!\n");
}

int main(){
    greet();
    return 0;
}
```

Example: Function with One Parameter and No Return Value

```c
#include <stdio.h>

void printSquare(int n){
    printf("Square of %d is %d\n", n, n * n);
}

int main(){
    printSquare(5);
    return 0;
}
```

Example: Function with One Parameter and a Return Value

```c
#include <stdio.h>

int getSquare(int n){
    return n*n;
}

int main(){
    printf("The square of 5 is %d", getSquare(5));
    return 0;
}
```

Example: Function with Multiple Parameters and Return Value

```c
#include <stdio.h>

int add(int a, int b){
    return a + b;
}

int main(){
    int result = add(10, 20);
    printf("Sum = %d\n", result);
    return 0;
}
```

Example: Function Returning a Value Without Parameters

```c
#include <stdio.h>

int meaningOfLife(){
    return 42;
}

int main(){
    int meaning = meaningOfLife();
    printf("The meaning of life is %d\n", meaning);
    return 0;
}
```

Example: Function That Calls Another Function

```c
#include <stdio.h>

int square(int n){
    return n * n;
}

void showSquare(int x){
    printf("Square of %d is %d\n", x, square(x));
}

int main(){
    showSquare(7);
    return 0;
}
```

Recursive Functions

- A recursive function calls itself
- Must have a base case to stop recursion

```c
int factorial(int n){
    if(n == 0){
        return 1;
    } else{
        return n*factorial(n-1);
    }
}
```

Function Prototype

What Is a Function Prototype?

A function prototype is a declaration of a function without its body.

```c
int add(int a, int b);      // prototype only
```

- Ends with a semicolon.
- Parameter names are optional:
  ```c
  int add(int, int);
  ```
- Must match the definition exactly: return type, name, and parameter types.

Why Function Prototypes?

- The compiler reads code top to bottom
- A function must be known before it is called
- If `main()` calls a function defined later, the compiler has no information about that function
- A prototype supplies the missing information: return type, function name, and parameter types
- Prototypes enable type checking at compile time

Compiler Perspective

The compiler must know:

- How many arguments a function takes
- What their types are
- What return type to expect

Example: Prototypes in a Single C File

```c
#include <stdio.h>

int add(int a, int b);     // function prototype

int main(){
    int result = add(10, 20);
    printf("Result: %d", result);
    return 0;
}

int add(int a, int b){     // function definition
    return a + b;
}
```

Header Files

Introduction

- Sometimes it may be necessary to use the same functions/constants across multiple C files in a project
- In such cases, it is infeasible to define the functions/constants in each C file
- Header files (files with `.h` extension) help reuse function/constant definition across multiple C files
- Header files have no `main()` function

Example Header File

```c
#include <stdio.h>
#define PI 3.142

void sayHello(){
    printf("Hello World");
}

int add(int a, int b){
    return a+b;
}

float area(float radius){
    return PI*radius*radius;
}
```

Save this file as `utils.h` (you can use any name you want).

Using User-Created Header File

```c
#include <stdio.h>
#include "utils.h"    // NOTE: quotation marks

int main(){
    sayHello();
    printf("\n%d\n", add(3, 2));
    printf("%f", area(1.5));
}
```

Exercise

Exercises

1. Write a function to find the maximum of two numbers
2. Write a function that checks if an integer is even or odd
3. Write a function that takes three numbers and returns their average
4. Write a recursive function to calculate the sum of digits of an integer
5. Write a recursive function to calculate the GCD of two integers
6. Write a function that checks whether a given integer is prime
7. Write a function to print all prime numbers between 1 and 100