

Functions in C

Md. Aminul Islam Shazid

30 Oct 2025

Outline

- 1 Introduction
- 2 Syntax
- 3 Examples
- 4 Function Prototype
- 5 Header Files
- 6 Exercise

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

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

Example: Function with One Parameter

```
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

```
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

```
1  #include <stdio.h>
2
3  void greet(){
4      printf("Hello, World!\n");
5  }
6
7  int main(){
8      greet();
9      return 0;
10 }
```


Example: Function with One Parameter and No Return Value

```
1  #include <stdio.h>
2
3  void printSquare(int n){
4      printf("Square of %d is %d\n", n, n * n);
5  }
6
7  int main(){
8      printSquare(5);
9      return 0;
10 }
```

Example: Function with One Parameter and a Return Value

```
1  #include <stdio.h>
2
3  int getSquare(int n){
4      return n*n;
5  }
6
7  int main(){
8      printf("The square of 5 is %d", getSquare(5));
9      return 0;
10 }
```

Example: Function with Multiple Parameters and Return Value

```
1  #include <stdio.h>
2
3  int add(int a, int b){
4      return a + b;
5  }
6
7  int main(){
8      int result = add(10, 20);
9      printf("Sum = %d\n", result);
10     return 0;
11 }
```

Example: Function Returning a Value Without Parameters

```
1  #include <stdio.h>
2
3  int meaningOfLife(){
4      return 42;
5  }
6
7  int main(){
8      int meaning = meaningOfLife();
9      printf("The meaning of life is %d\n", meaning);
10     return 0;
11 }
```

Example: Function That Calls Another Function

```
1  #include <stdio.h>
2
3  int square(int n){
4      return n * n;
5  }
6
7  void showSquare(int x){
8      printf("Square of %d is %d\n", x, square(x));
9  }
10
11 int main(){
12     showSquare(7);
13     return 0;
14 }
```

Recursive Functions

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

```
1 int factorial(int n){  
2     if(n == 0){  
3         return 1;  
4     } else{  
5         return n*factorial(n-1);  
6     }  
7 }
```

Function Prototype

What Is a Function Prototype?

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

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

- Ends with a semicolon.
- Parameter names are optional:

```
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

```
1  #include <stdio.h>
2
3  int add(int a, int b);    // function prototype
4
5  int main(){
6      int result = add(10, 20);
7      printf("Result: %d", result);
8      return 0;
9  }
10
11 int add(int a, int b){    // function definition
12     return a + b;
13 }
```

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

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