

EEE: 103

Computer Programming

Functions in C

Modular Programming

Instructor: Sk Tahmed Salim Rafid

Introduction to Functions

A function is a block of code that performs a specific task.

Benefits of Using Functions:

- 🕒 ✓ Better readability and modularity
- 🕒 ✓ Reusability of code
- 🕒 ✓ Easier debugging
- 🕒 ✓ Structure and organization

Basic Structure of a Function

1. Function Declaration (Prototype)

Tells compiler about function before use

2. Function Definition

Contains the actual code/logic

3. Function Call

Executes the function

Function Syntax

Function Declaration

```
return_type function_name(parameter_list);
```

Example:

```
int add(int x, int y);
```

Function Definition

```
return_type function_name(parameter_list) {  
    // function body  
}
```

Example:

```
int add(int x, int y) {
```

Function Prototype

Tells the compiler about the function before it is used

Tells compiler about:

- ▶ Function name
- ▶ Return type
- ▶ Parameter types

Used before the main() function

```
int add(int x, int y); // prototype
```

Function Definition Example

Complete function with definition

```
#include <stdio.h>

int add(int x, int y) {
    return x + y;
}

void main() {
    int result = add(5, 3);
    printf("Sum = %d\n", result);
}
```

Output:

Sum = 8

Types of Functions

Three main categories based on return type and parameters

A. Void Function (No Return Value)

Function that performs action but returns nothing

```
#include <stdio.h>

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

void main() {
    greet();
}
```

Output:

Hello!

B. Function With Return Value

Function that calculates and returns a result

```
#include <stdio.h>

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

void main() {
    int r = square(6);
    printf("%d", r);
}
```

Output:

36

C. Function with Parameters (Pass-by-Value)

C uses pass by value - function receives a copy of the variable

```
#include <stdio.h>

void work(int x) {
    printf("I got %d in work", x);
}

void main() {
    int a = 5;
    work(a);
}
```

Output:

I got 5 in work

Variables Scope

Local Variables vs Global Variables

Local Variables

- Declared inside a function
- Exist only during function execution
- Not accessible from other functions
- Each function can have its own local variable with same name

Local Variables - Example

Each function has its own x variable

```
#include <stdio.h>

void test() {
    int x = 10; // local variable
    printf("%d\n", x);
}

void main() {
    int x = 5;
    printf("%d\n", x);
    test();
    printf("%d", x);
}
```

Output:

```
5
10
5
```

Global Variables

- Declared outside all functions
- Accessible by all functions in the program
- Created when program starts
- Destroyed when program ends

Global Variables - Example

Variable g is accessible everywhere

```
#include <stdio.h>

int g = 50; // global variable

void show() {
    printf("Inside show: %d\n", g);
}

void main() {
    printf("Inside main: %d\n", g);
    g = 20;
    show();
}
```

Output:

```
Inside main: 50
Inside show: 20
```

Local vs Global - Combined Example

Local variable shadows global variable

```
#include <stdio.h>

int g = 20; // global

void func() {
    int g = 5; // local (shadows global)
    printf("Local g = %d\n", g);
}

void main() {
    printf("Global g = %d\n", g);
    func();
}
```

Output:

```
Global g = 20
Local g = 5
```

Practice Problems (Homework)

- ▶ Question 1: Write a function that returns the factorial of a number
- ▶ Question 2: Write a function to convert Celsius to Fahrenheit
- ▶ Question 3: Write a void function that prints the sum of first N numbers
- ▶ Question 4: Write a function that checks whether a number is prime
- ▶ Question 5: Write a function to compute power(a, b) without using pow()

Thank You

Keep Coding!