IEEE RAS
ROBOTICS & AUTOMATION SOCIETY
Zagazig student chapter

TASK 2

Embedded Systems Track

DATE :

27/3/2025

PRESENTED BY :

MOHAMED KADRY

Table Of Content

1 **Defining and Using Functions in C**

1.1 Introduction to Functions

A function in C is a block of reusable code designed to perform a specific task. Functions help in breaking a complex program into smaller, manageable parts, making the code more readable, modular, and easier to debug.

1.2 Function Declaration and Definition

Function Declaration:
It tells the compiler about the function's name, return type, and parameters before its actual definition.

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

Function Definition:
It contains the actual code and logic of the function.

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

1.3 Function Parameters and Return Types

- **Parameters: These are the values passed to the function when called.**
- **Return Type: Defines the type of value the function returns. If a function does not return a value, it uses void.**

1.4 Calling Functions

A function is executed when it is called within the main function or another function.

```
int result = add(5, 3); // Calling the function
```

1.5 Scope of Variables in Functions

- **Local Variables: Declared inside a function and cannot be accessed outside it.**
- **Global Variables: Declared outside all functions and accessible throughout the program.**

2 Performing Arithmetic Operations with Numeric Variables in C

2.1 Basic Arithmetic Operations

C supports fundamental arithmetic operations:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%) (remainder of division)

```c
int a = 10, b = 3;
int sum = a + b; // 13
int mod = a % b; // 1
```

2.2 Operator Precedence and Associativity

- Operators like *, /, % have higher precedence than +, -.
- Parentheses () can be used to change evaluation order.

2.3 Type Conversion in Arithmetic Operations

- Implicit Conversion: Converts smaller data types to larger ones automatically.

```c
int x = 5;
float y = 2.5;
float result = x + y; // x is converted to float
```

```c
float avg = (float)(a + b) / 2;
```

2.4 Arithmetic Functions in C

C provides standard library functions for arithmetic operations:

- pow() - Exponentiation
- sqrt() - Square root
- abs() - Absolute value

```c
#include <math.h>
double root = sqrt(25.0); // Returns 5.0
```

3 What is Recursion?

3.1 Definition of Recursion

Recursion is a programming technique where a function calls itself to solve a problem by breaking it into smaller subproblems.

3.2 How Recursion Works

A recursive function must have:

- A base case (stopping condition)
- A recursive case (calls itself with a smaller problem)

Example (Factorial Calculation):

```
int factorial(int n) {
    if (n == 0) return 1; // Base case
    else return n * factorial(n - 1); // Recursive case
}
```

3.3 Base Case and Recursive Case

- Base Case: Prevents infinite recursion.
- Recursive Case: Reduces problem complexity step by step.

4 Types of Recursion

4.1 Direct Recursion

A function calls itself directly.

```
void func() {
    func(); // Direct recursion
}
```

4.2 Indirect Recursion

Functions call each other in a cyclic manner.

```
void funcA() { funcB(); }
void funcB() { funcA(); }
```

4.3 Tail Recursion

The recursive call is the last operation before returning the result, allowing compiler optimizations.

```
int factorial(int n, int acc) {
    if (n == 0) return acc;
    return factorial(n - 1, n * acc); // Tail recursion
}
```

4.4 Non-Tail Recursion

The recursive call is not the last operation in the function.

```
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1); // Non-tail recursion
}
```

4.5 Nested Recursion

A recursive function calls itself inside another recursive call.

```
int ackermann(int m, int n) {
    if (m == 0) return n + 1;
    else if (n == 0) return ackermann(m - 1, 1);
    else return ackermann(m - 1, ackermann(m, n - 1));
}
```