Introduction to Functions in C

# Objective:

The objective of this assignment is to understand the concept of functions in C programming, including function declaration, definition, and calling. You will also learn how to pass arguments to functions and return values from functions.

# Introduction:

A function is a block of code that performs a specific task. In C, functions are used to break down a program into smaller, manageable parts, making it easier to read, maintain, and debug. Functions help in code reuse, reducing redundancy, and improving organization.

In C, functions are classified into two types:

* **Built-in functions**: Functions that are provided by the C standard library (e.g., printf(), scanf()).
* **User-defined functions**: Functions that are defined by the programmer to perform specific tasks.

# Structure of a Function:

A C function generally consists of the following parts:

1. **Function declaration (or prototype)**: It provides the function name, return type, and parameters (if any).
2. **Function definition**: It provides the actual code of the function.
3. **Function call**: It invokes the function to perform its task.

# Syntax:

1. **Function Declaration:**

return\_type function\_name(parameter\_list);

**Example:**

int add(int, int);

1. **Function Definition:**

return\_type function\_name(parameter\_list) {

// body of the function

}

**Example:**

int add(int a, int b) {

return a + b;

}

1. **Function Call:**

function\_name(arguments);

Example:

int result = add(3, 4);

# **Programs:**

## Question: Write a C program that takes two floating-point numbers as input and performs the basic arithmetic operations (addition, subtraction, multiplication, and division) on them. The program should display the result of each operation rounded to two decimal places.

*#include* <stdio.h>

*float* *add*(*float* a, *float* b) {

*return* a *+* b;

}

*float* *sub*(*float* a, *float* b) {

*return* a *-* b;

}

*float* *mul*(*float* a, *float* b) {

*return* a *\** b;

}

*float* *div*(*float* a, *float* b) {

*return* a */* b;

}

*int* *main*() {

*float* *a*, *b*;

*scanf*("%f %f", *&a*, *&b*);

*printf*("%.2f\n", *add*(*a*, *b*));

*printf*("%.2f\n", *sub*(*a*, *b*));

*printf*("%.2f\n", *mul*(*a*, *b*));

*printf*("%.2f\n", *div*(*a*, *b*));

*return* 0;

}

**Output:**

9.00

1.40

19.76

1.37

## **Question:** Write a C program that takes an integer n as input, prints all the divisors of n, and then prints the factorial of n.

*#include* <stdio.h>

*void* *divisor*(*int* n) {

*for* (*int* *i* *=* 1;*i* *<=* n;*i++*) {

*if* (n *%* *i* *==* 0) {

*printf*("%d ", *i*);

        }

    }

}

*int* *fact*(*int* n) {

*if* (n *==* 0 *||* n *==* 1) *return* 1;

*if* (n *<* 0) *return* *-*1;

*int* *fact* *=* 1;

*for* (*int* *i* *=* 1;*i* *<=* n;*i++*) {

*fact* *\*=* *i*;

    }

*return* *fact*;

}

*int* *main*() {

*int* *n*;

*scanf*("%d", *&n*);

*divisor*(*n*);

*printf*("\n");

*printf*("%d", *fact*(*n*));

*return* 0;

}

**Output:**

For Input :6

1 2 3 6

720

## **Question:** Write a C program that takes an integer n as input and computes the factorial of n. If the input is a negative number, the program should print an error message.

*#include* <stdio.h>

*#define* *ll* *long* *long*

*ll* *int* *fact*(*int* n) {

*if* (n *<* 0) {

*printf*("Invalid input!");

*return* 0;

    }

*else* *if* (n *==* 0 *||* n *==* 1) {

*return* 1;

    }

*return* n *\** *fact*(n *-* 1);

}

*int* *main*() {

*printf*("Enter n (n<0):: ");

*int* *n*;

*scanf*("%d", *&n*);

*printf*("%d! = %d",*n*, *fact*(*n*));

*return* 0;

}

**Output:**

For input 5:

5! = 120

For input -3:

Invalid input!

## **Question:** Write a C program that takes an integer n as input and checks whether it is even or odd. The program should print "even" if the number is even and "odd" if the number is odd.

*#include* <stdio.h>

*char\** *check*(*int* n) { *//argument*

*if* (n *%* 2 *==* 0)

*return* "even";

*else*

*return* "odd";

}

*int* *main*() {

*int* *num*;

*scanf*("%d", *&num*);

*printf*("%s ", *check*(*num*)); *//parameter*

*return* 0;

}

**Output:**

For input 4:

even

For input 7:

odd

# Conclusion

Functions help make C programs modular, reusable, and easier to maintain. Mastering functions is essential for efficient programming.