RECURSIVE MATH TOOLBOX

Course : B.tech(CSE – AI & ML)
Subject : Programming in C

Name – Lakshya Dhaneshwar
Guided By – Naina Devi

Rungta International Skills University, Bhilai

AGENDA

- Introduction to Recursion
- Objective of the Mini Project
- Problem Statement
- Concept of Recursion
- Description of Functions Used
- Program Explanation
- Sample Output
- Applications
- Advantages & Limitations
- Conclusion

INTRODUCTION

- Recursion is a programming technique where a function calls itself.
- It helps solve complex problems by breaking them into smaller sub-problems.
- Recursive solutions are often simple, clean, and close to mathematical formulas.
- This mini project demonstrates recursion using common mathematical problems.

OBJECTIVE OF THE MINI PROJECT

- To understand the concept of recursion in C.
- To implement recursive functions for mathematical operations.
- To demonstrate how base cases and recursive calls work.
- To improve problem-solving and logical thinking skills.

PROBLEM STATEMENT

- Design a C program named **“Recursive Math Toolbox”** that performs:
- Factorial calculation
- Fibonacci number generation
- Power calculation
- Sum of digits

All operations must be implemented using **recursive functions.**

WHAT IS RECURSION?

Recursion is when a function calls itself.
Every recursive function has:

- **Base Case** – stops recursion
- **Recursive Case** – calls itself

- Without a base case, recursion leads to infinite calls.

Example:
Factorial
$n! = n \times (n-1)!$

FUNCTIONS USED IN THE PROGRAM

Function Name	Purpose
`factorial()`	Calculates factorial of a number
`fibonacci()`	Finds Fibonacci number
`power()`	Calculates power of a number
`sumDigits()`	Finds sum of digits

FACTORIAL FUNCTION

- **Definition:**
  Factorial of n = n × (n−1) × (n−2) × … × 1
- **Base Case:**
  If $n \leq 1 \longrightarrow$ return 1
- **Recursive Call:**
  factorial(n − 1)
- **Example:**
  $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

FIBONACCI FUNCTION

- **Definition:**

  Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, …
- **Formula:**

  $F(n) = F(n-1) + F(n-2)$
- **Base Cases:**
- $F(0) = 0$
- $F(1) = 1$

POWER FUNCTION

- **Definition:**

  $x^n = x \times x^{n-1}$

- **Base Case:**

  If exponent = 0 → return 1

- **Special Case:**

  Handles negative exponent using reciprocal.

- **Example:**

  $2^5 = 2 \times 2 \times 2 \times 2 \times 2 = 32$

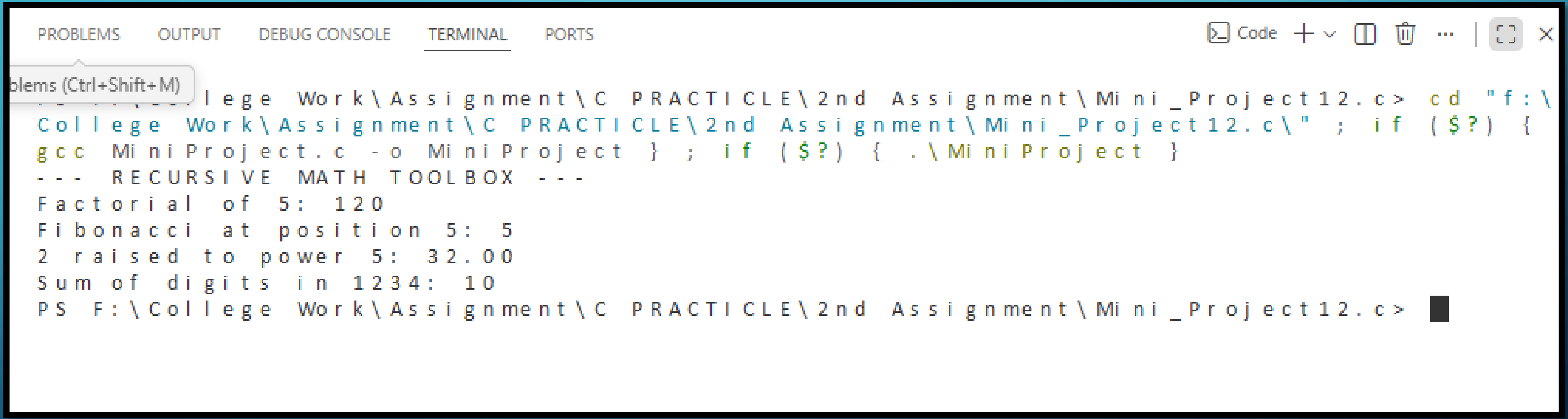
SUM OF DIGITS FUNCTION

- **Definition:**
  Adds digits of a number recursively.
- **Logic:**
- Last digit = n % 10
- Remaining number = n / 10
- **Base Case:**
  If n = 0 → return 0
- **Example:**
  1234 → 1 + 2 + 3 + 4 = 10

MAIN FUNCTION EXPLANATION

- Declares a number (num = 5)
- Calls all recursive functions
- Displays results using printf()
- Acts as a controller for the program

SAMPLE OUTPUT



```
lege Work\Assignment\C PRACTICLE\2nd Assignment\Mini_Project12.c> cd "f:\
College Work\Assignment\C PRACTICLE\2nd Assignment\Mini_Project12.c\" ; if ($?) {
gcc MiniProject.c -o MiniProject } ; if ($?) { .\MiniProject }
--- RECURSIVE MATH TOOLBOX ---
Factorial of 5: 120
Fibonacci at position 5: 5
2 raised to power 5: 32.00
Sum of digits in 1234: 10
PS F:\College Work\Assignment\C PRACTICLE\2nd Assignment\Mini_Project12.c> 
```

APPLICATIONS OF RECURSION

- Mathematical computations
- Tree and graph traversal
- Sorting algorithms (Quick Sort, Merge Sort)
- Dynamic programming
- Compiler design

ADVANTAGES

- Code is shorter and cleaner
- Easy to understand mathematical problems
- Reduces complex logic into simple steps

LIMITATIONS

- Higher memory usage (stack memory)
- Slower than iterative approach for large inputs
- Risk of stack overflow if base case is missing

CONCLUSION

- Recursion is a powerful programming concept.
- This mini project demonstrates recursion through practical examples.
- Understanding recursion improves coding skills and logic building.
- The project successfully meets all objectives.