💻Activity 5: Recursive Fibonacci Program💻

01.RESEARCH(Understanding the topic)

Recursion is a technique in which a function calls itself repeatedly to solve a problem by breaking it into smaller subproblems. Every recursive function is built on two key parts:

- Base Case: The stopping condition that prevents infinite calling.
- Recursive Case: The part where the function calls itself with a modified value.

In the Fibonacci problem, recursion fits naturally because Fibonacci itself is defined using its previous two values: $F(n) = F(n-1) + F(n-2)$.

Using recursion with pointers in C helps students understand two important concepts together:

1. How recursion reduces a problem step-by-step until the base case.
2. How pointers allow passing memory addresses instead of direct values.

This combination helps visualize how memory and function calls interact, especially when multiple recursive calls are executed.

Recursive techniques are widely used in:

Mathematical problems (factorial, Fibonacci, power functions).

Searching large directory structures.

Working with data structures like binary trees, graphs, and linked lists.

Algorithms such as quick sort, merge sort, DFS, and backtracking

Sources:

GeeksforGeeks – Recursion Basics

https://www.geeksforgeeks.org/recursion/

W3Schools – C Functions & Recursion
https://www.w3schools.com/c/c_functions_recursion.php

TutorialsPoint – Recursion in C

https://www.tutorialspoint.com/cprogramming/c_recursion.htm

02.Analysis(Breaking down the problem)

Understanding the problem:

The objective of my project is to generate the Fibonacci series using a recursive function that takes a pointer as its argument. The program receives a value n and computes the Fibonacci number for each term from 0 to n-1.

The recursive function checks two base conditions:

- If the number is 0 → return 0
- If the number is 1 → return 1

For any other value, the function creates two smaller subproblems:
n-1 and n-2.
Their addresses are passed to the function recursively. This reduces the problem size in each step.

As recursion continues, new stack frames are created for each call. Pointer usage ensures that the function reads the current value from the memory location passed, improving understanding of memory-based operations. Once the base cases are reached, recursion begins returning values back up the call chain. These returned values are added to produce the final Fibonacci number.

Finally, the main program prints the entire Fibonacci series using repeated recursive calls. This approach effectively demonstrates how recursion works internally and how a simple mathematical formula can be implemented using pointer-based recursive logic.

03.Ideate(Planning our solution):

Problem Statement

The aim of this program is to calculate Fibonacci values using recursion in C. The user will enter an integer n, and the program will either generate the Fibonacci series up to n terms or compute the depending on the requirement. The program uses the recursive definition:
$F(n) = F(n-1) + F(n-2)$
with essential base conditions $F(0) = 0$ and $F(1) = 1$. Recursion breaks a big problem into smaller subproblems by repeatedly calling the same function with reduced values until the base condition is reached. The program will then return values step by step and finally display the correct Fibonacci output.

Algorithm

```
Step 1: Start
Step 2: Input integer n
Step 3: Call the recursive function fibonacci(n)
Step 4: Inside the function:
    • If n == 0, return 0
    • Else if n == 1, return 1
    • Otherwise return fibonacci(n-1) + fibonacci(n-2)
Step 5: Receive the returned Fibonacci value(s)
Step 6: Display the nth Fibonacci number or series
Step 7: Stop
```

To design this program, the main idea was to use recursion because Fibonacci naturally fits the concept of repeated breakdown. The Fibonacci sequence builds each number using the previous two numbers, which aligns perfectly with recursive structure.

The first step was identifying the base cases because recursion must stop somewhere; otherwise, it leads to infinite calls. So we fix the foundation:

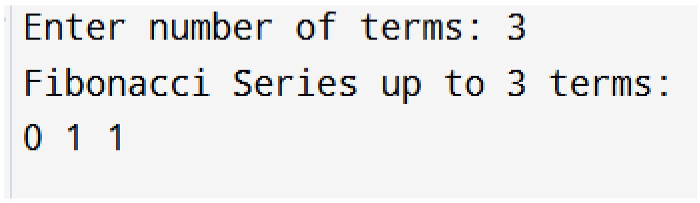
- When n = 0, Fibonacci is 0

- When n = 1, Fibonacci is 1

04.Build(Understanding the program)

```
#include <stdio.h>

int fibonacci(int *n) {
    if (*n == 0)
        return 0;
    if (*n == 1)
        return 1;
    int a = *n - 1;
    int b = *n - 2;

    return fibonacci(&a) + fibonacci(&b);
}

int main() {
    int i, n;

    printf("Enter number of terms: ");
    scanf("%d", &n);

    printf("Fibonacci Series up to %d terms:\n", n);

    for (i = 0; i < n; i++) {
        int x = i;
        printf("%d ", fibonacci(&x));
    }

    return 0;
}
```

05.Output



```
Enter number of terms: 3
Fibonacci Series up to 3 terms:
0 1 1
```

06.Conclusion

This project successfully demonstrates how recursion can be used to generate the Fibonacci series by breaking the problem into smaller subproblems. The recursive function uses clear base conditions and pointer-based calls to compute each term without using loops. The program produces accurate results and highlights the power of recursion in solving mathematical problems in a simple and logical way.

07.Implementation