Lab-06

Recursion Theory

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

- A recursive algorithm takes one step toward solution and then recursively call itself to further move. The algorithm stops once we reach the solution.
- Since called function may further call itself, this process might continue forever. So it is essential to provide a base case to terminate this recursion process.

Algorithms 1.

Factorial of a number

Step 1: Start

Step 2: Read number *n*

Step 3: Call factorial(n)

Step 4: Print factorial

Step 5: Stop factorial(n)

Step 1: If n==0 then return 1

Step 2: Else return **n*factorial(n-1)**

2. N^{th} term of Fibonacci series

Step 1: Start

Step 2: Read number *n*

Step 3: Call fibo(n)

Step 4: Print the required term

Step 5: Stop

fibo(n)

Step 1: If n==0 then return 0

Step 2: Else if (n==1||n==2) then return 1

Step 3: Else return **fibo(n-1)+fibo(n-2)**

3. Tower of Hanoi

To move n disk from A to C using B as auxiliary **Step**

1: Declare and initialize necessary variables.

n=no. of disks

'A','B','C' for three pegs being used

Step 2: if n==1

Move the single disk from A to C and stop

Step 3: Otherwise

- Move the top n-1 disks from A to B using C as auxiliary
- Move the remaining disk from A to C
- Move the n-1 disks from B to C using A as auxiliary **Step 4:** Stop

Source codes:

1. Factorial of a number

```c
#include <stdio.h>
int fact (int n)
{
    if(n==0)
{
        return 1;
    }
else
    return (n*fact(n-1));
}
int main()
{
int n;
    printf("Enter a number:");
scanf("%d",&n);
    printf("FActorial of %d =
%d",n,fact(n));
   return 0; }
```

Process feedback link: https://app.processfeedback.org/coding/c_2025-02-03_20-07_58143c05-84cf-46c2-8776-f517c6c86601?report=true

2. Nth term of Fibonacci series

```c
#include <stdio.h>
int fibo(int n)
{
    if(n==0)
return 0;
if(n==1||n==2)
return 1;
else
    return (fibo(n-1)+fibo(n-2));
}
int main()
{     int
n;
    printf("Enter the term:");
scanf("%d",&n);
    printf("%dth term= %d",n,fibo(n));
return 0;
}
```

Process feedback link: https://app.processfeedback.org/coding/c_2025-02-03_20-14_1eee65e9-229c-434e-b1ae-718c0faac3fa?report=true

3. Tower of Hanoi

```c
#include <stdio.h>
void TOH(int n,char A,char B,char C)
{
    if(n>0)
    {
        TOH(n-1,A,C,B);
        printf("\n Move disk %d from rod %c to rod %c", n, A,C);
        TOH(n-1,B,A,C);
    } }
int main()
{     int
n;
    printf("Enter no of disks:");
    scanf("%d",&n);
TOH(n,'A','B','C');     return
0;
}
```

Process feedback link: https://app.processfeedback.org/coding/c_2025-02-03_20-21_ca650b7f-511a-4a08-8edc-5101cfd9579e?report=true

Discussion and Conclusion:

n this lab report on recursion theory in, we explored the fundamental concepts of recursion, its implementation, and its applications. Through various experiments, we analyzed recursive functions, including base and recursive cases, and their impact on time and space complexity. The results demonstrated that while recursion simplifies problem-solving by breaking tasks into smaller subproblems, excessive recursion can lead to stack overflow and inefficiency. Optimizations like tail recursion and memoization can improve performance. In conclusion, recursion is a powerful technique, but careful consideration of its efficiency is crucial when designing algorithms.

Biraj Pandey

ACE079BCT023