Recursion

When a function calls itself (either directly or indirectly) then it is called recursive function and process is called recursion.”

Principles of Recursion

- Original Problem can be decompose into small size of same type of problem.
- Recursive call must reduce the size of problem.
- Base condition must be present.
- Base condition must be reachable.

In order for the definition not to be circular, it must have two properties:

- There must be certain arguments called base value for which the function doesn’t refers to itself.
- Each time the function does refers to itself , the argument of the function must be closer to a base value.

A recursive function with these two properties is also said to be well-defined.

Recursion is done with the help of stack data structure.

A stack is a **linear data structure,** a collection of items of the same type. In a stack, the insertion and deletion of elements happen only at one endpoint. The behavior of a stack is described as “**Last In, First Out” (LIFO).**

Stack operations:

1. Push
2. Pop

What is the difference between recursion and iteration

Recursion: A program is called recursive when an entity calls itself.

Iterative: when a program is call iterative when there is a loop.

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.

Property	Recursion	Iteration
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).
Space Complexity	The space complexity is higher than iterations.	Space complexity is lower.
Stack	Here the stack is used to store local variables when the function is called.	Stack is not used.
Speed	Execution is slow since it has the overhead of maintaining and updating the stack.	Normally, it is faster than recursion as it doesn't utilize the stack.
Memory	Recursion uses more memory as compared to iteration.	Iteration uses less memory as compared to recursion.
Overhead	Possesses overhead of repeated function calls.	No overhead as there are no function calls in iteration.
Infinite Repetition	If the recursive function does not meet to a termination condition or the base case is not defined or is never reached then it leads to stack overflow error and there is a chance that system may crash in infinite recursion.	If the control condition of the iteration statement never becomes false or control variable does not reach the termination value, then it will cause infinite loop. On infinite loop, it uses the CPU cycles again and again.

Algorithm of Factorial with Recursion

ALGORITHM Fact (N)

Input: Any positive number N

Output: factorial of N

```
BEGIN:
        IF N<0
                Display negative numbers not allowed
                EXIT
        ELSE IF N==0 || N==1 THEN                        //BASE Condition
                RETURN 1
        ELSE
                RETURN N * Fact(N-1)              //Recursive Call
END;
```

```
 // find the factorial of a number using recursion
#include<stdio.h>
long int fact (int);  // function prototype or declaration
void main()
{
   long int f;
   int n;
   printf("enter no");
   scanf("%d",&n);
   f=fact(n);   // function calling
   printf("factorial is \t%ld",f);
}
//function definition
  long int fact(int num)
  {
     if(num<0)
     {
          printf("negative number not allowed");
          exit(0);
        }
        else if (num==0 ||num==1)        //base condtion
            return(1);
        else
            return (num *fact(num-1));    //recursive call
}

// Write a program to find sum of Fibonacci series using recursion.
#include<stdio.h>
int fib(int);      // function prototype
  int main()
  {
     int n,i,sum=0;
     printf("Enter number of term in fibonacci series\n");
     scanf("%d",&n);
     for(i=1;i<=n;i++)
     {
        sum=sum+fib(i);             //function calling
     }
     printf("Sum of the series is: %d",sum);
     return 0;
  }
// function definition
  int fib(int n)
  {
    if(n==1)                          // base condition
     return 0;
    else if(n==2)                            //base condition
```

```
     return 1;
    else
       return (fib(n-1)+fib(n-2));                    // recursive call
  }
```

```
// Write a program to print the Fibonacci series using recursion.
#include<stdio.h>
  int fib(int);    // function prototype
  int main()
  {
     int n,i,sum=0;
     printf("Enter number of term in fibbo series\n");
     scanf("%d",&n);
     for(i=1;i<=n;i++)
     {
       printf("%d\t",fib(i));       // function calling
     }
     return 0;
  }
//function definition
int fib(int n)
  {
    if(n==1)                            //base condition
     return 0;
    else if(n==2)                              //base condition
     return 1;
    else
       return (fib(n-1)+fib(n-2));                    //recursive call
  }
```