Recursion

What is Recursion in C?

Recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem. “A recursive function is a function that calls itself with a smaller input until it reaches a base case, at which point it returns a result”.

Types of Recursion

There are two main types of recursion in C:

1. Tail recursion
2. Non-tail recursion

Tail Recursion

- This is a type of recursion where the recursive call is the last thing the function does before returning a value.
- Tail-recursive functions are generally preferred over non-tail-recursive functions because they can be optimized by the compiler.

Non Tail Recursion

- This is a type of recursion where the recursive call is not the last thing the function does before returning a value.
- Non-tail-recursive functions are generally less efficient than tail-recursive functions because they cannot be optimized by the compiler.

Recursive Functions

A **recursive function** is a function that solves a problem by calling itself on a smaller subproblem. It typically consists of two main parts:

1. **Base Case** – the condition under which the recursion stops.
2. **Recursive Case** – where the function calls itself with modified arguments, gradually approaching the base case.

Applications of Recursion

Recursion is widely used to solve different kinds of problems from simple ones like printing linked lists to being extensively used in AI. Some of the common uses of recursion are:

- Tree-Graph Algorithms
- Mathematical Problems
- Divide and Conquer
- Dynamic Programming
- In Postfix to Infix Conversion
- Searching and Sorting Algorithms

Advantages of Recursion

The advantages of using recursive methods over other methods are:

- Recursion can effectively reduce the length of the code.
- Some problems are easily solved by using recursion like the tower of Hanoi and tree traversals.
- Data structures like linked lists, trees, etc. are recursive by nature so recursive methods are easier to implement for these data structures.

Disadvantages of Recursion

As with almost anything in the world, recursion also comes with certain limitations some of which are:

1. Recursive functions make our program a bit slower due to function call overhead.
2. Recursion functions always take extra space in the function call stack due to separate stack frames.
3. Recursion methods are difficult to understand and implement.

Memory Allocation of Recursive Functions

- Each recursive call creates a new copy of that function in the memory.

- Once some data is returned by the function, the copy is removed from the memory.
- Since all the variables and other stuff declared inside function get stored in the stack, therefore a separate stack is maintained at each recursive call.
- Once the value is returned from the corresponding function, the stack gets destroyed.
- Recursion involves so much complexity in resolving and tracking the values at each recursive call.
- Therefore we need to maintain the stack and track the values of the variables defined in the stack.

Some Important :

Why Recursion?

- Recursion involves calling the same function again, and hence, has a very small length of code.
- It is especially good for working on things that have many possible branches and are too complex for an iterative approach.
- Recursion is best suited in trees and graphs problems.

Base Condition :

In a recursive function, there must be a terminating condition to stop the recursion. That terminating condition is known as the base condition.

Let’s understand recursion by solving the problem of **Factorial using Recursion**.

Algorithm :

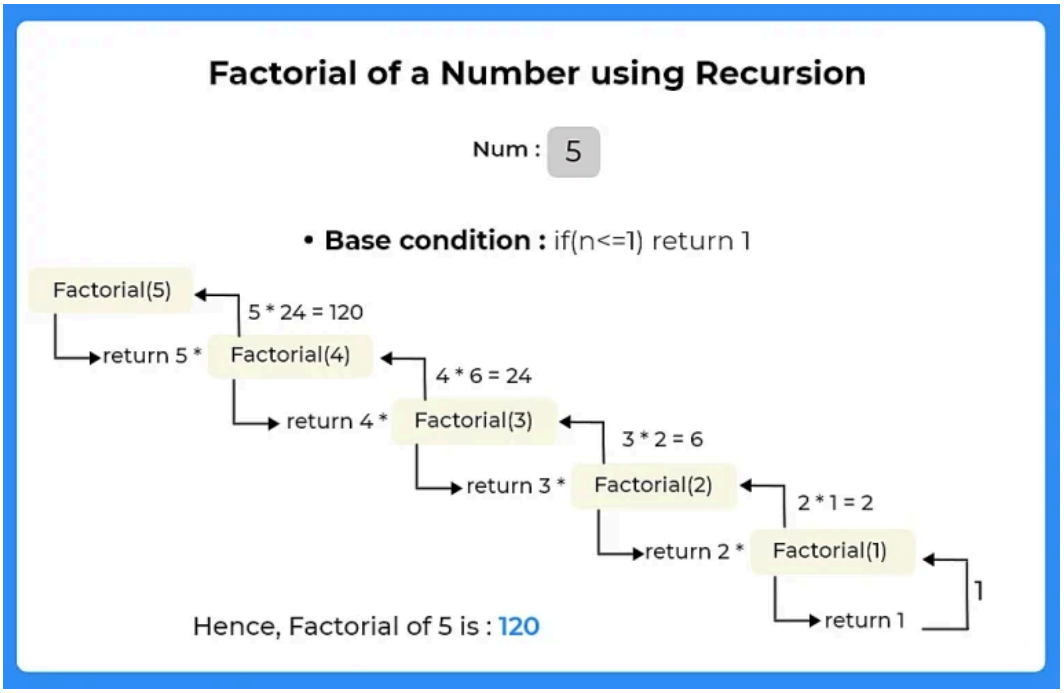
- Create a function fact(int n),
- Inside fact(), define the termination condition : if(n<=1) return 1
- Otherwise return **n*fact(n-1)** //here, we are calling function fact() again by decrement the value of n.

Time-Complexity : **O(n)**

Space-Complexity : **O(1)**

Example of a recursive function in C that calculates the factorial of a given number:

```
#include<stdio.h>
//Function to calculate the factorial
int fact(int n){

  if(n<=1) return 1;   //Base Condition
  else return (n*fact(n-1));   //Recurcive Call of Fact() function

}

//Driver Code
int main(){

  int n=5;
  printf("Factorial of %d is %d", n, fact(n));

}
```


Factorial of a Number using Recursion
Num : 5
• Base condition : if(n<=1) return 1
Factorial(5)
5 * 24 = 120
return 5 * Factorial(4)
4 * 6 = 24
return 4 * Factorial(3)
3 * 2 = 6
return 3 * Factorial(2)
2 * 1 = 2
return 2 * Factorial(1)
1
return 1
Hence, Factorial of 5 is : 120