C++ Course Notes | Recursion | Week 11

Hello everyone welcome to the weekly lecture notes

Topics to be covered:

- Introduction to recursive functions
- Working rules of recursive functions

Topic 1: Introduction to recursive functions

let's say if someone ask you to calculate 5! , you can say give me 4! and I will just multiply is to 5 and return it to you then the other person will say that no, you give me 3! and then I will multiply is by 4 to give you 4! and so on...

In this example everyone is doing the same task . A recursive function works in the same way.

More formally, a recursive function is a function that calls itself again and again until certain conditions are met. It has basically two parts:

1. A precondition that is used to stop this recursive call (Halting Condition).
2. A function that is capable of calling itself (recursive call).

Halting Condition:

Just like how we have a condition in an iterative statement to terminate the loop similarity, it must have a halting condition to terminate the recursive call; otherwise, it will result in an overflow error as it will go inside an infinite recursive call.

Why Do We Need Recursion?

Recursion is generally used when dealing with complex problems and problems that form a hierarchical pattern; it solves the original problem breaking into the smaller subproblems.Recursion is inefficient in cases where the same value is calculated again and again. It requires extra memory on the stack for each recursive call.

Topic 2: Working of Recursive functions

A recursive function has 3 parts -

1. Base statement - It is the statement at which the recurrence will terminate. It is generally a condition for which the solution is known or can be calculated easily. Without it the recurrence will continue forever.
2. Recurrence statement - It is the statement which calls the function again.
3. The rest of the function where we do our computations.

Syntax

In C++ we can write a recursive function using the following syntax:

```
methodName (N parameters )
{
if(haltCondition){
return result
}
return methodName (N parameters )
}
```
