📘 Recursion in C++

🧠 What is Recursion?

Recursion is a programming technique where **a function calls itself** to solve a problem. Each recursive call should bring the problem closer to a **base case** (also known as **exit condition**), which is a condition that stops the recursion.

Key Components:

1. **Base Case** – The condition under which the function stops calling itself.
2. **Recursive Case** – The part where the function calls itself with a modified argument.

🔁 Why Use Recursion?

Recursion is particularly useful for problems that can be broken down into smaller, similar subproblems, such as:

- Factorials
- Fibonacci numbers
- Tree traversal
- Backtracking problems (e.g., Sudoku, N-Queens)
- Divide and conquer algorithms (e.g., Merge Sort, Quick Sort)

🪄 When to Use Recursion

Use recursion when:

- The problem has a natural recursive structure.
- You can define a clear base case.
- The depth of recursion is manageable.

🧮 Example 1: Factorial Function

The factorial of a number (n) is defined as:

`n! = n * (n-1) * (n-2) * ... * 1`

C++ Code:

```cpp
#include <iostream>
using namespace std;

int factorial(int n) {
    if (n <= 1) return 1; // Base case
    return n * factorial(n - 1); // Recursive case
}

int main() {
    int num = 5;
    int result = factorial(num);
    cout << "Factorial of " << num << " is " << result << endl;
}
```

Output:

```
Factorial of 5 is 120
```

🧬 Example 2: Fibonacci Sequence

The Fibonacci sequence is defined as:

`F(n) = F(n-1) + F(n-2), where F(0) = 0, F(1) = 1`

C++ Code:

```cpp
#include <iostream>
using namespace std;

int fibonacci(int n) {
    if (n == 0) return 0; // Base case
    if (n == 1) return 1; // Base case
    return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case
}

int main() {
    int n = 6;
    cout << "Fibonacci number at position " << n << " is " << fibonacci(n) << endl;
    return 0;
}
```

Output:

```
Fibonacci number at position 6 is 8
```

⚠️ Pitfalls of Recursion

- **Stack Overflow**: Too many recursive calls can exhaust the call stack.
- **Performance**: Naive recursion (like in Fibonacci) can be inefficient due to repeated calculations.

Optimization Tip:

Use memoization or dynamic programming to improve performance.

🎯 Quiz!

Here's a short quiz on the topic: quiz

🔭 Markdown Viewer

How to view the markdown files in a browser...

- Markdown Viewer

🧠 Lecture Practices

Here are the lecture Practices...

- Day 4
- Day 5
- Day 6

🔍 Lecture Quizzes

Here are the lecture quizzes...

- Day 4
- Day 5
- Day 6

Weekly Topics

Here are the topics for the week...

- Recursion
- Pseudocode
- Sorting
- Searching
- Maps
- Time Complexity