Tutorial 18 - Recursion & Recursive Functions in C++

Recursion in C++

- **Definition**: Recursion is a process where a function calls itself.
- **Recursive Function**: A function that calls itself to solve smaller instances of a problem.
- **Base Case**: Necessary to prevent infinite recursion.
- **Recursive Case**: The condition under which the function continues calling itself.

Factorial Function Example (Recursive)

- **Problem**: Calculate the factorial of a number.
- **Formula**:
  - `n! = n * (n-1)!`
  - Base case: `factorial(1) = 1`
- **Code**:

```
int factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * factorial(n - 1);
}

```

Main Program:

```
#include<iostream>
using namespace std;
int main() {
    int a;
    cout << "Enter a number: ";
    cin >> a;
    cout << "The factorial of " << a << " is " << factorial(a) << endl;
    return 0;
}

```

Steps:
For `factorial(4)`, the function follows:
`4 * factorial(3)` → `4 * 3 * factorial(2)` → `4 * 3 * 2 * factorial(1)` → `4 * 3 * 2 * 1 = 24`.

Fibonacci Sequence (Recursive)

- **Problem**: Find the Fibonacci number at a specific position.
- **Formula**:
  - `fib(n) = fib(n-1) + fib(n-2)`
  - Base case: `fib(0) = 1` and `fib(1) = 1`
- **Code**:

```
int fib(int n) {
```

```
    if (n < 2) {
        return 1;
    }
    return fib(n - 2) + fib(n - 1);
}

```

Main Program:

```
#include<iostream>
using namespace std;
int main() {
    int a;
    cout << "Enter a number: ";
    cin >> a;
    cout << "The term in Fibonacci sequence at position " << a << " is " << fib(a) << endl;
    return 0;
}

```

Steps:
For `fib(5)`, the function performs:
`fib(5) = fib(3) + fib(4)` → `fib(4) = fib(2) + fib(3)` → and so on.

Key Points to Note

1. **Recursive Functions**:
   - Useful for problems that can be broken down into smaller subproblems (e.g., factorial, Fibonacci).
   - **Base Case** is essential to terminate the recursion.
2. **Factorial Function**:
   - `factorial(4) = 4 * 3 * 2 * 1 = 24`
3. **Fibonacci Function**:
   - Finds the Fibonacci number at a specific position, e.g., `fib(5) = 8`, `fib(6) = 13`.
4. **Performance**:
   - Recursion is not always the most efficient method for all problems. It can lead to performance issues in cases where repetitive calculations are involved (e.g., Fibonacci sequence).

Summary for Notebook

1. **Recursion**: Function calling itself to solve smaller instances of a problem.
2. **Base Case**: Terminating condition in recursion.
3. **Factorial Example**: `factorial(n) = n * factorial(n-1)` with base case `factorial(1) = 1`.
4. **Fibonacci Example**: `fib(n) = fib(n-1) + fib(n-2)` with base cases `fib(0) = 1` and `fib(1) = 1`.
5. **Recursive Functions** are efficient for problems like factorial and Fibonacci but can be inefficient for large inputs due to repeated calculations.