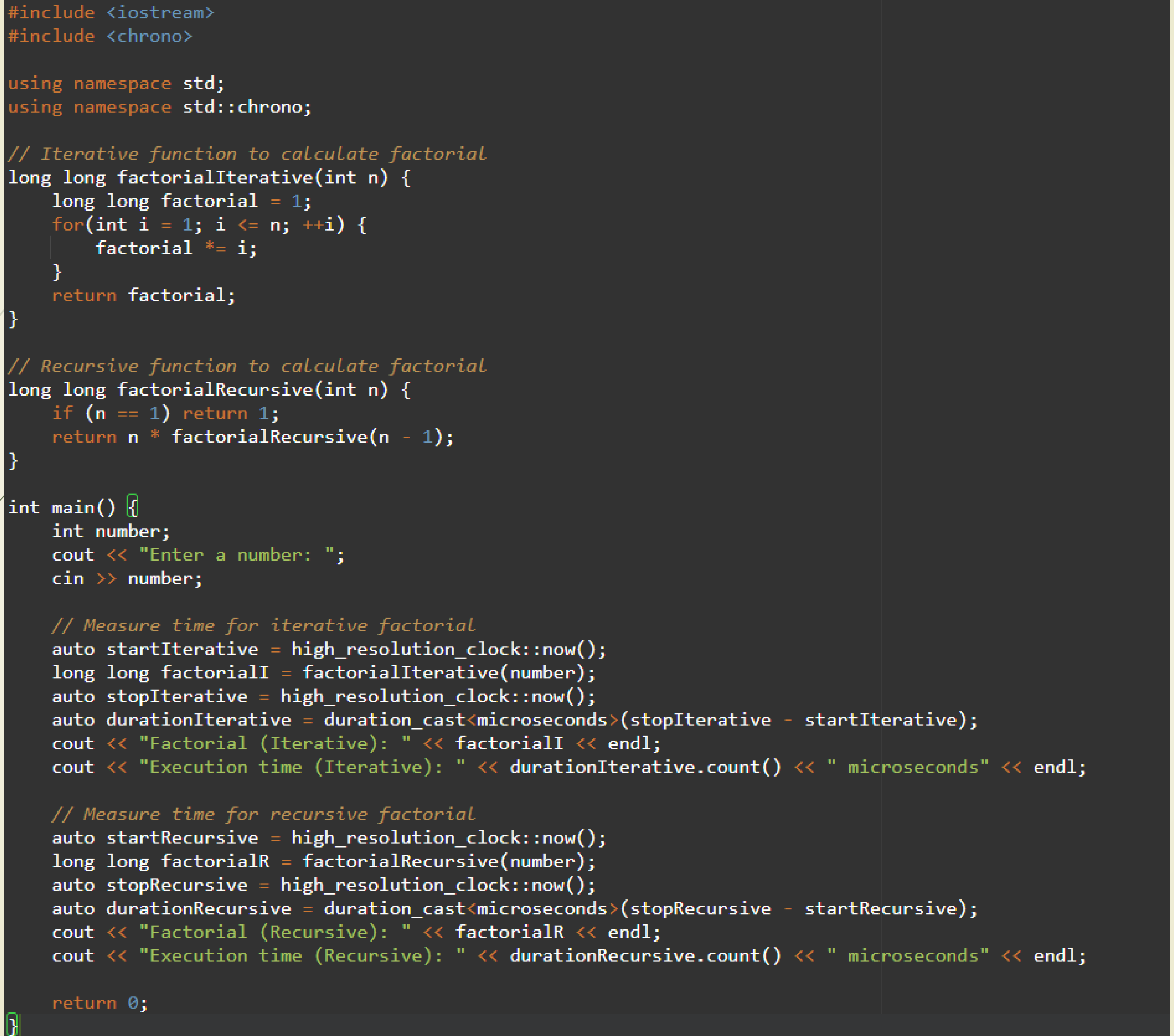
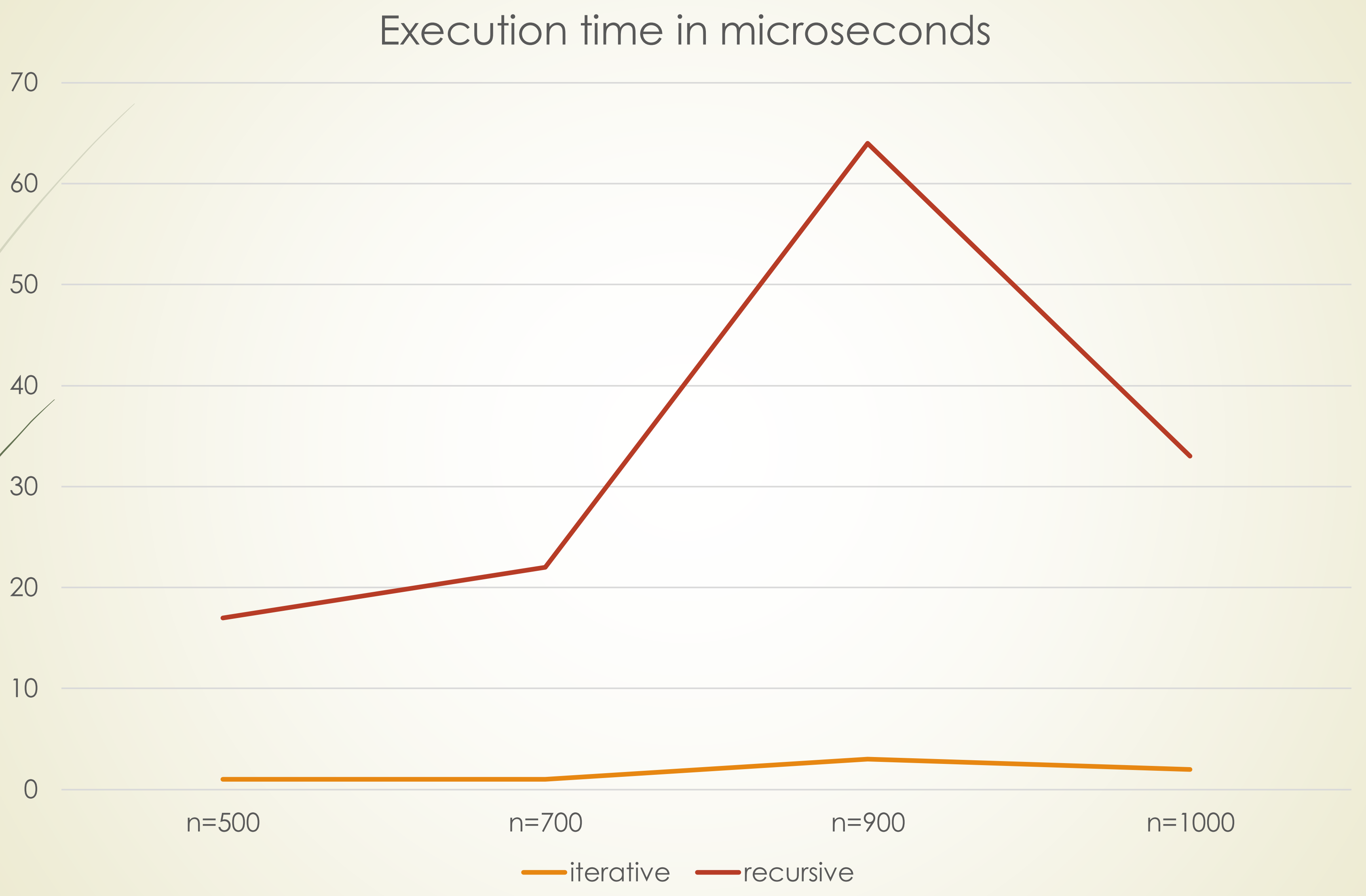
Recursive&Iterative



```cpp
#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

// Iterative function to calculate factorial
long long factorialIterative(int n) {
    long long factorial = 1;
    for(int i = 1; i <= n; ++i) {
        factorial *= i;
    }
    return factorial;
}

// Recursive function to calculate factorial
long long factorialRecursive(int n) {
    if (n == 1) return 1;
    return n * factorialRecursive(n - 1);
}

int main() {
    int number;
    cout << "Enter a number: ";
    cin >> number;

    // Measure time for iterative factorial
    auto startIterative = high_resolution_clock::now();
    long long factorialI = factorialIterative(number);
    auto stopIterative = high_resolution_clock::now();
    auto durationIterative = duration_cast<microseconds>(stopIterative - startIterative);
    cout << "Factorial (Iterative): " << factorialI << endl;
    cout << "Execution time (Iterative): " << durationIterative.count() << " microseconds" << endl;

    // Measure time for recursive factorial
    auto startRecursive = high_resolution_clock::now();
    long long factorialR = factorialRecursive(number);
    auto stopRecursive = high_resolution_clock::now();
    auto durationRecursive = duration_cast<microseconds>(stopRecursive - startRecursive);
    cout << "Factorial (Recursive): " << factorialR << endl;
    cout << "Execution time (Recursive): " << durationRecursive.count() << " microseconds" << endl;

    return 0;
}
```


Execution time in microseconds
70
60
50
40
30
20
10
0
n=500
n=700
n=900
n=1000
iterative
recursive

Two separate functions were implemented:
factorialIterative(int n): Calculates n! using a for-loop.
factorialRecursive(int n): Calculates n! using a recursive algorithm.

Stack Overflow
The recursive implementation is susceptible to stack overflow for large values of *n* due to the increasing number of function calls placed on the call stack.

Execution Time
***** Iterative implementation showed a consistent and gradual increase in execution time with the increase of *n*.
* Recursive implementation exhibited an exponential growth in execution time as *n* increased, which is consistent with the recursive function's overhead of maintaining a call stack.