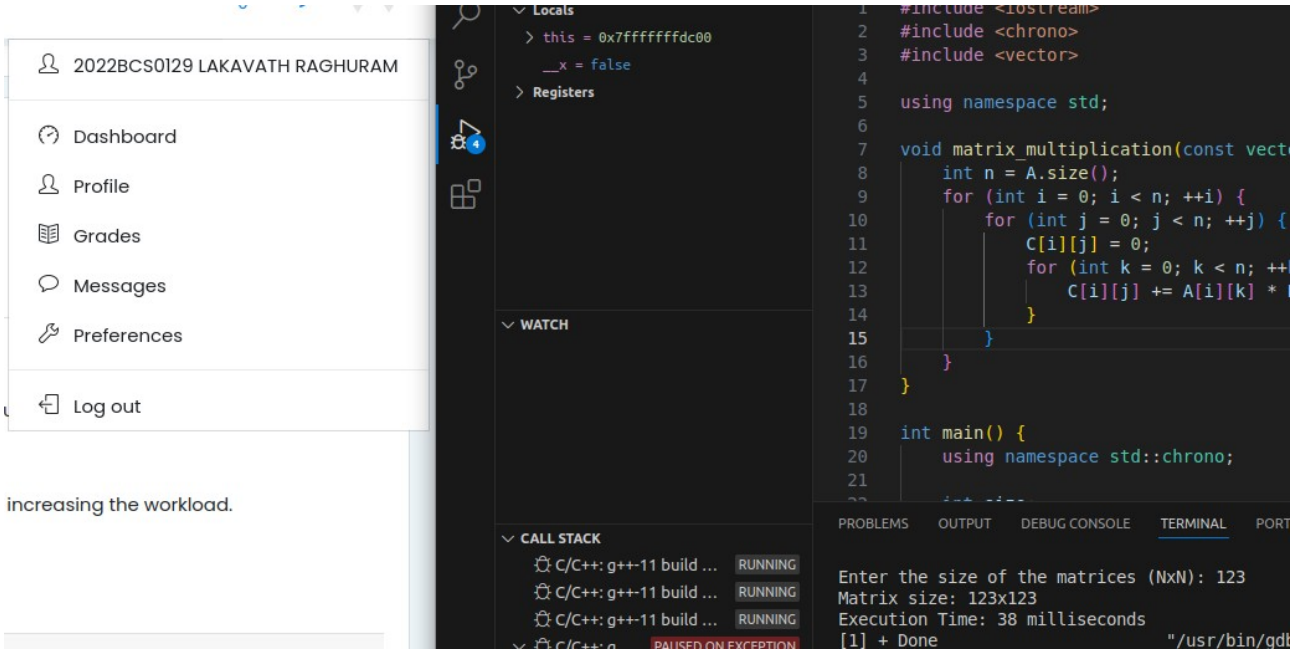
ICS 311 PARALLEL AND DISTRIBUTED COMPUTING

SET6-LAB3-23.8.24

LAKAVATH RAGHURAM 2022BCS0129

For the three programs (Fibonacci, Matrix Multiplication, and Prime numbers), do the following:

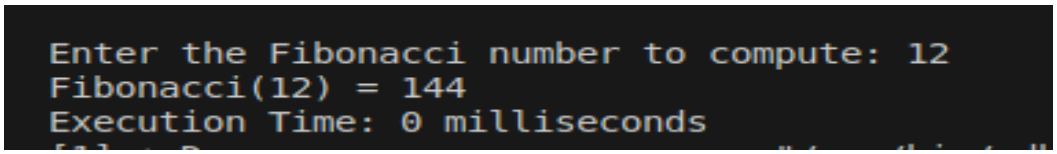


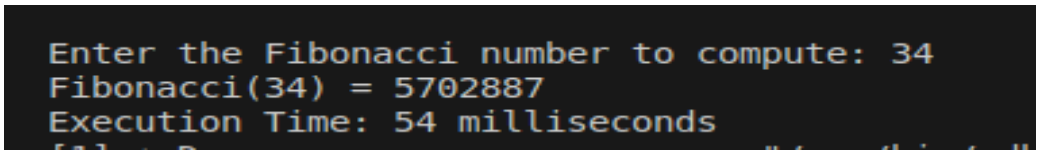
FIBONACCI

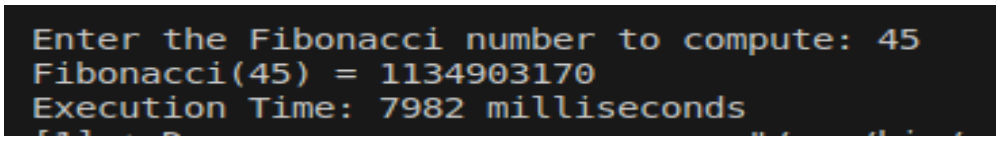
Code : fib.cpp

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

unsigned long long fibonacci(int n) {
    if (n <= 1)
        return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
int main() {
    using namespace std::chrono;
    int n;
    std::cout << "Enter the Fibonacci number to compute: ";
    std::cin >> n;
    auto start = high_resolution_clock::now();
    unsigned long long result = fibonacci(n);
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(end - start).count();
    std::cout << "Fibonacci(" << n << ") = " << result << std::endl;
    std::cout << "Execution Time: " << duration << " milliseconds" << std::endl;

    return 0;
}
```







MATRIX MULTIPLICATION

Code:

```
#include <iostream>
#include <chrono>
#include <vector>

using namespace std;
```

```
void matrix_multiplication(const vector<vector<int>>& A, const vector<vector<int>>& B,
vector<vector<int>>& C) {
    int n = A.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            C[i][j] = 0;
            for (int k = 0; k < n; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

int main() {
    using namespace std::chrono;
    int size;
    std::cout << "Enter the size of the matrices (NxN): ";
    std::cin >> size;
    vector<vector<int>> A(size, vector<int>(size, 1));
    vector<vector<int>> B(size, vector<int>(size, 2));
    vector<vector<int>> C(size, vector<int>(size, 0));

    auto start = high_resolution_clock::now();
    matrix_multiplication(A, B, C);
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(end - start).count();
    std::cout << "Matrix size: " << size << "x" << size << std::endl;
    std::cout << "Execution Time: " << duration << " milliseconds" << std::endl;
    return 0;
}
```

```
Enter the size of the matrices (NxN): 123
Matrix size: 123x123
Execution Time: 38 milliseconds
[1] + Done                       "/usr/bin/gdb
```

```
Enter the size of the matrices (NxN): 1000 1000
Matrix size: 1000x1000
Execution Time: 13804 milliseconds
```

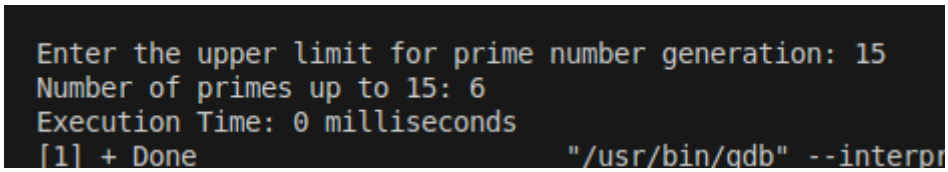
```
Enter the size of the matrices (NxN): 1100
Matrix size: 1100x1100
Execution Time: 17843 milliseconds
```

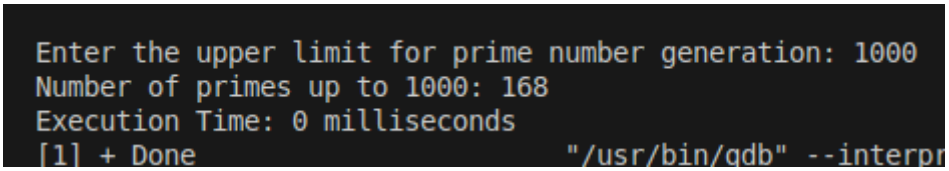
PRIME NUMBERS

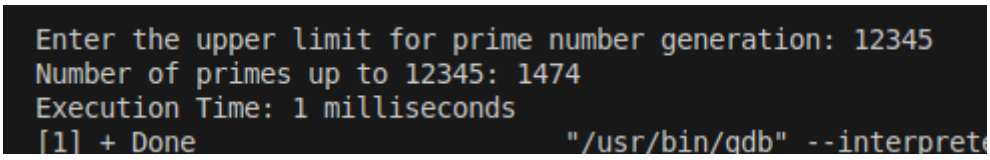
code: prime.cpp

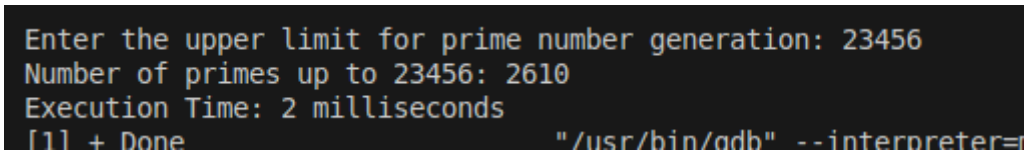
```
#include <iostream>
#include <chrono>
#include <vector>

using namespace std;

vector<int> generate_primes(int limit) {
    vector<int> primes;
    vector<bool> is_prime(limit + 1, true);
    is_prime[0] = is_prime[1] = false;

    for (int i = 2; i <= limit; ++i) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (int j = i * i; j <= limit; j += i) {
                is_prime[j] = false;
            }
        }
    }
    return primes;
}
int main() {
```

```
    using namespace std::chrono;
    int limit;
    std::cout << "Enter the upper limit for prime number generation: ";
    std::cin >> limit;
    auto start = high_resolution_clock::now();
    vector<int> primes = generate_primes(limit);
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(end - start).count();

    std::cout << "Number of primes up to " << limit << ": " << primes.size() << std::endl;
    std::cout << "Execution Time: " << duration << " milliseconds" << std::endl;

    return 0;
}
```









1. Write programs to find the execution time of the main function.

FIBONACCI : INPUT=12 time :0 milliseconds

matrix multiplication input=1000 time:13804milliseocnds

prime numbers: input=12345 :time: 1 milliseconds

2. Create a table for each program illustrating the execution time by increasing the workload.

Fibonacci table:

Fibonacci number	Execution Time (milliseconds)
12	0
34	54
45	7982

Matrix multiplication table:

Matrix Size (NxN)	Execution Time (milliseconds)
123	38
1000	13804
1100	17842

Prime number table

Upper Limit	Execution Time (milliseconds)
1000	0
12345	1
23456	2