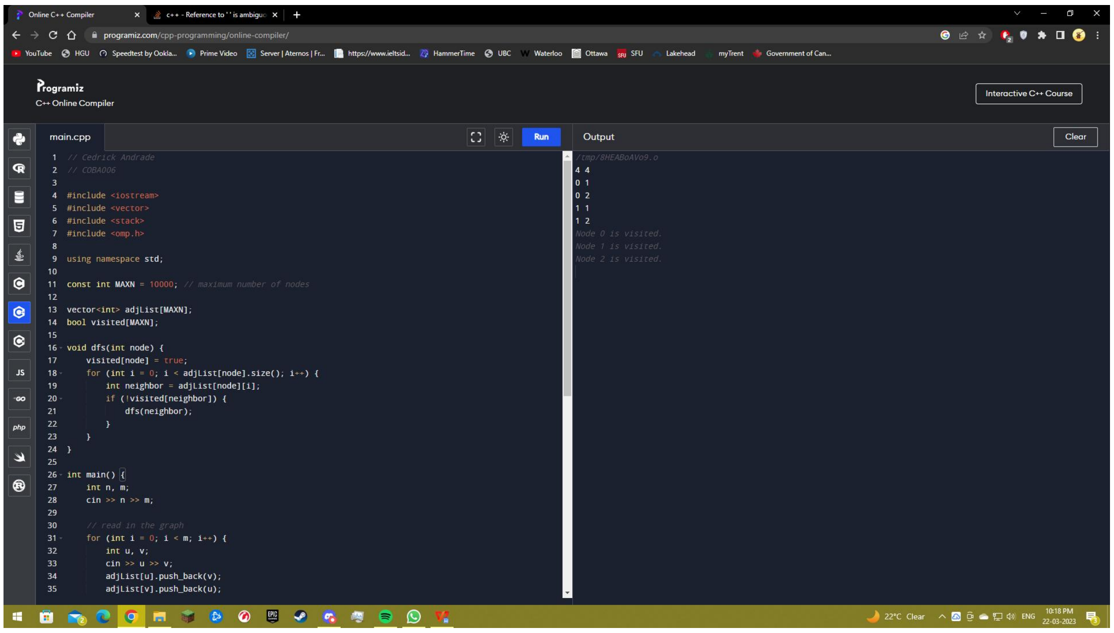
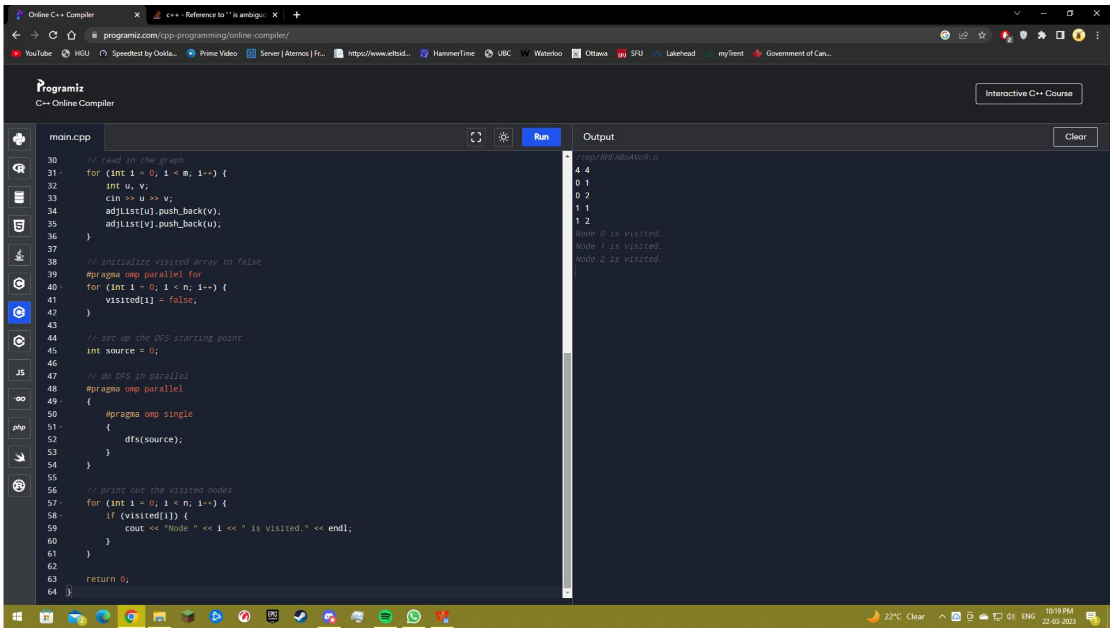
Cedrick Andrade
COBA006

Assignment 1 A



```
// Cedrick Andrade
// COBA006

#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

using namespace std;

const int MAXN = 10000; // maximum number of nodes

vector<int> adjList[MAXN];
bool visited[MAXN];

void dfs(int node) {
    visited[node] = true;
    for (int i = 0; i < adjList[node].size(); i++) {
        int neighbor = adjList[node][i];
        if (!visited[neighbor]) {
            dfs(neighbor);
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    // read in the graph
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adjList[u].push_back(v);
        adjList[v].push_back(u);
```

```
/tmp/8HEABoAVo9.o
4 4
0 1
0 2
1 1
1 2
Node 0 is visited.
Node 1 is visited.
Node 2 is visited.
```

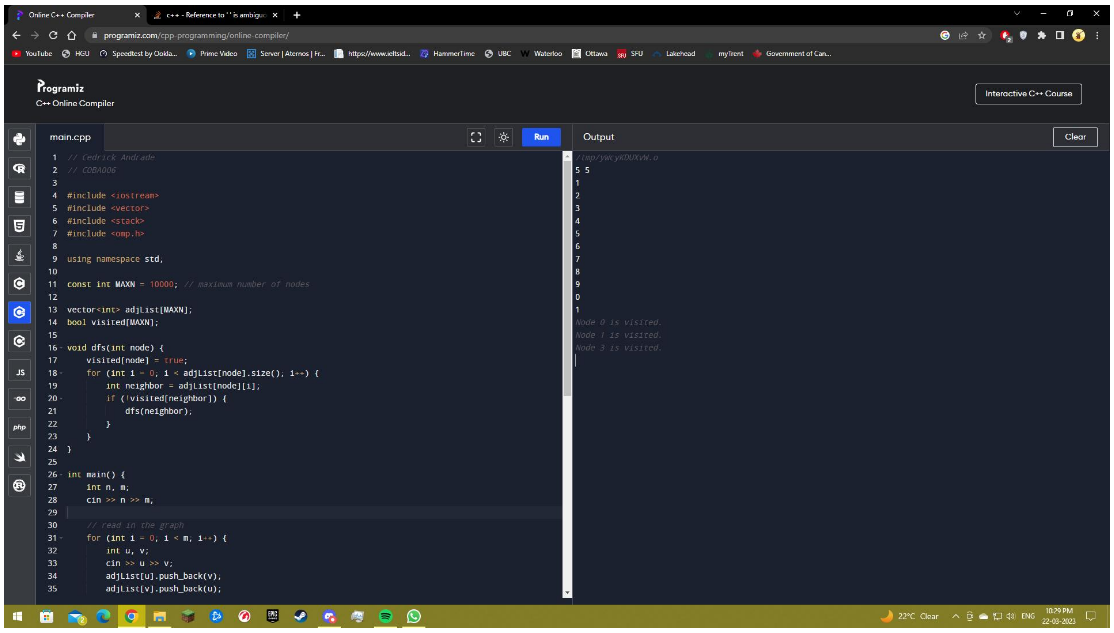


```
    // read in the graph
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adjList[u].push_back(v);
        adjList[v].push_back(u);
    }

    // initialize visited array to false
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }

    // set up the DFS starting point
    int source = 0;

    // do DFS in parallel
    #pragma omp parallel
    {
        #pragma omp single
        {
            dfs(source);
        }
    }

    // print out the visited nodes
    for (int i = 0; i < n; i++) {
        if (visited[i]) {
            cout << "Node " << i << " is visited." << endl;
        }
    }

    return 0;
}
```

```
/tmp/8HEABoAVo9.o
4 4
0 1
0 2
1 1
1 2
Node 0 is visited.
Node 1 is visited.
Node 2 is visited.
```

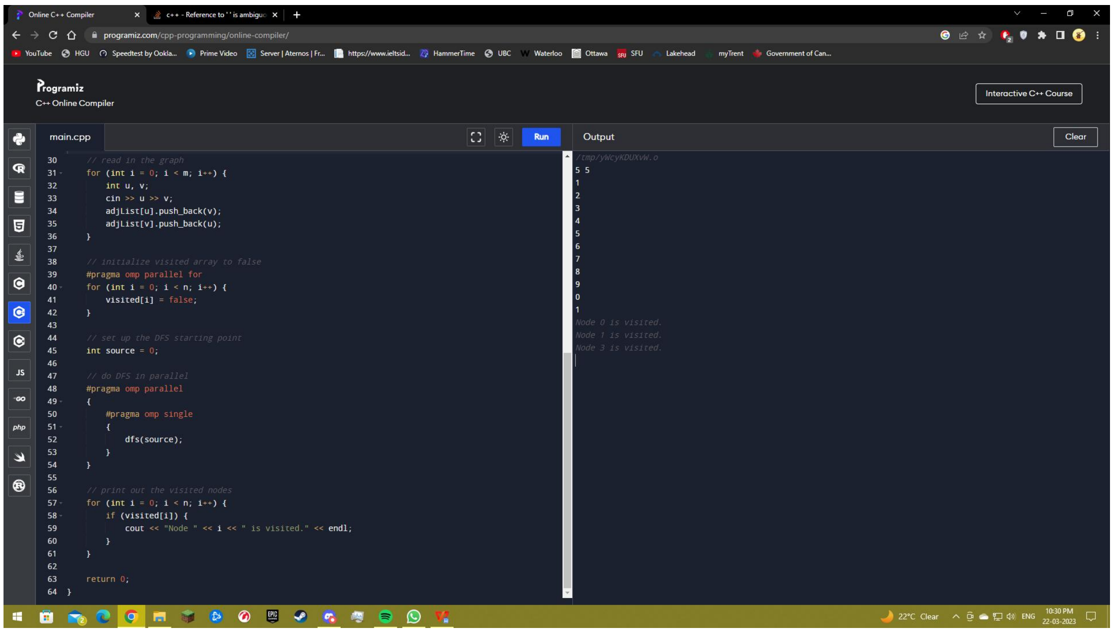
Cedrick Andrade
COBA006

Assignment 1B



```cpp
// Cedrick Andrade
// COBA006

#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

using namespace std;

const int MAXN = 10000; // maximum number of nodes

vector<int> adjList[MAXN];
bool visited[MAXN];

void dfs(int node) {
    visited[node] = true;
    for (int i = 0; i < adjList[node].size(); i++) {
        int neighbor = adjList[node][i];
        if (!visited[neighbor]) {
            dfs(neighbor);
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    // read in the graph
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adjList[u].push_back(v);
        adjList[v].push_back(u);
```

```
/tmp/yWcyKDUXvW.o
5 5
1
2
3
4
5
6
7
8
9
0
1
Node 0 is visited.
Node 1 is visited.
Node 3 is visited.
```



```cpp
    // read in the graph
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adjList[u].push_back(v);
        adjList[v].push_back(u);
    }

    // initialize visited array to false
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }

    // set up the DFS starting point
    int source = 0;

    // do DFS in parallel
    #pragma omp parallel
    {
        #pragma omp single
        {
            dfs(source);
        }
    }

    // print out the visited nodes
    for (int i = 0; i < n; i++) {
        if (visited[i]) {
            cout << "Node " << i << " is visited." << endl;
        }
    }

    return 0;
}
```

```
/tmp/yWcyKDUXvW.o
5 5
1
2
3
4
5
6
7
8
9
0
1
Node 0 is visited.
Node 1 is visited.
Node 3 is visited.
```