**LAB #13**

**Name:** Hassan Farooq Siddiqi

**Roll NO:** 099

**Section: BBSE-3A**

**Code:**

**BFS With Tree**

#include<iostream>

using namespace std;

struct TreeNode

{

int value;

TreeNode\* left , \*right;

TreeNode(int v = -1){

value = v;

left = nullptr;

right = nullptr;

}

};

class DepthFrstSearchTree

{

private:

TreeNode\* root;

TreeNode\* insertNode(TreeNode\* node , int value){

if(node ==nullptr){

return new TreeNode(value);

}

if(value <node->value){

node->left = insertNode(node->left, value);

}else{

node->right = insertNode(node->right, value);

}

return node;

}

void preOrder(TreeNode\* node){

if(node != nullptr){

cout <<node->value<<" ";

preOrder(node->left);

preOrder(node->right);

}

}

public:

DepthFrstSearchTree(){

root = nullptr;

}

~DepthFrstSearchTree(){}

void insert(int value){

root = insertNode(root, value);

}

void display(){

preOrder(root);

cout<<endl;

}

};

int main(){

DepthFrstSearchTree dfs;

int values[] = {7,4,8,9, 3,2 };

for(int value : values){

dfs.insert(value);

}

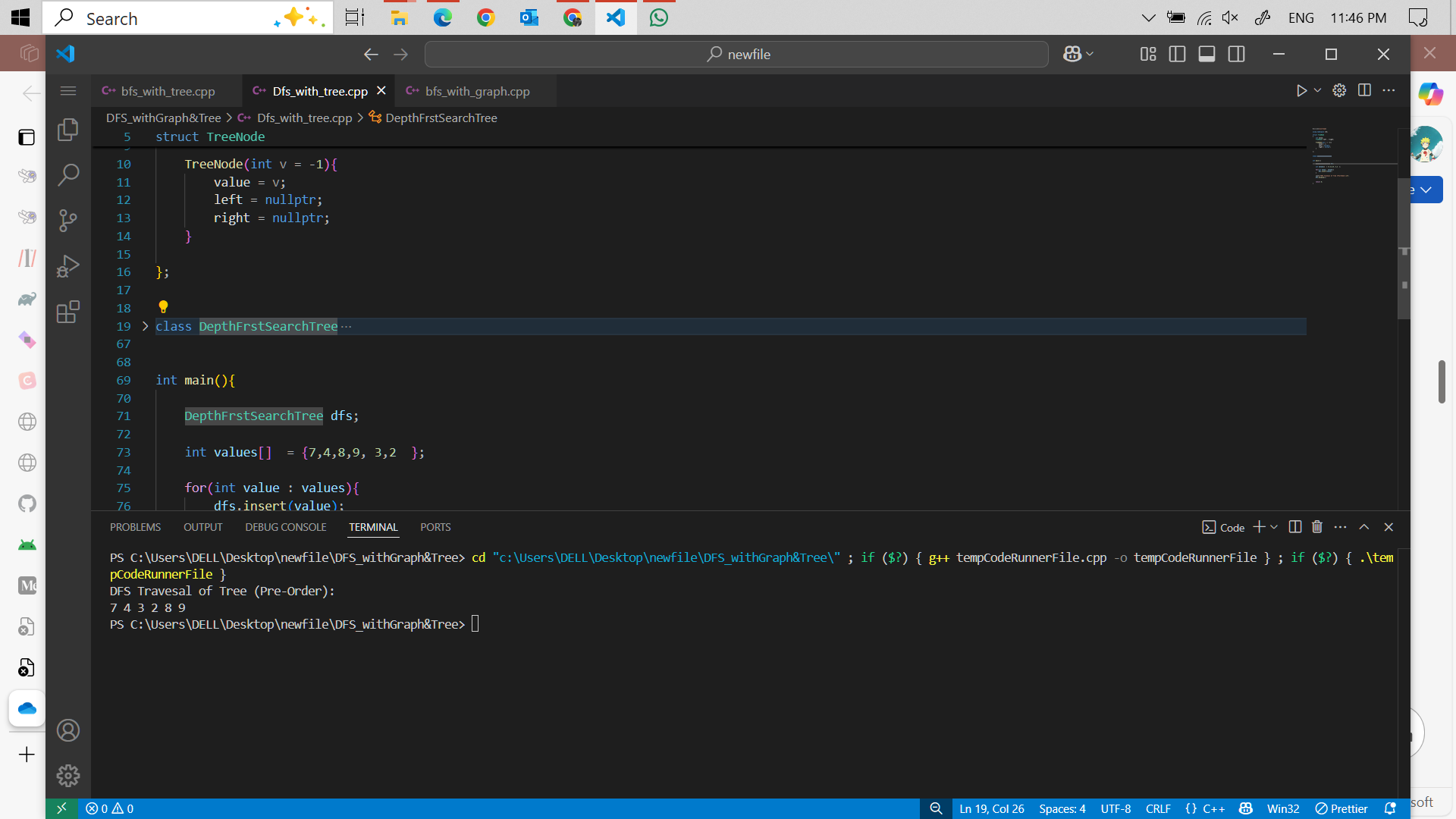
cout<<"DFS Travesal of Tree (Pre-Order):\n";

dfs.display();

return 0;

}

**Output:**



**Code:**

**DFS with Graph**

#include<iostream>

using namespace std;

const int MAX = 100;

struct Node

{

int vertex;

Node\* next;

Node(int vertex){

this->vertex = vertex;

next = nullptr;

}

};

class Graph

{

private:

Node\* adjencyList[MAX];

bool visted[MAX];

int numVertices;

void DFSUtil(int v){

visted[v] = true;

cout<< v <<" ";

Node\* temp = adjencyList[v];

while (temp !=nullptr)

{

int adjVetex = temp->vertex;

if(!visted[adjVetex]){

DFSUtil(adjVetex);

}

temp = temp->next;

}

}

public:

Graph(int vertices = -1){

numVertices = vertices;

for(int i =0 ; i<vertices; i++){

adjencyList[i] = nullptr;

visted[i] = false;

}

}

void insert(int u, int v){

Node\* newNode = new Node(v);

newNode->next = adjencyList[u];

adjencyList[u] = newNode;

}

void ApplyDFS(int start ){

//Reset visted array

for(int i =0 ; i<numVertices; i++){

visted[i] = false;

}

DFSUtil(start);

cout<<endl;

}

~Graph(){}

};

int main(){

Graph g(4);

g.insert(0,1);

g.insert(0, 2);

g.insert(1, 2);

g.insert(2, 0);

g.insert(2, 3);

g.insert(3, 3);

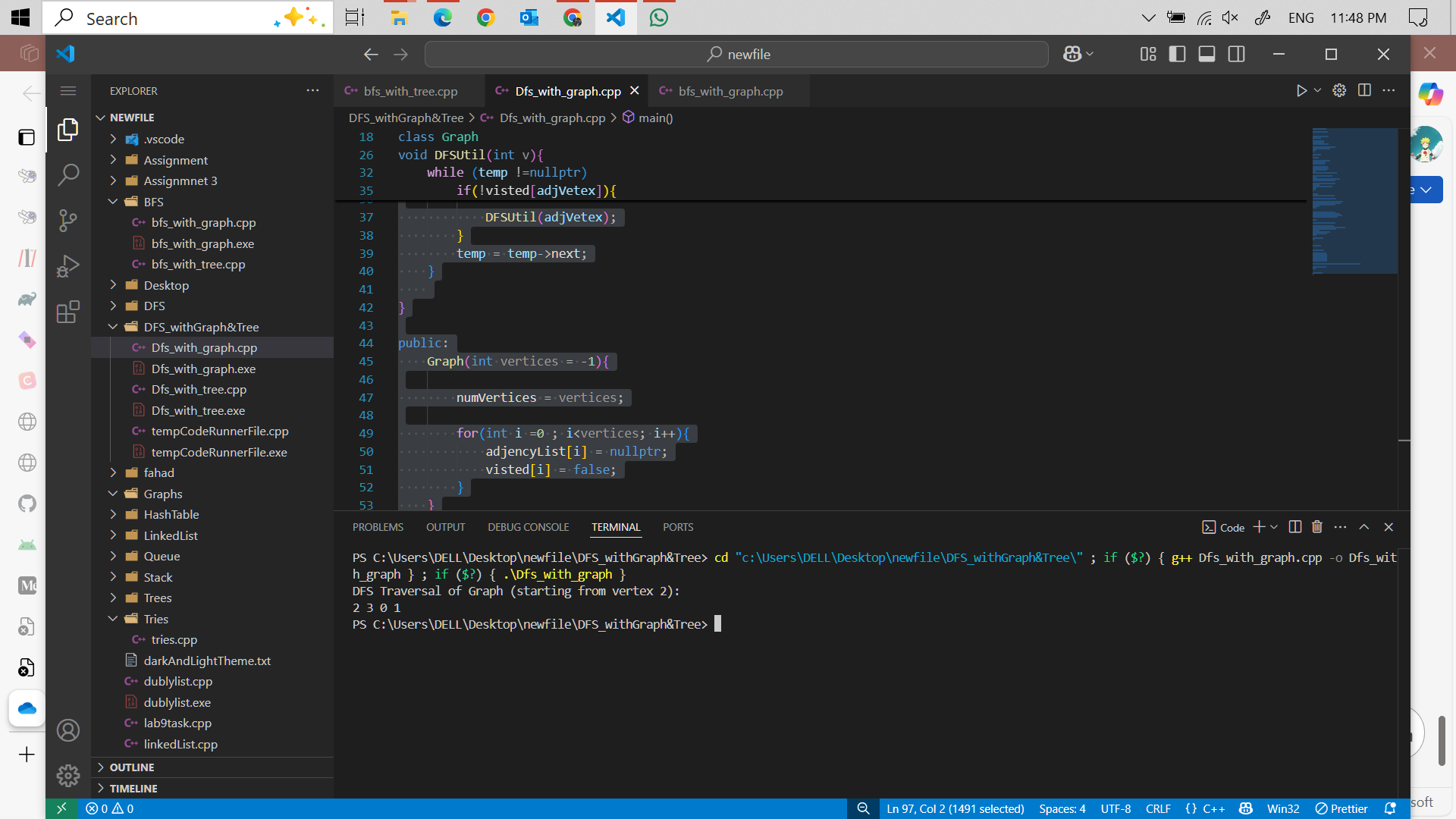
cout<<"DFS Traversal of Graph (starting from vertex 2):\n";

g.ApplyDFS(2);

return 0;

}

**Output:**



**Code:**

**BFS with Tree**

#include <iostream>

using namespace std;

struct TreeNode {

int value;

TreeNode\* left;

TreeNode\* right;

TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}

};

struct QueueNode {

TreeNode\* treeNode;

QueueNode\* next;

QueueNode(TreeNode\* node) : treeNode(node), next(nullptr) {}

};

class Queue {

private:

QueueNode \*front, \*rear;

public:

Queue() : front(nullptr), rear(nullptr) {}

void enqueue(TreeNode\* node) {

QueueNode\* newNode = new QueueNode(node);

if (rear == nullptr) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

TreeNode\* dequeue() {

if (front == nullptr) return nullptr;

QueueNode\* temp = front;

front = front->next;

if (front == nullptr) rear = nullptr;

TreeNode\* node = temp->treeNode;

delete temp;

return node;

}

bool isEmpty() {

return front == nullptr;

}

};

void bfs\_tree(TreeNode\* root) {

if (root == nullptr) return;

Queue q;

q.enqueue(root);

while (!q.isEmpty()) {

TreeNode\* node = q.dequeue();

cout << node->value << " ";

if (node->left != nullptr) {

q.enqueue(node->left);

}

if (node->right != nullptr) {

q.enqueue(node->right);

}

}

}

int main() {

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

cout << "BFS Traversal of Tree:\n";

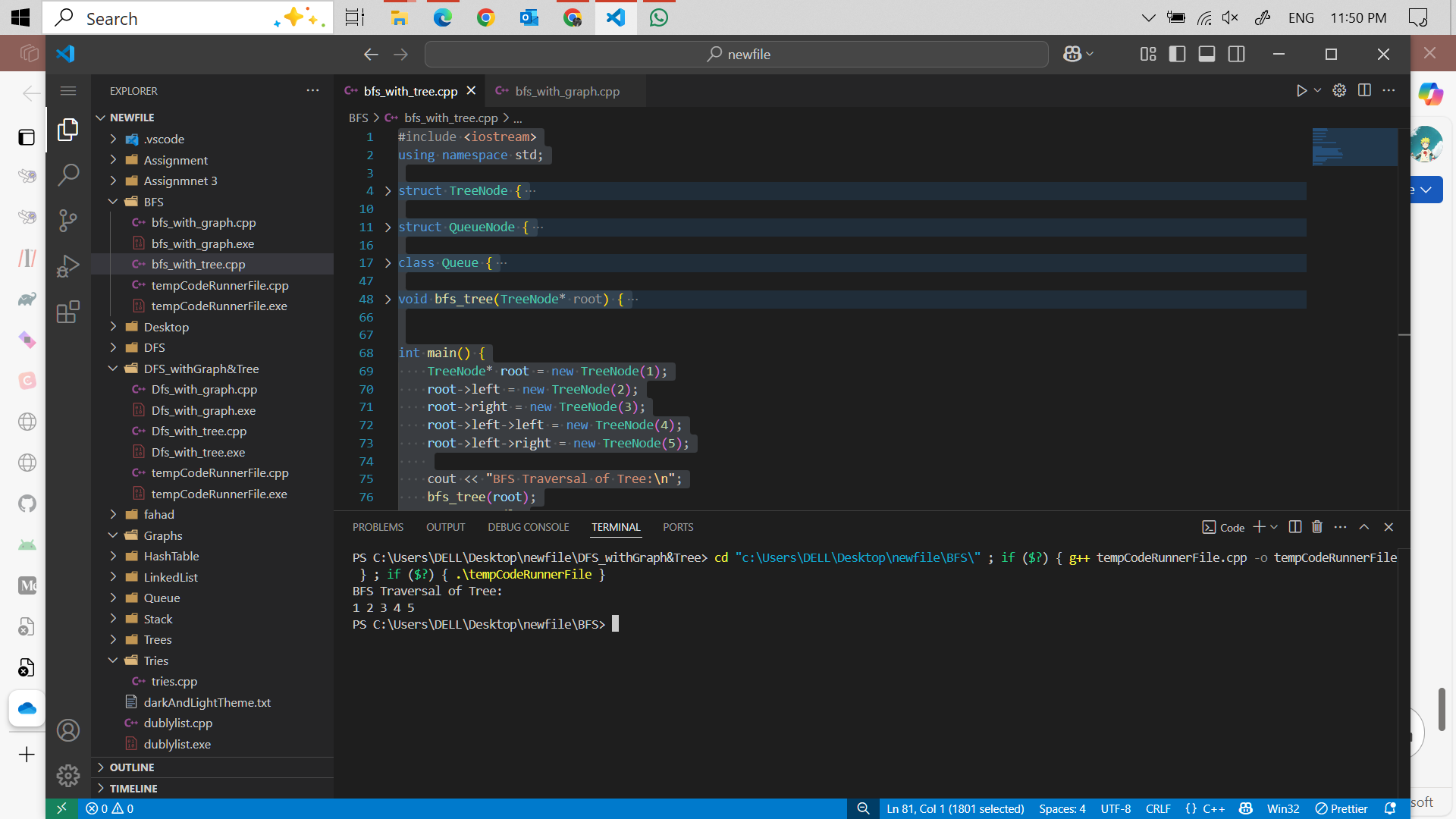
bfs\_tree(root);

cout << endl;

return 0;

}

**Output:**



**Code:**

**BFS with Graph**

#include <iostream>

using namespace std;

const int MAX = 100;

struct Node {

int vertex;

Node\* next;

Node(int v) : vertex(v), next(nullptr) {}

};

struct QueueNode {

int vertex;

QueueNode\* next;

QueueNode(int v) : vertex(v), next(nullptr) {}

};

class Queue {

private:

QueueNode \*front, \*rear;

public:

Queue() : front(nullptr), rear(nullptr) {}

void enqueue(int v) {

QueueNode\* newNode = new QueueNode(v);

if (rear == nullptr) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

int dequeue() {

if (front == nullptr) return -1;

QueueNode\* temp = front;

front = front->next;

if (front == nullptr) rear = nullptr;

int v = temp->vertex;

delete temp;

return v;

}

bool isEmpty() {

return front == nullptr;

}

};

class GraphBFS {

private:

Node\* adjList[MAX];

bool visited[MAX];

int numVertices;

public:

GraphBFS(int vertices) : numVertices(vertices) {

for (int i = 0; i < numVertices; i++) {

adjList[i] = nullptr;

visited[i] = false;

}

}

void insert\_graph(int u, int v) {

Node\* newNode = new Node(v);

newNode->next = adjList[u];

adjList[u] = newNode;

}

void bfs\_graph(int start) {

// Reset visited array

for (int i = 0; i < numVertices; i++) {

visited[i] = false;

}

Queue q;

visited[start] = true;

q.enqueue(start);

while (!q.isEmpty()) {

int current = q.dequeue();

cout << current << " ";

Node\* temp = adjList[current];

while (temp != nullptr) {

int adjVertex = temp->vertex;

if (!visited[adjVertex]) {

visited[adjVertex] = true;

q.enqueue(adjVertex);

}

temp = temp->next;

}

}

}

};

int main() {

GraphBFS g(4);

g.insert\_graph(0, 1);

g.insert\_graph(0, 2);

g.insert\_graph(1, 2);

g.insert\_graph(2, 0);

g.insert\_graph(2, 3);

g.insert\_graph(3, 3);

cout << "BFS Traversal of Graph (starting from vertex 2):\n";

g.bfs\_graph(2);

cout << endl;

return 0;

}

**Output :**

