

```

#include <iostream>
using namespace std;
// Node Structure
struct Node
{
    int data;
    Node *next;
};
// Graph class
class Graph
{
    int numVertices;
    Node **adjList;

public:
    // Constructor
    Graph(int vertices)
    {
        numVertices = vertices;
        adjList = new Node *[vertices];

        for (int i = 0; i < vertices; i++)
        {
            adjList[i] = NULL;
        }
    }
    // Function to add new Edge to Adjency list
    void addEdge(int src, int dest)
    {
        Node *newNode = new Node;
        newNode->data = dest;
        newNode->next = adjList[src];
        adjList[src] = newNode;

        newNode = new Node;
        newNode->data = src;
        newNode->next = adjList[dest];
        adjList[dest] = newNode;
    }

    // BFS Traversal Function
    void BFS(int startVertex)
    {

```

```

bool *visited = new bool[numVertices];
for (int i = 0; i < numVertices; i++)
{
    visited[i] = false;
}

int *queue = new int[numVertices];
int front = 0;
int rear = 0;

visited[startVertex] = true;
queue[rear++] = startVertex;

cout << "BFS Traversal: ";

while (front < rear)
{
    int currentVertex = queue[front++];
    cout << currentVertex << " ";

    Node *temp = adjList[currentVertex];
    while (temp != NULL)
    {
        int adjNode = temp->data;

        if (!visited[adjNode])
        {
            visited[adjNode] = true;
            queue[rear++] = adjNode;
        }
        temp = temp->next;
    }
}
cout << endl;
delete[] visited;
delete[] queue;
}

};

// Main Function
int main()
{
    int vertices = 5;
    Graph g(vertices);

```

```
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 4);

    g.BFS(0);

    return 0;
}
```