```
// BFS
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cstddef>
#include <omp.h>
#include <bits/stdc++.h>
using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph {
        int V; // No. of vertices

        // Pointer to an array containing adjacency
        // lists
        vector<list<int> > adj;

public:
        Graph(int V); // Constructor

        // function to add an edge to graph
        void addEdge(int v, int w);

        // prints BFS traversal from a given source s
        void BFS(int s);
};

Graph::Graph(int V)
{
        this->V = V;
        adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
        adj[v].push_back(w); // Add w to v’s list.
}

void Graph::BFS(int s)
{
        // Mark all the vertices as not visited
        vector<bool> visited;
        visited.resize(V, false);

        // Create a queue for BFS
        list<int> queue;

        // Mark the current node as visited and enqueue it
        visited[s] = true;
        queue.push_back(s);

        while (!queue.empty()) {
                // Dequeue a vertex from queue and print it
                s = queue.front();
                cout << s << " ";
                queue.pop_front();

                // Get all adjacent vertices of the dequeued
                // vertex s. If a adjacent has not been visited,
                // then mark it visited and enqueue it
                for (auto adjacent : adj[s]) {
                        if (!visited[adjacent]) {
                                visited[adjacent] = true;
                                queue.push_back(adjacent);
                        }
                }
        }
}

// Driver program to test methods of graph class
int main()
{
        // Create a graph given in the above diagram
        Graph g(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        cout << "Following is Breadth First Traversal "
                << "(starting from vertex 2) \n";
        g.BFS(2);

        return 0;
}
```