```
// C++ program to print DFS traversal from
// a given vertex in a given graph
#include <bits/stdc++.h>
using namespace std;

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

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

    // A recursive function used by DFS
    void DFSUtil(int v, bool visited[]);

public:
    Graph(int V); // Constructor

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

    // DFS traversal of the vertices
    // reachable from v
    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

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

void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and
    // print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent
    // to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}

// DFS traversal of the vertices reachable from v.
```

```
// It uses recursive DFSUtil()
void Graph::DFS(int v)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function
    // to print DFS traversal
    DFSUtil(v, visited);
}

// Driver code
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 Depth First Traversal"
            " (starting from vertex 2) \n";
    g.DFS(2);

    return 0;
}
```