// C++ Implementation of Kosaraju's algorithm to print all SCCs

#include <iostream>

#include <list>

#include <stack>

using namespace std;

class Graph

{

    int V;    // No. of vertices

    list<int> \*adj;    // An array of adjacency lists

    // Fills Stack with vertices (in increasing order of finishing

    // times). The top element of stack has the maximum finishing

    // time

    void fillOrder(int v, bool visited[], stack<int> &Stack);

    // A recursive function to print DFS starting from v

    void DFSUtil(int v, bool visited[]);

public:

    Graph(int V);

    void addEdge(int v, int w);

    // The main function that finds and prints strongly connected

    // components

    void printSCCs();

    // Function that returns reverse (or transpose) of this graph

    Graph getTranspose();

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list<int>[V];

}

// A recursive function to print DFS starting from v

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);

}

Graph Graph::getTranspose()

{

    Graph g(V);

    for (int v = 0; v < V; v++)

    {

        // Recur for all the vertices adjacent to this vertex

        list<int>::iterator i;

        for(i = adj[v].begin(); i != adj[v].end(); ++i)

        {

            g.adj[\*i].push\_back(v);

        }

    }

    return g;

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::fillOrder(int v, bool visited[], stack<int> &Stack)

{

    // Mark the current node as visited and print it

    visited[v] = true;

    // 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])

            fillOrder(\*i, visited, Stack);

    // All vertices reachable from v are processed by now, push v

    Stack.push(v);

}

// The main function that finds and prints all strongly connected

// components

void Graph::printSCCs()

{

    stack<int> Stack;

    // Mark all the vertices as not visited (For first DFS)

    bool \*visited = new bool[V];

    for(int i = 0; i < V; i++)

        visited[i] = false;

    // Fill vertices in stack according to their finishing times

    for(int i = 0; i < V; i++)

        if(visited[i] == false)

            fillOrder(i, visited, Stack);

    // Create a reversed graph

    Graph gr = getTranspose();

    // Mark all the vertices as not visited (For second DFS)

    for(int i = 0; i < V; i++)

        visited[i] = false;

    // Now process all vertices in order defined by Stack

    while (Stack.empty() == false)

    {

        // Pop a vertex from stack

        int v = Stack.top();

        Stack.pop();

        // Print Strongly connected component of the popped vertex

        if (visited[v] == false)

        {

            gr.DFSUtil(v, visited);

            cout << endl;

        }

    }

}

// Driver program to test above functions

int main()

{

    // Create a graph given in the above diagram

    Graph g(5);

    g.addEdge(1, 0);

    g.addEdge(0, 2);

    g.addEdge(2, 1);

    g.addEdge(0, 3);

    g.addEdge(3, 4);

    cout << "Following are strongly connected components in "

            "given graph \n";

    g.printSCCs();

    return 0;

}