#include <iostream>

#include <list>

using namespace std;

class Graph{

int NoVertex;

list<int> \*adj;

bool isCyclicUtil(int v, bool visited[], int parent);

public:

Graph(int);

void addEdge(int Src, int dest);

void bfs(int src);

bool isCyclic();

};

Graph::Graph(int num){

NoVertex = num;

adj = new list<int>[num];

}

void Graph::addEdge(int src, int dest){

adj[src].push\_back(dest);

adj[dest].push\_back(src);

}

void Graph::bfs(int src){

list<int> queue;

list<int>::iterator qItr;

int currVertex;

bool visited[NoVertex] = {false};

visited[src] = true;

cout<<src<<" ";

queue.push\_back(src);

while(!queue.empty()){

currVertex = queue.front();

queue.pop\_front();

for(qItr = adj[currVertex].begin(); qItr != adj[currVertex].end(); qItr++){

if(visited[\*qItr] == false){

visited[\*qItr] = true;

cout<<\*qItr<<" ";

queue.push\_back(\*qItr);

}

}

}

}

// A recursive function that uses visited[] and parent to detect

// cycle in subgraph reachable from vertex v.

bool Graph::isCyclicUtil(int v, bool visited[], int parent){

// Mark the current node as visited

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 an adjacent is not visited, then recur for that adjacent

if (!visited[\*i])

{

if (isCyclicUtil(\*i, visited, v))

return true;

}

// If an adjacent is visited and not parent of current vertex,

// then there is a cycle.

else if (\*i != parent)

return true;

}

return false;

}

bool Graph::isCyclic(){

bool visited[NoVertex] = {false};

for(int u = 0 ; u < NoVertex ; u++){

if(!visited[u]){

if(isCyclicUtil(u, visited, -1))

return true;

}

}

return false;

}

int main(){

Graph g1(8);

g1.addEdge(0,1);

g1.addEdge(1,2);

g1.addEdge(1,3);

g1.addEdge(2,4);

g1.addEdge(2,5);

g1.addEdge(3,5);

g1.addEdge(4,5);

g1.addEdge(4,6);

g1.addEdge(5,6);

g1.addEdge(6,7);

g1.bfs(4);

cout<<endl;

g1.isCyclic()?cout<<"Cyclic graph":cout<<"Non-cyclic Graph";

return 0;

}