//Shreya Gowda

//CISP 430

//Assignment #7

#include <iostream>

using namespace std;

struct Node{

int tag;

Node\* next;

};

//directed adjacency list graph implementation

class ListGraph{

private:

int vertices;

int edges;

Node\*\* adj\_list;

bool\* checked;

public:

ListGraph(const int numV){

vertices=numV;

edges=0;

bool\* checked= new bool[numV];

for(int i=0; i<numV; i++){

checked[i]=false;

}

adj\_list=new Node\*[numV];

for(int i=0; i<numV; i++){

adj\_list[i]=nullptr;

}

}

void setEdge(int begin, int end){

Node\* head=adj\_list[begin-1];

Node\* newNode = new Node;

newNode->tag=end;

newNode->next=head;

adj\_list[begin-1] = newNode;

edges++;

}

void display()

{

Node\* n;

for(int i=0; i<vertices; i++){

n=adj\_list[i];

cout<<i+1<<" : ";

while (n != nullptr) {

cout << n->tag<<", ";

n = n->next;

}

cout << endl;

}

}

void numPaths(int start, int finish){

int num=0;

for(int i=0; i<vertices; i++){

checked[i]=false;

}

findPath(start, finish, &num);

cout<<"The number of paths from vertex "<<start<<" to vertex "<<finish<<" is "<<num<<".";

}

void findPath(int start, int finish, int\* num){

checked[start-1]=true;

if(start==finish){

\*num+=1;

}

else{

Node\* node=adj\_list[start-1];

while(node!=nullptr){

if(!checked[node->tag-1]){

findPath(node->tag, finish, num);

}

node=node->next;

}

}

checked[start-1] = false;

}

~ListGraph() {

Node\* node;

Node\* next;

   for(int i=0; i<vertices; i++){

node=adj\_list[i];

while (node != nullptr) {

next=node->next;

delete node;

node=next;

}

}

   delete[] adj\_list;

}

};

int main() {

const int numVertices=6;

ListGraph g1(numVertices);

g1.setEdge(1, 2);

g1.setEdge(2, 4);

g1.setEdge(3, 1);

g1.setEdge(3, 2);

g1.setEdge(3, 5);

g1.setEdge(4, 6);

g1.setEdge(5, 2);

g1.setEdge(5, 4);

g1.setEdge(5, 6);

g1.numPaths(3, 6);

}

