#include<iostream>

SOURCE CODE

using namespace std;

int gcd\_iterative(int *a*,int *b*){

    do{

        if(*a*>*b*){

            if(*a*%*b*==0){

                return *b*;

            }

            else{

*a*%=*b*;

            }

        }

        else if(*a*<*b*){

            if(*b*%*a*==0){

                return *a*;

            }

            else{

*b*%=*a*;

            }

        }

    }while(*a*!=*b*);

    return *a*;

}

int gcd\_recurr(int *a*,int *b*){

    if(*a*%*b*==0){

        return *b*;

    }

    else if(*b*%*a*==0){

        return *a*;

    }

    if(*a*>*b*){

        return gcd\_recurr(*a*%*b*,*b*);

    }

    if(*a*<*b*){

        return gcd\_recurr(*a*,*b*%*a*);

    }

    if(*a*==*b*){

        return *a*;

    }

}

int main(){

int a,b;

cout<<"\*\*\*\*\*\*\*This program is to find GCD\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Enter the two numbers: "<<endl;

cin>>a>>b;

cout<<"GCD(iterative) of "<<a<<" and "<<b<<" is: "<<gcd\_iterative(a,b)<<endl;

cout<<"GCD(Recurrsive) of "<<a<<" and "<<b<<" is: "<<gcd\_recurr(a,b)<<endl;

return 0;

}

OUTPUT

// incomplete for Deletion on head

SOURCE CODE

#include<iostream>

#include<queue>

using namespace std;

class *edge*{

public:

    char dest;

*edge*\* link;

    edge(char *ch*){

        dest=*ch*;

        link=NULL;

    }

};

class *node*{

public:

    char info;

*node*\* next;

*edge*\* adj;

    node(char *val*){

        info=*val*;

        next=NULL;

        adj=NULL;

    }

};

//insert node At last

void insertNode(*node*\* &*start*,char *ele*){

*node*\* n=**new** *node*(*ele*);

        if(*start*==NULL){

*start*=n;

            return;

        }

*node*\*temp=*start*;

        while(temp->next!=NULL){

            temp=temp->next;

        }

        temp->next=n;

        return;

}

// Find Node

*node*\* findNode(*node*\* *start*,char *ch*){

*node*\* temp=*start*;

        if(*start*==NULL){

            return NULL;

        }

        while(temp!=NULL){

            if(temp->info==*ch*){

                return temp;

            }

            temp=temp->next;

        }

        return NULL;

}

//Create Egde between two nodes

void createEdge(*node*\* &*start*,char *ch1*,char *ch2*){

*node*\* temp=*start*;

*node*\* n1=findNode(temp,*ch1*);

*node*\* n2=findNode(temp,*ch2*);

    if(n1!=NULL and n2!=NULL){

*edge*\* ed=**new** *edge*(*ch2*);

        if(n1->adj==NULL){

            n1->adj=ed;

            return;

        }

*edge*\* temp\_edge=n1->adj;

        if(temp\_edge->link==NULL){

            temp\_edge->link=ed;

            return;

        }

        while(temp\_edge->link!=NULL){

            temp\_edge=temp\_edge->link;

        }

        temp\_edge->link=ed;

        return;

    }

    else{

        cout<<"Edge is not possible"<<endl;

        return;

    }

}

//Delete edge

void DeleteEdge(*node*\* *start*,char *ch1*,char *ch2*){

*node*\* n1=findNode(*start*,*ch1*);

    if(n1==NULL){

        return;

    }

    if(n1->adj==NULL){

        return;

    }

*edge*\* temp\_E=n1->adj;

    if(temp\_E->dest == *ch2*){

        // edge\* todele=temp\_E;

        n1->adj=temp\_E->link;

        delete temp\_E;

        return;

    }

    while(temp\_E->link!=NULL and temp\_E->link->dest!=*ch2*){

        temp\_E=temp\_E->link;

        // cout<<"\*";

    }

*edge*\* todelete=temp\_E->link;

    if(temp\_E->link!=NULL){

    temp\_E->link=temp\_E->link->link;

    }

    delete todelete;

}

*node*\* DeleteNode(*node*\* *start*,char *val*){

    if(*start*==NULL){

        return NULL;

    }

    // todelete node from adj list of vertices

*node*\* tp=*start*;

    while(tp!=NULL){

        DeleteEdge(*start*,tp->info,*val*);

        tp=tp->next;

    }

    // todelete adj list of Delnode

*node*\* tp2=findNode(*start*,*val*);

*edge*\* del=tp2->adj;

    while(del!=NULL){

        char current\_edge\_info = del->dest;

        del=del->link;

        DeleteEdge(*start*,*val*,current\_edge\_info);

    }

    if(*start*->info == *val*){

*node*\* new\_start = *start*->next;

        delete *start*;

        return new\_start;

    }

*node*\* temp=*start*;

    while(temp->next != NULL){

        if(temp->next->info==*val*){

            break;

        }

        temp=temp->next;

    }

*node*\* todelete=temp->next;

    if(todelete == NULL){

        cout<<"Element doesn't exist in graph";

        return *start*;

    }

    temp->next=temp->next->next;

    delete todelete;

return *start*;

}

//print node

void printNode(*node*\* *temp*){

    if(*temp*==NULL){

        cout<<"EMPTY LIST"<<endl;

        return;

    }

    while(*temp*!=NULL){

        cout<<*temp*->info<<"->";

*temp*=*temp*->next;

    }

    cout<<"NULL"<<endl;

}

// print adjacency list

void printAdj(*node*\* *start*,char *ch*){

*node*\* temp=findNode(*start*,*ch*);

    if(temp==NULL){

        cout<<"EMPTY LIST"<<endl;

        return;

    }

    cout<<temp->info<<"-> ";

*edge*\* temp\_E=temp->adj;

    while(temp\_E!=NULL){

        cout<<temp\_E->dest<<",";

        temp\_E=temp\_E->link;

    }

    cout<<"NULL"<<endl;

}

// calculate indegree

void Cal\_degree(*node*\* *start*,char *ch*){

*node*\* temp=findNode(*start*,*ch*);

    if(temp==NULL){

        cout<<"EMPTY LIST"<<endl;

        return;

    }

    int count=0;

    cout<<"Degree of "<<temp->info<<"-> ";

*edge*\* temp\_E=temp->adj;

    while(temp\_E!=NULL){

        count++;

        temp\_E=temp\_E->link;

    }

    cout<<" "<<count<<endl;

}

int cal\_indegree(*node*\* *start*,char *ch*){   // not good approach

*node*\* temp1=findNode(*start*,*ch*);

    if(temp1==NULL){

        cout<<"NODE IS NOT PRESENT IN LIST"<<endl;

        return -1;

    }

*node*\* temp=*start*;

    int count=0;

    while(temp!=NULL){

*edge*\* temp\_E=temp->adj;

        while(temp\_E!=NULL){

            if(temp\_E->dest==*ch*){

                count++;

            }

            temp\_E=temp\_E->link;

        }

        temp=temp->next;

    }

    return count;

}

bool checkEdge(*node*\* *start*,char *ch1*,char *ch2*){

*node*\* temp=findNode(*start*,*ch1*);

*edge*\* temp\_E=temp->adj;

    while(temp\_E!=NULL){

        if(temp\_E->dest==*ch2*){

            return true;

        }

        temp\_E=temp\_E->link;

    }

    return false;

}

//Topological Sort

void topologicalSort(*node*\* *start*){

    queue<char> q1;

*node*\* temp=*start*;

    while(temp!=NULL){

        if(cal\_indegree(*start*,temp->info)==0){

            q1.push(temp->info);

            cout<<"temp-info: "<<temp->info<<endl;

        }

        temp=temp->next;

    }

    while(!q1.empty()){

        char c=q1.front();

        q1.pop();

*node*\* dummy=findNode(*start*,c);

*edge*\* temp\_E=dummy->adj;

        cout<<c<<" ";

        while(temp\_E!=NULL){

            char ch=temp\_E->dest;

            if(cal\_indegree(*start*,ch)==1 and checkEdge(*start*,c,ch)){

                q1.push(ch);

            }

            temp\_E=temp\_E->link;

        }

*start*=DeleteNode(*start*,c);

    }

    cout<<endl;

}

int main(){

    char c,ch;

    int n,choice;

*node*\* start=NULL;

    cout<<"Enter the number of vertices: ";

    cin>>n;

    while(n){

        cout<<"Enter vertex: ";

        cin>>c;

        insertNode(start,c);

        n--;

    }

    int edges;

    cout<<"Enter the number of edges: ";

    cin>>edges;

    while(edges){

        char ch1,ch2;

        cout<<"Enter Vertices: "<<endl;

        cin>>ch1>>ch2;

        createEdge(start,ch1,ch2);

        edges--;

    }

    cout<<"ALL THE EDGES ARE INSERTED SUCCESSFULLY"<<endl;

    do{

        cout<<"\*\*\*\*\*\*\*\*THIS PROGRAM IS GRAPH REPRESENTATION\*\*\*\*\*"<<endl;

        cout<<"1. INSERT NODE"<<endl;

        cout<<"2. INSERT EDGE"<<endl;

        cout<<"3. PRINT NODE-LIST"<<endl;

        cout<<"4. PRINT ADJACENCY-LIST"<<endl;

        cout<<"5. DELETE NODE"<<endl;

        cout<<"6. DELETE EDGE"<<endl;

        cout<<"7. CALCULATE DEGREE OF A NODE"<<endl;

        cout<<"8. CALCULATE INDEGREE OF A NODE"<<endl;

        cout<<"9. TOPOLOGICAL SORT"<<endl;

        cout<<"Enter your choice: ";

        cin>>choice;

        switch(choice){

            case 1:{

                cout<<"Enter the number of vertices: ";

                cin>>n;

                while(n){

                    cout<<"Enter vertex: ";

                    cin>>c;

                    insertNode(start,c);

                    n--;

                }

                cout<<"ALL THE NODES ARE INSERTED SUCCESSFULLY\n"<<endl;

                break;

            }

            case 2:{

                cout<<"Enter the number of edges: ";

                cin>>edges;

                while(edges){

                    char ch1,ch2;

                    cout<<"Enter Vertices: "<<endl;

                    cin>>ch1>>ch2;

                    createEdge(start,ch1,ch2);

                    edges--;

                }

                cout<<"ALL THE EDGES ARE INSERTED SUCCESSFULLY\n"<<endl;

                break;

            }

            case 3:{

                cout<<"Node list is: ";

                printNode(start);

                cout<<endl;

                break;

            }

            case 4:{

                cout<<"Enter the Element for Adjacencylist: ";

                cin>>c;

                cout<<"Adjacencylist is: "<<endl;

                printAdj(start,c);

                cout<<endl;

                break;

            }

            case 5:{

                cout<<"Enter the Node you want to delete: ";

                cin>>c;

                start=DeleteNode(start,c);

                cout<<"DELETE SUCCESSFULLY\n"<<endl;

                break;

            }

            case 6:{

                char ch1,ch2;

                cout<<"Enter the Edges you want to delete: ";

                cin>>ch1>>ch2;

                DeleteEdge(start,ch1,ch2);

                cout<<"DELETE SUCCESSFULLY\n"<<endl;

                break;

            }

            case 7:{

                // calculte degree

                char c;

                cout<<"Enter the vertix to find its Degree: ";

                cin>>c;

                Cal\_degree(start,c);

                cout<<endl;

                break;

            }

            case 8:{

                char c;

                cout<<"Enter the vertix to find its InDegree: ";

                cin>>c;

                cout<<cal\_indegree(start,c)<<endl<<endl;

                break;

            }

            case 9:{

                cout<<"AFTER SORTING: "<<endl;

                topologicalSort(start);

                break;

            }

            default:{

                cout<<"Enter the valid choice!!!"<<endl;

                break;

            }

        }

        cout<<"Do u wish to continue(y/n): ";

        cin>>ch;

        cout<<endl;

    }while(ch!='n');

    return 0;

}

OUTPUT

// incomplete for Deletion on head

SOURCE CODE

#include<iostream>

#include<queue>

using namespace std;

class *edge*{

public:

    char dest;

*edge*\* link;

    edge(char *ch*){

        dest=*ch*;

        link=NULL;

    }

};

class *node*{

public:

    char info;

*node*\* next;

*edge*\* adj;

    node(char *val*){

        info=*val*;

        next=NULL;

        adj=NULL;

    }

};

//insert node At last

void insertNode(*node*\* &*start*,char *ele*){

*node*\* n=**new** *node*(*ele*);

        if(*start*==NULL){

*start*=n;

            return;

        }

*node*\*temp=*start*;

        while(temp->next!=NULL){

            temp=temp->next;

        }

        temp->next=n;

        return;

}

// Find Node

*node*\* findNode(*node*\* *start*,char *ch*){

*node*\* temp=*start*;

        if(*start*==NULL){

            return NULL;

        }

        while(temp!=NULL){

            if(temp->info==*ch*){

                return temp;

            }

            temp=temp->next;

        }

        return NULL;

}

//Create Egde between two nodes

void createEdge(*node*\* &*start*,char *ch1*,char *ch2*){

*node*\* temp=*start*;

*node*\* n1=findNode(temp,*ch1*);

*node*\* n2=findNode(temp,*ch2*);

    if(n1!=NULL and n2!=NULL){

*edge*\* ed=**new** *edge*(*ch2*);

        if(n1->adj==NULL){

            n1->adj=ed;

            return;

        }

*edge*\* temp\_edge=n1->adj;

        if(temp\_edge->link==NULL){

            temp\_edge->link=ed;

            return;

        }

        while(temp\_edge->link!=NULL){

            temp\_edge=temp\_edge->link;

        }

        temp\_edge->link=ed;

        return;

    }

    else{

        cout<<"Edge is not possible"<<endl;

        return;

    }

}

//Delete edge

void DeleteEdge(*node*\* *start*,char *ch1*,char *ch2*){

*node*\* n1=findNode(*start*,*ch1*);

    if(n1==NULL){

        return;

    }

    if(n1->adj==NULL){

        return;

    }

*edge*\* temp\_E=n1->adj;

    if(temp\_E->dest == *ch2*){

        // edge\* todele=temp\_E;

        n1->adj=temp\_E->link;

        delete temp\_E;

        return;

    }

    while(temp\_E->link!=NULL and temp\_E->link->dest!=*ch2*){

        temp\_E=temp\_E->link;

        // cout<<"\*";

    }

*edge*\* todelete=temp\_E->link;

    if(temp\_E->link!=NULL){

    temp\_E->link=temp\_E->link->link;

    }

    delete todelete;

}

*node*\* DeleteNode(*node*\* *start*,char *val*){

    if(*start*==NULL){

        return NULL;

    }

    // todelete node from adj list of vertices

*node*\* tp=*start*;

    while(tp!=NULL){

        DeleteEdge(*start*,tp->info,*val*);

        tp=tp->next;

    }

    // todelete adj list of Delnode

*node*\* tp2=findNode(*start*,*val*);

*edge*\* del=tp2->adj;

    while(del!=NULL){

        char current\_edge\_info = del->dest;

        del=del->link;

        DeleteEdge(*start*,*val*,current\_edge\_info);

    }

    if(*start*->info == *val*){

*node*\* new\_start = *start*->next;

        delete *start*;

        return new\_start;

    }

*node*\* temp=*start*;

    while(temp->next != NULL){

        if(temp->next->info==*val*){

            break;

        }

        temp=temp->next;

    }

*node*\* todelete=temp->next;

    if(todelete == NULL){

        cout<<"Element doesn't exist in graph";

        return *start*;

    }

    temp->next=temp->next->next;

    delete todelete;

    return *start*;

}

//print node

void printNode(*node*\* *temp*){

    if(*temp*==NULL){

        cout<<"EMPTY LIST"<<endl;

        return;

    }

    while(*temp*!=NULL){

        cout<<*temp*->info<<"->";

*temp*=*temp*->next;

    }

    cout<<"NULL"<<endl;

}

// print adjacency list

void printAdj(*node*\* *start*,char *ch*){

*node*\* temp=findNode(*start*,*ch*);

    if(temp==NULL){

        cout<<"EMPTY LIST"<<endl;

        return;

    }

    cout<<temp->info<<"-> ";

*edge*\* temp\_E=temp->adj;

    while(temp\_E!=NULL){

        cout<<temp\_E->dest<<",";

        temp\_E=temp\_E->link;

    }

    cout<<"NULL"<<endl;

}

// calculate indegree

void Cal\_degree(*node*\* *start*,char *ch*){

*node*\* temp=findNode(*start*,*ch*);

    if(temp==NULL){

        cout<<"EMPTY LIST"<<endl;

        return;

    }

    int count=0;

    cout<<"Degree of "<<temp->info<<"-> ";

*edge*\* temp\_E=temp->adj;

    while(temp\_E!=NULL){

        count++;

        temp\_E=temp\_E->link;

    }

    cout<<" "<<count<<endl;

}

int cal\_indegree(*node*\* *start*,char *ch*){   // not good approach

*node*\* temp1=findNode(*start*,*ch*);

    if(temp1==NULL){

        cout<<"NODE IS NOT PRESENT IN LIST"<<endl;

        return -1;

    }

    // cout<<"#"<<endl;

*node*\* temp=*start*;

    int count=0;

    while(temp!=NULL){

*edge*\* temp\_E=temp->adj;

        while(temp\_E!=NULL){

            if(temp\_E->dest==*ch*){

                count++;

            }

            temp\_E=temp\_E->link;

        }

        temp=temp->next;

    }

    return count;

}

bool checkEdge(*node*\* *start*,char *ch1*,char *ch2*){

*node*\* temp=findNode(*start*,*ch1*);

*edge*\* temp\_E=temp->adj;

    while(temp\_E!=NULL){

        if(temp\_E->dest==*ch2*){

            return true;

        }

        temp\_E=temp\_E->link;

    }

    return false;

}

//Topological Sort

void topologicalSort(*node*\* *start*){

    queue<char> q1;

*node*\* temp=*start*;

    while(temp!=NULL){

        if(cal\_indegree(*start*,temp->info)==0){

            q1.push(temp->info);

            cout<<"temp-info: "<<temp->info<<endl;

        }

        temp=temp->next;

    }

    while(!q1.empty()){

        char c=q1.front();

        q1.pop();

*node*\* dummy=findNode(*start*,c);

*edge*\* temp\_E=dummy->adj;

        cout<<c<<" ";

        while(temp\_E!=NULL){

            char ch=temp\_E->dest;

            if(cal\_indegree(*start*,ch)==1 and checkEdge(*start*,c,ch)){

                q1.push(ch);

            }

            temp\_E=temp\_E->link;

        }

*start*=DeleteNode(*start*,c);

    }

    cout<<endl;

}

int main(){

    char c,ch;

    int n,choice;

*node*\* start=NULL;

    cout<<"Enter the number of vertices: ";

    cin>>n;

    while(n){

        cout<<"Enter vertex: ";

        cin>>c;

        insertNode(start,c);

        n--;

    }

    int edges;

    cout<<"Enter the number of edges: ";

    cin>>edges;

    while(edges){

        char ch1,ch2;

        cout<<"Enter Vertices: "<<endl;

        cin>>ch1>>ch2;

        createEdge(start,ch1,ch2);

        edges--;

    }

    cout<<"ALL THE EDGES ARE INSERTED SUCCESSFULLY"<<endl;

    do{

        cout<<"\*\*\*\*\*\*\*\*THIS PROGRAM IS GRAPH REPRESENTATION\*\*\*\*\*"<<endl;

        cout<<"1. INSERT NODE"<<endl;

        cout<<"2. INSERT EDGE"<<endl;

        cout<<"3. PRINT NODE-LIST"<<endl;

        cout<<"4. PRINT ADJACENCY-LIST"<<endl;

        cout<<"5. DELETE NODE"<<endl;

        cout<<"6. DELETE EDGE"<<endl;

        cout<<"7. CALCULATE DEGREE OF A NODE"<<endl;

        cout<<"8. CALCULATE INDEGREE OF A NODE"<<endl;

        cout<<"9. TOPOLOGICAL SORT"<<endl;

        cout<<"Enter your choice: ";

        cin>>choice;

        switch(choice){

            case 1:{

                cout<<"Enter the number of vertices: ";

                cin>>n;

                while(n){

                    cout<<"Enter vertex: ";

                    cin>>c;

                    insertNode(start,c);

                    n--;

                }

                cout<<"ALL THE NODES ARE INSERTED SUCCESSFULLY\n"<<endl;

                break;

            }

            case 2:{

                cout<<"Enter the number of edges: ";

                cin>>edges;

                while(edges){

                    char ch1,ch2;

                    cout<<"Enter Vertices: "<<endl;

                    cin>>ch1>>ch2;

                    createEdge(start,ch1,ch2);

                    edges--;

                }

                cout<<"ALL THE EDGES ARE INSERTED SUCCESSFULLY\n"<<endl;

                break;

            }

            case 3:{

                cout<<"Node list is: ";

                printNode(start);

                cout<<endl;

                break;

            }

            case 4:{

                cout<<"Enter the Element for Adjacencylist: ";

                cin>>c;

                cout<<"Adjacencylist is: "<<endl;

                printAdj(start,c);

                cout<<endl;

                break;

            }

            case 5:{

                cout<<"Enter the Node you want to delete: ";

                cin>>c;

                start=DeleteNode(start,c);

                cout<<"DELETE SUCCESSFULLY\n"<<endl;

                break;

            }

            case 6:{

                char ch1,ch2;

                cout<<"Enter the Edges you want to delete: ";

                cin>>ch1>>ch2;

                DeleteEdge(start,ch1,ch2);

                cout<<"DELETE SUCCESSFULLY\n"<<endl;

                break;

            }

            case 7:{

                // calculte degree

                char c;

                cout<<"Enter the vertix to find its Degree: ";

                cin>>c;

                Cal\_degree(start,c);

                cout<<endl;

                break;

            }

            case 8:{

                char c;

                cout<<"Enter the vertix to find its InDegree: ";

                cin>>c;

                cout<<cal\_indegree(start,c)<<endl<<endl;

                break;

            }

            case 9:{

                cout<<"AFTER SORTING: "<<endl;

                topologicalSort(start);

                break;

            }

            default:{

                cout<<"Enter the valid choice!!!"<<endl;

                break;

            }

        }

        cout<<"Do u wish to continue(y/n): ";

        cin>>ch;

        cout<<endl;

    }while(ch!='n');

    return 0;

}

// Adjacencylist is: a-> c,b,d,NULL

// b-> d,c,NULL

// c-> NULL

// d-> c,e,NULL

// e-> c,NULL

/\*

            a----c

            |\  /|\

            | \/ | \

            | /\ |  e

            |/  \| /

            b----d/

\*/

OUTPUT