Question 1. and 2.

Program to represent a finite automata and check if a string is accepted by the automata or not

//states start from 0 and allowed symbols are alphabets

#include<stdio.h>

#include<string.h>

int main(){

    int num\_of\_states,num\_of\_alphabets;

    printf("\nEnter the number of States of the DFA: ");

    scanf("%d",&num\_of\_states);

    printf("\nEnter the number of symbols: ");

    scanf("%d",&num\_of\_alphabets);

    int transition\_table[num\_of\_states][num\_of\_alphabets];

    for(int i=0;i<num\_of\_states;i++){

        printf("\nFor the state %d",i);

        for(int j=0;j<num\_of\_alphabets;j++){

            printf("\nand the alphabet %c the next state is : ",('a'+j));

            scanf("%d",&transition\_table[i][j]);

        }

    }

    int initial\_state,num\_of\_final\_states;

    printf("\nEnter the initial state for the dfa: ");

    scanf("%d",&initial\_state);

    printf("\nEnter the number of final states: ");

    scanf("%d",&num\_of\_final\_states);

    int final\_states[num\_of\_final\_states];

    printf("\nEnter the final states: ");

    for(int i=0;i<num\_of\_final\_states;i++)

        scanf("%d",&final\_states[i]);

    printf("\nGiven Transition table for the DFA : \n");

    for(int i=0;i<num\_of\_states;i++){

        printf("\t%d",i);

    }

    for(int j=0;j<num\_of\_alphabets;j++){

        printf("\n%c",('a'+j));

        for(int i=0;i<num\_of\_states;i++)

            printf("\t%d",transition\_table[i][j]);

    }

    char string[256];

    int yes=0;

    do{

        printf("\nEnter a string to check if it is accepted by the dfa : ");

        getchar();

        gets(string);

        int curr\_state=initial\_state,index=0;

        while(string[index]!='\0'){

            if(transition\_table[curr\_state][string[index]-'a']<0) break;

            curr\_state=transition\_table[curr\_state][string[index]-'a'];

            index++;

        }

        if(index<strlen(string)) printf("\nGiven string is not accepted by the DFA!!");

        else {

            for(int i=0;i<num\_of\_final\_states;i++){

                if(final\_states[i]==curr\_state){

                    printf("\nGiven string is accepted by the DFA!!");

                    goto AA;

                }

            }

            printf("\nGiven string not accepted by the DFA!!");

        }

        AA:

        printf("\n Do you want to check another string: (1 for yes, 0 for no): ");

        scanf("%d",&yes);

    }while(yes);

    return 0;

}

Question 3.

Program to check if a string is accepted by a NFA or not

//states start from 0 and symbols can be any character other than 'e' as 'e' is used to denote epsilon

#include<stdio.h>

#include<stdlib.h>

typedef struct edge{

    char symbol;

    int vertex;

    struct edge\* next;

}edge;

void push(edge\*\* head,char symbol,int vertex){

    edge \*temp=malloc(sizeof(edge));

    temp->symbol=symbol;

    temp->vertex=vertex;

    temp->next=NULL;

    if(\*head==NULL) {

        \*head=temp;

        return;

    }

    edge \*curr=\*head;

    while(curr->next!=NULL){

        curr=curr->next;

    }

    curr->next=temp;

}

int dfs(int node,edge\*\* nodes,char\* string,int index,int\* final){

    if(string[index]=='\0'){

        if(final[node]) return 1;

        else return 0;

    }

    edge \*curr=nodes[node];

    while(curr){

        if(curr->symbol=='e'){

            if(dfs(curr->vertex,nodes,string,index,final)) return 1;

        }

        else if(curr->symbol==string[index]) {

           if(dfs(curr->vertex,nodes,string,index+1,final)) return 1;

        }

        curr=curr->next;

    }

    return 0;

}

int main(){

    int num\_of\_states,num\_of\_transitions;

    printf("\nEnter the number of States in the NFA: ");

    scanf("%d",&num\_of\_states);

    edge \*nodes[num\_of\_states];

    for(int i=0;i<num\_of\_states;i++) nodes[i]=NULL;

    int final[num\_of\_states],vertex,yes;

    printf("\nEnter the number of transitions in the nfa: ");

    scanf("%d",&num\_of\_transitions);

    printf("\nEnter the transitons in the format: state state character (character e is used to denote epsilon transition )\n");

    for(int i=0;i<num\_of\_transitions;i++){

        int st,en;

        char ch;

        scanf("%d %d %c",&st,&en,&ch);

        push(&nodes[st],ch,en);

    }

    for(int i=0;i<num\_of\_states;i++){

        printf("\nIs the state: %d, final state or not (1 for yes, 0 for no): ",i);

        scanf("%d",&final[i]);

    }

    char string[256];

    yes=0;

    do{

        printf("\nEnter a string to check if it is accepted by the nfa : ");

        getchar();

        gets(string);

        if(dfs(0,nodes,string,0,final)) printf("\nEntered string is accepted by the nfa!!");

        else printf("\nEntered string is not accepted by the nfa!!");

        printf("\nDO you want to check another string? (1 for yes, 0 for no): ");

        scanf("%d",&yes);

    }while(yes);

    return 0;

}

Question 4.

Program to implement Thompson construction of NFA from RE

/\* Its a C++ program

    State starts from 0, and any character other than e can be used as symbols, character 'e' is used to denote epsilon

\*/

#include<bits/stdc++.h>

using namespace std;

struct node{

    char symbol;

    int to;

    node \*next;

};

int priority(char ch){

    if(ch=='\*') return 3;

    else if(ch=='.') return 2;

    else if(ch=='|') return 1;

    return -1;

}

void addconcat(string &reg){

    string nreg;

    for(int i=0;i<int(reg.size()-1);i++){

        nreg+=reg[i];

        if(reg[i]=='.' or reg[i]=='|' or reg[i]=='+' or reg[i]=='(' or reg[i+1]=='|' or reg[i+1]=='+' or reg[i+1]==')' or reg[i+1]=='.' or (reg[i]==')' and reg[i+1]=='\*')) continue;

        nreg+='.';

    }

    nreg+=reg.back();

    reg=nreg;

}

string to\_postfix(string &reg){

    stack<char> st;

    string postfix;

    for(int i=0;i<reg.size();i++){

        if((reg[i]>='a' and reg[i]<='z') or (reg[i]>='A' and reg[i]<='Z')) postfix+=reg[i];

        else if(reg[i]=='(') st.push(reg[i]);

        else if(reg[i]==')') {

            while(!st.empty() and st.top()!='('){

                postfix+=st.top();

                st.pop();

            }

            if(st.empty()) return "";

            st.pop();

        }else{

            while(!st.empty() and priority(reg[i])<=priority(st.top())){

                postfix+=st.top();

                st.pop();

            }

            st.push(reg[i]);

        }

    }

    while(!st.empty()){

        postfix+=st.top();

        st.pop();

    }

    return postfix;

}

node \*newnode(char ch,int to){

    node \*temp=new node;

    temp->symbol=ch;

    temp->to=to;

    temp->next=NULL;

    return temp;

}

int addconcat(vector<node\*> &nfa,vector<vector<int>> &st){

    if(st.size()<2) return 1;

    int s0=st[st.size()-2][1];

    int s1=st[st.size()-1][0];

    int newfirst=st[st.size()-2][0];

    int newlast=st[st.size()-1][1];

    st.pop\_back();

    st.pop\_back();

    node \*curr=nfa[s0],\*temp=nfa[s1];

    while(curr->next){

        curr=curr->next;

    }

    while(temp){

        if(temp->to!=-1){

            curr->next=newnode(temp->symbol,temp->to);

            curr=curr->next;

        }

        temp=temp->next;

    }

    temp=nfa[s1];

    curr=temp->next;

    while(curr){

        free(temp);

        temp=curr;

        curr=curr->next;

    }

    free(curr);

    nfa[s1]=newnode('e',-1);

    st.push\_back({newfirst,newlast});

    return 0;

}

int addor(vector<node\*> &nfa,vector<vector<int>> &st){

    if(st.size()<2) return 1;

    int s10=st[st.size()-2][0],s11=st[st.size()-2][1];

    int s20=st[st.size()-1][0],s21=st[st.size()-1][1];

    int newstart=nfa.size();

    node \*temp=newnode('e',s10);

    temp->next=newnode('e',s20);

    nfa.push\_back(temp);

    int newlast=nfa.size();

    temp=newnode('e',-1);

    nfa.push\_back(temp);

    st.pop\_back();

    st.pop\_back();

    st.push\_back({newstart,newlast});

    node \*curr=nfa[s11];

    while(curr->next){

        curr=curr->next;

    }

    curr->next=newnode('e',newlast);

    curr=nfa[s21];

    while(curr->next){

        curr=curr->next;

    }

    curr->next=newnode('e',newlast);

    return 0;

}

int addstar(vector<node\*> &nfa,vector<vector<int>> &st){

    int s0=st[st.size()-1][0],s1=st[st.size()-1][1];

    int newstart=nfa.size(),newlast=nfa.size()+1;

    st.pop\_back();

    st.push\_back({newstart,newlast});

    node \*temp=newnode('e',s0);

    temp->next=newnode('e',newlast);

    nfa.push\_back(temp);

    temp=newnode('e',-1);

    nfa.push\_back(temp);

    node \*curr=nfa[s1];

    while(curr->next){

        curr=curr->next;

    }

    curr->next=newnode('e',s0);

    curr=curr->next;

    curr->next=newnode('e',newlast);

    return 0;

}

int table(vector<node\*> &nfa){

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

    cout<<"| from state\t|with input->state\t|"<<endl;

    for(int i=0;i<nfa.size();i++){

        cout<<"| "<<i+1<<"          \t|";

        node\* head = nfa[i];

        while(head!=NULL){

            if(head->to==-1){

                head = head->next;

                continue;

            }

            cout<<" "<<head->symbol<<"->"<<head->to+1<<"  \t";

            head=head->next;

        }

        cout<<endl;

    }

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

}

int main(){

    string reg;

    cout<<"Enter the regular expression: \n";

    cin>>reg;

    string postfix;

    vector<node\*> nfa;

    addconcat(reg);

    postfix=to\_postfix(reg);

    cout<<"postfix expression is = "<<postfix<<"\n";

    vector<vector<int>> st;

    cout<<reg<<"\n";

    for(int i=0;i<postfix.size();i++){

        if((postfix[i]>='a' and postfix[i]<='z') or (postfix[i]>='A' and postfix[i]<='Z')){

            node \*s0=newnode(postfix[i],nfa.size()+1);

            node \*s1=newnode('e',-1);

            nfa.push\_back(s0);

            nfa.push\_back(s1);

            st.push\_back({int(nfa.size())-2,int(nfa.size())-1});

        }else if(postfix[i]=='\*') {

            if(addstar(nfa,st)) {

                cout<<"Invalid";

                return 0;

            }

        }

        else if(postfix[i]=='.') {

            if(addconcat(nfa,st)) {

                cout<<"Invalid";

                return 0;

            }

        }

        else if(postfix[i]=='|') {

            if(addor(nfa,st)){

                cout<<"Invalid";

                return 0;

            }

        }

    }

    cout<<"table for nfa: \n";

    table(nfa);

    cout<<"starting state for the nfa is : "<<st[st.size()-1][0]+1<<"\n";

    cout<<"final state for the nfa is : "<<st[st.size()-1][1]+1<<"\n";

    return 0;

}

Question 5.

Program to find epsilon closure of all states

/\* Its a C++ program

    State starts from 0, and any character other than e can be used as symbols, character 'e' is used to denote epsilon

\*/

#include<bits/stdc++.h>

using namespace std;

struct node{

    char symbol;

    int to;

};

void find\_closure(vector<int> &states,vector<vector<node>> &nfa){

    stack<int> st;

    for(int i=0;i<int(states.size());i++){

        if(states[i]) st.push(i);

    }

    while(!st.empty()){

        int temp=st.top();

        st.pop();

        for(int j=0;j<int(nfa[temp].size());j++){

            if(nfa[temp][j].symbol=='e' and states[nfa[temp][j].to]==0) {

                states[nfa[temp][j].to]=1;

                st.push(nfa[temp][j].to);

            }

        }

    }

}

int main(){

    int num\_of\_states,num\_of\_final\_states,initial\_state,num\_of\_transitions;

    cout<<"\nEnter the number of states in the nfa: ";

    cin>>num\_of\_states;

    vector<vector<node>> nfa(num\_of\_states);

    cout<<"\nEnter the initial state of the nfa:  ";

    cin>>initial\_state;

    cout<<"\nEnter the number of final states: ";

    cin>>num\_of\_final\_states;

    cout<<"\nNow enter the final states: ";

    vector<int> isfinal(num\_of\_states);

    for(int i=0;i<num\_of\_final\_states;i++){

        int state;

        cin>>state;

        isfinal[state]=1;

    }

    cout<<"\nEnter the number of transitions for the nfa: ";

    cin>>num\_of\_transitions;

    cout<<"\nEnter the transitons in the format: state state character (character 'e' is used to denote 'epsilon')\n";

    for(int i=0;i<num\_of\_transitions;i++){

        int st,en;

        char ch;

        cin>>st>>en>>ch;

        node temp;

        temp.symbol=ch;

        temp.to=en;

        nfa[st].push\_back(temp);

    }

    for(int i=0;i<num\_of\_states;i++){

        cout<<"\n\nEpsilon Closure of state "<<i<<" is = ";

        vector<int> states(num\_of\_states);

        states[i]=1;

        find\_closure(states,nfa);

        cout<<"\n{ ";

        for(int j=0;j<num\_of\_states;j++){

            if(states[j])

                cout<<j<<", ";

        }

        cout<<"}";

    }

    return 0;

}

Question 6.

Program to convert a NFA to a DFA by ε closure method.

/\* Its a C++ program

    State starts from 0, and any character other than e can be used as symbols, character 'e' is used to denote epsilon

\*/

#include<bits/stdc++.h>

using namespace std;

//Max states this program can work upon = 60

struct node{

    char symbol;

    int to;

};

void find\_closure(vector<int> &states,vector<vector<node>> &nfa){

    stack<int> st;

    for(int i=0;i<int(states.size());i++){

        if(states[i]) st.push(i);

    }

    while(!st.empty()){

        int temp=st.top();

        st.pop();

        for(int j=0;j<int(nfa[temp].size());j++){

            if(nfa[temp][j].symbol=='e' and states[nfa[temp][j].to]==0) {

                states[nfa[temp][j].to]=1;

                st.push(nfa[temp][j].to);

            }

        }

    }

}

void move(vector<int> &states,vector<vector<node>> &nfa,char ch){

    vector<int> new\_states(states.size());

    for(int i=0;i<int(states.size());i++){

        if(states[i]){

            for(int j=0;j<int(nfa[i].size());j++){

                if(nfa[i][j].symbol==ch) {

                    new\_states[nfa[i][j].to]=1;

                }

            }

        }

    }

    states=new\_states;

}

void print\_dfa(map<long long,vector<node>> &dfa,int initial\_state,vector<int> &isfinal,int num\_of\_states){

    map<long long ,int> mp;

    int curr=0;

    set<long long> final\_states;

    for(map<long long,vector<node>>::iterator it=dfa.begin();it!=dfa.end();it++){

        mp[it->first]=curr;

        long long mask=it->first;

        for(int i=0;i<num\_of\_states;i++){

            if(mask&1<<i and isfinal[i]) {

                final\_states.insert(mask);

                break;

            }

        }

        curr++;

    }

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<"\n";

    cout<<"| from state\t|with input->state\t|"<<"\n";

    for(map<long long,vector<node>>::iterator it=dfa.begin();it!=dfa.end();it++){

        cout<<"| "<<mp[it->first]<<"          \t|";

        for(int i=0;i<int((it->second).size());i++){

            cout<<" "<<(it->second)[i].symbol<<"->"<<mp[(it->second)[i].to]<<"  \t";

        }

        cout<<"\n";

    }

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

    cout<<"\n\n Initial state for the dfa is :  "<<mp[initial\_state];

    cout<<"\nFinal States of the dfa : ";

    for(set<long long>::iterator it=final\_states.begin();it!=final\_states.end();it++) cout<<mp[\*it]<<" ";

}

int main(){

    int num\_of\_states,num\_of\_final\_states,initial\_state,num\_of\_transitions;

    cout<<"\nEnter the number of states in the nfa: ";

    cin>>num\_of\_states;

    vector<vector<node>> nfa(num\_of\_states);

    cout<<"\nEnter the initial state of the nfa:  ";

    cin>>initial\_state;

    cout<<"\nEnter the number of final states: ";

    cin>>num\_of\_final\_states;

    cout<<"\nNow enter the final states: ";

    vector<int> isfinal(num\_of\_states);

    for(int i=0;i<num\_of\_final\_states;i++){

        int state;

        cin>>state;

        isfinal[state]=1;

    }

    cout<<"\nEnter the number of transitions for the nfa: ";

    cin>>num\_of\_transitions;

    set<char> symbols;

    cout<<"\nEnter the transitons in the format: state state character (character 'e' is used to denote 'epsilon')\n";

    for(int i=0;i<num\_of\_transitions;i++){

        int st,en;

        char ch;

        cin>>st>>en>>ch;

        node temp;

        temp.symbol=ch;

        temp.to=en;

        nfa[st].push\_back(temp);

        symbols.insert(ch);

    }

    map<long long,vector<node>> dfa;

    stack<long long> st;

    vector<int> states(num\_of\_states);

    states[initial\_state]=1;

    find\_closure(states,nfa);

    long long mask=0;

    for(int i=0;i<num\_of\_states;i++){

        if(states[i]) mask|=1<<i;

    }

    initial\_state=mask;

    st.push(mask);

    while(!st.empty()){

        long long temp=st.top();

        dfa[temp]=vector<node>();

        st.pop();

        for(set<char>::iterator it=symbols.begin();it!=symbols.end();it++){

            if(\*it=='e') continue;

            states=vector<int>(num\_of\_states);

            for(int i=0;i<num\_of\_states;i++){

                if(temp&1<<i) states[i]=1;

            }

            move(states,nfa,\*it);

            find\_closure(states,nfa);

            mask=0;

            for(int i=0;i<num\_of\_states;i++){

                if(states[i]) mask|=1<<i;

            }

            if(dfa.find(mask)==dfa.end()){

                st.push(mask);

            }

            node curr;

            curr.symbol=\*it;

            curr.to=mask;

            dfa[temp].push\_back(curr);

        }

    }

    print\_dfa(dfa,initial\_state,isfinal,num\_of\_states);

    return 0;

}

Question 7.

Program to minimize a dfa

/\*states are number from 0

  symbols are int integer form like 0,1,2,3

  list p[2] stores the sets representing dfa

\*/

#include<bits/stdc++.h>

using namespace std;

vector<vector<int>> p[2];

int find\_index(int x){

    for(int i=0; i<p[1].size(); i++) {

        for(int j=0; j<p[1][i].size(); j++) {

            if(p[1][i][j] == x) return i;

        }

    }

}

void printf\_dfa(vector<vector<int>> &transition\_table,int intitial\_sate,vector<int>&isfinal){

    int num\_of\_symbols=transition\_table[0].size(),num\_of\_states=p[1].size();

    cout<<"\nMinimized Dfa table: ";

    cout<<"\n---------------------\n";

    for(int i=0;i<num\_of\_symbols;i++){

        cout<<"\t"<<i;

    }

    for(int i=0;i<num\_of\_states;i++){

        cout<<"\n"<<i;

        for(int j=0;j<num\_of\_symbols;j++){

            cout<<"\t"<<find\_index(transition\_table[p[1][i][0]][j]);

        }

    }

    cout<<"\nInitial State for the dfa : "<<find\_index(intitial\_sate);

    set<int> s;

    for(int i=0;i<int(isfinal.size());i++){

        if(!isfinal[i]) continue;

        s.insert(find\_index(isfinal[i]));

    }

    cout<<"\nFinal states for the dfa are : ";

    for(set<int>::iterator it=s.begin();it!=s.end();it++){

        cout<<\*it<<" ";

    }

}

int main(){

    int num\_of\_states,num\_of\_symbols;

    cout<<"Enter the number of states for the dfa: ";

    cin>>num\_of\_states;

    cout<<"\nEnter the number of symbols: ";

    cin>>num\_of\_symbols;

    vector<vector<int>> transition\_table(num\_of\_states,vector<int>(num\_of\_symbols));

    for(int i=0;i<num\_of\_states;i++){

        cout<<"\nFor the state "<<i<<": ";

        for(int j=0;j<num\_of\_symbols;j++){

            cout<<"\nand the symbol "<<j<<" the next state is : ";

            cin>>transition\_table[i][j];

        }

    }

    int initial\_state,num\_of\_final;

    cout<<"\nEnter the initial state for the dfa: ";

    cin>>initial\_state;

    cout<<"\nEnter the number of final states: ";

    cin>>num\_of\_final;

    vector<int> isfinal(num\_of\_states);

    cout<<"\nEnter the final states: ";

    for(int i=0;i<num\_of\_final;i++){

        int temp;

        cin>>temp;

        isfinal[temp]=1;

    }

    vector<int> v[2];

    for(int i=0;i<num\_of\_states;i++){

        v[isfinal[i]].push\_back(i);

    }

    p[0].push\_back(v[0]);

    p[0].push\_back(v[1]);

    p[1]=p[0];

    int change;

    while(1){

        p[0]=p[1]; //p[0] is the old set , p[1] new set

        for(int i=0;i<p[1].size();i++){

            for(int j=1;j<p[1][i].size();j++){

                change=0;

                for(int k=0;k<num\_of\_symbols;k++){

                    if(find\_index(transition\_table[p[1][i][j]][k])!=find\_index(transition\_table[p[1][i][0]][k])){

                        change=1;

                        break;

                    }

                }

                if(!change) continue;

                for(int x=p[0].size();x<p[1].size();x++){

                    change=0;

                    for(int k=0;k<num\_of\_symbols;k++){

                        if(find\_index(transition\_table[p[1][i][j]][k])!=find\_index(transition\_table[p[1][x][0]][k])){

                            change=1;

                            break;

                        }

                    }

                    if(!change){

                        p[1][x].push\_back(p[1][i][j]);

                        p[1][i].erase(p[1][i].begin()+j);

                        break;

                    }

                }

                if(change){

                    p[1].push\_back({p[1][i][j]});

                    p[1][i].erase(p[1][i].begin()+j);

                }

            }

        }

        if(p[0].size()==p[1].size()) break;

    }

    printf\_dfa(transition\_table,initial\_state,isfinal);

    return 0;

}

All the above programs have been compiled, run (on windows) and tested on some sample inputs to verify the correctness.