ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

COMPILER DESIGN LAB

EXP 3 - NFA to DFA

AIM :

To convert a given NFA into its equivalent DFA using the subset construction method while also showing the transition table in a C++ compiler.

REQUIREMENTS :

1. Knowledge of the concepts of Finite Automata and its types
2. Knowledge of the definition of an NFA and the transition function with states and input symbols.
3. Knowledge of the methods to convert Non-Deterministic Finite Automata(NFA) into its equivalent Deterministic Finite Automata(DFA) - the idea of subset construction from the power set.
4. Ability to recognise final state and hence construct the DFA diagram
5. Online GDB compiler to execute and implement code

THEORY :

NFA:

- NFA stands for non-deterministic finite automata. It is easier to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains the ε transition.

CONVERSION - NFA TO DFA:

- Observe the NFA diagram and list down Q (all states) and the input symbols
- Now construct the power set of Q (list down all possible subsets)
- Map all the subsets of Q with input symbols in the transition table . On giving which input to which state , what transition happens - this is what the transition table will tell us.
- Rename the subsets of Q as A,B,C,D… and put a star near the final states. Now , in the next version of transition table , replace the values with these new symbols A,B,C,D… eg : write null state as A , {q0} as B , {q1} as C , {q2} as D , {q0,q1} as E , etc.

- Now in the final version of the transition table , only pick up the important states which actually reflect the changes in transition . These last few states would be shown in the DFA diagram as per their variation with the incoming input symbols.

ALGORITHM :

STEP 1: In the main function we initialise a 3d-array vector to dynamically store the data for the transition table of NFA and then eventually convert it to DFA
STEP 2: We take input for the total number of states in NFA and also total number of elements in the alphabet (no. of input symbols)
STEP 3: Then for each state we build nested for loops wherein for each state and for each input we take the number of output states
STEP 4:Next whatever we get as a 3D array we insert it to the main 3D vector table
STEP 5: We now call the print function wherein we display the 3d table elements
STEP 6: Next for the transition table of DFA we need to do the epsilon closure for that we need to compute 4 nested for loops within a while loop
STEP 7: Use the graph adjacency technique to track if for any of the state whether it is directing to a new state/adjacency location(this indicates state change according to given input symbol).If yes we take that row data and store it into another 3-d matrix which contains the solution
STEP 8: Now print the dfa table by creating a printdfa function which displays the 3D array data

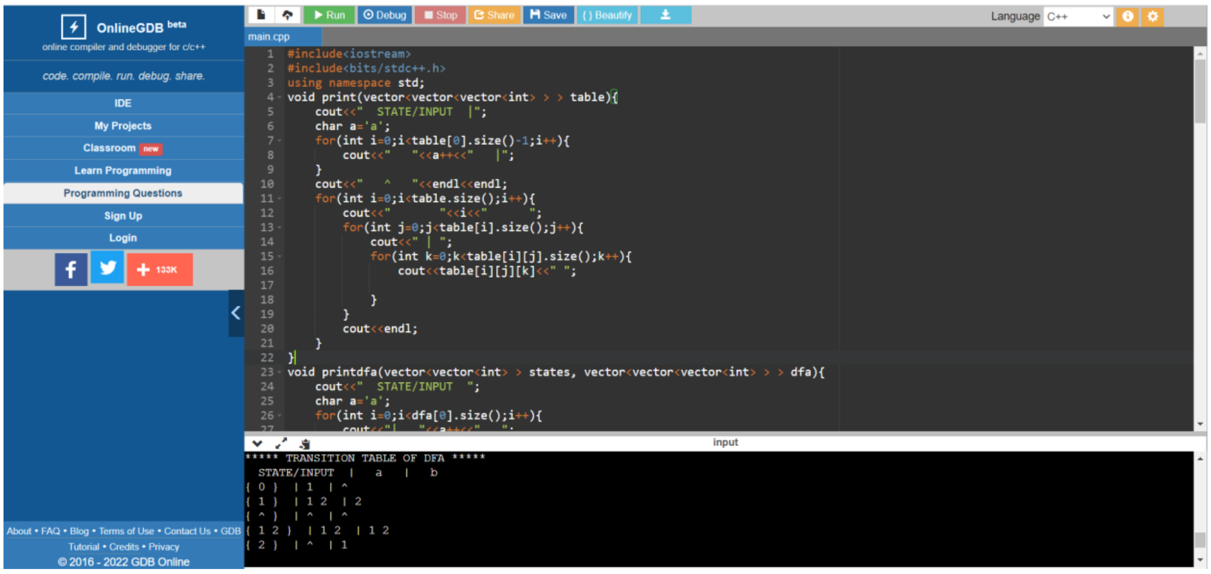
SOURCE CODE :

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
void print(vector<vector<vector<int> > > table){
	cout<<"  STATE/INPUT  |";
	char a='a';
	for(int i=0;i<table[0].size()-1;i++){
		cout<<"   "<<a++<<"   |";
	}
	cout<<"   ^   "<<endl<<endl;
	for(int i=0;i<table.size();i++){
		cout<<"       "<<i<<"      ";
		for(int j=0;j<table[i].size();j++){
			cout<<" | ";
			for(int k=0;k<table[i][j].size();k++){
				cout<<table[i][j][k]<<" ";

			}
		}
		cout<<endl;
	}
}

void printdfa(vector<vector<int> > states, vector<vector<vector<int> > > dfa){
	cout<<"  STATE/INPUT  ";
	char a='a';
	for(int i=0;i<dfa[0].size();i++){
		cout<<"|   "<<a++<<"   ";
	}
	cout<<endl;
	for(int i=0;i<states.size();i++){
		cout<<"{ ";
		for(int h=0;h<states[i].size();h++)
			cout<<states[i][h]<<" ";
		if(states[i].empty()){
			cout<<"^ ";
		}
		cout<<"} ";
		for(int j=0;j<dfa[i].size();j++){
			cout<<" | ";
			for(int k=0;k<dfa[i][j].size();k++){
				cout<<dfa[i][j][k]<<" ";

			}
			if(dfa[i][j].empty()){
				cout<<"^ ";
```

```
                        }
                }
                cout<<endl;
        }
}
vector<int> closure(int s,vector<vector<vector<int> > > v){
        vector<int> t;
        queue<int> q;
        t.push_back(s);
        int a=v[s][v[s].size()-1].size();
        for(int i=0;i<a;i++){
                t.push_back(v[s][v[s].size()-1][i]);
                //cout<<"t[i]"<<t[i]<<endl;
                q.push(t[i]);
        }
        while(!q.empty()){
                int f=q.front();
                q.pop();
                if(!v[f][v[f].size()-1].empty()){
                        int u=v[f][v[f].size()-1].size();
                        for(int i=0;i<u;i++){
                                int y=v[f][v[f].size()-1][i];
                                if(find(t.begin(),t.end(),y)==t.end()){
                                        //cout<<"y"<<y<<endl;
                                        t.push_back(y);
                                        q.push(y);
                                }
                        }
                }
        }
        return t;
}
int main(){
        int n,alpha;
        cout<<"************************ NFA to DFA ************************"<<endl<<endl;
        cout<<"Enter total number of states in NFA : ";
        cin>>n;
        cout<<"Enter number of elements in alphabet(no of input symbols) : ";
        cin>>alpha;
        vector<vector<vector<int> > > table;
        for(int i=0;i<n;i++){
                cout<<"For state "<<i<<endl;
                vector< vector< int > > v;
                char a='a';
                int y,yn;
                for(int j=0;j<alpha;j++){
                        vector<int> t;
                        cout<<"Enter no. of output states for input "<<a++<<" : ";
                        cin>>yn;
                        cout<<"Enter output states :"<<endl;
                        for(int k=0;k<yn;k++){
                                cin>>y;
                                t.push_back(y);
                        }
                        v.push_back(t);
                }
                vector<int> t;
                cout<<"Enter no. of output states for input ^ : ";
                cin>>yn;
                cout<<"Enter output states :"<<endl;
                for(int k=0;k<yn;k++){
                        cin>>y;
                        t.push_back(y);
                }
                v.push_back(t);
                table.push_back(v);
        }
        cout<<"***** TRANSITION TABLE OF NFA *****"<<endl;
        print(table);
        cout<<endl<<"***** TRANSITION TABLE OF DFA *****"<<endl;
        vector<vector<vector<int> > > dfa;
        vector<vector<int> > states;
        states.push_back(closure(0,table));
        queue<vector<int> > q;
        q.push(states[0]);
        while(!q.empty()){
                vector<int> f=q.front();
                q.pop();
                vector<vector<int> > v;
                for(int i=0;i<alpha;i++){
                        vector<int> t;
                        set<int> s;
                        for(int j=0;j<f.size();j++){

                                for(int k=0;k<table[f[j]][i].size();k++){
                                        vector<int> cl= closure(table[f[j]][i][k],table);
                                        for(int h=0;h<cl.size();h++){
                                                if(s.find(cl[h])==s.end())
                                                s.insert(cl[h]);
                                        }
                                }
                        }
                        for(set<int >::iterator u=s.begin(); u!=s.end();u++)
                                t.push_back(*u);
                        v.push_back(t);
                        if(find(states.begin(),states.end(),t)==states.end())
```

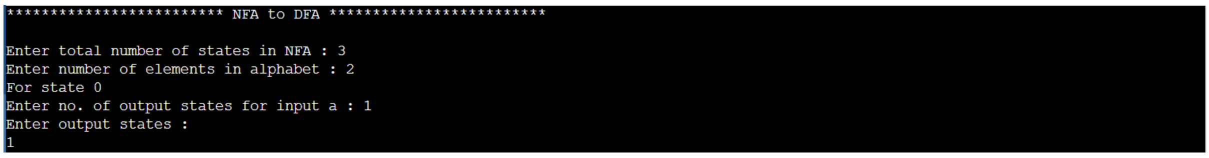
```
                        {
                                states.push_back(t);
                                q.push(t);
                        }
                }
                dfa.push_back(v);
        }
        printdfa(states,dfa);
}
```

SCREENSHOTS OF OUTPUT :

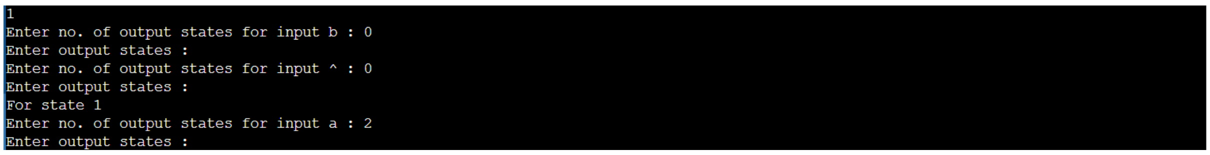


```cpp
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
void print(vector<vector<vector<int> > > table){
    cout<<"  STATE/INPUT  |";
    char a='a';
    for(int i=0;i<table[0].size()-1;i++){
        cout<<"   "<<a++<<"   |";
    }
    cout<<"   ^   "<<endl<<endl;
    for(int i=0;i<table.size();i++){
        cout<<"       "<<i<<"       ";
        for(int j=0;j<table[i].size();j++){
            cout<<" | ";
            for(int k=0;k<table[i][j].size();k++){
                cout<<table[i][j][k]<<" ";

            }
        }
        cout<<endl;
    }
}
void printdfa(vector<vector<int> > states, vector<vector<vector<int> > > dfa){
    cout<<"  STATE/INPUT  ";
    char a='a';
    for(int i=0;i<dfa[0].size();i++){
```

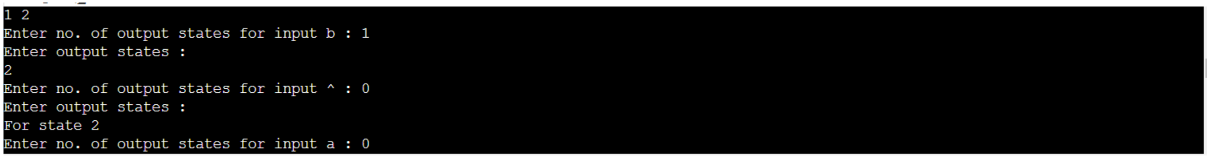
```
***** TRANSITION TABLE OF DFA *****
  STATE/INPUT  |   a   |   b
{ 0 }  | 1  | ^
{ 1 }  | 1 2  | 2
{ ^ }  | ^  | ^
{ 1 2 }  | 1 2  | 1 2
{ 2 }  | ^  | 1
```



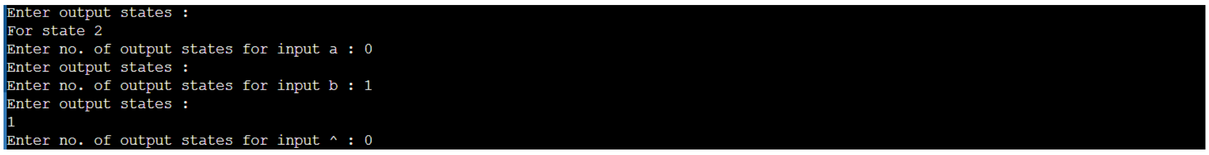
```
************************* NFA to DFA *************************

Enter total number of states in NFA : 3
Enter number of elements in alphabet : 2
For state 0
Enter no. of output states for input a : 1
Enter output states :
1
```



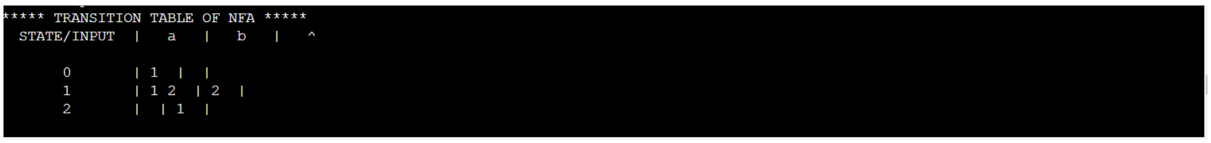
```
1
Enter no. of output states for input b : 0
Enter output states :
Enter no. of output states for input ^ : 0
Enter output states :
For state 1
Enter no. of output states for input a : 2
Enter output states :
```



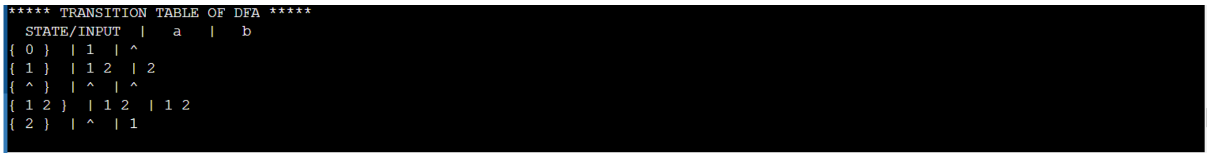
```
1 2
Enter no. of output states for input b : 1
Enter output states :
2
Enter no. of output states for input ^ : 0
Enter output states :
For state 2
Enter no. of output states for input a : 0
```



```
Enter output states :
For state 2
Enter no. of output states for input a : 0
Enter output states :
Enter no. of output states for input b : 1
Enter output states :
1
Enter no. of output states for input ^ : 0
```



```
***** TRANSITION TABLE OF NFA *****
  STATE/INPUT  |   a   |   b   |   ^

      0        | 1  |  |
      1        | 1 2  | 2  |
      2        |  | 1  |
```



```
***** TRANSITION TABLE OF DFA *****
  STATE/INPUT  |   a   |   b
{ 0 }  | 1  | ^
{ 1 }  | 1 2  | 2
{ ^ }  | ^  | ^
{ 1 2 }  | 1 2  | 1 2
{ 2 }  | ^  | 1
```

OBSERVATION :

Dynamic inputs were taken from the user regarding the number of states in NFA and the input symbols. The output states were noted for each input state and corresponding input symbol. Based on the code written , epsilon closures were found and the NFA states were converted into their equivalent DFA.

RESULT :

Thus we have successfully executed a C++ program to convert NFA to DFA and displayed the transition table.