Exercise – 3: Construct the DFA from the given NFA.

Aim: To convert the given NFA into a DFA (not a optimized one).

Theory:

Non-deterministic automata is a finite state machine which may make transitions to two different states for an event causing decision making for the transition of states difficult. It also has ϵ-transitions. In order to design the lexical analyzer, first the regular expressions are convertered into NFA and NFA has to be converted into DFA. The resultant DFA will have unique transitions and will not have ϵ-transitions.

Steps:

1. Read the NFA transition table available from the previous exercises or we can create our own transition table by working out from the given RE
2. Initialize all the required data structures for
    a. NFA transition table
    b. DFA transition table
    c. Set of States (SOS)
3. Start with the initial state of NFA
4. Find the Closure (may give more than one state and hence we call it as Set of State (SOS) of this initial state to get the initial state of DFA
5. Find the transitions from each state of DFA for each operand and find the e-closure of this state. Make entry of the resultant set of states in the DFA transition table
6. Repeat step 5 till all the states (SOS) of DFA are processed for the possible transitions

Modules:

- Reading the transition of NFA and initializing the DFA transition table
    - `readNFA`
    - `initDFATables`
- Finding e-closure of a given state(s)
    - `eClosure`
- Move from one state of DFA to another state of DFA for an operand
    - `move`
- Subset construction module which construct the sub of states of the NFA states, which are labelled to be the states of DFA
    - `constructSubset`
- Other modules are for reading the transition table of NFA, checking the duplicates of the set of states, sorting the set of states, displaying the table etc

Algorithm:

For the details of the modules *constructSubset*, *eClosure*, and *move*, refer to the algorithms given in figure 3.26.

Data Structures and Prototype Declarations:

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#define UNCHECKED 0
#define CHECKED 1
#define EMPTYSTACK -1
```

```
#define MAXSTACKSIZE 50

#define NOOPERANDS 50
#define NOOPERATORS 50
class SOS
{
public:
	int *sos;
	int nos;
	SOS(int);
	void dispSOS();
};

class DFA
{
private:
	int maxDFAStates;
	int noDFAStates;
	int noNFAStates;
	int noSymbols;
	int nfaStartState;
	int nfaFinalState;
	int noOperands;
	char operands[NOOPERANDS];
	int **nfaTransTable;
	int **dfaTransTable;
	int *dfaStatesChecked;
public:
	DFA();
```

```
        SOS **dfaSOS;
        SOS* eClosure(SOS*);
        SOS* move(SOS* ss, char b);
        void constructSubset();
        int isPresent(int a[],int n,int x);
        void readNFA(FILE *fp);
        void initDFATables();
        int findOpdIndex(char r);
        void displayNFATransTable();
        void displayDFATransTable();
        SOS* eClosure(SOS*);
        int checkDFAStates(SOS *ss);
        int isSOSEqual(SOS *ss1, SOS *ss2);
        SOS *sort(SOS *ss);
};
```

// Source code listing

```
#include "decl.h"
#include "cstack.h"
#include "istack.h"
#include "dfa.h"
SOS::SOS(int ns)
{
        int i;
        nos = ns;
        sos = (int*)malloc(sizeof(long int)*nos);
        for(i=0;i<nos;i++)
                sos[i] = -1;
}
```

```
void SOS::dispSOS()
{
	int i;
	printf("[ ");
	for(i=0;i<nos;i++)
		if(sos[i] != -1)
			printf("%d ",sos[i]);
	printf("]");
}

DFA::DFA()
{
	noDFAStates = 0;
}

SOS* DFA::sort(SOS *ss)
{
	int i,j,t;
	for(i=0;i<ss->nos-1;i++)
		for(j=i+1;j<ss->nos;j++)
			if(ss->sos[i] > ss->sos[j])
			{
				t=ss->sos[i];
				ss->sos[i] = ss->sos[j];
				ss->sos[j]=t;
			}
	return ss;
}
```

```
int DFA::isSOSEqual(SOS *ss1, SOS *ss2)
{
	int i;
	for(i=0;i<ss1->nos;i++)
		if(ss1->sos[i] != ss2->sos[i])
			return 0;
	return 1;
}

int DFA::checkDFAStates(SOS *ss)
{
	int i;
	for(i=0;i<noDFAStates;i++)
		if(isSOSEqual(ss,dfaSOS[i]))
			return i;
	return -1;
}

void DFA::constructSubset()
{
	SOS *ss0, *ss, *ss1,*ss2;
	char b;
	int dfaState,opdIdx, retState,i;
	noDFAStates = 0;
	ss0 = new SOS (noNFAStates);
	ss0->sos[0]= nfaStartState;
	ss = eClosure(ss0);
	dfaSOS[noDFAStates++]  = ss;
```

```
for(dfaState = 0;dfaState < noDFAStates;dfaState++)
{
	ss = dfaSOS[dfaState];
	if(dfaStatesChecked[dfaState] == UNCHECKED)
	{
		for(opdIdx=0;opdIdx < noOperands;opdIdx++)
		{
			b=operands[opdIdx];
			ss1=move(ss,b);
			ss2 = eClosure(ss1);
			if(ss2->nos == 0)
			{
				dfaTransTable[dfaState][opdIdx]=-1;
				continue;
			}
			retState = checkDFAStates(ss2);

			if(retState == -1)
			{
				dfaSOS[noDFAStates++]=ss2;
				dfaTransTable[dfaState][opdIdx]=noDFAStates-1;

			}
			else
				dfaTransTable[dfaState][opdIdx]=retState;

		}
		dfaStatesChecked[dfaState] = CHECKED;
```

```
            }

      }
}

SOS* DFA::eClosure(SOS* ss)
{
      IStack s;
      SOS *tss;
      int nos;
      int p,q;
      int i,stCount=0;
      //Push all the states onto stack
      nos = ss->nos;
      tss= new SOS(nos);
      for(i=0;i<ss->nos;i++)
            if(ss->sos[i] != -1)
            {
                  s.push(ss->sos[i]);
            }

      while(!(s.checkempty()))
      {

            q=s.topAndPop();
            tss->sos[stCount++]=q;
            p=nfaTransTable[q][noOperands];   //to get epsilon1

            if(p != -1 && (!isPresent(tss->sos,stCount, p) ) )
```

```
                  s.push(p);
            p=nfaTransTable[q][noOperands+1];   //to get epsilon2

            if(p != -1 && (!isPresent(tss->sos,stCount, p) ) )
                  s.push(p);
      }
      tss->nos=stCount;
      return sort(tss);
}

SOS* DFA::move(SOS *ss, char b)
{
      SOS *ss1, *ss2;     //ss, ss1 are the set of states, with nos as the
number of states

      int i,st=-1,k=0,state=-1;
      int opdIdx;
      int nos;
      nos = ss->nos;
      ss1 = new SOS(nos);
      ss2 = new SOS(nos);
      opdIdx = findOpdIndex(b);
      if(opdIdx != -1)
            for(i=0;i < nos;i++)
            {
                  state = ss->sos[i];
                  st = nfaTransTable[state][opdIdx];
                  if(st != -1 && (!isPresent(ss1->sos,k, st) ) )  //Check for
duplicates
```

```
                {

                        ss1->sos[k] = st;
                        k++;
                }
            }
      return ss1;
}

int DFA::isPresent(int a[],int n,int x)
{
      int i;
      for(i=0;i<n;i++)
            if(a[i] == x)
                  return 1;
      return 0;
}
void DFA::displayDFATransTable()
{
      int i,j,k;

      printf("\n\n\tResultant DFA\n\n");
      printf("-------------------------\n");

      printf("State\t");
      for(i=0;i<noOperands;i++)
            printf("\t%c ",operands[i]);
      printf("\n------------------------\n");
      for(i=0;i<noDFAStates;i++)
```

```
	{
		dfaSOS[i]->dispSOS();
		for(j=0;j<noOperands;j++)
		{
			if(dfaTransTable[i][j] != -1)
				printf("\t%d ",dfaTransTable[i][j]);
			else
				printf("\t-  ");
		}
		printf("\n");
	}
	printf("-------------------------\n");
}

void DFA::readNFA(FILE *fp)
{
	int i,j;
	char ch;
	fscanf(fp,"%d%d%d%d",&noNFAStates,&noSymbols,&nfaStartState,&nfaFinalSt
ate);
	noOperands = noSymbols-2;
	maxDFAStates = noNFAStates;

	for(i=0;i<noOperands;i++)
	{
		fscanf(fp," %c",&operands[i]);
		//printf("%c\t",operands[i]);
	}
```

```
	nfaTransTable=(int**)malloc(noNFAStates*sizeof(long int));
	for(i=0;i<noNFAStates;i++)
	{
		nfaTransTable[i]=(int*)malloc(noSymbols*sizeof(long int));
	}
	for(i=0;i<noNFAStates;i++)
	{
		for(j=0;j<noSymbols;j++)
				fscanf(fp,"%d",&nfaTransTable[i][j]);
	}
}

void DFA::displayNFATransTable()
{
	int i,j;
	printf("\n\n\tGiven e-NFA\n\n");
	printf("--------------------------------\n");
	printf("State\t");
	for(i=0;i<noOperands;i++)
		printf("%c   ",operands[i]);
	printf("eps1  eps2\n\n");
	printf("--------------------------------\n");
	for(i=0;i<noNFAStates;i++)
	{
	  printf("%d\t",i);
		for(j=0;j<noSymbols;j++)
			if(nfaTransTable[i][j]==-1)
		   printf("-\t");
		else
```

```
                printf("%d\t",nfaTransTable[i][j]);
            printf("\n");
        }
        printf("--------------------------------\n");
}

void DFA::initDFATables()
{
        int i,j;

        SOS *ss;
        dfaTransTable=(int**)malloc(maxDFAStates*sizeof(long int));
        dfaStatesChecked = (int *)malloc(maxDFAStates*sizeof(long int));
        for(i=0;i<maxDFAStates;i++)
        {
            dfaTransTable[i]=(int*)malloc(noOperands*sizeof(long int));
            dfaStatesChecked[i] = UNCHECKED;
        }

        for(i=0;i<maxDFAStates;i++)
        {
            for(j=0;j<noOperands;j++)
            {
                dfaTransTable[i][j]=-1;
            }
        }
        dfaSOS=(SOS**)malloc(maxDFAStates*sizeof(long int));

        for (i=0;i<maxDFAStates;i++)
```

```
	{
		dfaSOS[i] = new SOS (noNFAStates);
	}
}

//Find the index of the operand

int DFA::findOpdIndex(char r)
{
	int o;
	for(o=0;o<noOperands;o++)
	{
		if(r==operands[o])
			return (o);
	}
	return -1;
}
```

Output

Case 1: output for the NFA table obtained from RE (a|b)

Given e-NFA

```
-------------------------------
State a   b   eps1  eps2


-------------------------------
0     1     -     -     -
1     -     -     5     -
2     -     3     -     -
```

```
3     -     -     5     -
4     -     -     0     2
5     -     -     -     -
--------------------------------
```

Resultant DFA

```
-------------------------
State       a     b
------------------------
[ 0 2 4 ]   1     2
[ 1 5 ]     -     -
[ 3 5 ]     -     -
```

(- indicates that there are no transitions)

Case 2: output for the NFA table obtained from RE (a|b)

```
Given e-NFA
--------------------------------
State a   b   eps1  eps2
--------------------------------
0     1     -     -     -
1     -     -     2     -
2     -     3     -     -
3     -     -     -     -
--------------------------------
```

Resultant DFA

```
-------------------------
State       a     b
-----------------------
[ 0 ]       1     -
[ 1 2 ]     -     2
[ 3 ]       -     -
-------------------------
(- indicates that there are no transitions)
```

Exercises for the students:

1. Verify the output for the given NFA.
2. Use this DFA transition table for recognizning simple words.
3. Use this DFA to recognize the tokens of various control statements such as if, goto, while, for, etc.
4. Use this DFA to identify the token representing the Integers, Real Numbers and Numbers with Exponents.