Exercise – 2: Find the non-deterministic finite automata from the regular expression given in Postfix form.

Aim: To construct the regular expression into a non-deterministic finite automata and display the transitions table.

Theory:

Any lexical specification can be given in a regular expression. In order to separate token, the token recognizer has to be designed which starts from conversion of regular expression into NFA.The output of this exercise can be used as the input for the next exercise i.e the conversion of NFA into DFA.

Steps:

1. Initialize the transition table
2. Read the regular expression in postfix form and allocate space for the transitions table
3. Develop the code for the five operations such as:
    - concatenation (.)
    - Alternate operation (|)
    - Closure operations *, +, ?

Modules:

- Initialising the table (table is indexed by state number in row and operands and epsilon in column)
    - *initNFATransTable*

- Conversion of RE to NFA (make entries in the transition table for the given state and symbols)
    - *reg2nfa*
- Display the table
    - *displayTransTable*
- Storing the NFA transition table in the file for the next NFA to DFA conversion operation
    - *writeNFA*

Algorithm:

Refer to Figure 3.8 in section 3.4.2 describing the Thompson algorithm and the subsequent discussions with examples.

Source code development:

The *makefile* (project file) has the following files compiled and linked developed in a Linux environment.

```
scan:     scan.o nfa.o stack.o
          g++ scan.o nfa.o stack.o -o scan
scan.o:   scan.cc nfa.h stack.h decl.h
          g++ -c scan.cc
nfa.o:    nfa.cc nfa.h stack.h decl.h
          g++ -c nfa.cc
stack.o:  stack.cc stack.h decl.h
          g++ -c stack.cc
```

Data Structure and Prototype Declarations

```
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
typedef char ElementType;
#define NOOPERANDS 50
#define NOOPERATORS 50
typedef long int Pointer;
class NFA
{
private:
      int startState;
      int finalState;
      int noOperands;
      int noSymbols;
      int noOperators;
      int maxStates;
      int noStates;
      int count;
      ElementType operators[NOOPERATORS];
      ElementType operands[NOOPERANDS];
      int **transTable;

      void initNFATransTable(int rows,int cols);
      int checkOperator(ElementType element);
```

```
        int isMember(ElementType e);
        void findNoChars(ElementType reg[]);
        int findOpdIndex(ElementType r,ElementType reg[]);
public:
        NFA();
        void reg2nfa(ElementType reg[]);
        void displayTransTable();
        void writeNFA(FILE *fp);

};
```

Source code for the implementation of the RE to NFA conversion operations

```
NFA::NFA()
{
        int i,j,n;
        startState=0;
        finalState=0;
        noOperands=0;
        noSymbols=0;
        noOperators=0;
        maxStates=0;
        noStates=0;
        count=0;
        for(i=0;i<NOOPERATORS;i++)
                operators[i] = ' ';
        for(i=0;i<NOOPERANDS;i++)
                operands[i] = ' ';
}
```

```
int NFA::checkOperator(ElementType element)
{
      int flag=0;

      int k;
      switch(element)
      {
         case '*':
         case '+':
         case '?':
              return 1;

         case '.':
         case '|':
              return 2;
         default: return 0;
     }
}

int NFA::isMember(ElementType e)
{
      int i;
      for(i=0;i<noOperands;i++)
            if(operands[i] == e)
            {
                  printf("%c %c\n",operands[i],e);
                  return 1;
            }
```

```
	return 0;
}

void NFA::findNoChars(ElementType reg[])
{
	int i=0;
	for(i=0;reg[i]!='\0';i++)
	{
		if(reg[i]>='a'&&reg[i]<='z')
		{
			if(!isMember(reg[i]))
			{
				operands[noOperands]=reg[i];
					noOperands++;
			}
			count++;
		}
		else if(reg[i]=='*'||reg[i]=='|'||reg[i]=='+'||reg[i]=='?')
		{
			operators[noOperators];
				noOperators++;
			count++;
		}
	}
}

void NFA::initNFATransTable(int rows, int cols)
{
	int i,j,n;
```

```
	transTable=(int**)malloc(rows*sizeof(Pointer));

	for(n=0;n<rows;n++)
	{
		transTable[n]=(int*)malloc(cols*sizeof(Pointer));
	}

	for(i=0;i<rows;i++)
	{
		for(j=0;j<cols;j++)
		{

			transTable[i][j]=-1;

		}
	}
}

void NFA::displayTransTable()
{
	int i,j;
	printf("\n\n\tResultant e-NFA\n\n");
	printf("--------------------------------\n");
	printf("State\t");
	for(i=0;i<noSymbols;i++)
		printf("%c   ",operands[i]);
	printf("eps1  eps2\n\n");
	printf("--------------------------------\n");
```

```
	for(i=0;i<noStates;i++)
	{
	  printf("%d\t",i);
		for(j=0;j<noSymbols;j++)
			if(transTable[i][j]==-1)
			   printf("-\t");
			else
			   printf("%d\t",transTable[i][j]);
		printf("\n");
	}
	printf("--------------------------------\n");
}

void NFA::writeNFA(FILE *fp)
{
	int i,j;
	fprintf(fp,"%d\t%d\t%d\t%d\t",noStates,noSymbols,startState,finalState)
;
	for(i=0;i<noOperands;i++)
		fprintf(fp,"   %c\t",operands[i]);
	fprintf(fp,"\n");
	for(i=0;i<noStates;i++)
	{
		//printf("%d:  ");
		for(j=0;j<noSymbols;j++)
				fprintf(fp," %d\t",transTable[i][j]);
		fprintf(fp,"\n");
	}
}
```

```
int NFA::findOpdIndex(ElementType r,ElementType reg[])
{
	int o,x;
	//x=noofoperands(reg);
	for(o=0;o<noOperands;o++)
	{
		if(r==operands[o])
			return (o);
	}
	return -1;
}


void NFA::reg2nfa(ElementType reg[])
{
	int index=-1,opdIndex=1;
	int
i=0,secondStartState,secondAcceptState,firstAcceptState,firstStartState;
	Stack s;
	#define fe noOperands
	#define se noOperands+1

	findNoChars(reg);
	maxStates=2*count;
	noSymbols = noOperands+2;  //to include epsilon

	initNFATransTable(maxStates,noSymbols);
```

```
s.createStack(2*count);


for(i=0;reg[i]!='\0';i++)
{
	if(checkOperator(reg[i])==0)  //operand
	{
		index++;
		startState=index;
		s.push(index);
		opdIndex=findOpdIndex(reg[i],reg);
		transTable[index][opdIndex]=index+1;
		index++;
		finalState=index;
		s.push(index);
	}
	else if(reg[i]=='|')
	{
		secondAcceptState=s.topAndPop();//SECOND ACCEPTING
		secondStartState=s.topAndPop();//SECOND START
		firstAcceptState=s.topAndPop();//FIRST ACCEPTING
		firstStartState=s.topAndPop();//FIRST START
			index++;
		startState=index;  //
		s.push(index);
		transTable[index][fe]=firstStartState;
		transTable[index][se]=secondStartState;
		index++;
		finalState=index;   //
		transTable[firstAcceptState][fe]=index;
```

```
        transTable[secondAcceptState][fe]=index;
        s.push(index);
    }
    else if(reg[i]=='.')
    {
        secondAcceptState=s.topAndPop();
        secondStartState=s.topAndPop();
        firstAcceptState=s.topAndPop();
        firstStartState=s.topAndPop();
        startState = firstStartState;  //
        transTable[firstAcceptState][fe]=secondStartState;
        s.push(firstStartState);
        s.push(secondAcceptState);
    }
    else if(reg[i]=='*')
    {
        index++;
        startState=index;   //
        firstAcceptState=s.topAndPop();
        firstStartState=s.topAndPop();
        s.push(index);
        transTable[index][se]=firstStartState;
        transTable[index][fe]=++index;
        finalState=index;   //
        transTable[firstAcceptState][fe]=firstStartState;
        transTable[firstAcceptState][se]=index;
        s.push(index);
    }
    else if(reg[i]=='+')
```

```
	{
		firstAcceptState=s.topAndPop();
		firstStartState=s.topAndPop();
		index++;
		startState = index;//
		transTable[index][fe]=firstStartState;
		s.push(index);
		index++;
		finalState = index;//
		s.push(index);
		transTable[firstAcceptState][fe]=index;
		transTable[firstAcceptState][se]=firstStartState;
	}
	else if(reg[i]=='?')
	{
		firstAcceptState=s.topAndPop();
		firstStartState=s.topAndPop();
		index++;
		startState=index; //
		s.push(index);
		transTable[index][fe]=firstStartState;
		transTable[index][se]=++index;
		finalState=index;
		transTable[firstAcceptState][fe]=index;
		s.push(index);
	}

}
noStates = index+1;
```

```
}
```

Main function calling the routines for the conversion of regular expression into NFA

```
#include "decl.h"
#include "stack.h"
#include "nfa.h"
int main()
{
	NFA nfa;
	char reg[50];
	int nofc,nfod;
	int st,ft;
	FILE *fp;
	char fname[20];
	printf("Enter Regular expression in Post fix form\n");
	scanf("%s",reg);
	printf("Reg expression is: %s\n",reg);

	nfa.reg2nfa(reg);
	nfa.displayTransTable();

	printf("Enter the file name for storing NFA states\n");
	scanf("%s",fname);
	fp=fopen(fname,"w");
	if(NULL == fp)
	{
		printf("%s:Error in file open.. Could not write NFA into
file\n",fname);
```

```
            exit(-1);
      }
      nfa.writeNFA(fp);
      return 1;

}
```

Output

Case 1: Output for the regular expression a|b

```
The following numbers indicates the total number of NFA states, number of symbols, start state, final state, the operands

6    4  4  5  a   b
```

Transition Table

NFA	a	b	ϵ1	ϵ2
0	1	-1	-1	-1
1	-1	-1	5	-1
2	-1	3	-1	-1
3	-1	-1	5	-1
4	-1	-1	0	2
5	-1	-1	-1	-1

(-1 indicates no transition)

Case 1: Output for the regular expression ab

```
4    4    3  a   b
```

Transition Table

NFA	a	b	ϵ1	ϵ2
0	1	-1	-1	-1
1	-1	-1	2	-1
2	-1	3	-1	-1
3	-1	-1	-1	-1

Exerice for the students:

Use this program and verify the transition table for the regular expression (a|b)abb .