ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

# COMPILER DESIGN LAB

## *EXP 2 - RE to NFA*

**AIM :**

To convert a given regular expression into its non deterministic finite automata representation using C++

**REQUIREMENTS :**

1. Knowledge of languages and regular expression
2. Knowledge of NFA and its state diagram
3. Online GDB compiler to execute and implement code

**THEORY :**

Regular Expression :

A regular expression is a sequence of characters that specifies a search pattern in text. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation

Non Deterministic Finite Automata :

In NFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In

other words, the exact state to which the machine moves cannot be determined. Hence, it is called Non-deterministic Automaton. As it has a finite number of states, the machine is called Non-deterministic Finite Machine or Non-deterministic Finite Automaton.
An NFA is represented by digraphs called state diagrams.

- The vertices represent the states.
- The arcs labelled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

**ALGORITHM :**

**STEP 1 :** A structure is created for each kind of regular expression that we want to create into NDFA.
**STEP 2 :** Under each structure , we define 3 variables - start state , alphabet input and n-state.
**STEP 3 :** Each state changes into another state based on the alphabet input symbol . All these are defined in the structures
**STEP 4 :** A switch case is made with 4 choices (4 different regular expressions) and based on the choice selected , an NFA diagram is shown as output.

**SOURCE CODE :**

```
#include<bits/stdc++.h>
```

```
#include<stdio.h>
#include<conio.h>
using namespace std;
struct node
{
char start;
char alp;
node *nstate;
}*p,*p1,*p2,*p3,*p4,*p5,*p6,*p7,*p8;
char e='e';
void disp();
void re1()
{
p1=new(node);
p2=new(node);
p3=new(node);
p4=new(node);
p1->start='0';
p1->alp='e';
p1->nstate=p2;
p2->start='1';
p2->alp='a';
p2->nstate=p3;
p3->start='2';
p3->alp='e';
p3->nstate=p4;
p4->start='3';
p4->nstate=NULL;
disp();
getch();
}
void re2()
{
p1=new(node);
p2=new(node);
p3=new(node);
p4=new(node);
```

```
p5=new(node);
p6=new(node);
p7=new(node);
p8=new(node);
p1->start='0';
p1->alp='e';
p1->nstate=p2;
p2->start='1';
p2->alp='a';
p2->nstate=p3;
p3->start='2';
p3->alp='e';
p3->nstate=p4;
p4->start='5';
p4->alp=' ';
p4->nstate=p5;
p5->start='0';
p5->alp='e';
p5->nstate=p6;
p6->start='3';
p6->alp='b';
p6->nstate=p7;
p7->start='4';
p7->alp='e';
p7->nstate=p8;
p8->start='5';
p8->alp=' ';
p8->nstate=NULL;
disp();
getch();
}
void re3()
{
p1=new(node);
p2=new(node);
p3=new(node);
p1->start='0';
```

```
p1->alp='a';
p1->nstate=p2;
p2->start='1';
p2->alp='b';
p2->nstate=p3;
p3->start='2';
p3->alp=' ';
p3->nstate=NULL;
disp();
getch();
}
void re4()
{
p1=new(node);
p2=new(node);
p3=new(node);
p4=new(node);
p5=new(node);
p6=new(node);
p7=new(node);
p8=new(node);
p1->start='0';
p1->alp='e';
p1->nstate=p2;
p2->start='1';
p2->alp='a';
p2->nstate=p3;
p3->start='2';
p3->alp='e';
p3->nstate=p4;
p4->start='3';
p4->alp=' ';
p4->nstate=p5;
p5->start='0';
p5->alp='e';
p5->nstate=p6;
p6->start='3';
```

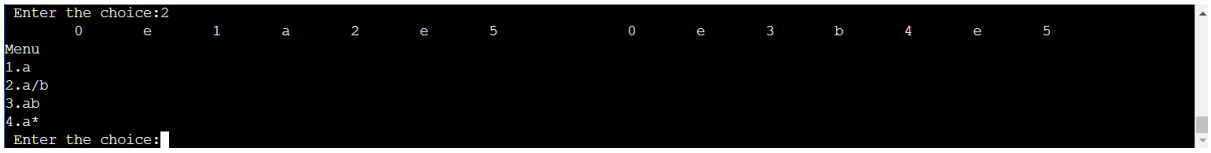
```
p6->alp=' ';
p6->nstate=p7;
p7->start='2';
p7->alp='e';
p7->nstate=p8;
p8->start='1';
p8->alp=' ';
p8->nstate=NULL;
disp();
getch();
}
void disp()
{
p=p1;
while(p!=NULL)
{
cout<<"\t"<<p->start<<"\t"<<p->alp;
p=p->nstate;
}
}
int main()
{
p=new(node);
int ch=1;
while(ch!=0)
{
cout<<"\nMenu"<<"\n1.a"<<"\n2.a/b"<<"\n3.ab"<<"\n4.a*";
cout<<"\n Enter the choice:";
cin>>ch;
switch(ch)
{
case 1:
{
re1();
break;
}
case 2:
```

```
{
re2();
break;
}
case 3:
{
re3();
break;
}
case 4:
{
re4();
break;
}
default:
{
exit(0);
}
}
}
}
return 0;
}
```

## SCREENSHOT OF OUTPUTS :

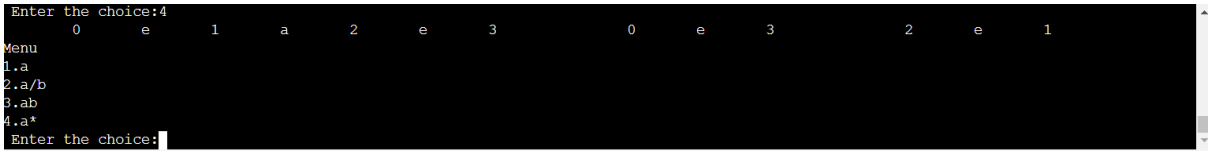
1. a


2. a/b




3. ab


4. a*




**OBSERVATION :**

All of the switch case choices were selected and for each of the regular expressions , its equivalent NFA diagram was shown in the console . Output verification process was completed.

**RESULT :**

Thus we have successfully converted the given REs into NFA using C++.