Compiler Design Lab

NAME: Priyanka Srinivas
REGISTER NUMBER: RA1911026010014
SECTION: K1

Exercise 2: Conversion of a regular expression to NFA

AIM: To write a program to implement the conversion of a regular expression to NFA

ALGORITHM:

1. Start the program.
2. Take the input expression from the user.
3. Create three different variables to store the NFA expression, postfix and display at every transition step.
4. Create separate parts to check for different operators.
5. Initialize different cases to deal with the different operators.
   A. For '.', '*' and '+' initialize a different method using stack.
6. Collect the output expression in NFA.
7. Display the output.
8. Stop the program.

PROGRAM:

```
#include <iostream>
#include <stack>
#include <string>
#include <algorithm>
#include <vector>
using namespace std;

class node
{
     public:
          char input;
          int to;
          node *next;
};

int prec(char c)
{
```

```
     if (c == '*')
          return 3;
     else if (c == '.')
          return 2;
     else if (c == '+')
          return 1;
     else
          return -1;
}

string post(string s)
{
     stack<char> st;
     st.push('N');
     int l = s.length();
     string ns;
     for (int i = 0; i < l; i++)
     {
          if((s[i]>='a' && s[i]<='z') || (s[i]>='A' && s[i]<='Z'))
               ns += s[i];
          else if(s[i]=='(')
               st.push('(');
          else if (s[i] == ')')
          {
               while (st.top() != 'N' && st.top() != '(')
               {
                    char c = st.top();
                    st.pop();
                    ns += c;
               }
               if(st.top()=='(')
               {
                    char c = st.top();
                    st.pop();
               }
          }
          else
          {
               while(st.top()!='N' && prec(s[i])<=prec(st.top()))
               {
                    char c = st.top();
                    st.pop();
                    ns += c;
               }
               st.push(s[i]);
          }
```

```
	}
	while (st.top() != 'N')
	{
		char c = st.top();
		st.pop();
		ns += c;
	}
	return ns;
}

void printnode(vector<node *> v)
{
	cout << "_____________________________________________" << endl;
	cout << "| from state\t| input\t| tostates" << endl;
	for (int i = 0; i < v.size(); i++)
	{
		cout << "| " << i << "          \t|";
		node *head = v[i];
		cout << head->input;
		bool first = true;
		while (head != NULL)
		{
			if (first)
			{
				cout << "     \t|";
				first = false;
			}
			else
			{
				cout << "     \t";
			}
			cout << head->to;
			head = head->next;
		}
		cout << endl;
		// cout<<"\t\t\t\t\t\t|"<<endl;
	}
	cout << "_____________________________________________" << endl;
}

node *makenode(char in)
{
	node *a = new node;
	a->input = in;
	a->to = -1;
	a->next = NULL;
```

```
	return a;
}

node *copynode(node *a)
{
	node *b = new node;
	b->input = -1;
	b->to = -1;
	b->next = NULL;
	return b;
}

void andd(vector<node *> &v, vector<vector<int> > &st)
{
	int x, y;
	int first, last1;
	y = st[st.size() - 1][0];
	x = st[st.size() - 2][1];
	first = st[st.size() - 2][0];
	last1 = st[st.size() - 1][1];

	st.pop_back();
	st.pop_back();

	vector<int> ptemp;
	ptemp.push_back(first);
	ptemp.push_back(last1);
	st.push_back(ptemp);

	node *last = v[y];
	node *lnode = v[x];
	node *temp = copynode(last);
	// temp->to = -1;
	while (lnode->next != NULL)
	{
		lnode = lnode->next;
	}
	lnode->next = temp;
	lnode->to = y;
}

void orr(vector<node *> &v, vector<vector<int> > &st)
{
	int x, y, x1, y1;
	x = st[st.size() - 2][0];
	y = st[st.size() - 1][0];
```

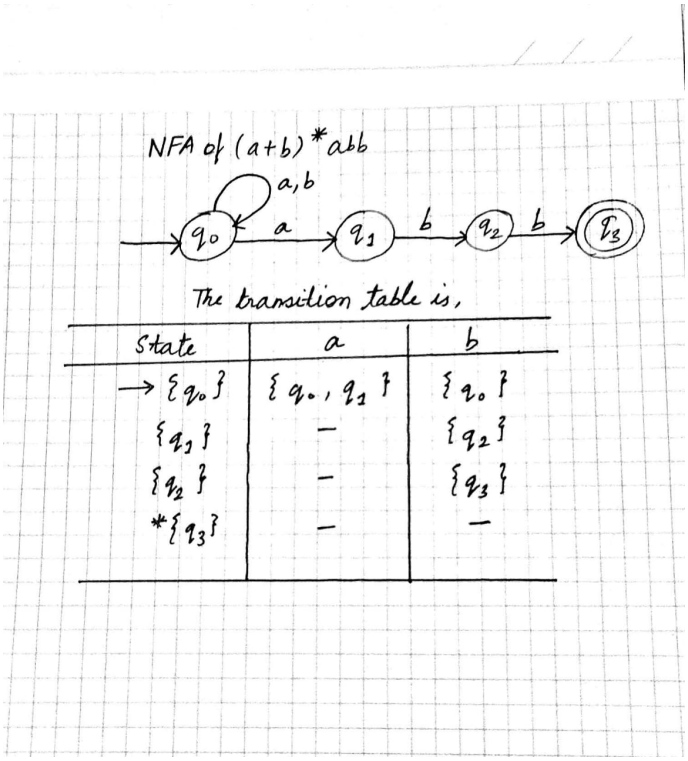
```
x1 = st[st.size() - 2][1];
y1 = st[st.size() - 1][1];
node *start = makenode('e');
node *end = makenode('e');
v.push_back(start);
int firstnode = v.size() - 1;
v.push_back(end);
int endnode = v.size() - 1;

st.pop_back();
st.pop_back();

vector<int> ptemp;
ptemp.push_back(firstnode);
ptemp.push_back(endnode);
st.push_back(ptemp);

for (int i = 0; i < v.size() - 2; i++)
{
     node *h = v[i];
     while (h->next != NULL)
     {
          if (h->to == x || h->to == y)
          {
               h->to = firstnode;
          }
          h = h->next;
     }
}

node *temp = copynode(v[x]);
node *temp1 = copynode(v[y]);
node *t = v[firstnode];
while (t->next != NULL)
{
     t = t->next;
}
t->to = x;
t->next = temp;
t->next->to = y;
t->next->next = temp1;

node *adlink = v[x1];
while (adlink->next != NULL)
{
     adlink = adlink->next;
```

```
	}

	adlink->to = endnode;
	adlink->next = copynode(end);

	node *adlink1 = v[y1];
	while (adlink1->next != NULL)
	{
		adlink1 = adlink1->next;
	}
	adlink1->to = endnode;
	adlink1->next = copynode(end);
}

void closure(vector<node *> &v, vector<vector<int> > &st)
{
	int x, x1;
	x = st[st.size() - 1][0];
	x1 = st[st.size() - 1][1];
	node *s = makenode('e');
	// node* e = makenode('e');
	v.push_back(s);
	int firstnode = v.size() - 1;
	// v.push_back(e);
	// int endnode = v.size() -1;
	st.pop_back();
	vector<int> ptemp;
	ptemp.push_back(x);
	ptemp.push_back(firstnode);
	st.push_back(ptemp);

	for (int i = 0; i < v.size() - 2; i++)
	{
		node *h = v[i];
		while (h->next != NULL)
		{
			if (h->to == x)
			{
				h->to = firstnode;
			}
			h = h->next;
		}
	}

	node *t = v[x1];
	while (t->next != NULL)
```

```
	{
		t = t->next;
	}
	t->to = x;
	t->next = copynode(t);
	t->next->to = firstnode;
	t->next->next = copynode(s);
}

int main()
{
	string in;
	cout << "Enter a regular expression\n";
	cin >> in;
	string o;
	vector<node *> v;
	o = post(in);
	cout << "\npostfix expression is " << o << endl;
	vector<vector<int> > st;
	int firstnode = 0;
	for (int l = 0; l < o.length(); l++)
	{
		if (o[l] != '+' && o[l] != '*' && o[l] != '.')
		{
			node *temp = makenode(o[l]);
			v.push_back(temp);
			vector<int> ptemp;
			ptemp.push_back(v.size() - 1);
			ptemp.push_back(v.size() - 1);
			st.push_back(ptemp);
		}
		else if (o[l] == '.')
		{
			andd(v, st);
		}
		else if (o[l] == '+')
		{
			orr(v, st);
		}
		else if (o[l] == '*')
		{
			closure(v, st);
		}
	}
	cout << "\ntrainsition table for given regular expression is -
\n";
```

```
    printnode(v);
    cout << endl;
    cout << "starting node is ";
    cout << st[st.size() - 1][0] << endl;
    cout << "ending node is ";
    cout << st[st.size() - 1][1] << endl;
    return 0;
}
```

TRANSITION TABLE:



The transition table is,

State	a	b
→ {q0}	{q0, q1}	{q0}
{q1}	–	{q2}
{q2}	–	{q3}
*{q3}	–	–

OUTPUT:

```
C:\Users\priya\Desktop\Coding\compiler-design-lab\lab-2>output.exe
Enter a regular expression
(a+b)*abb

postfix expression is ab+abb*

trainsition table for given regular expression is -
_____________________________________________
| from state    | input | tostates
| 0             |a      |3      -1
| 1             |b      |3      -1
| 2             |e      |0      1       -1
| 3             |e      |-1
| 4             |a      |-1
| 5             |b      |-1
| 6             |b      |6      7       -1
| 7             |e      |-1
_____________________________________________

starting node is 6
ending node is 7
```

RESULT:

A regular expression was converted to NFA and verified successfully using transition table.