**PROGRAM – 3**

**AIM :** WAP to convert Regular Expression to NFA . Output Transition table .

**PROGRAM :**

#include <bits/stdc++.h>

using namespace std ;

vector<vector<string> > v ;

int state = -1 ;

string intTostring(int num) {

stringstream ss ;

ss << num ;

return ss.str() ;

}

int getPrecedence(char ch) {

if(ch == '|')

return 1 ;

if(ch == '.')

return 2 ;

if(ch == '\*')

return 3 ;

}

string convertToPostfix(string str) {

vector<char> pre\_op ;

pre\_op . push\_back('|') ;

pre\_op . push\_back('.') ;

pre\_op . push\_back('\*') ;

string postStr = "" ;

vector<char> ops ;

for(int i = 0 ; i < str . length() ; i ++) {

if(str[i] == 'a' || str[i] == 'b')

postStr = postStr + string(1 , str[i]) ;

if(str[i] == '|' || str[i] == '\*' || str[i] == '.') {

if(ops . empty()) {

ops . push\_back(str[i]) ;

} else {

if(getPrecedence(str[i]) > getPrecedence(ops[ops . size() - 1])) {

ops . push\_back(str[i]) ;

} else {

while(getPrecedence(str[i]) <= getPrecedence(ops[ops . size() - 1])) {

char op = ops[ops . size() - 1] ;

ops . pop\_back() ;

postStr = postStr + string(1 , op) ;

}

ops . push\_back(str[i]) ;

}

}

}

}

while(! ops . empty()) {

char op = ops[ops . size() - 1] ;

ops . pop\_back() ;

postStr = postStr + string(1 , op) ;

}

return postStr ;

}

void addAorB(int &in , int &fi , char ch) {

vector<string> temp ;

in = state + 1 ;

fi = state + 2 ;

state += 2 ;

temp . push\_back(intTostring(in)) ;

temp . push\_back(intTostring(fi)) ;

temp . push\_back(string(1 , ch)) ;

v . push\_back(temp) ;

temp . clear() ;

}

void conc(int &in , int &fi , int in1 , int fi1 , int in2 , int fi2) {

in = in1 ;

fi = fi2 ;

vector<string> temp ;

temp . push\_back(intTostring(fi1)) ;

temp . push\_back(intTostring(in2)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

}

void uni(int &in , int &fi , int in1 , int fi1 , int in2 , int fi2) {

in = state + 1 ;

fi = state + 2 ;

state += 2 ;

vector<string> temp ;

temp . push\_back(intTostring(in)) ;

temp . push\_back(intTostring(in1)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

temp . push\_back(intTostring(in)) ;

temp . push\_back(intTostring(in2)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

temp . push\_back(intTostring(fi1)) ;

temp . push\_back(intTostring(fi)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

temp . push\_back(intTostring(fi2)) ;

temp . push\_back(intTostring(fi)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

}

void clo(int &in , int &fi , int in1 , int fi1) {

in = state + 1 ;

fi = state + 2 ;

state += 2 ;

vector<string> temp ;

temp . push\_back(intTostring(fi1)) ;

temp . push\_back(intTostring(in1)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

temp . push\_back(intTostring(in)) ;

temp . push\_back(intTostring(in1)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

temp . push\_back(intTostring(in)) ;

temp . push\_back(intTostring(fi)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

temp . push\_back(intTostring(fi1)) ;

temp . push\_back(intTostring(fi)) ;

temp . push\_back("E") ;

v . push\_back(temp) ;

temp . clear() ;

}

void solve(int &in , int &fi , string str) {

str = convertToPostfix(str) ;

vector<pair<int , int> > vec ;

for(int i = 0 ; i < str . length() ; i ++) {

if(str[i] == 'a' || str[i] == 'b') {

addAorB(in , fi , str[i]) ;

vec . push\_back(make\_pair(in , fi)) ;

} else {

if(str[i] == '.') {

int in2 = vec[vec . size() - 1] . first ;

int fi2 = vec[vec . size() - 1] . second ;

int in1 = vec[vec . size() - 2] . first ;

int fi1 = vec[vec . size() - 2] . second ;

vec . pop\_back() ;

vec . pop\_back() ;

conc(in , fi , in1 , fi1 , in2 , fi2) ;

vec . push\_back(make\_pair(in , fi)) ;

}

if(str[i] == '|') {

int in2 = vec[vec . size() - 1] . first ;

int fi2 = vec[vec . size() - 1] . second ;

int in1 = vec[vec . size() - 2] . first ;

int fi1 = vec[vec . size() - 2] . second ;

vec . pop\_back() ;

vec . pop\_back() ;

uni(in , fi , in1 , fi1 , in2 , fi2) ;

vec . push\_back(make\_pair(in , fi)) ;

}

if(str[i] == '\*') {

int in1 = vec[vec . size() - 1] . first ;

int fi1 = vec[vec . size() - 1] . second ;

vec . pop\_back() ;

clo(in , fi , in1 , fi1) ;

vec . push\_back(make\_pair(in , fi)) ;

}

}

}

}

void printSol() {

cout << "State\ta\tb\tE\n" ;

for(int i = 0 ; i < v . size() ; i ++) {

if(v[i][2] == "a") {

cout << v[i][0] << "\t" << v[i][1] << "\t\_\t\_\n" ;

}

if(v[i][2] == "b") {

cout << v[i][0] << "\t\_\t" << v[i][1] << "\t\_\n" ;

}

if(v[i][2] == "E") {

cout << v[i][0] << "\t\_\t\_\t" << v[i][1] << "\n" ;

}

}

}

int main() {

string str ;

input :

cout << "Input unparanthesised Regular Expression (R.E.) having only a b | . \* -\n" ;

cin >> str ;

bool valid = true ;

for(int i = 0 ; i < str . length() ; i ++) {

if(str[i] != 'a' && str[i] != 'b' && str[i] != '|' && str[i] != '.' && str[i] != '\*') {

cout << "Invalid input ... Please try again ...\n" ;

valid = false ;

break ;

}

}

if(! valid)

goto input ;

int in , fi ;

solve(in , fi , str) ;

cout << "\nTransition table of NFA represented by above R.E. -\n\n" ;

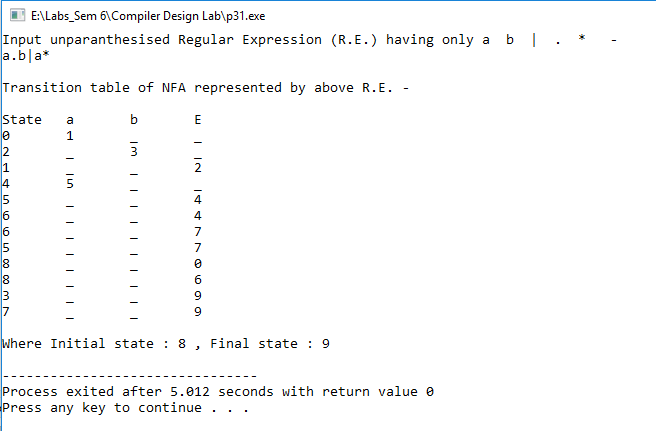
printSol() ;

cout << "\nWhere Initial state : " << in << " , Final state : " << fi << "\n" ;

return 0 ;

}

**Output :**

****