1. Implementation of a lexical Analyzer

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

using namespace std;

int isKeyword(char buffer[]){

char keywords[32][10] = {"auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef","union",

"unsigned","void","volatile","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main(){

char ch, buffer[15], operators[] = "+-\*/%=";

ifstream fin("program.txt");

int i,j=0;

if(!fin.is\_open()){

cout<<"error while opening the file\n";

exit(0);

}

while(!fin.eof()){

ch = fin.get();

for(i = 0; i < 6; ++i){

if(ch == operators[i])

cout<<ch<<" is operator\n";

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

cout<<buffer<<" is keyword\n";

else

cout<<buffer<<" is indentifier\n";

}

}

fin.close();

return 0;

}

2. Implementation of shift reduce parsing Algorithm

#include<stdio.h>

#include<string.h>

int k=0,z=0,i=0,j=0,c=0;

char a[16],ac[20],stk[15],act[10];

void check();

int main()

{

puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->id");

puts("enter input string ");

gets(a);

c=strlen(a);

strcpy(act,"SHIFT->");

puts("stack \t input \t action");

for(k=0,i=0; j<c; k++,i++,j++)

{

if(a[j]=='i' && a[j+1]=='d')

{

stk[i]=a[j];

stk[i+1]=a[j+1];

stk[i+2]='\0';

a[j]=' ';

a[j+1]=' ';

printf("\n$%s\t%s$\t%sid",stk,a,act);

check();

}

else

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n$%s\t%s$\t%ssymbols",stk,a,act);

check();

}

}

}

void check()

{

strcpy(ac,"REDUCE TO E");

for(z=0; z<c; z++)

if(stk[z]=='i' && stk[z+1]=='d')

{

stk[z]='E';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

j++;

}

for(z=0; z<c; z++)

if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

for(z=0; z<c; z++)

if(stk[z]=='E' && stk[z+1]=='\*' && stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

for(z=0; z<c; z++)

if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

}

// Input string: id+id\*id

3. Generation of a code for a given intermediate code

#include <stdio.h >

#include <stdio.h >

#include<conio.h>

#include <string.h >

int main() {

char icode[10][30], str[20], opr[10];

int i = 0;

// clrscr();

printf("\n Enter the set of intermediate code (terminated by exit):\n");

do

{

scanf("%s", icode[i]);

} while (strcmp(icode[i++], "exit") != 0);

printf("\n target code generation");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

i = 0;

do {

strcpy(str, icode[i]);

switch (str[3]) {

case '+':

strcpy(opr, "ADD ");

break;

case '-':

strcpy(opr, "SUB ");

break;

case '\*':

strcpy(opr, "MUL ");

break;

case '/':

strcpy(opr, "DIV ");

break;

}

printf("\n\tMov %c,R%d", str[2], i);

printf("\n\t%s%c,R%d", opr, str[4], i);

printf("\n\tMov R%d,%c", i, str[0]);

} while (strcmp(icode[++i], "exit") != 0);

getch();

}

///

a=a\*b

c=f\*h

g=a\*h

f=Q+w

t=q-j

exit

///

4. Simulation of Symbol Table

// C++ program to implement Symbol Table

#include <iostream>

using namespace std;

const int MAX = 100;

class Node {

string identifier, scope, type;

int lineNo;

Node\* next;

public:

Node()

{

next = NULL;

}

Node(string key, string value, string type, int lineNo)

{

this->identifier = key;

this->scope = value;

this->type = type;

this->lineNo = lineNo;

next = NULL;

}

void print()

{

cout << "Identifier's Name:" << identifier

<< "\nType:" << type

<< "\nScope: " << scope

<< "\nLine Number: " << lineNo << endl;

}

friend class SymbolTable;

};

class SymbolTable {

Node\* head[MAX];

public:

SymbolTable()

{

for (int i = 0; i < MAX; i++)

head[i] = NULL;

}

int hashf(string id); // hash function

bool insert(string id, string scope,

string Type, int lineno);

string find(string id);

bool deleteRecord(string id);

bool modify(string id, string scope,

string Type, int lineno);

};

// Function to modify an identifier

bool SymbolTable::modify(string id, string s,

string t, int l)

{

int index = hashf(id);

Node\* start = head[index];

if (start == NULL)

return "-1";

while (start != NULL) {

if (start->identifier == id) {

start->scope = s;

start->type = t;

start->lineNo = l;

return true;

}

start = start->next;

}

return false; // id not found

}

// Function to delete an identifier

bool SymbolTable::deleteRecord(string id)

{

int index = hashf(id);

Node\* tmp = head[index];

Node\* par = head[index];

// no identifier is present at that index

if (tmp == NULL) {

return false;

}

// only one identifier is present

if (tmp->identifier == id && tmp->next == NULL) {

tmp->next = NULL;

delete tmp;

return true;

}

while (tmp->identifier != id && tmp->next != NULL) {

par = tmp;

tmp = tmp->next;

}

if (tmp->identifier == id && tmp->next != NULL) {

par->next = tmp->next;

tmp->next = NULL;

delete tmp;

return true;

}

// delete at the end

else {

par->next = NULL;

tmp->next = NULL;

delete tmp;

return true;

}

return false;

}

// Function to find an identifier

string SymbolTable::find(string id)

{

int index = hashf(id);

Node\* start = head[index];

if (start == NULL)

return "-1";

while (start != NULL) {

if (start->identifier == id) {

start->print();

return start->scope;

}

start = start->next;

}

return "-1"; // not found

}

// Function to insert an identifier

bool SymbolTable::insert(string id, string scope,

string Type, int lineno)

{

int index = hashf(id);

Node\* p = new Node(id, scope, Type, lineno);

if (head[index] == NULL) {

head[index] = p;

cout << "\n"

<< id << " inserted";

return true;

}

else {

Node\* start = head[index];

while (start->next != NULL)

start = start->next;

start->next = p;

cout << "\n"

<< id << " inserted";

return true;

}

return false;

}

int SymbolTable::hashf(string id)

{

int asciiSum = 0;

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

asciiSum = asciiSum + id[i];

}

return (asciiSum % 100);

}

// Driver code

int main()

{

SymbolTable st;

string check;

cout << "\*\*\*\* SYMBOL\_TABLE \*\*\*\*\n";

// insert 'if'

if (st.insert("if", "local", "keyword", 4))

cout << " -successfully";

else

cout << "\nFailed to insert.\n";

// insert 'number'

if (st.insert("number", "global", "variable", 2))

cout << " -successfully\n\n";

else

cout << "\nFailed to insert\n";

// find 'if'

check = st.find("if");

if (check != "-1")

cout << "Identifier Is present\n";

else

cout << "\nIdentifier Not Present\n";

// delete 'if'

if (st.deleteRecord("if"))

cout << "if Identifier is deleted\n";

else

cout << "\nFailed to delete\n";

// modify 'number'

if (st.modify("number", "global", "variable", 3))

cout << "\nNumber Identifier updated\n";

// find and print 'number'

check = st.find("number");

if (check != "-1")

cout << "Identifier Is present\n";

else

cout << "\nIdentifier Not Present";

return 0;

}

5.Implementation of Operator precedence Parsing Algorithm

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#define node struct tree1

int getOperatorPosition(char);

int matrix[5][5]=

{ {1,0,0,1,1},

{1,1,0,1,1},

{0,0,0,2,3},

{1,1,3,1,1},

{0,0,0,3,2} };

int tos=-1;

void matrix\_value(void);

void show\_tree(node \*);

int isOperator(char);

struct tree1

{

char data;

node \*lptr;

node \*rptr;

}\*first;

struct opr

{

char op\_name;

node \*t;

}

oprate[50];

char cur\_op[5]={'+','\*','(',')','['};

char stack\_op[5]={'+','\*','(',')',']'};

int main()

{

char exp[10];

int ssm=0,row=0,col=0;

node \*temp;

// clrscr();

printf("enter exp:");

scanf("%s",exp);

matrix\_value();

while(exp[ssm]!='\0')

{

if(ssm==0)

{

tos++;

oprate[tos].op\_name=exp[tos];

}

else

{

if(isOperator(exp[ssm])==-1)

{

oprate[tos].t=(node\*)malloc(sizeof(node));

oprate[tos].t->data=exp[ssm];

oprate[tos].t->lptr='\0';

oprate[tos].t->rptr='\0';

}

else

{

row=getOperatorPosition(oprate[tos].op\_name);

col=getOperatorPosition(exp[ssm]);

if(matrix[row][col]==0)

{

tos++;

oprate[tos].op\_name=exp[ssm];

}

else if(matrix[row][col]==1)

{

temp=(node\*)malloc(sizeof(node));

temp->data=oprate[tos].op\_name;

temp->lptr=(oprate[tos-1].t);

temp->rptr=(oprate[tos].t);

tos--;

oprate[tos].t=temp;

ssm--;

}

else if(matrix[row][col]==2)

{

temp=oprate[tos].t;

tos--;

oprate[tos].t=temp;

}

else if(matrix[row][col]==3)

{

printf("\b expression is invalid...\n");

printf("%c %c can not occur simuktaneously\n", oprate[tos].op\_name,exp[ssm]);

break;

}

}

}ssm++;

}

printf("show tree \n\n\n");

show\_tree(oprate[tos].t);

printf("over");

getch();

return 0;

}

int isOperator(char c)

{

int i=0;

for(i=0;i<5;i++)

{

if(c==cur\_op[i]||c==stack\_op[i])

break;

}

if(i==5)

return (-1);

else

return 1;

}

int getOperatorPosition(char c)

{

int i;

for(i=0;i<=5;i++)

{

if(c==cur\_op[i]||c==stack\_op[i])

break;

}

return i;

}

void show\_tree(node \*start)

{

if(start->lptr !=NULL)

show\_tree(start->lptr);

if(start->rptr !=NULL)

show\_tree(start->rptr);

printf("%c \n",start->data);

}

void matrix\_value(void)

{

int i,j;

printf("operator precedence matrix\n");

printf("==========================\n");

for(i=0;i<5;i++)

{

printf("%c",stack\_op[i]);

}

printf("\n");

for(i=0;i<5;i++)

{

printf("%c",cur\_op[i]);

for(j=0;j<5;j++)

{

if(matrix[i][j]==0)

printf("<");

else if(matrix[i][j]==1)

printf(">");

else if(matrix[i][j]==2)

printf("=");

else if(matrix[i][j]==3)

printf(" ");

}

printf("\n");

}

}

6. Finding First and Follow of a given grammar.

// C program to calculate the First and

// Follow sets of a given grammar

#include<stdio.h>

#include<ctype.h>

#include<string.h>

// Functions to calculate Follow

void followfirst(char, int, int);

void follow(char c);

// Function to calculate First

void findfirst(char, int, int);

int count, n = 0;

// Stores the final result

// of the First Sets

char calc\_first[10][100];

// Stores the final result

// of the Follow Sets

char calc\_follow[10][100];

int m = 0;

// Stores the production rules

char production[10][10];

char f[10], first[10];

int k;

char ck;

int e;

int main(int argc, char \*\*argv)

{

int jm = 0;

int km = 0;

int i, choice;

char c, ch;

count = 8;

// The Input grammar

strcpy(production[0], "E=TR");

strcpy(production[1], "R=+TR");

strcpy(production[2], "R=#");

strcpy(production[3], "T=FY");

strcpy(production[4], "Y=\*FY");

strcpy(production[5], "Y=#");

strcpy(production[6], "F=(E)");

strcpy(production[7], "F=i");

int kay;

char done[count];

int ptr = -1;

// Initializing the calc\_first array

for(k = 0; k < count; k++) {

for(kay = 0; kay < 100; kay++) {

calc\_first[k][kay] = '!';

}

}

int point1 = 0, point2, xxx;

for(k = 0; k < count; k++)

{

c = production[k][0];

point2 = 0;

xxx = 0;

// Checking if First of c has

// already been calculated

for(kay = 0; kay <= ptr; kay++)

if(c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

// Function call

findfirst(c, 0, 0);

ptr += 1;

// Adding c to the calculated list

done[ptr] = c;

printf("\n First(%c) = { ", c);

calc\_first[point1][point2++] = c;

// Printing the First Sets of the grammar

for(i = 0 + jm; i < n; i++) {

int lark = 0, chk = 0;

for(lark = 0; lark < point2; lark++) {

if (first[i] == calc\_first[point1][lark])

{

chk = 1;

break;

}

}

if(chk == 0)

{

printf("%c, ", first[i]);

calc\_first[point1][point2++] = first[i];

}

}

printf("}\n");

jm = n;

point1++;

}

printf("\n");

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

char donee[count];

ptr = -1;

// Initializing the calc\_follow array

for(k = 0; k < count; k++) {

for(kay = 0; kay < 100; kay++) {

calc\_follow[k][kay] = '!';

}

}

point1 = 0;

int land = 0;

for(e = 0; e < count; e++)

{

ck = production[e][0];

point2 = 0;

xxx = 0;

// Checking if Follow of ck

// has already been calculated

for(kay = 0; kay <= ptr; kay++)

if(ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

// Function call

follow(ck);

ptr += 1;

// Adding ck to the calculated list

donee[ptr] = ck;

printf(" Follow(%c) = { ", ck);

calc\_follow[point1][point2++] = ck;

// Printing the Follow Sets of the grammar

for(i = 0 + km; i < m; i++) {

int lark = 0, chk = 0;

for(lark = 0; lark < point2; lark++)

{

if (f[i] == calc\_follow[point1][lark])

{

chk = 1;

break;

}

}

if(chk == 0)

{

printf("%c, ", f[i]);

calc\_follow[point1][point2++] = f[i];

}

}

printf(" }\n\n");

km = m;

point1++;

}

}

void follow(char c)

{

int i, j;

// Adding "$" to the follow

// set of the start symbol

if(production[0][0] == c) {

f[m++] = '$';

}

for(i = 0; i < 10; i++)

{

for(j = 2;j < 10; j++)

{

if(production[i][j] == c)

{

if(production[i][j+1] != '\0')

{

// Calculate the first of the next

// Non-Terminal in the production

followfirst(production[i][j+1], i, (j+2));

}

if(production[i][j+1]=='\0' && c!=production[i][0])

{

// Calculate the follow of the Non-Terminal

// in the L.H.S. of the production

follow(production[i][0]);

}

}

}

}

}

void findfirst(char c, int q1, int q2)

{

int j;

// The case where we

// encounter a Terminal

if(!(isupper(c))) {

first[n++] = c;

}

for(j = 0; j < count; j++)

{

if(production[j][0] == c)

{

if(production[j][2] == '#')

{

if(production[q1][q2] == '\0')

first[n++] = '#';

else if(production[q1][q2] != '\0'

&& (q1 != 0 || q2 != 0))

{

// Recursion to calculate First of New

// Non-Terminal we encounter after epsilon

findfirst(production[q1][q2], q1, (q2+1));

}

else

first[n++] = '#';

}

else if(!isupper(production[j][2]))

{

first[n++] = production[j][2];

}

else

{

// Recursion to calculate First of

// New Non-Terminal we encounter

// at the beginning

findfirst(production[j][2], j, 3);

}

}

}

}

void followfirst(char c, int c1, int c2)

{

int k;

// The case where we encounter

// a Terminal

if(!(isupper(c)))

f[m++] = c;

else

{

int i = 0, j = 1;

for(i = 0; i < count; i++)

{

if(calc\_first[i][0] == c)

break;

}

//Including the First set of the

// Non-Terminal in the Follow of

// the original query

while(calc\_first[i][j] != '!')

{

if(calc\_first[i][j] != '#')

{

f[m++] = calc\_first[i][j];

}

else

{

if(production[c1][c2] == '\0')

{

// Case where we reach the

// end of a production

follow(production[c1][0]);

}

else

{

// Recursion to the next symbol

// in case we encounter a "#"

followfirst(production[c1][c2], c1, c2+1);

}

}

j++;

}

}

}

7. Construction of a NFA from a Regular expression

#include<bits/stdc++.h>

#include<iostream>

using namespace std;

int

main ()

{

char reg[20];

int q[20][3], i, j, len, a, b;

for (a = 0; a < 20; a++)

{

for (b = 0; b < 3; b++)

{

q[a][b] = 0;

}

}

cout<<"Enter the Regular Expression :-"<<endl;

cin >> reg;

len = strlen (reg);

i = 0;

j = 1;

while (i < len)

{

if (reg[i] == 'a' && reg[i + 1] != '|' && reg[i + 1] != '\*')

{

q[j][0] = j + 1;

j++;

}

if (reg[i] == 'b' && reg[i + 1] != '|' && reg[i + 1] != '\*')

{

q[j][1] = j + 1;

j++;

}

if (reg[i] == 'e' && reg[i + 1] != '|' && reg[i + 1] != '\*')

{

q[j][2] = j + 1;

j++;

}

if (reg[i] == 'a' && reg[i + 1] == '|' && reg[i + 2] == 'b')

{

q[j][2] = ((j + 1) \* 10) + (j + 3);

j++;

q[j][0] = j + 1;

j++;

q[j][2] = j + 3;

j++;

q[j][1] = j + 1;

j++;

q[j][2] = j + 1;

j++;

i = i + 2;

}

if (reg[i] == 'b' && reg[i + 1] == '|' && reg[i + 2] == 'a')

{

q[j][2] = ((j + 1) \* 10) + (j + 3);

j++;

q[j][1] = j + 1;

j++;

q[j][2] = j + 3;

j++;

q[j][0] = j + 1;

j++;

q[j][2] = j + 1;

j++;

i = i + 2;

}

if (reg[i] == 'a' && reg[i + 1] == '\*')

{

q[j][2] = ((j + 1) \* 10) + (j + 3);

j++;

q[j][0] = j + 1;

j++;

q[j][2] = ((j + 1) \* 10) + (j - 1);

j++;

}

if (reg[i] == 'b' && reg[i + 1] == '\*')

{

q[j][2] = ((j + 1) \* 10) + (j + 3);

j++;

q[j][1] = j + 1;

j++;

q[j][2] = ((j + 1) \* 10) + (j - 1);

j++;

}

if (reg[i] == ')' && reg[i + 1] == '\*')

{

q[0][2] = ((j + 1) \* 10) + 1;

q[j][2] = ((j + 1) \* 10) + 1;

j++;

}

i++;

}

cout << "Transition functions are :- \n";

for (i = 0; i <= j; i++)

{

if (q[i][0] != 0)

cout << " q[" << i << ",a]-->" << q[i][0]<< endl;

if (q[i][1] != 0)

cout << " q[" << i << ",b]-->" << q[i][1]<< endl;

if (q[i][2] != 0)

{

if (q[i][2] < 10)

cout << " q[" << i << ",e]-->" << q[i][2] << endl;

else

cout << " q[" << i << ",e]-->" << q[i][2] /

10 << "&" << q[i][2] % 10 <<endl ;

}

}

return 0;

}

#include<bits/stdc++.h>

using namespace std;

int prio(char c)

{

if(c == '\*') return 3;

if(c == '.') return 2;

if(c == '+') return 1;

return 0;

}

string infix\_to\_postfix(string s)

{

stack<char> st;

st.push('(');

s += ')';

int n = s.size();

string ret;

for(int i=0; i<n; i++)

{

if(prio(s[i]))

{

while(!st.empty())

{

if(prio(st.top()) <= prio(s[i]))

{

st.push(s[i]);

break;

}

ret += st.top();

st.pop();

}

}

else if(s[i] == '(')

{

st.push(s[i]);

}

else if(s[i] == ')')

{

while(!st.empty())

{

if(st.top() == '(')

{

st.pop();

break;

}

ret += st.top();

st.pop();

}

}

else if(isalpha(s[i]))

{

ret += s[i];

}

}

return ret;

}

int nodes = 0;

int ALPHABET\_SZ = 3;

vector<vector<vector<int> > > transitions(30,vector<vector<int> >(ALPHABET\_SZ+1,vector<int>(0)));

pair<int,int> handlePlus(int ss1,int es1,int ss2,int es2)

{

//cout<<"i was called : "<<endl;

transitions[nodes+1][ALPHABET\_SZ].push\_back(ss1);

transitions[nodes+1][ALPHABET\_SZ].push\_back(ss2);

transitions[es1][ALPHABET\_SZ].push\_back(nodes+2);

transitions[es2][ALPHABET\_SZ].push\_back(nodes+2);

nodes = nodes+2;

return make\_pair(nodes-1,nodes);

}

pair<int,int> handleStar(int ss,int es)

{

//cout<<"\* was called : "<<endl;

transitions[es][ALPHABET\_SZ].push\_back(ss);

transitions[nodes+1][ALPHABET\_SZ].push\_back(ss);

transitions[nodes+1][ALPHABET\_SZ].push\_back(nodes+2);

transitions[es][ALPHABET\_SZ].push\_back(nodes+2);

nodes = nodes+2;

return make\_pair(nodes-1,nodes);

}

pair<int,int> handleDot(int ss2,int es2,int ss1,int es1)

{

//cout<<". was called :"<<endl;

transitions[es1][ALPHABET\_SZ].push\_back(ss2);

return make\_pair(ss1,es2);

}

pair<int,int> handleSingle(int alphabet)

{

transitions[nodes+1][alphabet-'a'].push\_back(nodes+2);

nodes = nodes+2;

return make\_pair(nodes-1,nodes);

}

void postfix\_to\_e\_nfa(string s)

{

stack<pair<int,int> > st;

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

{

//cout<<"NODES : "<<nodes<<endl;

if(s[i] == '\*')

{

pair<int,int> r = st.top();

st.pop();

st.push(handleStar(r.first,r.second));

}

else if(s[i]=='+')

{

pair<int,int> r1 = st.top();

st.pop();

pair<int,int> r2 = st.top();

st.pop();

st.push(handlePlus(r1.first,r1.second,r2.first,r2.second));

}

else if(s[i]=='.')

{

//cout<<"NODES BEFORE CALL : "<<nodes<<endl;

pair<int,int> r1 = st.top();

st.pop();

pair<int,int> r2 = st.top();

st.pop();

st.push(handleDot(r1.first,r1.second,r2.first,r2.second));

//cout<<" NODES AFTER CALL : "<<nodes<<endl;

}

else

{

st.push(handleSingle(s[i]));

}

}

}

int main()

{

cout << "Enter regular expression: ";

string regex; cin >> regex;

regex = infix\_to\_postfix(regex);

//cout<<"REGEX : "<<regex<<endl;

postfix\_to\_e\_nfa(regex);

for(int i=0;i<=nodes;i++)

{

for(int j=0;j<=ALPHABET\_SZ;j++)

{

char c = 'a';

for(int k=0;k<transitions[i][j].size();k++)

cout<<"q["<<i<<","<<(j==ALPHABET\_SZ?'E':char(c+j))<<"] -> "<<transitions[i][j][k]<<" "<<endl;

}

}

return 0;

}

#include<stdio.h>

#include<conio.h>

int main()

{

char m[20],t[10][10];

int n,i,j,r=0,c=0;

//clrscr();

printf("\n\t\t\t\tSIMULATION OF NFA");

printf("\n\t\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(i=0;i<10;i++)

{

for(j=0;j<10;j++)

{

t[i][j]=' ';

}

}

printf("\n\nEnter a regular expression:");

scanf("%s",m);

n=strlen(m);

for(i=0;i<n;i++)

{

switch(m[i])

{

case '|' : {

t[r][r+1]='E';

t[r+1][r+2]=m[i-1];

t[r+2][r+5]='E';

t[r][r+3]='E';

t[r+4][r+5]='E';

t[r+3][r+4]=m[i+1];

r=r+5;

break;

}

case '\*':{

t[r-1][r]='E';

t[r][r+1]='E';

t[r][r+3]='E';

t[r+1][r+2]=m[i-1];

t[r+2][r+1]='E';

t[r+2][r+3]='E';

r=r+3;

break;

}

case '+': {

t[r][r+1]=m[i-1];

t[r+1][r]='E';

r=r+1;

break;

}

default:

{

if(c==0)

{

if((isalpha(m[i]))&&(isalpha(m[i+1])))

{

t[r][r+1]=m[i];

t[r+1][r+2]=m[i+1];

r=r+2;

c=1;

}

c=1;

}

else if(c==1)

{

if(isalpha(m[i+1]))

{

t[r][r+1]=m[i+1];

r=r+1;

c=2;

}

}

else

{

if(isalpha(m[i+1]))

{

t[r][r+1]=m[i+1];

r=r+1;

c=3;

}

}

}

break;

}

}

printf("\n");

for(j=0;j<=r;j++)

printf(" %d",j);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n");

for(i=0;i<=r;i++)

{

for(j=0;j<=r;j++)

{

printf(" %c",t[i][j]);

}

printf(" | %d",i);

printf("\n");

}

printf("\nStart state: 0\nFinal state: %d",i-1);

getch();

}