**LAB 3**

**Aim: Write a program to convert NFA to DFA**

**Code:**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

void print(vector<vector<vector<int> > > table)

{

cout<<" STATE/INPUT |";

char a='a';

for(int i=0;i<table[0].size()-1;i++)

{

cout<<" "<<a++<<" |";

}

cout<<" ^ "<<endl<<endl;

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

cout<<" "<<i<<" ";

for(int j=0;j<table[i].size();j++){

cout<<" | ";

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

{

cout<<table[i][j][k]<<" ";

}

}

cout<<endl;

}

}

void printdfa(vector<vector<int> > states, vector<vector<vector<int> > > dfa)

{

cout<<" STATE/INPUT ";

char a='a';

for(int i=0;i<dfa[0].size();i++){

cout<<"| "<<a++<<" ";

}

cout<<endl;

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

cout<<"{ ";

for(int h=0;h<states[i].size();h++)

cout<<states[i][h]<<" ";

if(states[i].empty()){

cout<<"^ ";

}

cout<<"} ";

for(int j=0;j<dfa[i].size();j++){

cout<<" | ";

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

cout<<dfa[i][j][k]<<" ";

}

if(dfa[i][j].empty()){

cout<<"^ ";

}

}

cout<<endl;

}

}

vector<int> closure(int s,vector<vector<vector<int>>> v)

{

vector<int> t;

queue<int> q;

t.push\_back(s);

int a=v[s][v[s].size()-1].size();

for(int i=0;i<a;i++){

t.push\_back(v[s][v[s].size()-1][i]);

q.push(t[i]);

}

while(!q.empty()){

int f=q.front();

q.pop();

if(!v[f][v[f].size()-1].empty()){

int u=v[f][v[f].size()-1].size();

for(int i=0;i<u;i++){

int y=v[f][v[f].size()-1][i];

if(find(t.begin(),t.end(),y)==t.end())

{

t.push\_back(y);

q.push(y);

}

}

}

}

return t;

}

int main(){

int n,alpha;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RAJ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NFA to DFA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;

cout<<"Enter total number of states in NFA : ";

cin>>n;

cout<<"Enter number of elements in alphabet : ";

cin>>alpha;

vector<vector<vector<int> > > table;

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

cout<<"For state "<<i<<endl;

vector< vector< int > > v;

char a='a';

int y,yn;

for(int j=0;j<alpha;j++){

vector<int> t;

cout<<"Enter no. of output states for input "<<a++<<" : ";

cin>>yn;

cout<<"Enter output states :"<<endl;

for(int k=0;k<yn;k++){

cin>>y;

t.push\_back(y);

}

v.push\_back(t);

}

vector<int> t;

cout<<"Enter no. of output states for input ^ : ";

cin>>yn;

cout<<"Enter output states :"<<endl;

for(int k=0;k<yn;k++){

cin>>y;

t.push\_back(y);

}

v.push\_back(t);

table.push\_back(v);

}

cout<<"\*\*\*\*\* TRANSITION TABLE OF NFA \*\*\*\*\*"<<endl;

print(table);

cout<<endl<<"\*\*\*\*\* TRANSITION TABLE OF DFA \*\*\*\*\*"<<endl;

vector<vector<vector<int> > > dfa;

vector<vector<int> > states;

states.push\_back(closure(0,table));

queue<vector<int> > q;

q.push(states[0]);

while(!q.empty()){

vector<int> f=q.front();

q.pop();

vector<vector<int> > v;

for(int i=0;i<alpha;i++){

vector<int> t;

set<int> s;

for(int j=0;j<f.size();j++){

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

vector<int> cl= closure(table[f[j]][i][k],table);

for(int h=0;h<cl.size();h++){

if(s.find(cl[h])==s.end())

s.insert(cl[h]);

}

}

}

for(set<int >::iterator u=s.begin(); u!=s.end();u++)

t.push\_back(\*u);

v.push\_back(t);

if(find(states.begin(),states.end(),t)==states.end())

{

states.push\_back(t);

q.push(t);

}

}

dfa.push\_back(v);

}

printdfa(states,dfa);

}

**LAB 4**

**Aim: Write a program to calculate first and follow of a given LL(1) grammar.**

**Code:**

#include <ctype.h>

#include <stdio.h>

#include <string.h>

void followfirst(char, int, int);

void follow(char c);

void findfirst(char, int, int);

int count, n = 0;

char calc\_first[10][100];

char calc\_follow[10][100];

int m = 0;

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;

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

strcpy(production[0], "X=TnS");

strcpy(production[1], "X=Rm");

strcpy(production[2], "T=q");

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

strcpy(production[4], "S=p");

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

strcpy(production[6], "R=om");

strcpy(production[7], "R=ST");

int kay;

char done[count];

int ptr = -1;

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;

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

if (c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

findfirst(c, 0, 0);

ptr += 1;

done[ptr] = c;

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

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

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;

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;

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

if (ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

follow(ck);

ptr += 1;

donee[ptr] = ck;

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

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

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;

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')

{

followfirst(production[i][j + 1], i,

(j + 2));

}

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

{

follow(production[i][0]);

}

}

}

}

}

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

{

int j;

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))

{

findfirst(production[q1][q2], q1,

(q2 + 1));

}

else

first[n++] = '#';

}

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

{

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

}

else

{

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

}

}

}

}

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

{

int k;

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;

}

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

{

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

{

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

}

else

{

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

{

follow(production[c1][0]);

}

else

{

followfirst(production[c1][c2], c1,

c2 + 1);

}

}

j++;

}

}

}

**LAB 5**

**Aim: WAP to construct LL(1) parsing table for LL(1) grammar and validate the input string**

**Code:**

#include <ctype.h>

#include <stdio.h>

#include <string.h>

void followfirst(char, int, int);

void follow(char c);

void findfirst(char, int, int);

int count, n = 0;

char calc\_first[10][100];

char calc\_follow[10][100];

int m = 0;

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;

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

strcpy(production[0], "X=TnS");

strcpy(production[1], "X=Rm");

strcpy(production[2], "T=q");

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

strcpy(production[4], "S=p");

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

strcpy(production[6], "R=om");

strcpy(production[7], "R=ST");

int kay;

char done[count];

int ptr = -1;

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;

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

if (c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

findfirst(c, 0, 0);

ptr += 1;

done[ptr] = c;

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

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

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;

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;

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

if (ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

follow(ck);

ptr += 1;

donee[ptr] = ck;

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

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

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;

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')

{

followfirst(production[i][j + 1], i,

(j + 2));

}

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

{

follow(production[i][0]);

}

}

}

}

}

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

{

int j;

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))

{

findfirst(production[q1][q2], q1,

(q2 + 1));

}

else

first[n++] = '#';

}

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

{

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

}

else

{

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

}

}

}

}

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

{

int k;

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;

}

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

{

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

{

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

}

else

{

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

{

follow(production[c1][0]);

}

else

{

followfirst(production[c1][c2], c1,

c2 + 1);

}

}

j++;

}

}

}

**LAB 6**

**Aim: WAP to construct operator precedence parsing table for the given grammar and check validity of the string.**

**Code:**

#include <stdio.h>

#include<stdlib.h>

#include <string.h>

char \*input;

int i = 0;

char lasthandle[6], stack[50], handles[][5] = {")E(", "E\*E", "E+E", "i", "E^E"};

int top = 0, l;

char prec[9][9] = {'>','>','<','<','<','<','<','>','>',/\* - \*/ '>','>','<','<','<','<','<','>','>',/\* \* \*/ '>','>','>','>','<','<','<','>','>',/\* / \*/ '>','>','>','>','<','<','<','>','>',/\* ^ \*/ '>','>','>','>','<','<','<','>','>',/\* i \*/ '>','>','>','>','>','e','e','>','>',/\* ( \*/ '<','<','<','<','<','<','<','>','e',/\* ) \*/ '>','>','>','>','>','e','e','>','>',/\* $ \*/ '<','<','<','<','<','<','<','<','>',};

int getindex(char c)

{

switch (c)

{

case '+':

return 0;

case '-':

return 1;

case '\*':

return 2;

case '/':

return 3;

case '^':

return 4;

case 'i':

return 5;

case '(':

return 6;

case ')':

return 7;

case '$':

return 8;

}

}

int shift()

{

stack[++top] = \*(input + i++);

stack[top + 1] = '\0';

}

int reduce()

{

int i, len, found, t;

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

len = strlen(handles[i]);

if (stack[top] == handles[i][0] && top + 1 >= len) {

found = 1;

for (t = 0; t < len; t++){

if (stack[top - t] != handles[i][t]){

found = 0;

break;

}

}

if (found == 1) {

stack[top - t + 1] = 'E';

top = top - t + 1;

strcpy(lasthandle, handles[i]);

stack[top + 1] = '\0';

return 1; }

}

}

return 0;

}

void dispstack()

{

int j;

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

printf("%c", stack[j]);

}

void dispinput()

{

int j;

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

printf("%c", \*(input + j));

}

int main()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RAJ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int j;

input = (char \*)malloc(50 \* sizeof(char));

printf("\nEnter the string\n");

scanf("%s", input);

input = strcat(input, "$");

l = strlen(input);

strcpy(stack, "$");

printf("\nSTACK\tINPUT\tACTION");

while (i <= l){

shift();

printf("\n");

dispstack();

printf("\t");

dispinput();

printf("\tShift");

if (prec[getindex(stack[top])][getindex(input[i])] == '>'){

while (reduce()) {

printf("\n");

dispstack();

printf("\t");

dispinput();

printf("\tReduced: E->%s", lasthandle);}}}

if (strcmp(stack, "$E$") == 0)

printf("\nAccepted;");

else

printf("\nNot Accepted;");

}

**LAB 7**

**Aim: WAP to construct recursive descent parser**

**Code:**

#include <stdio.h>

#include <string.h>

#define SUCCESS 1

#define FAILED 0

int E(), Edash(), T(), Tdash(), F();

const char \*cursor;

char string[64];

int main()

{

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

puts("Enter the string: i+i+(i\*i)");

sscanf("i+i+(i\*i)", "%s", string);

cursor = string;

puts("");

puts("Input Action");

puts("--------------------------------");

if (E() && \*cursor == '\0')

{

puts("--------------------------------");

puts("String is successfully parsed");

return 0;

}

else

{

puts("--------------------------------");

puts("Error in parsing String");

return 1;

}

}

int E()

{

printf("%-16s E -> T E'\n", cursor);

if (T())

{

if (Edash())

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

int Edash()

{

if (\*cursor == '+')

{

printf("%-16s E' -> + T E'\n", cursor);

cursor++;

if (T())

{

if (Edash())

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

else

{

printf("%-16s E' -> $\n", cursor);

return SUCCESS;

}

}

int T()

{

printf("%-16s T -> F T'\n", cursor);

if (F())

{

if (Tdash())

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

int Tdash()

{

if (\*cursor == '\*')

{

printf("%-16s T' -> \* F T'\n", cursor);

cursor++;

if (F())

{

if (Tdash())

return SUCCESS;

else

return FAILED;

}

else

return FAILED;

}

else

{

printf("%-16s T' -> $\n", cursor);

return SUCCESS;

}

}

int F()

{

if (\*cursor == '(')

{

printf("%-16s F -> ( E )\n", cursor);

cursor++;

if (E())

{

if (\*cursor == ')')

{

cursor++;

return SUCCESS;

}

else

return FAILED;

}

else

return FAILED;

}

else if (\*cursor == 'i')

{

cursor++;

printf("%-16s F ->i\n", cursor);

return SUCCESS;

}

else

return FAILED;

}