**EXPERIMENT NO.:-** 1

**AIM:-** Program to convert Regular Expression(R.E.) to Non-Deterministic Finite Automata(N.F.A.)

**Algorithm:-**

1. The NFA representing the empty string is:



2. If the regular expression is just a character, eg. a, then the corresponding NFA is :



3. The union operator is represented by a choice of transitions from a node; thus a|b can be represented as:



4. Concatenation simply involves connecting one NFA to the other; eg. ab is:



5. The Kleene closure must allow for taking zero or more instances of the letter from the input; thus a\* looks like:



**Program Code:-**

# include <stdio.h>

# include <conio.h>

# include <string.h>

# include <ctype.h>

int ret[100];

static int pos=0;

static int sc=0;

void nfa(int st,int p,char \*s) { int i,sp,fs[15],fsc=0;

sp=st;pos=p;sc=st;

while(\*s!=NULL)

{if(isalpha(\*s))

{ret[pos++]=sp;

ret[pos++]=\*s;

ret[pos++]=++sc;}

if(\*s=='.')

{sp=sc;

ret[pos++]=sc;

ret[pos++]=238;

ret[pos++]=++sc;

sp=sc;}

if(\*s=='|')

{sp=st;

fs[fsc++]=sc;}

if(\*s=='\*')

{ret[pos++]=sc;

ret[pos++]=238;

ret[pos++]=sp;

2

ret[pos++]=sp;

ret[pos++]=238;

ret[pos++]=sc;

}

if (\*s=='(')

{char ps[50];

int i=0,flag=1;

s++;

while(flag!=0)

{ps[i++]=\*s;

if (\*s=='(')

flag++;

if (\*s==')')

flag--;

s++;}

ps[--i]='\0';

nfa(sc,pos,ps);

s--;

}

s++;

}

sc++;

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

{ret[pos++]=fs[i];

ret[pos++]=238;

ret[pos++]=sc;

3

}

ret[pos++]=sc-1;

ret[pos++]=238;

ret[pos++]=sc;

}

void main()

{ int i;

char \*inp;

clrscr();

printf("enter the regular expression :");

gets(inp);

nfa(1,0,inp);

printf("\nstate input state\n");

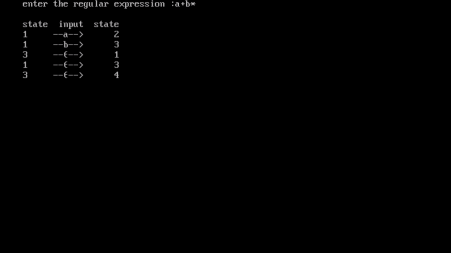
for(i=0;i<pos;i=i+3)

printf("%d --%c--> %d\n",ret[i],ret[i+1],ret[i+2]); printf("\n");

getch();

}

**Output:-**

**Result:-** The program to convert Regular Expression(R.E.) to Non-Deterministic Finite Automata(N.F.A.) was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 2

**AIM:-** Program to print transition table from given transition diagram

**Algorithm:-**

1. Initialize all the variables

2. Input the no. of states, no. of symbols

3. Record the transitions in a matrix

4. Output the transition table

5. Exit

**Program Code:-**

#include<iostream.h>

#include<conio.h>

int main()

{

int a[10][10],i,j,ns,ni;

clrscr();

cout<<”\n Enter the number of states \n”;

cin>>ns;

cout<<”\n Enter the number of inputs \n”;

cin>>ni;

cout<<”\n Enter the states”;

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

cin>>a[i][0];

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

{

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

{

cout<<”Enter the value for state a”<<i<<”for input”<<j-1<<”\n”; cin>>a[i][j];

}

}

cout<<” The transition table is:\n”;

cout<<”states”;

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

cout<<”input”<<i;

cout<<”\n”;

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

{

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

{

cout<<a[i][j]<<”\t”;

}

cout<<”\n”;

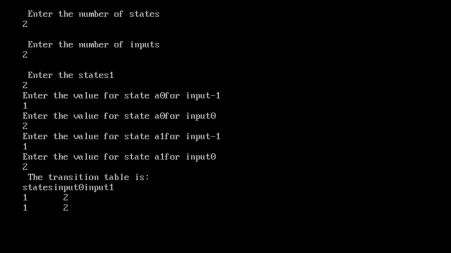
}

getch();

return 0;

}

**Output:-**

**Result:-** The program to print transition table from given transition diagram was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 3

**AIM:-** Program to convert NFA to Deterministic Finite Automata(D.F.A.) **Algorithm:-**

1. Convert into NFA using above rules for operators (union, concatenation and closure) and precedence.

2. Find Ɛ -closure of all states.

3. Start with epsilon closure of start state of NFA.

4. Apply the input symbols and find its epsilon closure. Dtran[state, input symbol] = Ɛ -closure(move(state, input symbol)) where Dtran àtransition function of DFA

6. Analyze the output state to find whether it is a new state. 7. If new state is found, repeat step 4 and step 5 until no more new states are found.

8. Construct the transition table for Dtran function.

9. Draw the transition diagram with start state as the Ɛ -closure (start state of NFA) and final state is the state that contains final state of NFA drawn.

**Program Code:-**

#include<stdio.h>

#include<conio.h>

#define MAX 20

//========================================================= struct nfa\_state

{

int a, b, eps1, eps2;

}NFA[20];

struct dfa\_state

{

9

int state[20],a[20],b[20];

}DFA[20];

int cur, initial\_state, final\_state;

int stack[MAX];

int top;

//========================================================= void push(int val)

{

stack[++top]=val;

}

int pop()

{

return stack[top--];

}

//========================================================= int priority(char op)

{

switch(op)

{

case '+': return 1;

case '.': return 2;

case '\*': return 3;

}

return 0;

}

//========================================================= void init\_nfa\_table()

{

int i;

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

{

NFA[i].a = NFA[i].b = -1;

NFA[i].eps1 = NFA[i].eps2 = -1;

}

}

//========================================================= void symbol(char c)

{

if(c=='a')

NFA[cur].a = cur+1;

if(c=='b')

NFA[cur].b = cur+1;

push(cur);

push(cur+1);

cur += 2;

}

//========================================================= void concat()

{

int first1, first2, last1, last2;

last2 = pop();

first2 = pop();

last1 = pop();

first1 = pop();

NFA[last1].eps1 = first2;

push(first1);

push(last2);

}

//========================================================= void parallel()

{

int first1, first2, last1, last2;

last2 = pop();

first2 = pop();

last1 = pop();

first1 = pop();

NFA[cur].eps1 = first1;

NFA[cur].eps2 = first2;

NFA[last1].eps1 = cur+1;

NFA[last2].eps2 = cur+1;

push(cur);

push(cur+1);

cur += 2;

}

//========================================================= void closure()

{

int first,last;

last = pop();

first = pop();

NFA[cur].eps1 = first;

NFA[cur].eps2 = cur+1;

NFA[last].eps1 = first;

NFA[last].eps2 = cur+1;

push(cur);

push(cur+1);

cur += 2;

}

//========================================================= void construct\_nfa(char \*postfix)

{

int i=0;

top=-1;

for(i=0; postfix[i]!='\0'; i++)

{

switch(postfix[i])

{

case 'a':

case 'b': symbol(postfix[i]);

break;

case '.': concat();

break;

case '+': parallel();

break;

case '\*': closure();

}

}

final\_state = pop();

initial\_state = pop();

}

//========================================================= void disp\_NFA()

{

int i;

printf("\nstate\ta\tb\tо");

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

{

if(i==initial\_state)

printf("\n->%d",i);

else

if(i==final\_state)

printf("\n\* %d",i);

else

printf("\n %d",i);

if(NFA[i].a==-1)

printf("\t-");

else

printf("\t{%d}",NFA[i].a);

if(NFA[i].b==-1)

printf("\t-");

else

printf("\t{%d}",NFA[i].b);

if(NFA[i].eps1!=-1)

{

printf("\t{%d",NFA[i].eps1);

if(NFA[i].eps2!=-1)

{

printf(",%d",NFA[i].eps2);

}

printf("}");

}

else

printf("\t-");

}

}

//========================================================= void init\_dfa\_table()

{

int i,j;

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

{

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

{

DFA[i].state[j]=-1;

DFA[i].a[j]=-1;

DFA[i].b[j]=-1;

}

}

}

//========================================================= void print\_state(int t[])

{

int i=0;

printf("[");

for(i=0;t[i]!=-1;i++)

printf("%d,",t[i]);

printf("\b]");

}

//========================================================= int isPresent(int T[], int v)

{

int i;

for(i=0;T[i]!=-1;i++)

if(T[i]==v)

return 1;

return 0;

}

//========================================================= void disp\_DFA(int n)

{

int i;

printf("\nstate\t\t\ta\t\t\tb");

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

{

printf("\n");

if(i==0)

printf("->");

if(isPresent(DFA[i].state,final\_state))

printf("\*");

print\_state(DFA[i].state);

printf("\t\t");

if(DFA[i].a[0]!=-1)

print\_state(DFA[i].a);

else

printf("\t-");

printf("\t\t");

if(DFA[i].b[0]!=-1)

print\_state(DFA[i].b);

else

printf("\t-");

}

}

//========================================================= void epsilon\_closure(int T[], int t[])

{

int i,v;

top=-1;

for(i=0;t[i]!=-1;i++)

push(t[i]);

i=0;

while(top!=-1)

{

v = pop();

if(isPresent(T,v)==0)

{

T[i++]=v;

}

if(NFA[v].eps1!=-1)

{

push(NFA[v].eps1);

}

if(NFA[v].eps2!=-1)

{

push(NFA[v].eps2);

}

}

}

//========================================================= void init\_t(int t[])

{

int i;

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

t[i]=-1;

}

//========================================================= int search(int n,int t2[])

{

int i,j;

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

{

for(j=0;t2[j]!=-1;j++)

if(isPresent(DFA[i].state,t2[j])==0)

break;

if(t2[j]==-1)

return 1;

}

return 0;

}

//========================================================= void copy(int t1[], int t2[])

{

int i;

for(i=0;t2[i]!=-1;i++)

t1[i]=t2[i];

}

//=========================================================

void main()

{

char postfix[20];

int t[20],v;

int n=0,i=0,j,k;

clrscr();

printf("\nEnter Regular Expression: "); scanf("%s",postfix);

printf("\nPostfix Expression: %s",postfix); getch();

init\_nfa\_table();

construct\_nfa(postfix);

clrscr();

disp\_NFA();

getch();

init\_dfa\_table();

init\_t(t);

t[0]=initial\_state;

epsilon\_closure(DFA[0].state,t);

init\_t(t);

for(j=0,k=0; DFA[0].state[j]!=-1 ; j++) {

v = DFA[0].state[j];

if(NFA[v].a!=-1)

{

if(isPresent(t,NFA[v].a)==0)

t[k++]=NFA[v].a;

}

}

epsilon\_closure(DFA[0].a,t);

init\_t(t);

for(j=0,k=0;DFA[0].state[j]!=-1;j++) {

v = DFA[0].state[j];

if(NFA[v].b!=-1)

{

22

if(isPresent(t,NFA[v].b)==0)

t[k++]=NFA[v].b;

}

}

epsilon\_closure(DFA[0].b,t);

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

{

if( search( n , DFA[i].a)==0 )

{

n++;

copy(DFA[n].state,DFA[i].a);

init\_t(t);

for( j=0,k=0; DFA[n].state[j]!=-1 ; j++)

{

v = DFA[n].state[j];

if(NFA[v].a!=-1)

{

if(isPresent(t,NFA[v].a)==0)

t[k++]=NFA[v].a;

}

}

epsilon\_closure(DFA[n].a,t);

init\_t(t);

for(j=0,k=0;DFA[n].state[j]!=-1;j++) {

v = DFA[n].state[j];

if(NFA[v].b!=-1)

{

if(isPresent(t,NFA[v].b)==0)

t[k++]=NFA[v].b;

}

}

epsilon\_closure(DFA[n].b,t);

}

if( search( n , DFA[i].b ) ==0)

{

n++;

copy(DFA[n].state,DFA[i].b);

init\_t(t);

for( j=0,k=0; DFA[n].state[j]!=-1 ; j++) {

v = DFA[n].state[j];

24

if( NFA[v].a!=-1)

{

if(isPresent(t,NFA[v].a)==0)

t[k++]=NFA[v].a;

}

}

epsilon\_closure(DFA[n].a,t);

init\_t(t);

for(j=0,k=0;DFA[n].state[j]!=-1;j++)

{

v = DFA[n].state[j];

if(NFA[v].b!=-1)

{

if(isPresent(t,NFA[v].b)==0)

t[k++]=NFA[v].b;

}

}

epsilon\_closure(DFA[n].b,t);

}

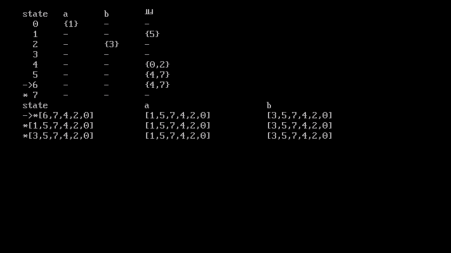
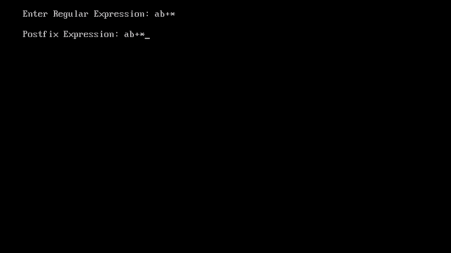
}

disp\_DFA(n);

getch();

}

**Output:-**

****

**Result:-** The program to convert NFA to Deterministic Finite Automata(D.F.A.) was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 4

**AIM:-** Program to convert Regular Expression(R.E.) to Deterministic Finite Automata(D.F.A.)

**Algorithm:-**

1. Augment the regular expression ***r*** with a special symbol **#** which is used as an end marker and any transition over # in a DFA will be an accepting. Hence, the new expression is ***r* #.**

2. Construct a syntax tree for ***r*#**

3. Traverse the tree to construct functions *nullable( )*, *firstpos( )*, *lastpos ( )*, and *followpos( )*

4. Based on the functions *firstpos( )*, *lastpos ( )*, and *followpos( )* and using the tree, the minimized DFA is constructed

**Program Code:-** #include<stdio.h> #include<conio.h> #include<stdlib.h> #include<string.h>

typedef struct tree {

char ch;

int pos;

int nullable;

int fpos[5];

int lpos[5];

struct tree \* lc; struct tree \* rc; }node;

int dfaa[30],df=0; typedef struct foll {

int follpos[10]; char ch;

}follpos;

follpos folltab[100]; char inpt[10];

void follow(node \*); node\* alloc(char ch)

{

node \* temp;

temp=(node \*)malloc(sizeof(node)); temp->nullable=1;

temp->lc=NULL;

temp->rc=NULL;

temp->ch=ch;

temp->fpos[0]=-1;

temp->lpos[0]=-1;

return temp;

}

void unio(int [],int []);

void sort(int []);

int check(int[] ,int);

void print\_follow(int n)

{

int i,j;

printf("FOLLOWPOS\n");

printf("POS\tNAME\tFOLLOWPOS\n"); for(i=1;i<=n;++i)

{

printf("%d\t%c\t",i,folltab[i].ch); j=0;

while(folltab[i].follpos[j]!=-1) {

printf("%d ",folltab[i].follpos[j]);

j++;

}

printf("\n");

}

}

void print\_nullable(node \*root) {

int i;

if(root!=NULL)

{

print\_nullable(root->lc); print\_nullable(root->rc); printf("%c\t",root->ch); i=0;

while(root->fpos[i]!=-1) {

printf("%d ",root->fpos[i]); i++;

}

printf("\t");

i=0;

while(root->lpos[i]!=-1) {

printf("%d ",root->lpos[i]); i++;

}

printf("\n");

}

}

node \* create(char str[],int \*l)

{

node \* nw;

nw=alloc(str[\*l]);

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

if(str[\*l]!='\*')

{

(\*l)--;

nw->nullable=0;

nw->rc=create(str,l);

}

(\*l)--;

nw->lc=create(str,l);

}

else

nw->nullable=0;

return nw;

}

void inorder(node \*root)

{

if(root!=NULL)

{

inorder(root->lc);

printf("%c",root->ch);

inorder(root->rc);

}

}

void create\_nullable(node \* root,int \*pos) {

if(root->lc!=NULL)

create\_nullable(root->lc,pos); if(root->rc!=NULL)

create\_nullable(root->rc,pos); if(root->lc==NULL && root->rc==NULL) {

root->pos=(\*pos);

root->fpos[0]=root->lpos[0]=(\*pos); root->fpos[1]=root->lpos[1]=-1; folltab[\*pos].ch=root->ch;

folltab[\*pos].follpos[0]=-1;

(\*pos)++;

}

else

{

if(root->ch=='|')

{

unio(root->fpos,root->lc->fpos); unio(root->fpos,root->rc->fpos);

unio(root->lpos,root->lc->lpos); unio(root->lpos,root->rc->lpos); }

else if(root->ch=='\*')

{

unio(root->fpos,root->lc->fpos); unio(root->lpos,root->lc->lpos); }

else if(root->ch=='.')

{

if(root->lc->nullable==1) {

unio(root->fpos,root->rc->fpos); }

unio(root->fpos,root->lc->fpos); sort(root->fpos);

if(root->rc->nullable==1) {

unio(root->lpos,root->lc->lpos); }

unio(root->lpos,root->rc->lpos); sort(root->lpos);

}

follow(root);

}

}

void follow(node \*root)

{

int i=0;

if(root->ch=='\*')

{

while(root->lpos[i]!=-1)

{

unio(folltab[root->lpos[i]].follpos,root->fpos); i++;

}

}

else if(root->ch=='.')

{

while(root->lc->lpos[i]!=-1)

{

unio(folltab[root->lc->lpos[i]].follpos,root->rc->fpos); i++;

}

}

}

void unio(int arr1[],int arr2[])

{

int i=0,j,k;

while(arr1[i]!=-1)

i++;

j=0;k=0;

while(arr2[j]!=-1)

{

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

if(arr2[j]==arr1[k])

break;

}

if(k==i)

{

arr1[i]=arr2[j];

i++;

}

j++;

}

arr1[i]=-1;

}

void sort(int a[])//insertion sort {

int i,j,temp;

for(i=1;a[i]!=-1;i++)

{

temp=a[i];

for(j=i-1;j>=0&&temp<a[j];j--) {

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

a[j+1]=temp;

}

}

}

int state[10][10];

void dfa()

{

int nos,m,i,j,k,temp[10];

j=0,k=0;

temp[0]=-1;

nos=1;

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

state[i][0]=-1;

i=0;

unio(state[0],folltab[1].follpos);

while(1)

{

for(i=0;inpt[i]!=NULL;++i)

{

for(j=0;state[k][j]!=-1;++j)

{

if(folltab[state[k][j]].ch==inpt[i]) { unio(temp,folltab[state[k][j]].follpos); } }

m=check(temp,nos);

if(m==-1)

{

unio(state[nos++],temp);

m=nos-1;

}

dfaa[df++]=m;

temp[0]=-1;

}

if(k==nos-1)

break;

k++;

}

}

int check(int temp[],int nos)

{

int i, j;

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

for(j=0;temp[j]!=-1;++j)

{

if(temp[j]!=state[i][j])

break;

}

if(temp[j]==-1 && state[i][j]==-1) return i;

}

return -1;

}

void display\_dfa()//displaying DFA table {

int i,j,k;

printf("\nDFA TABLE\n ");

for(i=0;inpt[i]!=NULL;i++)

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

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

{

printf("\n%c\t",j+65);

for(k=j\*i;k<(j\*i)+i;k++)

printf("%c\t",dfaa[k]+65);

}

getch();

}

void main()

{

int l,j,i,pos;

node\*root;

char str[50];

inpt[0]=NULL;

printf("Enter the postfix expression\n"); scanf("%s",str);

strcat(str,"#.\0");

l=strlen(str);

l--;

j=0;

for(i=0;i<l-1;++i){

j=0;

while(inpt[j]!=NULL)

{

if(inpt[j]==str[i])

break;

j++;

}

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

inpt[j]=str[i];

inpt[j+1]=NULL;

}

}

pos=1;

root=create(str,&l);

create\_nullable(root,&pos);

printf("NULLABLE TABLE\nElement\tFPOS\tLPOS\n"); print\_nullable(root->lc);

print\_follow(pos-2);

dfa();

display\_dfa();

}

**Output:-**

**Result:-** The program to convert Regular Expression(R.E.) to Deterministic Finite Automata(D.F.A.) was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 5

**AIM:-** Program to compute FIRST and FOLLOW sets

**Algorithm:-**

FIRST(X) for all grammar symbols X:

1. If X is terminal, FIRST(X) = {X}.

2. If X → ε is a production, then add ε to FIRST(X).

3. If X is a non-terminal, and X → Y1 Y2 … Yk is a production, and ε is in all of FIRST(Y1 ), …, FIRST(Yk ), then add ε to FIRST(X).

4. If X is a non-terminal, and X → Y1 Y2 … Yk is a production, then add a to FIRST(X) if for some i, a is in FIRST(Yi), and ε is in all of FIRST(Y1 ), …, FIRST(Yi-1 ).

FOLLOW(A) for all non-terminals A:

1. If $ is the input end-marker, and S is the start symbol, $ ∈ FOLLOW(S). 2. If there is a production, A → αBβ, then (FIRST(β) – ε) ⊆ FOLLOW(B). 3. If there is a production, A → αB, or a production A → αBβ, where ε ∈ FIRST(β), then FOLLOW(A) ⊆ FOLLOW(B).

**Program Code:-**

#include<stdio.h>

#include<conio.h>

#include<String.h>

int n,m=0,p,i=0,j=0;

char a[10][10],f[10];

void follow(char c);

void first(char c);

int main(){

int i,z;

char c,ch;

clrscr();

printf("Enter the no of prooductions:\n");

scanf("%d",&n);

printf("Enter the productions:\n");

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

scanf("%s%c",a[i],&ch);

do{

m=0;

printf("Enter the elemets whose fisrt & follow is to be found:"); scanf("%c",&c);

first(c);

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

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

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

printf("}\n");

strcpy(f," ");

flushall();

m=0;

follow(c);

printf("Follow(%c)={",c); for(i=0;i<m;i++)

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

printf("}\n");

printf("Continue(0/1)?"); scanf("%d%c",&z,&ch); }while(z==1);

return(0);

}

void first(char c)

{

int k;

if(!isupper(c))

f[m++]=c;

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

{

if(a[k][0]==c)

{

if(a[k][2]=='$')

follow(a[k][0]);

else if(islower(a[k][2])) f[m++]=a[k][2];

else first(a[k][2]);

}

}

}

void follow(char c)

{

if(a[0][0]==c)

f[m++]='$';

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

{

for(j=2;j<strlen(a[i]);j++) {

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

{

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

first(a[i][j+1]);

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

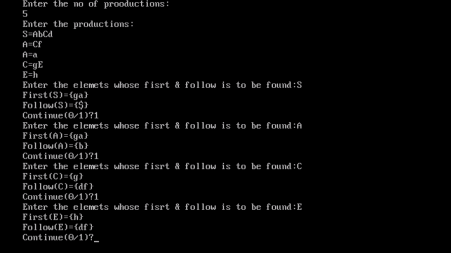
}

}

}

}

**Output:-**

****

**Result:-** The program to compute FIRST and FOLLOW sets was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 6

**AIM:-** Program to compute Leading and Trailing sets

**Algorithm:-**

**Lead** :- Lead is a list of all those terminals symbols("operators") which can appear first on any right hand side of a production.

For each non-terminal i.e. left hand side,a Lead list containing the first terminal in each production for that non-terminal.Where a non-terminal is the first symbol on the right hand side, include both it and the first terminal following.e.g. for

X → a.... / Bc

includes a,c and B in X's Lead List.

**Trail** :- Trail or Last is a similar list of those terminals which can appear Last.

For each non-terminal i.e. left hand side,a Last or trail list containing the last terminal in each production for that non terminal.Where a non-terminal is the last symbol on the right hand side, include both it and the last terminal .e.g. for

Y → ....u / ....vW

includes u,v and W in Last or Trail list.

**Program Code:-**

#include<iostream.h>

#include<string.h>

#include<conio.h>

int nt,t,top=0;

char s[50],NT[10],T[10],st[50],l[10][10],tr[50][50]; int searchnt(char a)

{

int count=-1,i;

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

{

if(NT[i]==a)

return i;

}

return count;

}

int searchter(char a)

{

int count=-1,i;

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

{

if(T[i]==a)

return i;

}

return count;

}

void push(char a) {

s[top]=a;

top++;

}

char pop()

{

top--;

return s[top];

}

void installl(int a,int b)

{

if(l[a][b]=='f')

{

l[a][b]='t';

push(T[b]);

push(NT[a]);

}

}

void installt(int a,int b) {

if(tr[a][b]=='f')

{

tr[a][b]='t';

push(T[b]);

push(NT[a]);

}

}

int main()

{

int i,s,k,j,n;

char pr[30][30],b,c;

clrscr();

cout<<"Enter the no of productions:"; cin>>n;

cout<<"Enter the productions one by one\n"; for(i=0;i<n;i++)

cin>>pr[i];

nt=0;

t=0;

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

{

if((searchnt(pr[i][0]))==-1)

NT[nt++]=pr[i][0];

}

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

{

for(j=3;j<strlen(pr[i]);j++)

{

if(searchnt(pr[i][j])==-1)

{

if(searchter(pr[i][j])==-1)

T[t++]=pr[i][j];

}

}

}

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

{

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

l[i][j]='f';

}

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

{

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

tr[i][j]='f';

}

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

{

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

{

if(NT[(searchnt(pr[j][0]))]==NT[i])

{

if(searchter(pr[j][3])!=-1)

installl(searchnt(pr[j][0]),searchter(pr[j][3])); else

{

for(k=3;k<strlen(pr[j]);k++)

{

if(searchnt(pr[j][k])==-1)

{

installl(searchnt(pr[j][0]),searchter(pr[j][k])); break;

}

}

}

}

}

}

while(top!=0)

{

b=pop();

c=pop();

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

{

if(pr[s][3]==b)

installl(searchnt(pr[s][0]),searchter(c)); }

}

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

{

cout<<"Leading["<<NT[i]<<"]"<<"\t{";

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

{

if(l[i][j]=='t')

cout<<T[j]<<",";

}

cout<<"}\n";

}

top=0;

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

{

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

{

if(NT[searchnt(pr[j][0])]==NT[i])

{

if(searchter(pr[j][strlen(pr[j])-1])!=-1)

installt(searchnt(pr[j][0]),searchter(pr[j][strlen(pr[j])-1])); else

{

for(k=(strlen(pr[j])-1);k>=3;k--)

{

if(searchnt(pr[j][k])==-1)

{

installt(searchnt(pr[j][0]),searchter(pr[j][k])); break;

}

}

}

}

}

}

while(top!=0)

{

b=pop();

c=pop();

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

{

if(pr[s][3]==b)

installt(searchnt(pr[s][0]),searchter(c)); }

}

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

{

cout<<"Trailing["<<NT[i]<<"]"<<"\t{"; for(j=0;j<t;j++)

{

if(tr[i][j]=='t')

cout<<T[j]<<",";

}

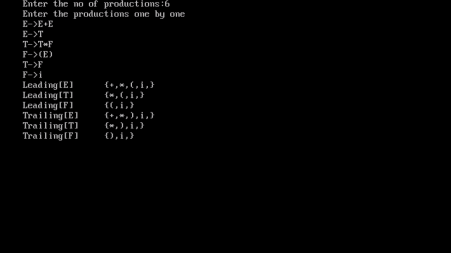
cout<<"}\n";

}

getch();

}

**Output:-**

****

**Result:-** The program to compute Leading and Trailing sets was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 7

**AIM:-** Program for the construction of predictive parsing table

**Algorithm:-**

Repeat: For each production A ∈ ∈of the grammar do For each terminal in FIRST( ∈)

add A ∈ ∈ to M[A, a]

if FIRST( ∈) contains ∈

add A ∈ ∈ to M[A, b] for each bin FOLLOW(A)

if ∈is in FIRST( ∈) and $ is in FOLLOW(A)

add A ∈ ∈ to M[A,$ ]

make each undefined entry of M be error

**Program Code:-**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char fin[10][20],st[10][20],ft[20][20],fol[20][20]; int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;

clrscr();

printf("enter the no. of coordinates\n"); scanf("%d",&n);

printf("enter the productions in a grammar\n"); for(i=0;i<n;i++)

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

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

fol[i][0]='\0';

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

{

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

{

j=3;

l=0;

a=0;

l1:if(!((st[i][j]>64)&&(st[i][j]<91)))

{

for(m=0;m<l;m++)

{

if(ft[i][m]==st[i][j])

goto s1;

}

ft[i][l]=st[i][j];

l=l+1;

s1:j=j+1;

}

else

{

if(s>0)

{

while(st[i][j]!=st[a][0])

{

a++;

}

b=0;

while(ft[a][b]!='\0')

{

for(m=0;m<l;m++)

{

if(ft[i][m]==ft[a][b])

goto s2;

}

ft[i][l]=ft[a][b];

l=l+1;

s2:b=b+1;

}

}

}

while(st[i][j]!='\0')

{

if(st[i][j]=='|')

{

j=j+1;

goto l1;

}

j=j+1;

}

ft[i][l]='\0';

}

}

printf("first pos\n");

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

printf("FIRS[%c]=%s\n",st[i][0],ft[i]); fol[0][0]='$';

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

{

k=0;

j=3;

if(i==0)

l=1;

else

l=0;

k1:while((st[i][0]!=st[k][j])&&(k<n)) {

if(st[k][j]=='\0')

{

k++;

j=2;

}

j++;

}

j=j+1;

if(st[i][0]==st[k][j-1])

{

if((st[k][j]!='|')&&(st[k][j]!='\0'))

{

a=0;

if(!((st[k][j]>64)&&(st[k][j]<91))) {

for(m=0;m<l;m++)

{

if(fol[i][m]==st[k][j])

goto q3;

}

fol[i][l]=st[k][j];

l++;

q3:

}

else

{

while(st[k][j]!=st[a][0])

{

a++;

}

p=0;

while(ft[a][p]!='\0')

{

if(ft[a][p]!='@')

{

for(m=0;m<l;m++)

{

if(fol[i][m]==ft[a][p])

goto q2;

}

fol[i][l]=ft[a][p];

l=l+1;

}

else

e=1;

q2:p++;

}

if(e==1)

{

e=0;

goto a1;

}

}

}

else

{

a1:c=0;

a=0;

while(st[k][0]!=st[a][0])

{

a++;

}

while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0])) {

for(m=0;m<l;m++)

{

if(fol[i][m]==fol[a][c])

goto q1;

}

fol[i][l]=fol[a][c];

l++;

q1:c++;

}

}

goto k1;

}

fol[i][l]='\0';

}

printf("follow pos\n");

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

printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]); printf("\n");

s=0;

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

{

j=3;

while(st[i][j]!='\0')

{

if((st[i][j-1]=='|')||(j==3))

{

for(p=0;p<=2;p++)

{

fin[s][p]=st[i][p];

}

t=j;

for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++) {

fin[s][p]=st[i][j];

j++;

}

fin[s][p]='\0';

if(st[i][k]=='@')

{

b=0;

a=0;

while(st[a][0]!=st[i][0])

{

a++;

}

while(fol[a][b]!='\0')

{

printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]); b++;

}

}

else if(!((st[i][t]>64)&&(st[i][t]<91))) printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]); else

{

b=0;

a=0;

while(st[a][0]!=st[i][3])

{

a++;

}

while(ft[a][b]!='\0')

{

printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]); b++;

}

}

s++;

}

if(st[i][j]=='|')

j++;

}

}

getch();

}

**Output:-**

****

**Result:-** The program for the construction of predictive parsing table was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 8

**AIM:-** Program for recursive descent parsing

**Algorithm:-**

1. Read the input string.

2. Write procedures for the non terminals

3. Verify the next token equals to non terminals if it satisfies match the non terminal.

4. If the input string does not match print error.

**Program Code:-**

#include <stdio.h>

#include <conio.h>

char input[100];

char prod[100][100];

int pos=-1,l,st=-1;

char id,num;

void E();

void T();

void F();

void advance();

void Td();

void Ed();

void advance()

{

pos++;

if(pos<l)

{

if(input[pos]>='0'&& input[pos]<='9')

{

num=input[pos];

id='\0';

}

if((input[pos]>='a' || input[pos]>='A')&&(input[pos]<='z' || input[pos]<='Z'))

{id=input[pos];

num='\0';

}

}

}

void E()

{

strcpy(prod[++st],"E->TE'"); T();

Ed();

}

void Ed()

{

int p=1;

if(input[pos]=='+')

{

p=0;

strcpy(prod[++st],"E'->+TE'"); advance();

T();

Ed();

}

if(input[pos]=='-')

{ p=0;

strcpy(prod[++st],"E'->-TE'"); advance();

T();

Ed();

}

// Recursive Descent Parser

if(p==1)

{

strcpy(prod[++st],"E'->null");

}

}

void T()

{

strcpy(prod[++st],"T->FT'");

F();

Td();

}

void Td()

{

int p=1;

if(input[pos]=='\*')

{

p=0;

strcpy(prod[++st],"T'->\*FT'");

advance();

F();

Td();

}

if(input[pos]=='/')

{ p=0;

strcpy(prod[++st],"T'->/FT'"); advance();

F();

Td();

}

if(p==1)

strcpy(prod[++st],"T'->null"); }

void F()

{

if(input[pos]==id) {

strcpy(prod[++st],"F->id"); advance(); }

if(input[pos]=='(')

{

strcpy(prod[++st],"F->(E)"); advance();

E();

if(input[pos]==')') {

//strcpy(prod[++st],"F->(E)"); advance(); }

}

if(input[pos]==num)

{

strcpy(prod[++st],"F->num");

advance();

}

}

int main()

{

int i;

printf("Enter Input String "); scanf("%s",input);

l=strlen(input);

input[l]='$';

advance();

E();

if(pos==l)

{

printf("String Accepted\n"); for(i=0;i<=st;i++)

{

printf("%s\n",prod[i]); }

}

else

{

printf("String rejected\n"); }

getch();

return 0;

}

**Output:-**

****

**Result:-** The program for recursive descent parsing was successfully executed and the output was verified.

**EXPERIMENT NO.:-** 9

**AIM:-** Program for shift reduce parsing parsing

**Algorithm:-**

1. Get the input expression and store it in the input buffer.

2. Read the data from the input buffer one at the time.

3. Using stack and push & pop operation shift and reduce symbols with respect to production rules available.

4. Continue the process till symbol shift and production rule reduce reaches the start symbol.

5. Display the Stack Implementation table with corresponding Stack actions with input symbols.

**Program Code:-**

#include"stdio.h"

#include"stdlib.h"

#include"conio.h"

#include"string.h"

char ip\_sym[15],stack[15];

int ip\_ptr=0,st\_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main()

{

clrscr();

printf("\n\t\t SHIFT REDUCE PARSER\n");

printf("\n GRAMMER\n");

printf("\n E->E+E\n E->E/E");

printf("\n E->E\*E\n E->a/b");

printf("\n enter the input symbol:\t");

gets(ip\_sym);

printf("\n\t stack implementation table"); printf("\n stack\t\t input symbol\t\t action"); printf("\n\_\_\_\_\_\_\t\t \_\_\_\_\_\_\_\_\_\_\_\_\t\t \_\_\_\_\_\_\n"); printf("\n $\t\t%s$\t\t\t--",ip\_sym);

strcpy(act,"shift ");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

len=strlen(ip\_sym);

for(i=0;i<=len-1;i++)

{

stack[st\_ptr]=ip\_sym[ip\_ptr];

stack[st\_ptr+1]='\0';

ip\_sym[ip\_ptr]=' ';

ip\_ptr++;

printf("\n $%s\t\t%s$\t\t\t%s",stack,ip\_sym,act); strcpy(act,"shift ");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st\_ptr++;

}

st\_ptr++;

check();

}

void check()

{

int flag=0;

temp2[0]=stack[st\_ptr];

temp2[1]='\0';

if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b"))) {

stack[st\_ptr]='E';

if(!strcmpi(temp2,"a"))

printf("\n $%s\t\t%s$\t\t\tE->a",stack,

ip\_sym); else

printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip\_sym);

flag=1;

}

if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"\*"))||(!strcmpi(temp2,"/"))) {

flag=1;

}

if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E\*E"))) {

strcpy(stack,"E");

st\_ptr=0;

if(!strcmpi(stack,"E+E"))

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

else

if(!strcmpi(stack,"E\E"))

printf("\n $%s\t\t %s$\t\t\tE-

>E\E",stack,ip\_sym); else

printf("\n $%s\t\t%s$\t\t\tE->E\*E",stack,ip\_sym);

flag=1;

}

if(!strcmpi(stack,"E")&&ip\_ptr==len)

{

printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip\_sym);

getch();

exit(0);

}

if(flag==0)

{

printf("\n%s\t\t\t%s\t\t reject",stack,ip\_sym); exit(0);

}

return;

}

**Output:-**

**Result:-** The program for shift reduce parsing was successfully executed and the output was verified.