**KRISHNA ENGINEERING COLLEGE**

95, Loni Road, Mohan Nagar, Ghaziabad (Uttar Pradesh), Pin-201007

# 

**Bachelor of Technology**

**In**

**Computer Science And Engineering**

Practical File

Of

# Compiler Design Lab

(RCS-652)

**KRISHNA ENGINEERING COLLEGE**

**Department of Computer Science & Engineering**

**COMPILER DESIGN LAB (RCS-652)**

****

**SUBMITTED TO : MR**. PRASHANT NARESH

**SUBMITTED BY :**PALLAVI SHARMA

**ROLLNO :1816110131**

**YEAR :**3RD YEAR

**SECTION :**3C

**KRISHNA ENGINEERING COLLEGE**

**Department of Computer Science & Engineering**

**INDEX**

**COMPILER DESIGN LAB (RCS-652)**

|  |  |
| --- | --- |
| S.No. | LIST OF PRACTICALS |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | WAP to check whether the entered string is accepted or not for a given grammar. (**3-4**)  WAP to convert infix expression to postfix expression. (**5-8**)  WAP to convert infix expression to prefix expression. (**9-12**)  WAP to find the no. of tokens and list them according to their category in an expression (given/entered) (**13-15**)  WAP to construct an NFA from a regular expression (given) and display the transition table of NFA constructed. (**16-18**)  WAP to compute LEADING and TRAILING sets of a grammar. (**19-21**)  WAP to calculate FIRST and FOLLOW (**22-25**)  WAP in C to check whether the Grammar is Left-recursive and remove left recursion. (**26-28**)  WAP in C to draw a SLR parsing table for a given grammar. (**29-46**)  WAP in C to draw an operator precedence parsing table for the given grammar. (**47-51**)  WAP in C to draw a LL parsing table for a given grammar. (**52-56**) |

**PRACTICAL 1**

**AIM: WAP to check whether the entered string is accepted or not for a given grammar.**

**PROGRAM:**

Strings acceptable by grammar are of form: ab\*c(a+b)

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

char a[100];

int n, i;

void main()

{

clrscr();

printf(“\n enter string”);

scanf(“%s”,&a);

n=strlen(a);

if(a[0]==’a’ && (a[n-1]==’a’ || a[n-1]==’b’) && a[n-2]==’c’)

{

for(i=1; i<n-2; i++)

{

if(a[i]!=’b’)

{

printf(“\n string is not accepted”);

getch();

exit(0);

}

}

printf(“\n string is accepted”);

}

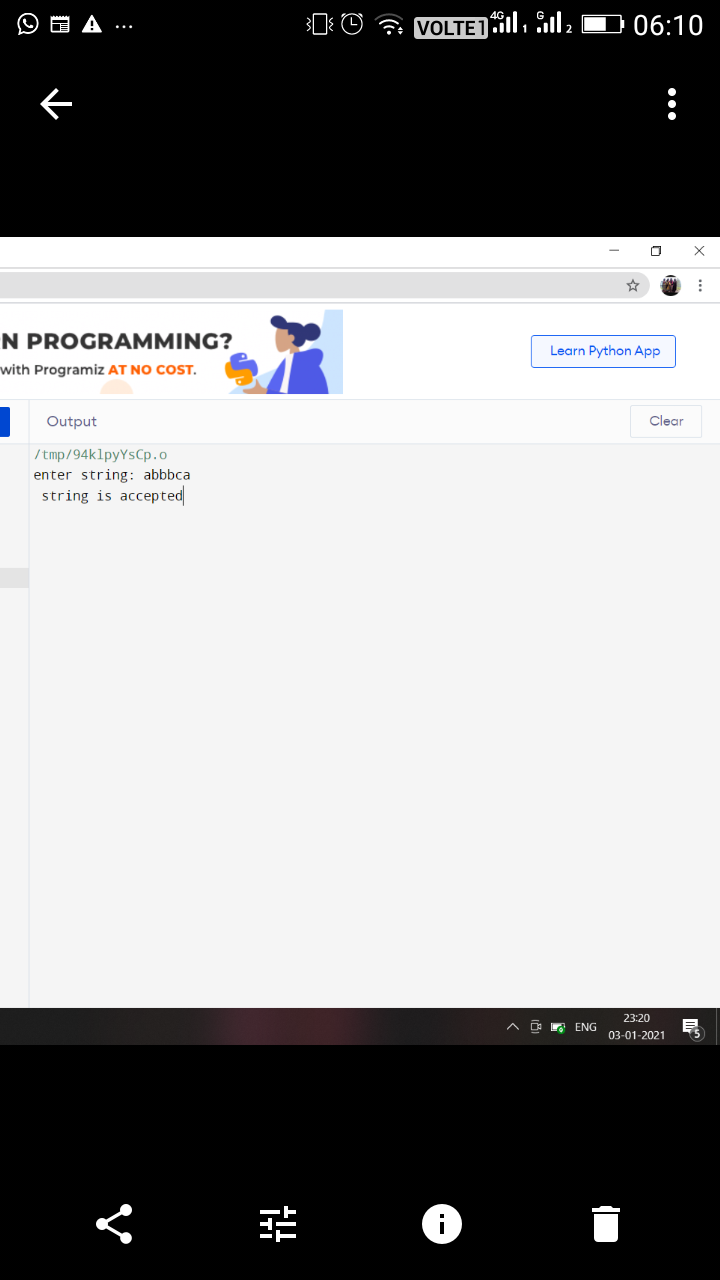
else

printf(“\n string is not accepted”);

getch();

}

**Output:**



**PRACTICAL 2**

**AIM: WAP to convert infix expression to postfix expression.**

Expression: A+(C\*D)\*F

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char str[]=”A+(C\*D)\*F”;

char stack[10];

int top=-1;

void push(char s)

{

top=top+1;

stack[top]=s;

}

char pop()

{

char item;

item=stack[top];

top--;

return(item);

}

int precede(char c)

{

if(c==47) // Division(/)

return(5);

if(c==42) // Multiplication(\*)

return(4);

if(c==43) //Addition(+)

return(3);

else

return(2);

}

void main()

{

char postfix[10];

int l, i=0, j=0;

char s, temp;

printf(“infix string: “);

puts(str);

l=strlen(str);

push(‘#’);

while(i<l)

{

s=str[i];

switch(s)

{

case ‘(‘:

push(s);

break;

case ‘)’:

temp=pop();

while(temp!=’(‘)

{

postfix[j]=temp;

j++;

temp=pop();

}

break;

case ‘+’:

case ‘-‘:

case ‘\*’:

case’/’:

while(precede(stack[top])>=precede(s))

{

temp=pop();

postfix[j]=temp;

j++;

}

push(s);

break;

default:

postfix[j++]=s;

break;

}

i++;

}

while(top>0)

{

temp=pop();

postfix[j++]=temp;

}

postfix[j++]=’\0’;

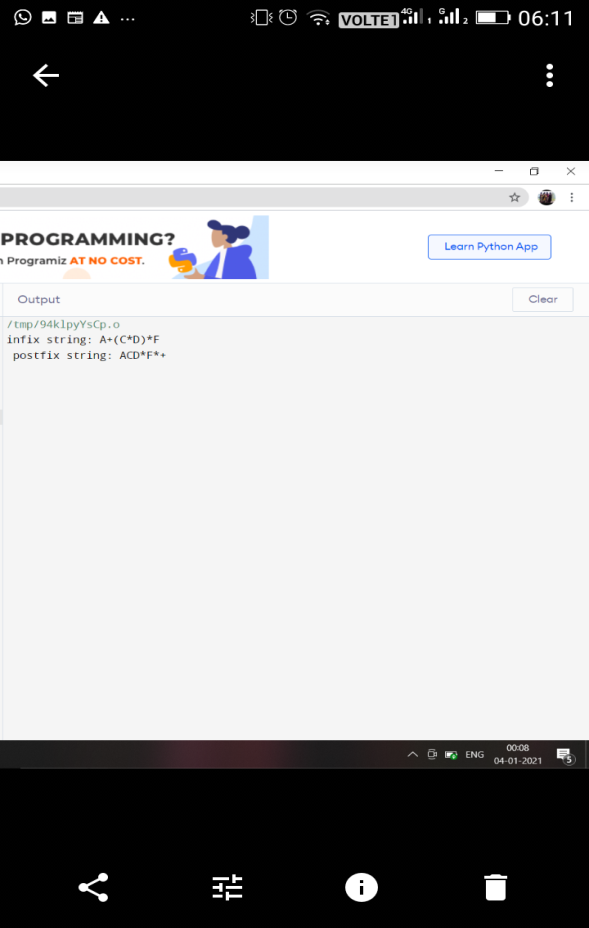
printf("\npostfix string");

puts(postfix);

getch();

}

**Output:**



**PRACTICAL 3**

**AIM: WAP to convert infix expression to prefix expression.**

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char str1[]="A+(C\*D)\*F";

char str[]="F\*(D\*C)+A";

char stack[10];

int top=-1;

void push(char s)

{

top=top+1;

stack[top]=s;

}

char pop()

{

char item;

item=stack[top];

top--;

return(item);

}

int precede(char c)

{

if(c==47) // Division(/)

return(5);

if(c==42) // Multiplication(\*)

return(4);

if(c==43) //Addition(+)

return(3);

else

return(2);

}

void main()

{

char prefix[10];

int l, i=0, j=0;

char s, temp;

printf("infix string: ");

puts(str);

l=strlen(str);

push('#');

while(i<l)

{

s=str[i];

switch(s)

{

case '(':

push(s);

break;

case ')':

temp=pop();

while(temp!='(')

{

prefix[j]=temp;

j++;

temp=pop();

}

break;

case '+':

case '-':

case '\*':

case '/':

while(precede(stack[top])>=precede(s))

{

temp=pop();

prefix[j]=temp;

j++;

}

push(s);

break;

default:

prefix[j++]=s;

break;

}

i++;

}

while(top>0)

{

temp=pop();

prefix[j++]=temp;

}

prefix[j++]='\0';

printf("\nprefix string");

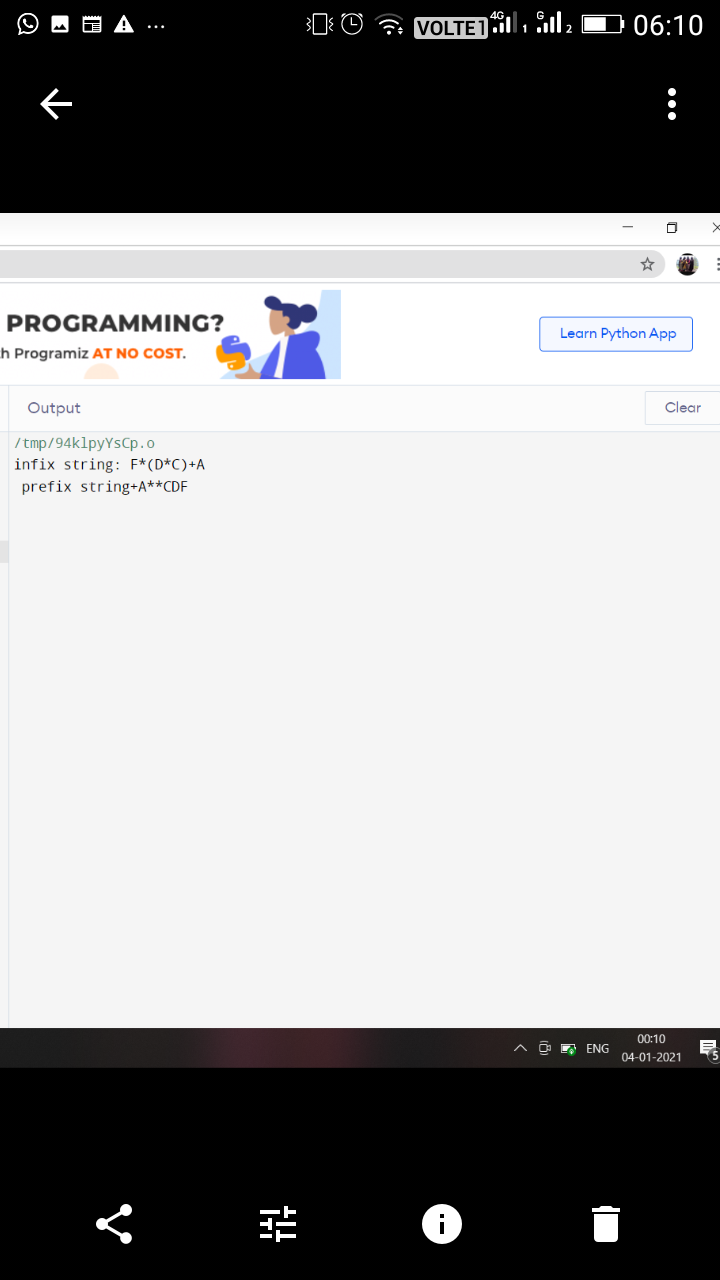
for(i=6;i>=0;i--)

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

getch();

}

**Output:**

**PRACTICAL 4**

**AIM: WAP to find the no. of tokens and list them according to their category in an expression (given/entered)**

**PROGRAM:**

Eg: a= b+c\*23-56^2

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

int con=0, var=0, op=0;

void check(char c)

{

if(isalpha(c))

var++;

if(c==47||c==42||c==43||c==45||c==61||c==94)

op++;

}

/\* ASCII values:

/ -> 47

\* -> 42

+ -> 43

- -> 45

= -> 61

^ -> 94

\*/

void main()

{

char str[13];

char c;

clrscr();

printf("\nenter string");

scanf("%s", &str);

for(int i=0; i<13; i++)

{

c=str[i];

check(c);

}

for(i=0; i<13; i++)

{

if(isdigit(str[i])&&isdigit(str[i+1]))

{

i=i+2;

con++;

}

else if(isdigit(str[i]))

con++;

}

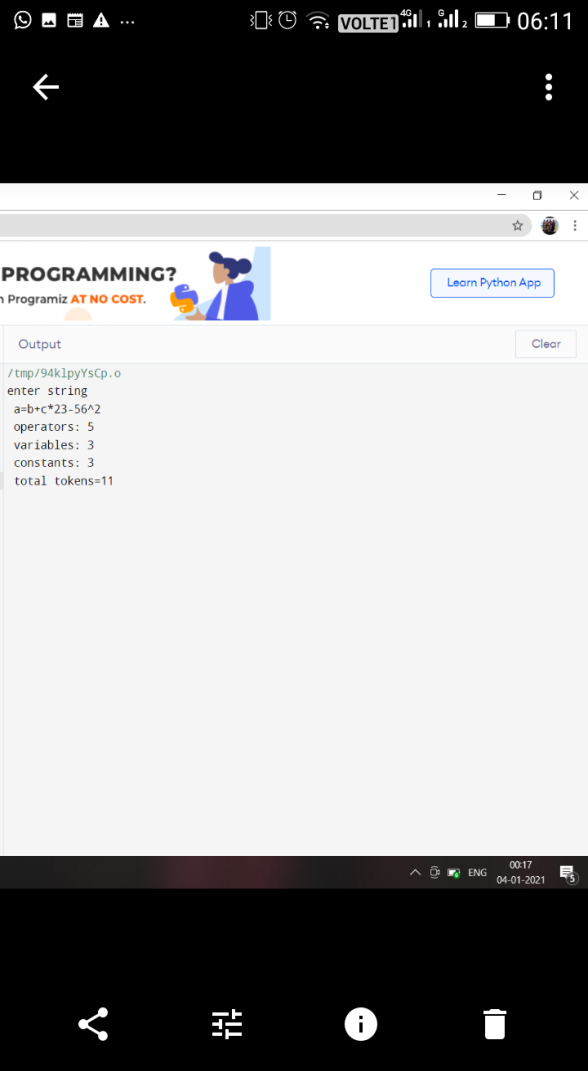
printf("\n operators: %d \nvariables: %d \nconstants: %d" , op, var, con);

printf("\ntotal tokens=%d", op+var+con);

getch();

}

**Output:**

****

**PRACTICAL 5**

**AIM: WAP to construct an NFA from a regular expression (given) and display the transition table of NFA constructed.**

1. What is FSM.
2. What is transition diagram.
3. What is E transition.
4. What is Thomsson rule.

Given regular expression: (a/b)\*

**PROGRAM:**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

void main()

{

clrscr();

char s[10];

int n,init=0,fin=1;

cout<<"enter R.E\n";

gets(s);

n=strlen(s);

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

{

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

fin+=2;

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

fin+=1;

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

fin+=4;

}

char c=238;

i=0;

int ch;

if(s[0]>=97&&s[0]<=122)

ch=1;

if(s[0]=='('&&s[4]==')')

ch=2;

switch(ch)

{

case 1:

if(s[i+1]=='/')

{

if(s[i+2]>=97 && s[i+2]<=122)

{

cout<<"\n"<<init+2<<"--"<<s[i]<<"-->"<<init+3;

cout<<"\n"<<init+4<<"--"<<s[i+2]<<"-->"<<init+5;

goto pt1;

}

}

case 2:

if(s[i+1]>=97 && s[i+1]<=122)

if(s[i+2]=='/')

{

if(s[i+3]>=97 && s[i+3]<=122)

{

cout<<"\n"<<init+2<<"--"<<s[i+1]<<"-->"<<init+3;

cout<<"\n"<<init+4<<"--"<<s[i+3]<<"-->"<<init+5;

if(s[i+5]=='\*')

{

goto pt;

}

else

goto pt1;

}

}

}

pt:

cout<<"\n"<<init<<"--"<<c<<"-->"<<init+1;

cout<<"\n"<<init<<"--"<<c<<"-->"<<fin;

pt1:

cout<<"\n"<<init+1<<"--"<<c<<"-->"<<init+2;

cout<<"\n"<<init+1<<"--"<<c<<"-->"<<init+4;

cout<<"\n"<<init+3<<"--"<<c<<"-->"<<init+6;

cout<<"\n"<<init+5<<"--"<<c<<"-->"<<init+6;

cout<<"\n"<<init+6<<"--"<<c<<"-->"<<init+1;

cout<<"\n"<<init+6<<"--"<<c<<"-->"<<fin;

getch();

}

1. What is NFA

**Output:**

**PRACTICAL 6**

**AIM: WAP to compute LEADING and TRAILING sets of a grammar(given).**

**Grammar: E🡪 E+T | T**

**T🡪 T\*F | F**

**F🡪 (E) | id**

**PROGRAM :**

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

char s,l[20],r[10],lead[10],trail[10];

int n,j,m;

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

{

lead[i]=NULL;

trail[i]=NULL;

}

cout<<"\nenter total no. of productions";

cin>>n;

int k=0;

m=0;

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

{

cout<<"\nenter the LHS of production";

cin>>l[i];

cout<<"\nenter the RHS of production";

cin>>r;

for(int j=0;j<2;j++)

{

if((r[j]=='(') || r[j]==')' || r[j]=='\*' || r[j]=='+' || r[j]=='-' || r[j]=='/' )

{

lead[k]=r[j];

k=k+1;

}

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

{

lead[k]=r[j];

lead[k+1]=r[j+1];

k=k+1;

}

}

for(j=1;j<=2;j++)

{

if((r[j]=='(') || r[j]==')' || r[j]=='\*' || r[j]=='+' || r[j]=='-' || r[j]=='/' )

{

trail[m]=r[j];

m=m+1;

}

if((r[j-1]=='i') && (r[j]=='d'))

{

trail[m]=r[j-1];

trail[m+1]=r[j];

m=m+1;

}

}

}

cout<<"\nthe Leading(A) is :\n";

cout<<"{ ";

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

{

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

cout<<lead[i]<<lead[i+1]<<" ";

else

cout<<lead[i]<<" ";

}

cout<<"}";

cout<<"\nthe Trailing(A) is :\n";

cout<<"{ ";

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

{

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

cout<<trail[i]<<trail[i+1]<<" ";

else

cout<<trail[i]<<" ";

}

cout<<"}";

getch();

}

**Output:**

****

**PRACTICAL 7**

**AIM: WAP to calculate FIRST and FOLLOW.**

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

char FT[5];

char FL[5];

void checkfirst(char x)

{

int i=0;

switch(x)

{

case 'a':

FT[i]='a'; i++;

break;

case 'b':

FT[i]='b'; i++;

break;

case 'e':

FT[i]='e'; i++;

break;

case ')':

FT[i]=')'; i++;

break;

case 'i':

FT[i]='i'; i++;

break;

case '@':

FT[i]='@'; i++;

break;

}

}

void checkfollow(char x)

{

int i=0;

switch(x)

{

case 'a':

FT[i]='a'; i++;

break;

case 'b':

FT[i]='b'; i++;

break;

case 'e':

FT[i]='e'; i++;

break;

case 't':

FL[i]='t'; i++;

break;

case 'i':

FT[i]='i'; i++;

break;

case '@':

FT[i]='@'; i++;

break;

}

}

void first(char y)

{ int i;

checkfirst(y);

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

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

}

void follow(char y)

{ int i;

FL[0]='$';

if(y=='e')

first(y);

checkfollow(y);

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

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

}

void main()

{

int i;

char S1[]="iCtSS'";

char S2[]="a";

char s1[]="eS'";

char s2[]="@";

char C1[]="b";

char X[]="tS";

char t1,t2,e1,e2,c1,x;

t1=S1[0];

t2=S2[0];

e1=s1[0];

e2=s2[0];

c1=C1[0];

x=X[0];

clrscr();

printf("\nFIRST [S]: ");

first(t1);

first(t2);

printf("\n\nFIRST [S']: ");

first(e1);

first(e2);

printf("\n\nFIRST [C]: ");

first(c1);

printf("\n\nFOLLOW [S]: ");

follow(e1);

printf("\n\nFOLLOW [S']: ");

follow(e1);

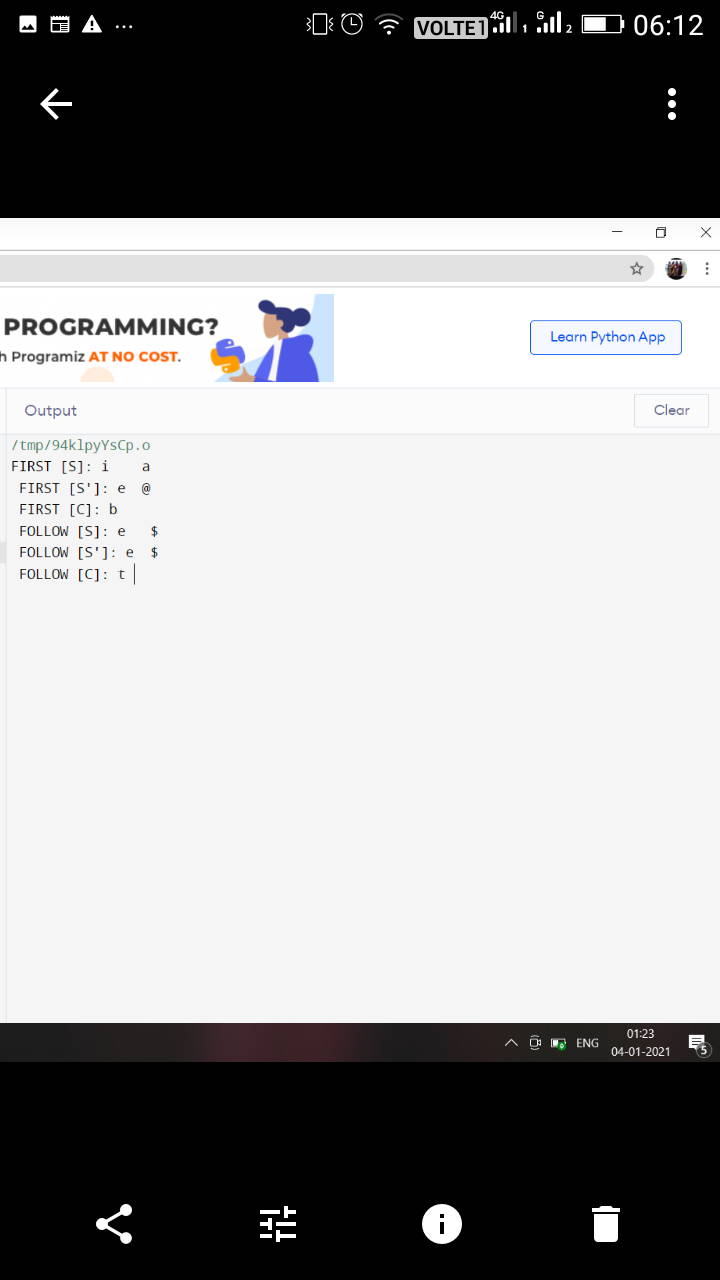
printf("\n\nFOLLOW [C]: ");

follow(x);

getch();

}

**Output:**



**PRACTICAL 8**

**AIM: WAP in C to check whether the Grammar is Left-recursive and remove left recursion.**

**PROGRAM:**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct production

{

char l;

char r[10];

int rear;

};

struct production prod[20],pr\_new[20];

int p=0,b=0,d,f,q,n,flag=0;

char terminal[20],nonterm[20],alpha[10];

char x,epsilon='^';

void main()

{

clrscr();

cout<<"Enter the number of terminals: ";

cin>>d;

cout<<"Enter the terminal symbols for your production: ";

for(int k=0;k<d;k++)

{

cin>>terminal[k];

}

cout<<"\nEnter the number of non-terminals: ";

cin>>f;

cout<<"Enter the non-terminal symbols for your production: ";

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

{

cin>>nonterm[k];

}

cout<<"\nEnter the number of Special characters(except non-terminals): ";

cin>>q;

cout<<"Enter the special characters for your production: ";

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

{

cin>>alpha[k];

}

cout<<"\nEnter the number of productions: ";

cin>>n;

for(k=0;k<=n-1;k++)

{

cout<<"Enter the "<< k+1<<" production: ";

cin>>prod[k].l;

cout<<"->";

cin>>prod[k].r;

prod[k].rear=strlen(prod[k].r);

}

for(int m=0;m<f;m++)

{

x=nonterm[m];

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

{

if((prod[j].l==x)&&(prod[j].r[0]==prod[j].l))

flag=1;

}

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

{

if((prod[i].l==x)&&(prod[i].r[0]!=x)&&(flag==1))

{

pr\_new[b].l=x;

for(int c=0;c<prod[i].rear;c++)

pr\_new[b].r[c]=prod[i].r[c];

pr\_new[b++].r[c]=alpha[p];

}

else if((prod[i].l==x)&&(prod[i].r[0]==x)&&(flag==1))

{

pr\_new[b].l=alpha[p];

for(int a=0;a<=prod[i].rear-2;a++)

pr\_new[b].r[a]=prod[i].r[a+1];

pr\_new[b++].r[a]=alpha[p];

pr\_new[b].l=alpha[p];

pr\_new[b++].r[0]=epsilon;

}

else if((prod[i].l==x)&&(prod[i].r[0]!=x)&&(flag==0))

{

pr\_new[b].l=prod[i].l;

strcpy(pr\_new[b].r,prod[i].r);

b++;

}

}

flag=0;

p++;

}

cout<<"\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n AFTER REMOVING LEFT RECURSION ";

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

for(int s=0;s<=b-1;s++)

{

cout<<"Production "<<s+1<<" is: ";

cout<<pr\_new[s].l;

cout<<"->";

cout<<pr\_new[s].r;

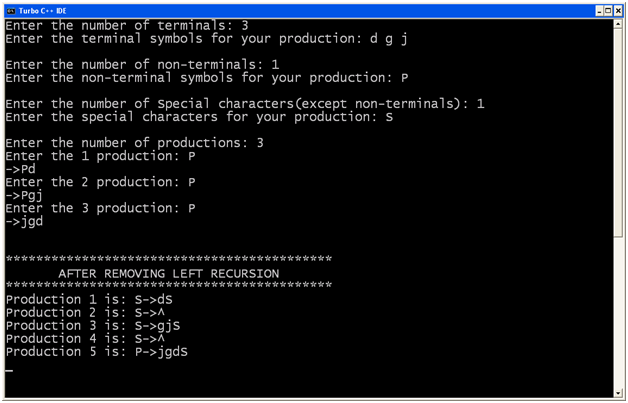
cout<<endl;

}

getche();

}

Output



**PRACTICAL 9**

**AIM: WAP in C to draw a SLR parsing table for a given grammar**

**PROGRAM:**

#include<stdio.h>

#include<ctype.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

#include<iostream.h>

#define epsilon '^'

// since I didn't know how to type epsilon symbol temporily I am using ^

char prod[20][20],T[20],NT[20],c[10][10],foll[10][10],fir[10][10];

int tt,tnt,tp,a;

int follow[20][20],first[20][20];

void first\_of(char);

int count(int j);

void rhs(int j);

void read\_tnt();

int rhs(int j);

void read\_tnt()

{

cout<<"For SLR parser: ";

cout<<"\nEnter number of terminals: ";

cin>>tt;

cout<<"\nEnter terminals: ";

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

T[i]=getche();

getch();

cout<<"\nEnter number of Non-terminals: ";

cin>>tnt;

cout<<"\nEnter Non-terminals: ";

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

NT[i]=getche();

getch();

}

void read\_prod()

{

int j;

char x=0;

cout<<"\n\nEnter number of productions: ";

cin>>tp;

cout<<"\n Enter productions: ";

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

{

j=x=0;

while(x!='\r')

{

prod[i][j]=x=getche();

j++;

}

cout<<"\n";

}

getch();

}

int nt\_no(char n)

{

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

if(NT[i]==n)

return(i);

return(-1);

}

int t\_no(char t)

{

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

if(T[i]==t)

return(i);

if(t=='$')

return(tt);

return(-1);

}

int terminal(char x)

{

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

if(T[i]==x)

return(1);

return(0);

}

int nonterminal(char x)

{

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

if(NT[i]==x)

return(1);

return(0);

}

int in\_rhs(char \*s,char x)

{

for(int i=0;i<=strlen(s);i++)

if(\*(s+i)==x)

return(i);

return(-1);

}

void find\_first()

{

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

first\_of(NT[i]);

}

void first\_of(char n)

{

int t1,t2,p1,cnt=0,i,j;

char x;

static int over[20];

p1=t\_no(epsilon);

if(terminal(n))

return;

t1=nt\_no(n);

if(over[t1])

return;

over[t1]=1;

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

{

t1=nt\_no(prod[i][0]);

if(prod[i][0]==n)

{

int k=0;

cnt=count(1);

rhs(i);

while(k<cnt)

{

x=c[i][k];

if(terminal(x))

{

t2=t\_no(x);

first[t1][t2]=1;

break;

}

else

{

t2=nt\_no(x);

first\_of(x);

for(int j=0;j<tt;j++)

if(p1!=j && first[t2][j])

first[t1][j]=1;

if(p1!=-1 && first[t2][p1])

k++;

else

break;

}

}

if(p1!=-1 && k>=cnt)

first[t1][p1]=1;

}

}

}

void follow\_of(char n)

{

int f,t1,t2,p1,t,cnt=0;

char x,beta;

static int over[20];

p1=t\_no(epsilon);

t1=nt\_no(n);

if(over[t1])

return;

over[t1]=1;

if(NT[0]==n)

follow[nt\_no(NT[0])][tt]=1;

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

{

rhs(i);

cnt=count(i);

t=in\_rhs(c[i],n);

if(t==-1)

continue;

for(int k=t+1;k<=cnt;k++)

{

rhs(i);

beta=c[i][k];

if(terminal(beta))

{

t2=t\_no(beta);

follow[t1][t2]=1;

break;

}

int bno;

for(int j=0;j<tt;j++)

{

bno=nt\_no(beta);

if((first[bno][j]) && (j!=p1))

follow[t1][j]=1;

}

if((p1!=-1) && (first[bno][p1]==1))

continue;

else if((t==(cnt-1)||(k>=cnt)))

{

follow\_of(prod[i][0]);

t1=nt\_no(prod[i][0]);

for(int l=0;l<=tt+1;l++)

if(follow[t][l])

follow[t1][l]=1;

}

}

}

}

int count(int j)

{

int c1=0;

for(int q=3;prod[j][q]!='\r';q++)

c1++;

return(c1);

}

void rhs(int j)

{

int a,h=0;

a=j;

for(int q=3;prod[j][q]!='\r';q++)

{

c[a][h]=prod[j][q];

h++;

}

}

void find\_follow()

{

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

follow\_of(NT[i]);

}

void show\_follow()

{

int b=0;

a=0;

cout<<"\n\n Follow Table For Grammar: \n";

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

{

b=0;

cout<<"\n FOLLOW ("<<NT[i]<<" )= { ";

for(int j=0;j<tt+1;j++)

if(follow[i][j] && j!=tt)

{

foll[a][b]=T[j];

b++;

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

}

else

if(j==tt)

{

foll[a][b]='$';

b++;

cout<<'$';

}

a++;

cout<<" } ";

}

getch();

}

void show\_first()

{

int b=0;

a=0;

cout<<"\n\n First Table For Grammar: \n";

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

{

b=0;

cout<<"\n FIRST ("<<NT[i]<<" )= { ";

for(int j=0;j<tt+1;j++)

if(first[i][j] && j!=tt)

{

fir[a][b]=T[j];

b++;

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

}

a++;

cout<<" } ";

}

getch();

}

void mainf(void)

{

clrscr();

read\_tnt();

read\_prod();

find\_first();

find\_follow();

show\_follow();

show\_first();

}

To construct parse table:

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

#include<iostream.h>

#include"c:\tc\bin\SLR.h"

int S=0,i=0,j=0,state[20];

char TNT[15];

struct node

{

int pno,dpos;

};

struct t

{

char s;

int n;

};

struct t1

{

struct t lr[10];

int gr[5];

};

struct t1 action[15];

struct node closure[10][10];

int g[15][10];

int l;

void sclosure(int,int);

int added(int);

int t\_into(char);

void print\_table(int);

void parser(void);

int find\_index(char);

int t\_ino(char);

void pop(void);

void push(char,int);

void find\_closure(int,int);

void SLR(void);

void main()

{

clrscr();

mainf();

getch();

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

TNT[i]=NT[i];

for(int j=0;j<tt;j++)

{

TNT[i]=T[j];

i++;

}

strcat(T,"$");

i=j=0;

SLR();

print\_table(S);

getch();

// clrscr();

// parser();

// getch();

}

void SLR()

{

int clno,no=0,x,y,z,len,cnt=-1,d=0;

closure[i][j].pno=0;

closure[i][j++].dpos=3;

find\_closure(no,3);

sclosure(i,j);

state[i]=j;

S=0;

do

{

cnt++;

z=state[cnt];

for(int k=0;k<tnt+tt;k++)

{

i++;

j=0;d=0;

for(int l=0;l<z;l++)

{

x=closure[cnt][1].pno;

y=closure[cnt][1].dpos;

if(prod[x][y]==TNT[k])

{

d=1;

closure[i][j].pno=x;

closure[i][j++].dpos=++y;

if((y<strlen(prod[x])) && (isupper(prod[x][y])))

find\_closure(x,y);

}

}

if(d==0)

{

i--;

continue;

}

sclosure(i,j);

state[i]=j;

clno=added(i-1);

if(clno==-1)

clno=i;

if(isupper(TNT[k]))

action[cnt].gr[k]=clno;

else

{

action[cnt].lr[k-tnt].s='S';

action[cnt].lr[k-tnt].n=clno;

}

if(added(i-1)!=-1)

i--;

else

{

S++;

for(l=0;l<state[i];l++)

{

if(closure[i][1].pno==0)

{

action[i].lr[tt].s='A';

continue;

}

len=(strlen(prod[closure[i][l].pno])-1);

if(len==closure[i][l].dpos)

{

char v=prod[closure[i][l].pno][0];

int u=nt\_no(v);

for(x=0;x<strlen(foll[u]);x++)

{

int w=t\_ino(foll[u][x]);

action[i].lr[w].s='R';

action[i].lr[w].n=closure[i][l].pno;

}

}

}

}

}

}

while(cnt!=S);

}

void print\_table(int states)

{

int lin=5;

cout<<"\n\n Parser Table: \n";

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

cout<<"\t"<<T[i];

cout<<"\t$";

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

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

cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

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

{

gotoxy(l,lin);

cout<<"I"<<i<<"\t";

for(int j=0;j<=tt;j++)

{

if(action[i].lr[j].s!='\x0')

{

if(action[i].lr[j].s=='A')

{

cout<<"Acc";

continue;

}

cout<<action[i].lr[j].s;

cout<<action[i].lr[j].n;

cout<<"\t";

}

else

cout<<"\t";

}

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

if(action[i].gr[j])

{

cout<<action[i].gr[j];

cout<<"\t";

}

else

cout<<"\t";

lin++;

cout<<"\n";

}

cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

}

void sclosure(int clno,int prodno)

{

struct node temp;

for(int i=0;i<prodno-1;i++)

{

for(int j=i+1;j<prodno;j++)

{

if(closure[clno][i].pno>closure[clno][j].pno)

{

temp=closure[clno][i];

closure[clno][i]=closure[clno][j];

closure[clno][j]=temp;

}

}

}

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

{

for(j=i+1;j<prodno;j++)

{

if((closure[clno][i].dpos>closure[clno][j].dpos) &&

(closure[clno][i].pno==closure[clno][j].pno))

{

temp=closure[clno][i];

closure[clno][i]=closure[clno][j];

closure[clno][j]=temp;

}

}

}

}

int added(int n)

{

int d=1;

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

{

if(state[k]==state[n+1])

{

d=0;

for(int j=0;j<state[k];j++)

{

if((closure[k][j].pno!=closure[n+1][j].pno) ||

(closure[k][j].dpos!=closure[n+1][j].dpos))

break;

else

d++;

}

if(d==state[k])

return(k);

}

}

return(-1);

}

void find\_closure(int no,int dp)

{

int k;

char temp[5];

if(isupper(prod[no][dp]))

{

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

{

if(prod[k][0]==prod[no][dp])

{

closure[i][j].pno=k;

closure[i][j++].dpos=3;

if(isupper(prod[k][3])&&

(prod[k][3]!=prod[k][0]))

find\_closure(k,3);

}

}

}

return;

}

int t\_ino(char t)

{

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

if(T[i]==t)

return(i);

return(-1);

}

char pops2;

struct node1

{

char s2;int s1;

};

struct node1 stack[10];

int pops1,top=0;

void parser(void)

{

int r,c;

struct t lr[10];

char t,acc='f',str[10];

cout<<"Enter I/p String To Parse: ";

cin>>str;

strcat(str,"$");

stack[0].s1=0;

stack[0].s2='\n';

cout<<"\n\n STACK";

cout<<"\t\t INPUT";

cout<<"\t\t ACTION";

cout<<"\n =====";

cout<<"\t\t =======";

cout<<"\t\t =======";

i=0;

cout<<"\n";

cout<<stack[top].s1;

cout<<" \t\t\t ";

for(int j=0;j<strlen(str);j++)

cout<<str[j];

do

{

r=stack[top].s1;

c=find\_index(str[i]);

if(c==-1)

cout<<"\n Error! Invalid String!";

return;

}

while(top!=0);

switch(action[r],lr[c].s)

{

case 'S':

{

push(str[i],action[r].lr[c].n);

i++;

cout<<"\t\t\t Shift";

break;

}

case 'R':

{

t=prod[action[r].lr[c].n][3];

do

{

pop();

}

while(pops2!=t);

t=prod[action[r].lr[c].n][0];

r=stack[top].s1;

c=find\_index(t);

push(t,action[r].gr[c-tt-1]);

cout<<"\t\t\t Reduce";

break;

}

case 'A':

{

cout<<"\t\t\t Accept";

cout<<"\n\n\n String accepted";

acc='t';

getch();

return;

}

default:

{

cout<<"\n\n\n Error! String not accepted!";

getch();

exit(0);

}

}

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

cout<<stack[j].s2<<stack[j].s1;

if(top<4)

cout<<"\t\t\t";

else

cout<<"\t\t";

for(j=i;j<strlen(str);j++)

cout<<str[j];

if(acc=='t')

return;

}

int find\_index(char temp)

{

for(int i=0;i<=tt+tnt;i++)

{

if(i<=tt)

{

if(T[i]==temp)

return(i);

}

else

if(NT[i-tt-1]==temp)

return(i);

}

return(-1);

}

void push(char t2,int t1)

{

++top;

stack[top].s1=t1;

stack[top].s2=t2;

return;

}

void pop(void)

{

pops1=stack[top].s1;

pops2=stack[top].s2;

--top;

return; }

**Output :**

Enter number of terminals: 5

Enter terminals:+\*()i

Enter number of non-terminals:3

Enter non-terminals:ETF

Enter number of productions:6

Enter productions:

E->E+T

E->T

T->T\*F

T->F

F->(E)

F->i

Follow table:

FOLLOW(E)={+ ) $}

FOLLOW(F)={+ \* ) $}

FOLLOW(T)={ + \* ) $}

First Table :

FIRST(E)={ ( i }

FIRST(E)={ ( i }

FIRST(E)={ ( i }

Expected parse table:

+ \* ( ) i $ E T F

I0 S4 S5 1 2 3

I1 S6 ACC

I2 R1 S7 R1 R1

I3 R3 R3 R3 R3

I4 S4 S5 ACC 8 2 3

I5 R5 R5 R5 R5

I6 ACC

I7 S4 S5 9

I8 S10 S11 ACC

I9 R2 R2 R2 R2

I10 ACC

I11 R4 R4 R4 R4

Enter i/p string: i+i\*i

STACK INPUT ACTION

0 i+i\*i$ Shift

0i5 +i\*i$ Reduce

0F3 +i\*i$ Reduce

0T2 +i\*i$ Reduce

0E1 +i\*i$ Shift

0E1+6 i\*i$ ERROR! STRING NOT ACCEPTED!

**PRACTICAL 10**

**AIM: WAP in C to draw an operator precedence parsing table for the given grammar**

**PROGRAM:**

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

int getOperatorPosition(char );

#define node struct tree1

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);

//node create\_node(char,\*node);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]={'+','\*','(',')',']'};

void 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];

}

elseif(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--;

}

elseif(matrix[row][col] == 2)

{

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

temp = oprate[tos].t;

tos--;

oprate[tos].t = temp;

}

elseif(matrix[row][col] == 3)

{

printf("\nExpression is Invalid...\n");

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

break;

}

}

}

ssm++;

}

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

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

printf("Over");

getch();

getch();

}

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);

elsereturn i;

}

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("< ");

elseif(matrix[i][j] == 1)

printf("> ");

elseif(matrix[i][j] == 2)

printf("= ");

elseif(matrix[i][j] == 3)

printf(" ");

}

printf("\n");

}

}

**OUTPUT:**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Enter Exp : [a+b\*c]

OPERATOR PRECEDENCE MATRIX

===========================

+ \* ( ) ]

+ > < < > >

\* > > < > >

( < < < =

) > > > >

[ < < < =

show tree

a

b

c

\*

+

Over

Enter Exp : [a+(b\*c)+d]

OPERATOR PRECEDENCE MATRIX

===========================

+ \* ( ) ]

+ > < < > >

\* > > < > >

( < < < =

) > > > >

[ < < < =

show tree

a

b

c

\*

+

d

+

Over

Enter Exp : [)]

OPERATOR PRECEDENCE MATRIX

===========================

+ \* ( ) ]

+ > < < > >

\* > > < > >

( < < < =

) > > > >

[ < < < =

**PRACTICAL 11**

**AIM: WAP in C to draw a LL parsing table for a given grammar**

**PROGRAM:**

#include <iostream.h>

#include <conio.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

void main()

{

clrscr();

int i=0,j=0,k=0,m=0,n=0,o=0,o1=0,var=0,l=0,f=0,c=0,f1=0;

char str[30],str1[40]="E",temp[20],temp1[20],temp2[20],tt[20],t3[20];

strcpy(temp1,'\0');

strcpy(temp2,'\0');

char t[10];

char array[6][5][10] = {

"NT", "<id>","+","\*",";",

"E", "Te","Error","Error","Error",

"e", "Error","+Te","Error","\0",

"T", "Vt","Error","Error","Error",

"t", "Error","\0","\*Vt","\0",

"V", "<id>","Error","Error","Error"

};

cout << "\n\tLL(1) PARSER TABLE \n";

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

{

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

{

cout.setf(ios::right);

cout.width(10);

cout<<array[i][j];

}

cout<<endl;

}

cout << endl;

cout << "\n\tENTER THE STRING :";

gets(str);

if(str[strlen(str)-1] != ';')

{

cout << "END OF STRING MARKER SHOULD BE ';'";

getch();

exit(1);

}

cout << "\n\tCHECKING VALIDATION OF THE STRING ";

cout <<"\n\t" << str1;

i=0;

while(i<strlen(str))

{

again:

if(str[i] == ' ' && i<strlen(str))

{

cout << "\n\tSPACES IS NOT ALLOWED IN SOURSE STRING ";

getch();

exit(1);

}

temp[k]=str[i];

temp[k+1]='\0';

f1=0;

again1:

if(i>=strlen(str))

{

getch();

exit(1);

}

for(int l=1;l<=4;l++)

{

if(strcmp(temp,array[0][l])==0)

{

f1=1;

m=0,o=0,var=0,o1=0;

strcpy(temp1,'\0');

strcpy(temp2,'\0');

int len=strlen(str1);

while(m<strlen(str1) && m<strlen(str))

{

if(str1[m]==str[m])

{

var=m+1;

temp2[o1]=str1[m];

m++;

o1++;

}

else

{

if((m+1)<strlen(str1))

{

m++;

temp1[o]=str1[m];

o++;

}

else

m++;

}

}

temp2[o1] = '\0';

temp1[o] = '\0';

t[0] = str1[var];

t[1] = '\0';

for(n=1;n<=5;n++)

{

if(strcmp(array[n][0],t)==0)

break;

}

strcpy(str1,temp2);

strcat(str1,array[n][l]);

strcat(str1,temp1);

cout << "\n\t" <<str1;

getch();

if(strcmp(array[n][l],'\0')==0)

{

if(i==(strlen(str)-1))

{

int len=strlen(str1);

str1[len-1]='\0';

cout << "\n\t"<<str1;

cout << "\n\n\tENTERED STRING IS VALID";

getch();

exit(1);

}

strcpy(temp1,'\0');

strcpy(temp2,'\0');

strcpy(t,'\0');

goto again1;

}

if(strcmp(array[n][l],"Error")==0)

{

cout << "\n\tERROR IN YOUR SOURCE STRING";

getch();

exit(1);

}

strcpy(tt,'\0');

strcpy(tt,array[n][l]);

strcpy(t3,'\0');

f=0;

for(c=0;c<strlen(tt);c++)

{

t3[c]=tt[c];

t3[c+1]='\0';

if(strcmp(t3,temp)==0)

{

f=0;

break;

}

else

f=1;

}

if(f==0)

{

strcpy(temp,'\0');

strcpy(temp1,'\0');

strcpy(temp2,'\0');

strcpy(t,'\0');

i++;

k=0;

goto again;

}

else

{

strcpy(temp1,'\0');

strcpy(temp2,'\0');

strcpy(t,'\0');

goto again1;

}

}

}

i++;

k++;

}

if(f1==0)

cout << "\nENTERED STRING IS INVALID";

else

cout << "\n\n\tENTERED STRING IS VALID";

getch(); }

OUTPUT

\*\*\*\*\*\*\*\*\*

LL(1) PARSER TABLE

NT <id> + \* ;

E Te Error Error Error

e Error +Te Error

T Vt Error Error Error

t Error \*Vt

V <id> Error Error Error

ENTER THE STRING :<id>+<id>\*<id>;

CHECKING VALIDATION OF THE STRING

E

Te

Vte

<id>te

<id>e

<id>+Te

<id>+Vte

<id>+<id>te

<id>+<id>\*Vte

<id>+<id>\*<id>te

<id>+<id>\*<id>e

<id>+<id>\*<id>

ENTERED STRING IS VALID

[/Code]