Compiler Design Lab

(RCS-652)

Laboratory Manual

For

Bachelor of Technology

In

Information Technology

Even Semester

Department of Information Technology

KRISHNA ENGINEERING COLLEGE

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

Course Outcomes of Compiler Design LAB (RCS-652)

1.Abilty to create lexical rules and grammars for a programming language

2.Ability to use Flex or similar tools to create a lexical analyzer and Yacc/Bison tools to create a parser.

3.Ability to implement a lexer without using Flex or any other lexer generation tools.

4.Ability to implement a parser such as a bottom-up SLR parser without using Yacc/Bison or any other compiler-generation tools.

5.Ability to implement semantic rules into a parser that performs attribution while parsing.

6.Abilty to design a compiler for a concise programming language.

KRISHNA ENGINEERING COLLEGE

Department of Computer Science & Engineering

List of Practical's

COMPILER DESIGN LAB (RCS-652)

S.No.	LIST OF PROGRAMS
1.	WAP to check whether the entered string is accepted or not for a given grammar.
2.	WAP to convert infix expression to postfix expression.
3.	WAP to convert infix expression to prefix expression.
4.	WAP to find the no. of tokens and list them according to their category in an expression (given/entered)
5.	WAP to construct an NFA from a regular expression (given) and display the transition table of NFA constructed.
6.	WAP to compute LEADING and TRAILING sets of a grammar (given).
7.	WAP to calculate FIRST and FOLLOW
8.	WAP in C to check whether the Grammar is Left-recursive and remove left recursion
9.	WAP in C to draw a SLR parsing table for a given grammar.
10.	WAP in C to draw an operator precedence parsing table for the given grammar
11.	WAP in C to draw a LL parsing table for a given grammar

Program 1

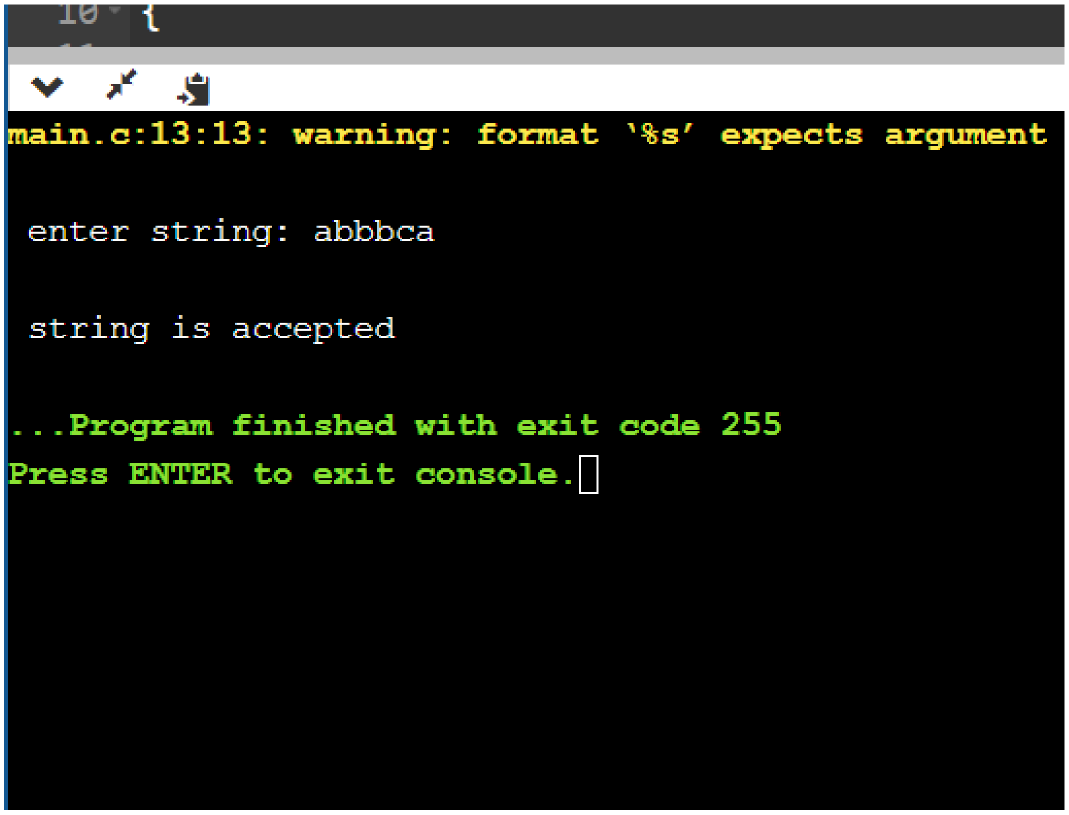
AIM: WAP to check whether the entered string is accepted or not for a given grammar.

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()
{

   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:
enter string: abbbca
 string is accepted



```
{
main.c:13:13: warning: format '%s' expects argument

 enter string: abbbca

 string is accepted

...Program finished with exit code 255
Press ENTER to exit console.
```

Program 2

AIM: WAP to convert infix expression to postfix expression.

Expression: A+(C*D)*F

PROGRAM:

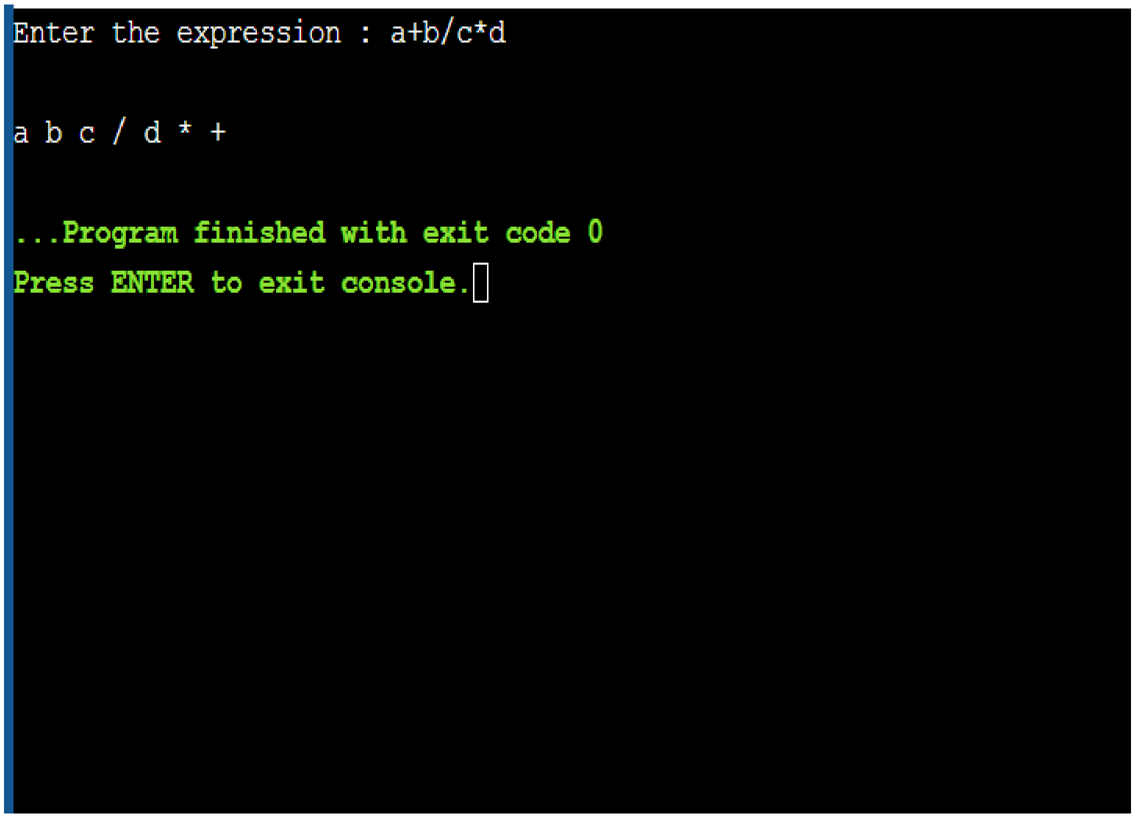
```
#include<stdio.h>
#include<conio.h>
#include<string.h>

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:

```
infix string: a+b/c*d
postfix string: abc/d*+
```



```
Enter the expression : a+b/c*d

a b c / d * +

...Program finished with exit code 0
Press ENTER to exit console.
```

Program 3

AIM: WAP to convert infix expression to prefix expression.

PROGRAM:

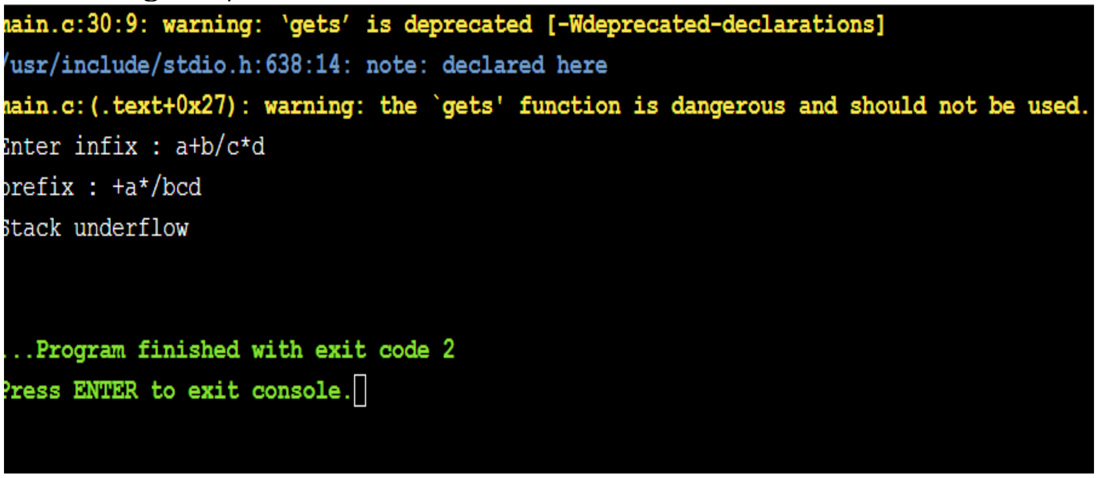
```
#include<stdio.h>
#include<conio.h>
#include<string.h>

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:

infix string: a+b/c*d
prefix string:+a*/bcd



```
nain.c:30:9: warning: 'gets' is deprecated [-Wdeprecated-declarations]
/usr/include/stdio.h:638:14: note: declared here
nain.c:(.text+0x27): warning: the `gets' function is dangerous and should not be used.
Enter infix : a+b/c*d
prefix : +a*/bcd
Stack underflow


...Program finished with exit code 2
Press ENTER to exit console.
```

Program 4

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

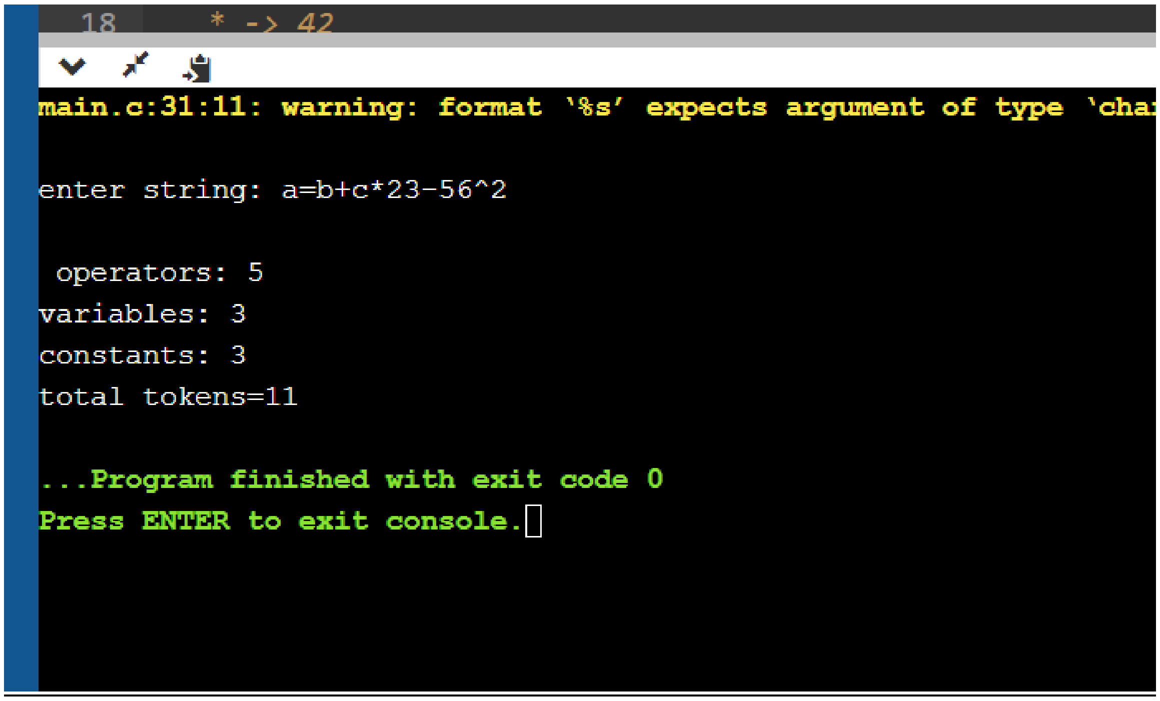
PROGRAM:

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;
 int i=0;
 printf("\nenter string: ");
 scanf("%s", &str);

 for(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();

}
```

<u>Output</u>:

```
enter string
a=b+c*23-56^2

operators: 5
variables: 3
constants: 3
total tokens=11
```



```
18        * -> 42
```

```
main.c:31:11: warning: format '%s' expects argument of type 'cha

enter string: a=b+c*23-56^2

 operators: 5
variables: 3
constants: 3
total tokens=11

...Program finished with exit code 0
Press ENTER to exit console.
```

Program 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

<u>Output</u>:

```
enter R.E
(a/b)*

2--a-->3
4--b-->5
0--î-->1
0--î-->7
1--î-->2
1--î-->4
3--î-->6
5--î-->6
6--î-->1
6--î-->7
```

Program 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();
}
```

<u>Output</u>:

```
enter total no. of productions: 6

enter the LHS of production: E
enter the RHS of production: E+T

enter the LHS of production: T
enter the RHS of production: T*F

enter the LHS of production: T
enter the RHS of production: F

enter the LHS of production: E
enter the RHS of production: T

enter the LHS of production: F
enter the RHS of production: (E)

enter the LHS of production :F
enter the RHS of production: id

the Leading(A) is :
{  +  *  (  id  }
the Trailing(A) is :
{  +  *  )  id  }
```

Program 8

AIM: WAP to calculate FIRST and FOLLOW.

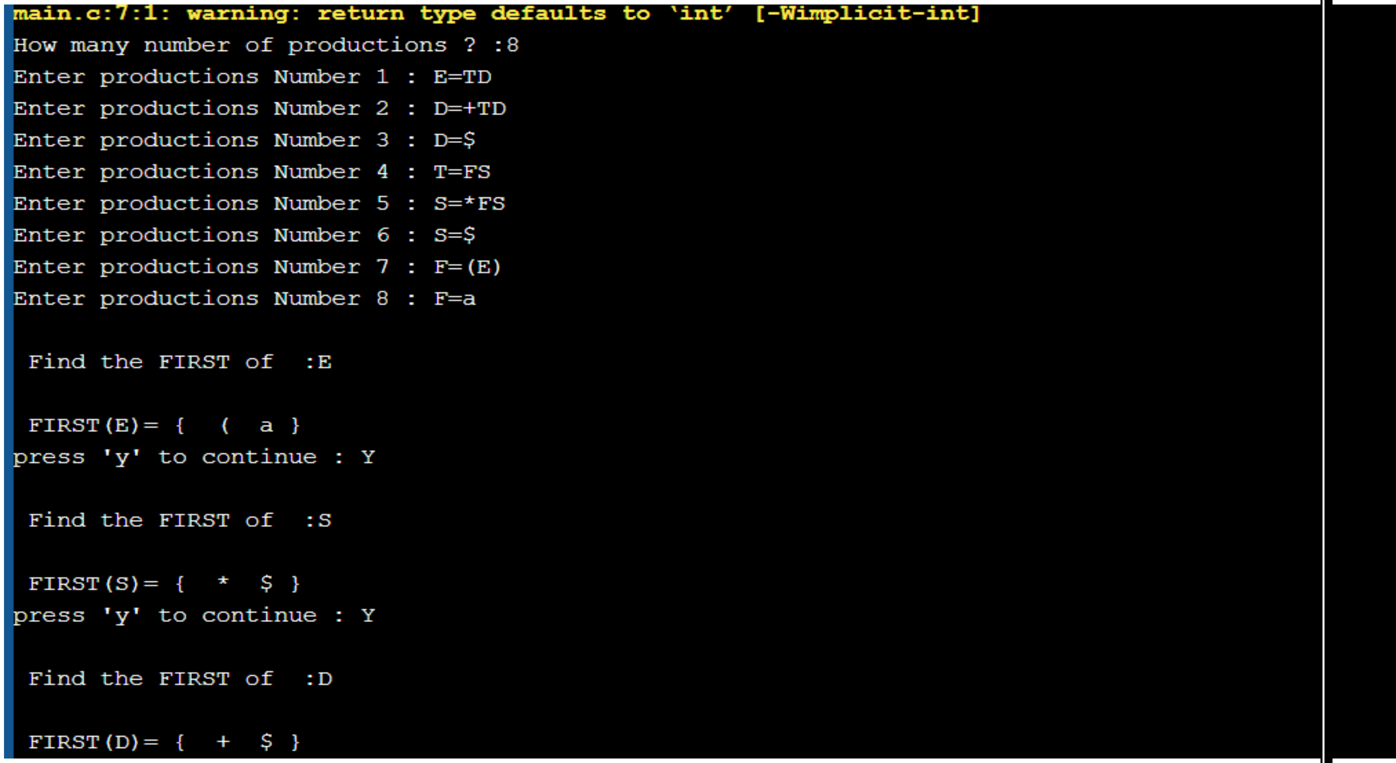
PROGRAM:

```
#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;
   }
}
```

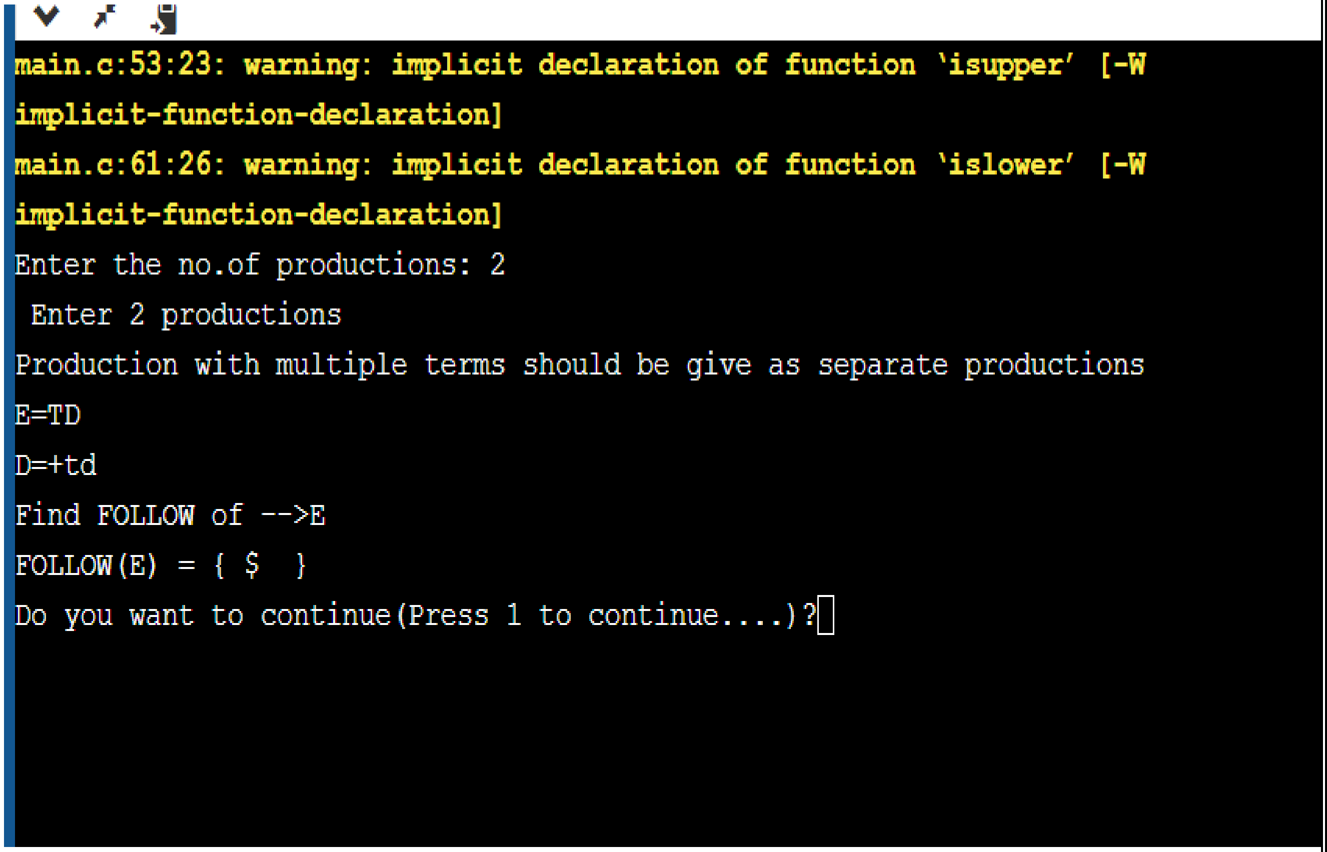
```
void checkfollow(char x)
{
  int i=0;
    switch(x)
  {
    case 'a':
    FT[i]='a'; i++;

    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:



```
main.c:7:1: warning: return type defaults to 'int' [-Wimplicit-int]
How many number of productions ? :8
Enter productions Number 1 : E=TD
Enter productions Number 2 : D=+TD
Enter productions Number 3 : D=$
Enter productions Number 4 : T=FS
Enter productions Number 5 : S=*FS
Enter productions Number 6 : S=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=a

 Find the FIRST of  :E

 FIRST(E)= {  (  a }
press 'y' to continue : Y

 Find the FIRST of  :S

 FIRST(S)= {  *  $ }
press 'y' to continue : Y

 Find the FIRST of  :D

 FIRST(D)= {  +  $ }
```



```
main.c:53:23: warning: implicit declaration of function 'isupper' [-W
implicit-function-declaration]
main.c:61:26: warning: implicit declaration of function 'islower' [-W
implicit-function-declaration]
Enter the no.of productions: 2
 Enter 2 productions
Production with multiple terms should be give as separate productions
E=TD
D=+td
Find FOLLOW of -->E
FOLLOW(E) = { $  }
Do you want to continue(Press 1 to continue....)?
```

Program 9

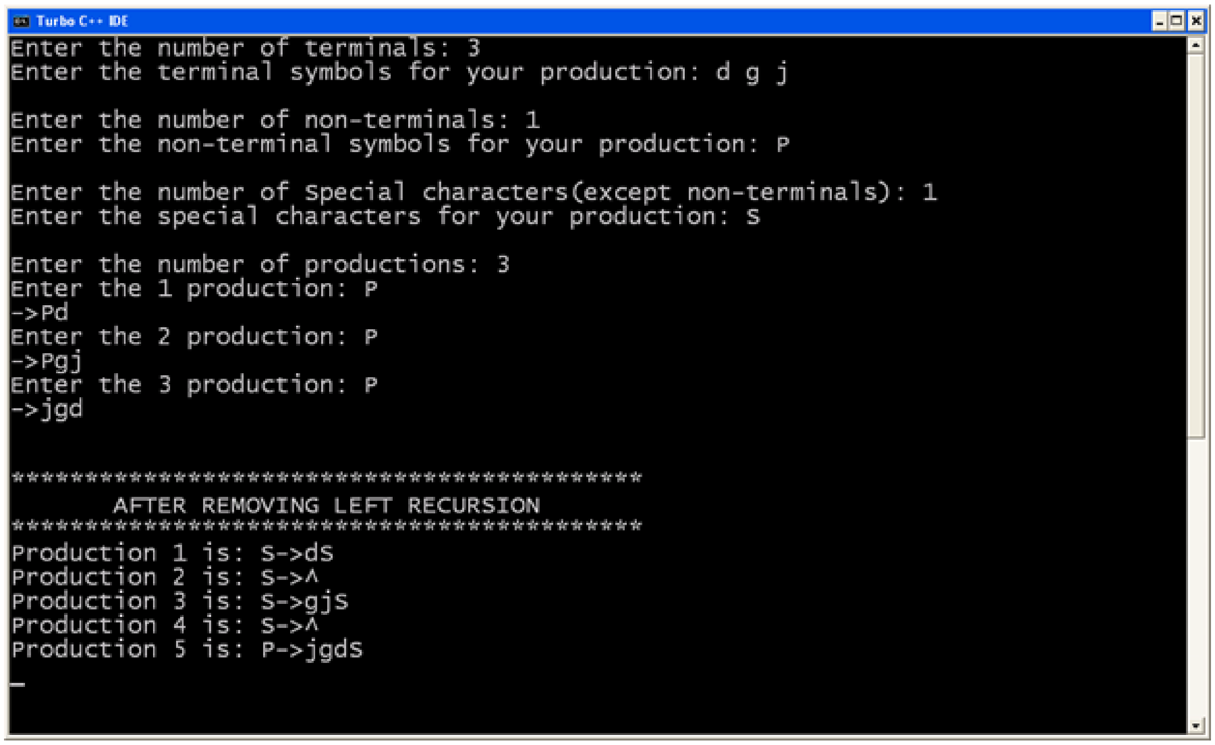
AIM: WAP in C to check whether the Grammar is Left-recursive and remove left recursion.

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



```
Enter the number of terminals: 3
Enter the terminal symbols for your production: d g j

Enter the number of non-terminals: 1
Enter the non-terminal symbols for your production: P

Enter the number of Special characters(except non-terminals): 1
Enter the special characters for your production: S

Enter the number of productions: 3
Enter the 1 production: P
->Pd
Enter the 2 production: P
->Pgj
Enter the 3 production: P
->jgd


*********************************************
       AFTER REMOVING LEFT RECURSION
*********************************************
Production 1 is: S->dS
Production 2 is: S->^
Production 3 is: S->gjS
Production 4 is: S->^
Production 5 is: P->jgdS

_
```

Prgramm-10

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

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!
```

Program 11

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
===========================
   + * ( ) ]
+  > < < > >
*  > > < > >
(  < < < =
)  > >   > >
[  < < <   =
```

Program-12

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]
```