Name:- Tanshi singh

Class:- CSE 3D

Roll no.1816110217

Faculty name:- Mr.Prashant Naresh

Subject:- Compiler Design Lab

Subject code:- (RCS-652)

Department of Computer science and engineering

KRISHNA ENGINEERING COLLEGE

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

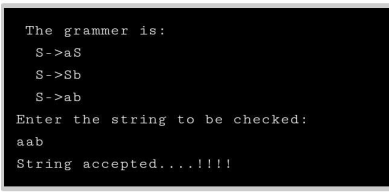
Write a program to check whether the entered string is accepted or not for a given grammar.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
        char string[50];
        int flag,count=0;
        clrscr();
        printf("The grammar is: S->aS, S->Sb, S->ab\n");
        printf("Enter the string to be checked:\n");
        gets(string);
        if(string[0]=='a') {
                flag=0;
                for (count=1;string[count-1]!='\0';count++) {
                        if(string[count]=='b') {
                                flag=1;
                                continue;
                        } else if((flag==1)&&(string[count]=='a')) {
                                printf("The string does not belong to the specified
grammar");
                                break;
                        } else if(string[count]=='a')
                        continue; else if(flag==1)&&(string[count]='\0')) {
                                printf("String accepted.....!!!!");
                                break;
                        } else {
                                printf("String not accepted");
                        }
                }
```

```
  getch();
}
```

Output:



Program 2

Write a program to convert infix expression to postfix expression.

PROGRAM:

```
 #include<stdio.h>
#include<ctype.h>

char stack[100];
int top = -1;

void push(char x)
```

```
{
   stack[++top] = x;
}

char pop()
{
   if(top == -1)
      return -1;
   else
      return stack[top--];
}

int priority(char x)
{
   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/')
      return 2;
   return 0;
}

int main()
{
   char exp[100];
   char *e, x;
   printf("Enter the expression : ");
   scanf("%s",exp);
   printf("\n");
   e = exp;

   while(*e != '\0')
   {
      if(isalnum(*e))
         printf("%c ",*e);
      else if(*e == '(')
         push(*e);
      else if(*e == ')')
      {
         while((x = pop()) != '(')
            printf("%c ", x);
      }
      else
      {
```

```
        while(priority(stack[top]) >= priority(*e))
           printf("%c ",pop());
        push(*e);
      }
      e++;
   }

   while(top != -1)
   {
      printf("%c ",pop());
   }return 0;
}
```

Output:

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

Program 3

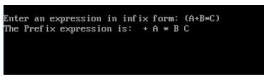
Write a program to convert infix expression to prefix expression.

PROGRAM:

```
# include <stdio.h>
# include <string.h>
# define MAX 20
void infixtoprefix(char infix[20],char prefix[20]);
void reverse(char array[30]);
char pop();
void push(char symbol);
int isOperator(char symbol);
int prcd(symbol);
int top=-1;
char stack[MAX];
main() {
        char infix[20],prefix[20],temp;
        printf("Enter infix operation: ");
        gets(infix);
        infixtoprefix(infix,prefix);
        reverse(prefix);
        puts((prefix));

void infixtoprefix(char infix[20],char prefix[20]) {
        int i,j=0;
        char symbol;
        stack[++top]='#';
        reverse(infix);
        for (i=0;i<strlen(infix);i++) {
                symbol=infix[i];
                if (isOperator(symbol)==0) {
                        prefix[j]=symbol;
                        j++;
                } else {
                        if (symbol==')') {
                                push(symbol);
                        } else if(symbol == '(') {
                                while (stack[top]!=')') {
                                        prefix[j]=pop();
                                        j++;
                                }
                                pop();
                        } else {
```

```
                        if (prcd(stack[top])<=prcd(symbol)) {
                                push(symbol);
                        } else {
                                while(prcd(stack[top])>=prcd(symbol)) {
                                        prefix[j]=pop();
                                        j++;
                                }
                                push(symbol);
                        }
                        //end for else
                }
        }
        //end for else
    }
    //end for for
    while (stack[top]!='#') {
            prefix[j]=pop();
            j++;
    }
    prefix[j]='\0';
}
```

Output:



Program 4

Write a program to find the no. of tokens and list them according to their category in an expression (given/entered)

PROGRAM:

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
bool isValidDelimiter(char ch) {
  if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
  ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
  ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
  ch == '[' || ch == ']' || ch == '{' || ch == '}')
  return (true);
  return (false);
}
bool isValidOperator(char ch){
  if (ch == '+' || ch == '-' || ch == '*' ||
  ch == '/' || ch == '>' || ch == '<' ||
  ch == '=')
  return (true);
  return (false);
}
// Returns 'true' if the string is a VALID IDENTIFIER.
bool isvalidIdentifier(char* str){
  if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
  str[0] == '3' || str[0] == '4' || str[0] == '5' ||
  str[0] == '6' || str[0] == '7' || str[0] == '8' ||
  str[0] == '9' || isValidDelimiter(str[0]) == true)
  return (false);
  return (true);
}
bool isValidKeyword(char* str) {
```

```
  if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") ||
!strcmp(str, "do") ||    !strcmp(str, "break") || !strcmp(str, "continue") ||
!strcmp(str, "int")
  || !strcmp(str, "double") || !strcmp(str, "float") || !strcmp(str, "return") ||
!strcmp(str,    "char") || !strcmp(str, "case") || !strcmp(str, "char")
  || !strcmp(str, "sizeof") || !strcmp(str, "long") || !strcmp(str, "short") ||
!strcmp(str, "typedef") || !strcmp(str, "switch") || !strcmp(str, "unsigned")
  || !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") ||
!strcmp(str, "goto"))
  return (true);
  return (false);
}
bool isValidInteger(char* str) {
  int i, len = strlen(str);
  if (len == 0)
  return (false);
  for (i = 0; i < len; i++) {
    if (str[i] != '0' && str[i] != '1' && str[i] != '2'&& str[i] != '3' && str[i] !=
'4' && str[i] != '5'
    && str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9' || (str[i] ==
'-' && i > 0))
    return (false);
  }
  return (true);
}
bool isRealNumber(char* str) {
  int i, len = strlen(str);
  bool hasDecimal = false;
  if (len == 0)
  return (false);
  for (i = 0; i < len; i++) {
    if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3' && str[i] !=
'4' && str[i]      != '5' && str[i] != '6' && str[i] != '7' && str[i] != '8'
    && str[i] != '9' && str[i] != '.' || (str[i] == '-' && i > 0))
    return (false);
      if (str[i] == '.')
    hasDecimal = true;
  }
  return (hasDecimal);
}
char* subString(char* str, int left, int right) {
  int i;
  char* subStr = (char*)malloc( sizeof(char) * (right - left + 2));
  for (i = left; i <= right; i++)
    subStr[i - left] = str[i];
```

```
  subStr[right - left + 1] = '\0';
  return (subStr);
}
void detectTokens(char* str) {
  int left = 0, right = 0;
  int length = strlen(str);
  while (right <= length && left <= right) {
    if (isValidDelimiter(str[right]) == false)
    right++;
    if (isValidDelimiter(str[right]) == true && left == right) {
      if (isValidOperator(str[right]) == true)
      printf("Valid operator : '%c'\n", str[right]);
      right++;
      left = right;
    } else if (isValidDelimiter(str[right]) == true && left != right || (right ==
length && left !=      right)) {
      char* subStr = subString(str, left, right - 1);
      if (isValidKeyword(subStr) == true)
        printf("Valid keyword : '%s'\n", subStr);
      else if (isValidInteger(subStr) == true)
        printf("Valid Integer : '%s'\n", subStr);
      else if (isRealNumber(subStr) == true)
        printf("Real Number : '%s'\n", subStr);
      else if (isvalidIdentifier(subStr) == true
        && isValidDelimiter(str[right - 1]) == false)
      printf("Valid Identifier : '%s'\n", subStr);
      else if (isvalidIdentifier(subStr) == false
        && isValidDelimiter(str[right - 1]) == false)
      printf("Invalid Identifier : '%s'\n", subStr);
      left = right;
    }
  }
  return;
}
int main(){
  char str[100] = "float x = a + 1b; ";
  printf("The Program is : '%s' \n", str);
  printf("All Tokens are : \n");
  detectTokens(str);
  return (0);
}
```

Output:

```
The Program is : 'float x = a + 1
All Tokens are :
Valid keyword : 'float'
Valid Identifier : 'x'
Valid operator : '='
Valid Identifier : 'a'
Valid operator : '+'
Invalid Identifier : '1b'
```

Program 5

Write a program 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 regular expression\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();
}
```

Output:

enter regular expression
(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

Write a program 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:

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

Write a program to calculate first and follow.

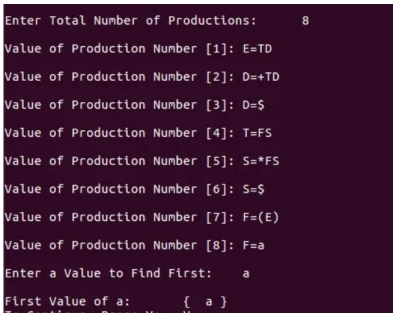
PROGRAM:

```
#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


Enter Total Number of Productions: 8
Value of Production Number [1]: E=TD
Value of Production Number [2]: D=+TD
Value of Production Number [3]: D=$
Value of Production Number [4]: T=FS
Value of Production Number [5]: S=*FS
Value of Production Number [6]: S=$
Value of Production Number [7]: F=(E)
Value of Production Number [8]: F=a
Enter a Value to Find First: a
First Value of a: { a }

Program 9

Write a program in C to check whether the Grammar is Left-recursive and remove left recursion.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
 int main () {
    char non_terminal;
    char beta,alpha;
    int num;
    char production[10][SIZE];
    int index=3;
    printf("Enter Number of Production : ");
    scanf("%d",&num);
     printf("Enter the grammar as E->E-A   :\n");
     for(int i=0;i<num;i++){
        scanf("%s",production[i]);
    }
    for(int i=0;i<num;i++){          printf("\nGRAMMAR : : : %s",production[i]);
        non_terminal=production[i][0];
        if(non_terminal==production[i][index]) {
            alpha=production[i][index+1];
            printf(" is left recursive.\n");
            while(production[i][index]!=0 && production[i][index]!='|')
                index++;
           if(production[i][index]!=0) {
               beta=production[i][index+1];
                printf("Grammar without left recursion:\n");
                 printf("'%c->%c%c\'",non_terminal,beta,non_terminal);
                printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
            }
          else                  printf(" can't be reduced\n");
      }
       else
            printf(" is not left recursive.\n");
     index=3;
   }
 }
```

Output

Prgramm-10

Write a program 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!
```

Program 11

Write a program 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

Write a program **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]
```