INDEX

S.No	Program	Date	Signature
1	Write a program to calculate first and follow of the given grammar.		
2	Write a program to test whether a given identifier is valid or not.		
3	Write a program to implement LL(1) parsing.		
4	Write a program for Lexical Analyzer.		
5	Write a program to develop recursive descent parsing.		
6	Write a program to implement LR(0) parsing.		
7	Write a program implement operator precedence parsing.		
8	Write a program to implement syntax tree expression generation.		
9	Study LEX and YACC tool.		
10	Write a lex Program to convert lowercase to uppercase and vice verssa.		
11	Write a lex program to find out total number of vowels from the given input string.		
12	Write a lex program to find out total number of vowels from the given input string.		

S.No	Program	Date	Signature
13	Write a lex program to convert decimal number to hexadecimal number in a file - lex program.		
14	Write a lex program to count the number of Positive numbers, Negative numbers & Fractions.		
15	Write a Lex code to count total number of tokens.		

Program 1

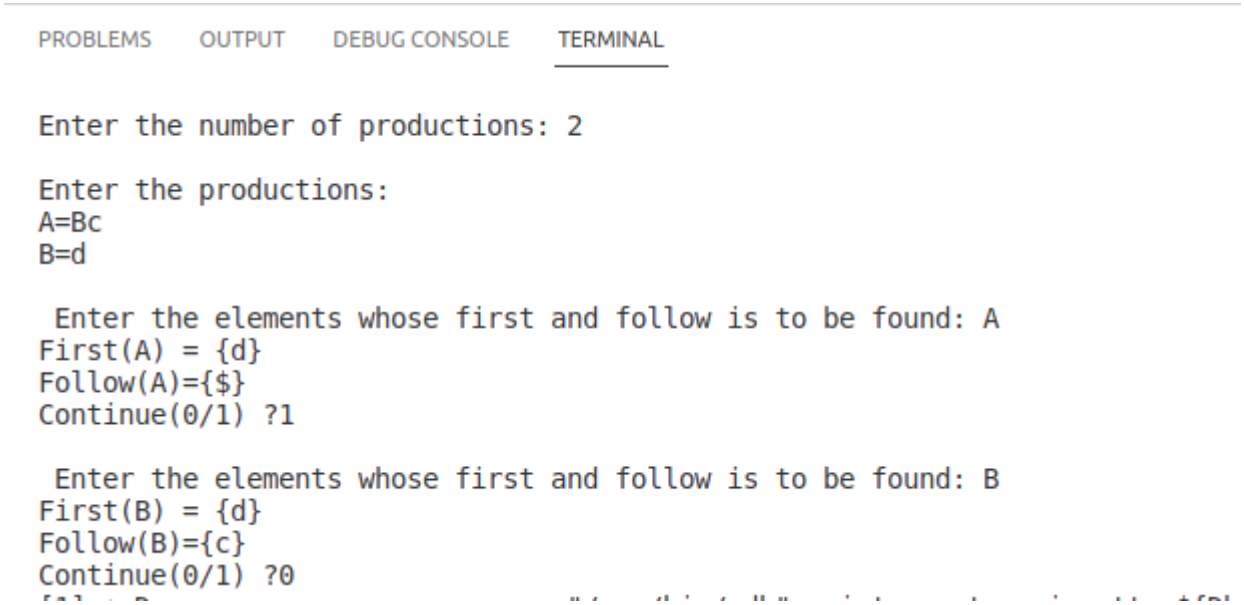
Objective : Write a program to calculate first and follow of the given grammar.

Code:

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>

int n=0,m=0,p,i=0,j=0;
char a[10][10],f[10];

void follow(char c);
void first(char c);

int main(){
        int i,z;
        char c,ch;
        printf("Enter the number of productions: ");
        scanf("%d",&n);
        printf("\nEnter the productions: \n");
        for(i=0;i<n;i++)
        scanf("%s%c",a[i],&ch);
        do{
                m=0;
                printf("\n Enter the elements whose first and follow is to be found: ");
                scanf("%c",&c);
                first(c);
                printf("First(%c) = {",c);
                for(i=0;i<m;i++)
                printf("%c",f[i]);
                printf("}\n");
                strcpy(f,"");
                m=0;
                follow(c);
                printf("Follow(%c)={",c);
                for(i=0;i<m;i++)
                printf("%c",f[i]);
                printf("}\n");
                printf("Continue(0/1) ?");
                scanf("%d%c",&z,&ch);
        }while(z==1);
        return 0;
}

void first(char c){
        int k;
        if(!isupper(c))
        f[m++]=c;
```

```
        for(k=0;k<n;k++){
                if(a[k][0]==c){
                        if(a[k][2]=='$')
                        follow(a[k][0]);
                        else if(islower(a[k][2]))
                        f[m++]=a[k][2];
                        else first(a[k][2]);
                }
        }
}

void follow(char c){
        if(a[0][0]==c)
        f[m++]='$';
        for(i=0;i<n;i++){
                for(j=2;j<strlen(a[i]);j++){
                        if(a[i][j]==c){
                                if(a[i][j+1]!='\0')
                                first(a[i][j+1]);
                                if(a[i][j+1]=='\0' && c!=a[i][0])
                                follow(a[i][0]);
                        }
                }
        }
}
```

Output:



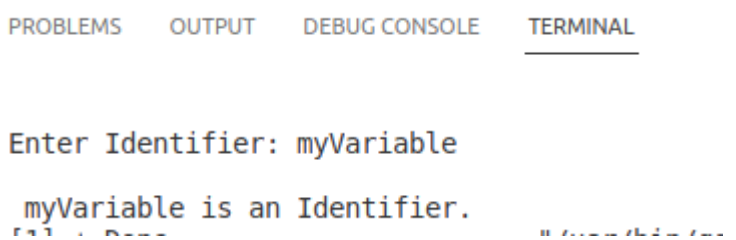
Program 2

Objective : Write a program to test whether a given identifier is valid or not.

Code:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main(){
  int i=0,flag=0;
  char keyw[10][10] = { "int", "float", "break", "long", "char", "for","if", "switch", "else", "while"};
  char a[10];
   printf("Enter Identifier: ");
   scanf("%s",a);
   for(i=0;i<10;i++){
      if((strcmp(keyw[i],a)==0))
      flag=1;
   }
   if(flag==1)
      printf("\n %s is Keyword",a);
   else{
      if(a[0]=='_'|| isalpha(a[0])!=0){
         for(i=1;a[i]!='\0';i++){
            if(isalnum(a[i])==0 && a[i]!='_')
             flag=1;
         }
      }
      else{
         flag=1;
      }
   }
   if(flag==0)
      printf("\n %s is an Identifier.",a);
   else
      printf("\n %s is not an identifier,",a);
   return 0;
}
```

Output:



Program 3

Objective : Write a program to implement LL(1) parsing.

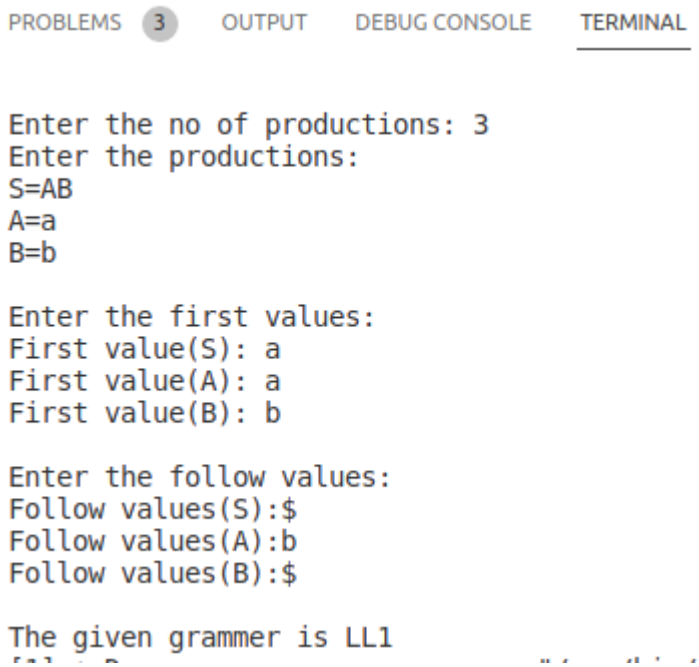
Code:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void main(){
   char pro[10][10],first[10][10],follow[10][10],nt[10],ter[10],res[10][10][10],temp[10];
   int npro,noter=0,nont=0,i,j,k,flag=0,count[10][10],row,col,l,m,n,index;

   for(i=0;i<10;i++){
      for(j=0;j<10;j++){
         count[i][j] = NULL;
         for(k=0;k<10;k++){
            res[i][j][k]=NULL;
         }
      }
   }
   printf("Enter the no of productions: ");
   scanf("%d",&npro);
   printf("Enter the productions: ");
   for(i=0;i<npro;i++){
      scanf("%s",pro[i]);
   }
   for(i=0;i<npro;i++){
      flag=0;
      for(j=0;j<nont;j++){
         if(nt[j]==pro[i][0])
         flag=1;
      }
      if(flag==0){
         nt[nont]=pro[i][0];
         nont++;
      }
   }
   printf("\nEnter the first values:\n");
   for(i=0;i<nont;i++){
      printf("First value(%c): ",nt[i]);
      scanf("%s",first[i]);
   }
   printf("\nEnter the follow values:\n");
   for(i=0;i<nont;i++){
      printf("Follow values(%c):",nt[i]);
      scanf("%s",follow[i]);
   }
   for(i=0;i<nont;i++){
      flag=0;
      for(j=0;j<strlen(first[i]);j++){
         for(k=0;k<noter;k++){
            if(ter[k]==first[i][j])
```

```
            flag = 1;
        }
        if(flag==0){
            if(first[i][j]!='#'){
                ter[noter]=first[i][j];
                noter++;
            }
        }
    }
}
for(i=0;i<nont;i++){
    flag=0;
    for(j=0;j<strlen(follow[i]);j++){
        for(k=0;k<noter;k++){
            if(ter[k]==follow[i][j])
            flag=1;
        }
        if(flag==0){
            ter[noter]=follow[i][j];
            noter++;
        }
    }
}
for(i=0;i<nont;i++){
    for(j=0;j<strlen(first[i]);j++){
        flag=0;
        if(first[i][j]=='#'){
            col=i;
            for(m=0;m<strlen(follow[col]);m++){
                for(l=0;l<noter;l++){
                    if(ter[l]==follow[col][m]){
                        row=1;
                    }
                }
                temp[0]=nt[col];
                temp[1]='_';
                temp[2]='>';
                temp[3]='#';
                temp[4]='\0';
                printf("temp %s",temp);
                strcpy(res[col][row],temp);
                count[col][row]+=1;
                for(k=0;k<10;k++)
                temp[k]=NULL;
            }
        }
        else{
            for(l=0;l<noter;l++){
                if(ter[l]==first[i][j])
                row=1;
            }
            for(k=0;k<npro;k++){
```

```
                if(nt[i]==pro[k][0]){
                   col=i;
                   if(pro[k][3]==first[i][j] && pro[k][0]==nt[col]){
                      strcpy(res[col][row],pro[k]);
                      count[col][row]+=1;
                   }
                   else{
                      if(isupper(pro[k][3]) &&  pro[k][0]==nt[col]){
                         flag=0;
                         for(m=0;m<nont;m++){
                            if(nt[m]==pro[k][3]){
                               index=m;
                               flag=1;
                            }
                         }
                         if(flag==1){
                            for(m=0;m<strlen(first[index]);m++){
                               if(first[i][j]==first[index][m]){
                                  strcpy(res[col][row],pro[k]);
                                  count[col][row]+=1;
                               }
                            }
                         }
                      }
                   }
                }
             }
          }
       }
    }
       flag=0;
       for(j=0;j<nont;j++){
          for(k=0;k<noter;k++){
             if(count[j][k]>1)
             flag=1;
          }
       }
       if(flag==1)
       printf("\n The given grammer is not LL1");
       else
       printf("\nThe given grammer is LL1");
}
```

Output:


PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
Enter the no of productions: 3
Enter the productions:
S=AB
A=a
B=b
Enter the first values:
First value(S): a
First value(A): a
First value(B): b
Enter the follow values:
Follow values(S):$
Follow values(A):b
Follow values(B):$
The given grammer is LL1

Program 4

Objective : Write a program for Lexical Analyzer.

Code:

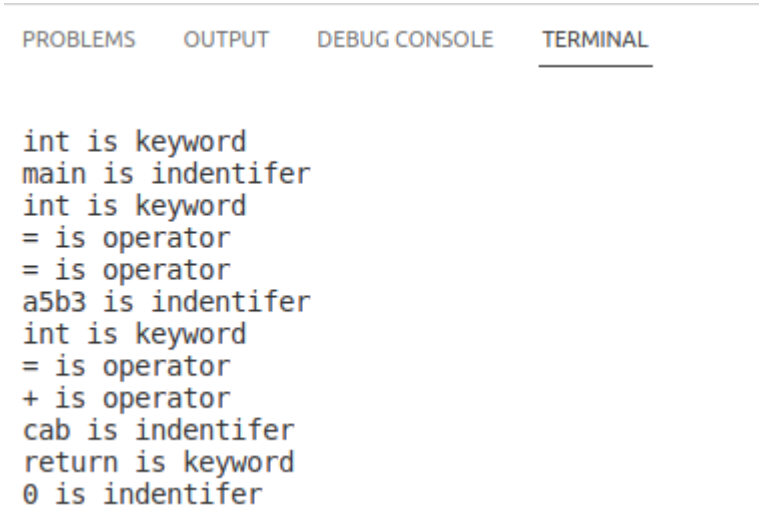
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

int isKeyword(char buffer[]){
   char keywords[32][10]={
 "auto","break","case","char","const","continue","default","do","double","else","enum","extern",
 "float","for","goto","if","int","long","register","return","short","signed","sizeof","static","struct",
 "switch","typedef","union","unsigned","void","volatile","while"};

   int i, flag=0;
   for(i=0;i<32;++i){
      if(strcmp(keywords[i],buffer)==0){
         flag=1;
         break;
      }
   }
   return flag;
}

int main(){
   char ch,buffer[15],operators[]="+-*/%=";
   FILE *fp;
   int i,j=0;
   fp=fopen("Program.txt","r");
   if(fp==NULL){
      printf(" Error while opening the file \n");
      exit(0);
   }

   while((ch=fgetc(fp))!=EOF){
      for(i=0;i<6;++i){
         if(ch==operators[i])
         printf("%c is operator\n",ch);
      }
      if(isalnum(ch)){
         buffer[j++]=ch;
      }
      else if((ch==' '||ch=='\n') && (j!=0)){
         buffer[j]='\0';
         j=0;
         if(isKeyword(buffer)==1)
         printf("%s is keyword\n",buffer);
         else
         printf("%s is indentifer\n",buffer);
      }
```

```
    }
    fclose(fp);
   return 0;
}
```

Program.txt:

```
int main(){
    int a=5,b=3;
    int c=a+b;
    return 0;
}
```

Output:



Program 5

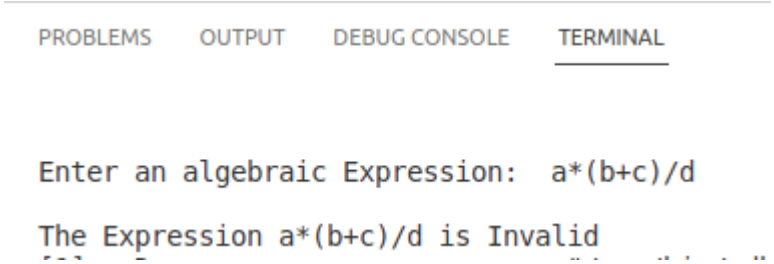
Objective : Write a program to develop recursive descent parsing.

Code:

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

void Tprime();
void Eprime();
void E();
void check();
void T();
char expression[10];
int count,flag;

int main(){
   count=0;
   flag=0;
   printf("\nEnter an algebraic Expression:\t");
   scanf("%s",expression);
   E();
   if(strlen(expression)==count && flag==0)
    printf("\nThe Expression %s is valid\n",expression);
   else
    printf("\nThe Expression %s is Invalid\n",expression);

}

void E(){
   T();
   Eprime();
}

void T(){
   check();
   Tprime();
}

void Tprime(){
   if(expression[count]=='*'){
      count++;
      check();
      Tprime();
   }
}

void check(){
   if(isalnum(expression[count]))
   count++;
   else if(expression[count]=='('){
```

```
        count++;
        E();
        if(expression[count]==')')
        count++;
        else
        flag=1;
    }
    else
    flag=1;
}

void Eprime(){
    if(expression[count]=='+'){
        count++;
        T();
        Eprime();
    }
}
```

Output:



Program 6

Objective : Write a program to implement LR(0) parsing.

Code:

```
#include<stdio.h>
#include<string.h>
char stack[30];
int top = -1;
void push(char c){
   top++;
   stack[top]=c;
}

char pop(){
   char c;
   if(top!=-1){
      c=stack[top];
      top--;
      return c;
   }
   return 'x';
}

void printstat(){
   int i;
   printf("\n\t\t\t $");
   for(i=0;i<=top;i++)
   printf("%c",stack[i]);
}

void main(){
   int i,j,k,l;
   char s1[20],s2[20],ch1,ch2,ch3;
   printf("\n\t LR PARSING");
   printf("\n\t Enter the expression: ");
   scanf("%s",s1);
   l=strlen(s1);
   j=0;
   printf("\n\t\t$");
   for(i=0;i<l;i++){
      if(s1[i]=='i' && s1[i+1]=='d'){
         s1[i]="/0";
         s1[i+1]='E';
         printstat();
         printf("id");
         push('E');
         printstat();
      }
      else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='*'||s1[i]=='/'||s1[i]=='d'){
         push(s1[i]);
         printstat();
```

```
        }
    }
    printstat();
    l=strlen(s2);
    while(1){
        ch1=pop();
        if(ch1=='x'){
            printf("\n\t\t\t$");
            break;
        }
        if(ch1=='+'||ch1=='/'||ch1=='*'||ch1=='-'){
            ch3=pop();
            if(ch3!='E'){
                printf("Error");
            }
            else{
                push('E');
                printstat();
            }
        }
        ch2=ch1;
    }
}
```

Output:


```
LR PARSING
Enter the expression: id*(id+id)

                $
                $id
                $E
                $E*
                $E*id
                $E*E
                $E*E+
                $E*E+id
                $E*E+E
                $E*E+E
                $E*E
                $E
                $
```

Program 7

Objective : Write a program to implement operator precedence parsing.

Code:

```
#include<stdio.h>
#include<string.h>
void main(){
   char stack[20],ip[20],opt[10][10][1],ter[10];
   int i,j,k,n,top=0,col,row;
   for(i=0;i<10;i++){
      stack[i]=NULL;
      ip[i]=NULL;
      for(j=0;j<10;j++){
         opt[i][j][1]=NULL;
      }
   }
   printf("Enter the no.of terminals: ");
   scanf("%d",&n);
   printf("\nEnter the terminals: ");
   scanf("%s",&ter);
   printf("\nEnter the table values:\n");
   for(i=0;i<n;i++){
      for(j=0;j<n;j++){
         printf("Enter the value for %c %c: ",ter[i],ter[j]);
         scanf("%s",opt[i][j]);
      }
   }
   printf("\n Operator Precedence Table \n");
   for (i=0;i<n;i++){
      printf("\t%c",ter[i]);
   }
   printf("\n");
   for(i=0;i<n;i++){
      printf("\n%c",ter[i]);
      for(j=0;j<n;j++)
      printf("\t%c",opt[i][j][0]);
   }
   stack[top]='$';
   printf("\n\n Enter the input string: ");
   scanf("%s",ip);
   i=0;
   printf("\nStack\t\t\tInput\t\t\tAction\n");
   printf("\n%s\t\t\t%s\t\t\t",stack,ip);
   while(i<=strlen(ip)){
      for(k=0;k<n;k++){
         if(stack[top]==ter[k])
         col=k;
         if(ip[i]==ter[k])
         row=k;
      }
      if((stack[top]=='$')&&(ip[i]=='$')){
```

```
            printf("\nString is accepted\n");
            break;
        }
        else if((opt[col][row][0]=='<')||(opt[col][row][0]=='=')){
            stack[++top]=opt[col][row][0];
            stack[++top]=ip[i];
            printf("Shift %c",ip[i]);
            i++;
        }
        else{
            if(opt[col][row][0]=='>'){
                while(stack[top]!='<')
                --top;
                top=top-1;
                printf("Reduce");
            }else{
                printf("\nString is not accepted\n");
                break;
            }
        }
        printf("\n");
        for(k=0;k<=top;k++)
            printf("%c",stack[k]);
        printf("\t\t\t");
        for(k=i;k<=strlen(ip);k++)
           printf("%c",ip[k]);
        printf("\t\t\t");

    }
}
```

Output:

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL

```
Enter the no.of terminals: 4

Enter the terminals: +*i$

Enter the table values:
Enter the value for + +: >
Enter the value for + *: <
Enter the value for + i: <
Enter the value for + $: >
Enter the value for * +: >
Enter the value for * *: >
Enter the value for * i: <
Enter the value for * $: >
Enter the value for i +: >
Enter the value for i *: >
Enter the value for i i: e
Enter the value for i $: >
Enter the value for $ +: <
Enter the value for $ *: <
Enter the value for $ i: <
Enter the value for $ $: a

 Operator Precedence Table
        +       *       i       $

+       >       <       <       >
*       >       >       <       >
i       >       >       e       >
$       <       <       <       a

 Enter the input string: i*i+i$

Stack                   Input                   Action

$                       i*i+i$                  Shift i
$<i                     *i+i$                   Reduce
$                       *i+i$                   Shift *
$<*                     i+i$                    Shift i
$<*<i                   +i$                     Reduce
$<*                     +i$                     Reduce
$                       +i$                     Shift +
$<+                     i$                      Shift i
$<+<i                   $                       Reduce
$<+                     $                       Reduce
$                       $
String is accepted
```

Program 8

Objective: Write a program to implement syntax tree expression generation.

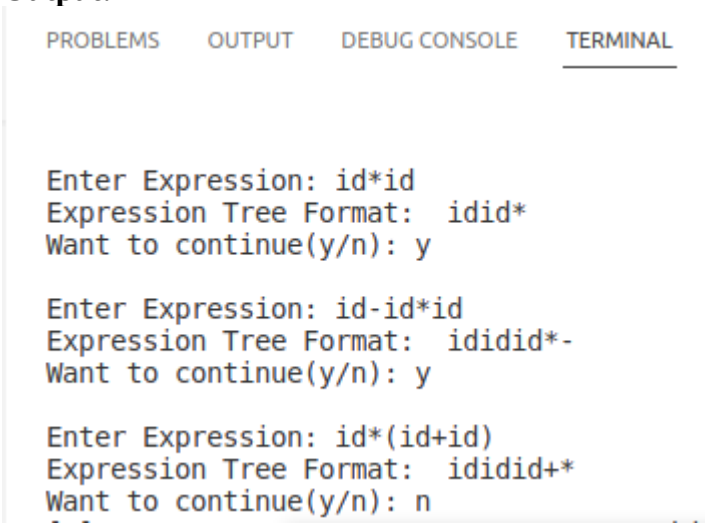
Code:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define Blank ' '
#define Tab '\t'
#define MAX 100

int pop();
char expression[MAX], syntaxtree[MAX];
int stack[MAX];
int top;
void expression_to_syntaxtree();
int prec(char symbol);
void push(int symbol);
int pop();
int white_space(char symbol);

int main(){
   int value;
   char choice='y';

   do{
      top=0;
      printf("\nEnter Expression: ");
       scanf("%[^\n]", expression);
       getchar();
      expression_to_syntaxtree();
      printf("Expression Tree Format:  %s \n",syntaxtree);
      printf("Want to continue(y/n): ");
      scanf(" %c",&choice);
      getchar();
   }while(choice=='y');
}
void expression_to_syntaxtree(){
   int i,p=0,type,precedence,len;
   char next;
   stack[top]='#';
   len=strlen(expression);
   expression[len]='#';
   for(i=0;expression[i]!='#';i++){
      if(!white_space(expression[i])){
         switch(expression[i]){
            case '(':
            push(expression[i]);
            break;
            case ')':
            while ((next=pop())!='('
```

```
                syntaxtree[p++]=next;
                break;
                case '+':
                case '-':
                case '*':
                case '/':
                case '%':
                case '^':
                precedence=prec(expression[i]);
                while(stack[top]!='#'&& precedence<=prec(stack[top]))
                syntaxtree[p++]=pop();
                push(expression[i]);
                break;
                default:
                syntaxtree[p++]=expression[i];
            }
        }
    }
    while (stack[top]!='#')
    syntaxtree[p++]=pop();
    syntaxtree[p]='\0';

}


int prec(char symbol)
{
    switch (symbol)
    {
    case '(':
    case ')': return 0;
    case '+':
    case '-':
    return 1;
    case '*':
    case '/':
    return 2;
    case '^':
    return 3;
    }
}

void push(int symbol){
    if(top>MAX){
        printf("Stack overflow\n");
        exit(1);
    }
    else{
        top=top+1;
        stack[top] = symbol;
    }
}
```

```
int pop(){
   if(top==-1){
      printf("Stack underflow \n");
      exit(2);
   }
   else{
      return(stack[top--]);
   }
}

int white_space(char symbol){
   if(symbol == Blank ||symbol == Tab || symbol == '\0')
   return 1;
   else
   return 0;
}
```

Output:



Program 9

Objective : Study LEX and YACC tool.

LEX:
Lex is officially known as a "Lexical Analyser".
It's main job is to break up an input stream into more usable elements.
Or in, other words, to identify the "interesting bits" in a text file.
For example, if you are writing a compiler for the C programming language, the symbols { } ( ) ; all have significance on their own. The letter a usually appears as part of a keyword or variable name, and is not interesting on it's own. Instead, we are interested in the whole word. Spaces and newlines are completely uninteresting, and we want to ignore them completely, unless they appear within quotes "like this" .
All of these things are handled by the Lexical Analyser.

YACC:
Yacc is officially known as a "parser".
It's job is to analyse the structure of the input stream, and operate of the "big picture".
In the course of it's normal work, the parser also verifies that the input is syntactically sound.
Consider again the example of a C-compiler. In the C-language, a word can be a function name or a variable, depending on whether it is followed by a (or a = There should be exactly one } for each { in the program.
YACC stands for "Yet Another Compiler Compiler". This is because this kind of analysis of text files is normally associated with writing compilers. However it can be applied to almost any situation where text-based input is being used.

Program 10

Objective : Write a lex Program to convert lowercase to uppercase & reverse – lex program.

Code:

```
lower [a-z]
CAPS [A-Z]
space [\t\n]

%%
{lower}   {printf("%c",yytext[0]-32);}
{CAPS}    {printf("%c",yytext[0]+32);}
{space}   ECHO;
.         ECHO;

%%
void main(){
   yylex();
}
```

Output:

```
ubuntu@user:~$ flex exp10.l
ubuntu@user:~$ cc lex.yy.c -lfl
ubuntu@user:~$ ./a.out
Hello
hELLO


Coverts LowerCase INTO UpperCase and Vice-Versa
cOVERTS lOWERcASE into uPPERcASE AND vICE-vERSA
ubuntu@user:~$
```

Program 11

Objective : Write a LEX program to find out total number of vowels from the given input string.

Code:

```
%{
   int count =0;
%}

%%

[aeiouAEIOU] {count++;ECHO;}

%%

int main(){
   yylex();
   printf("\n Number of vowels=%d\n",count);
   return 0;
}
```

Output:

```
ubuntu@user:~$ flex exp11.l
ubuntu@user:~$ cc lex.yy.c -lfl
ubuntu@user:~$ ./a.out
Count vowels in this string
Count vowels in this string

 Number of vowels=7
ubuntu@user:~$
```

Program 12

Objective : Write a lex program to count number of lines and characters in the input.

Code:

```
/* Dexlaring two counters one for number of lines other for number of characters*/
%{
   int no_of_lines=0;
   int no_of_chars=0;
%}

%%
\n  ++no_of_lines;
.   ++no_of_chars;
end return 0;
%%

int yywrap(){}
int main(int argc,char **argv){
   yylex();
   printf(" Number of lines=%d,number of chars=%d\n",no_of_lines,no_of_chars);
   return 0;
}
```

Output:

```
ubuntu@user:~$ flex exp12.l
ubuntu@user:~$ cc lex.yy.c -lfl
ubuntu@user:~$ ./a.out
Principles of Compiler design.
Dragon Book
Second Edition
Author: Alfred Aho
 Number of lines=4,number of chars=73
ubuntu@user:~$
```

Program 13

Objective : Write a lex program to convert decimal number to hexadecimal number in a file .

Code:

```
%{
   #include<stdio.h>
   int num,r,digit=0,count,pcount=0,i;
   char a[20];
%}

DIGIT[0-9]
%%
{DIGIT}+ { num=atoi(yytext);
          while(num!=0){
           r=num%16;
           digit='0'+r;
           if(digit>'9')
           digit+=7;
           a[count++]=digit;
           num=num/16;
          }
          for(i=count-1;i>=pcount;--i)
          printf("%c",a[i]);
          pcount=count;
}

.|\n  ECHO;
%%

int main(){
   yylex();
   return 0;
}
```

Output:

```
ubuntu@user:~$ flex exp13.l
ubuntu@user:~$ cc lex.yy.c -lfl
ubuntu@user:~$ ./a.out
16
10


15
F


235
EB
```

Program 14

Objective : Write a lex program to count the number of Positive numbers, Negative numbers and Fractions.

Code:

```
%{
   int postiveno=0;
   int negativeno=0;
   int postivefractions=0;
   int negativefractions=0;
%}

DIGIT[0-9]
%%
\+?{DIGIT}+   postiveno++;
-{DIGIT}+     negativeno++;
\+?{DIGIT}*\.{DIGIT}+ postivefractions++;
-{DIGIT}*\.{DIGIT}+   negativefractions++;
.;
%%

void main(){
   yylex();
   printf("\n Number of positive numbers: %d",postiveno);
   printf("\n Number of negative nummbers: %d",negativeno);
   printf("\n Number of positive fractions: %d",postivefractions);
   printf("\n Number of Negative fractions: %d\n",negativefractions);
}
```

Output:

```
ubuntu@user:~$ flex exp14.l
ubuntu@user:~$ cc lex.yy.c -lfl
ubuntu@user:~$ ./a.out
5 -10 15 -89.23 -7.234 67.23 90 1 2 3 -4 -5 -6 23.42

-45.2 -23.4 -3 -5 90 100 101 1999.122


 Number of positive numbers: 9
 Number of negative nummbers: 6
 Number of positive fractions: 3
 Number of Negative fractions: 4
ubuntu@user:~$
```

Program 15

Objective : Write a Lex program to count total number of tokens.

Code:

```
%{
   int n =0;
%}

%%
"while"|"if"|"else" {n++;printf("\t Keywords: %s",yytext);}
"int"|"float" {n++; printf("\t Keywords: %s",yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("\t identifier: %s",yytext);}
"<="|">="|"<"|">"|"=="|"="|"++"|"--"|"-"|"*"|"+"|"/" {n++;printf("\t operator: %s",yytext);}
[(){}|,;] {n++;printf("\t seprator: %s",yytext);}
[0-9]*"."[0-9]+ {n++;printf("\t float: %s",yytext);}
[0-9]+ {n++;printf("\t integer: %s",yytext);}
.;
%%

int main(){
   yylex();
   printf("\n Total number of tokens = %d\n",n);
}
```

Output:

```
ubuntu@user:~$ flex exp15.l
ubuntu@user:~$ cc lex.yy.c -lfl
ubuntu@user:~$ ./a.out
int i=6,j=9,k=20;
         Keywords: int   identifier: i   operator: =     integer: 6      seprator: ,
     identifier: j   operator: =     integer: 9      seprator: ,     identifier: k
 operator: =     integer: 20     seprator: ;
float p=(i*j+k)*45.67
         Keywords: float         identifier: p   operator: =     seprator: (    iden
tifier: i    operator: *     identifier: j   operator: +     identifier: k  seprator
: )      operator: *     float: 45.67

 Total number of tokens = 25
ubuntu@user:~$
```