FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

COMPILER DESIGN (303105350)

6TH SEMESTER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TABLE OF CONTENT

Experiment No: 1

Aim: Program to implement Lexical Analyzer.

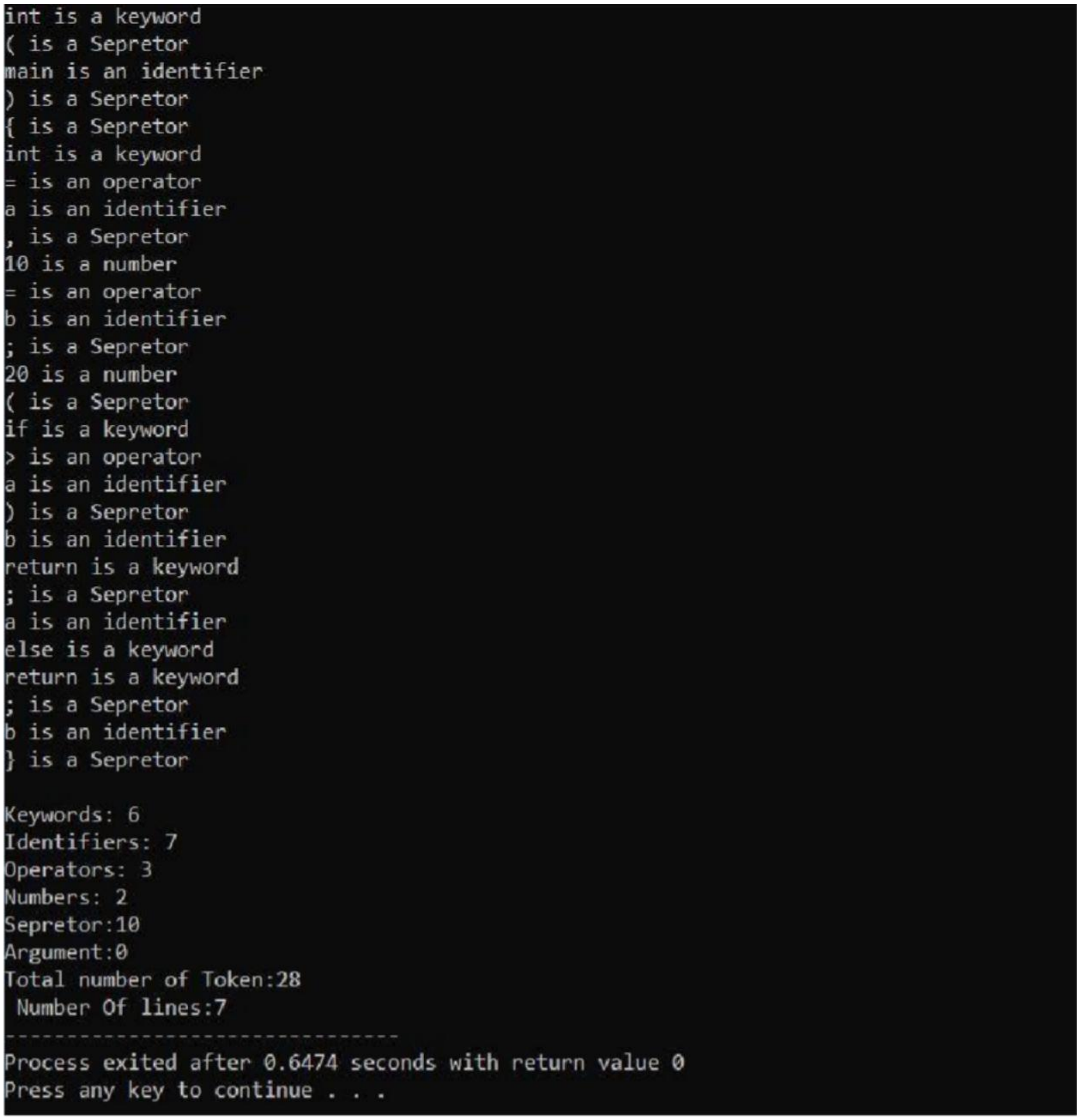
Program :

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
void keyw(char *p);
int i=0,id=0,kw=0,num=0,op=0,sp=0,ar=0,count=0,new_line=0;
char keys[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"};
main()
{
   char ch,str[25],seps[20]=" \t\n,;(){}[]#\"<>",oper[]="!%^&*-+=~|.<>/?"; int
   j;
   char fname[50];
   FILE *f1;
   f1 = fopen("Laxcode.txt","r");
   if(f1==NULL)
   {
      printf("file not found"); exit(0);
   }

   while((ch=fgetc(f1))!=EOF)
   {
         for(j=0;j<=14;j++)
      {
         if(ch==oper[j])
         {
                                    printf("%c is an operator\n",ch);
            op++;
            count++;
            str[i]='\0';
            keyw(str);
         }
      }
      if(ch=='\n'){ ne
         w_line++;
                }
```

```
for(j=0;j<=14;j++)
{
   if(i==-1)
            break; if(ch==seps[j])
   {
      if(ch=='#')
      {
         while(ch!='>')
         {
            printf("%c",ch);
            ch=fgetc(f1);
         }
         printf("%c is a header file\n",ch);
         i=-1;
         break;
      }
      if(ch=='"')
      {
         do
         {
            ch=fgetc(f1);
            printf("%c",ch);
         }while(ch!='"');
         printf("\b is an argument\n");
         i=-1;
         ar++; count++;
         break;
      }
      if(ch==',' || ch==';' || ch=='(' || ch==')' || ch=='{' || ch=='}' || ch=='[' ||
            ch==']'){ printf("%c is a Sepretor\n",ch);

            sp++;
            count++;
                              }
      str[i]='\0';
      keyw(str);
   }
}
if(i!=-1)
{
   str[i]=ch; i++;
}
else
   i=0;
```

```
    }
    printf("\nKeywords:            %d\nIdentifiers:            %d\nOperators:            %d\nNumbers:
%d\nSepretor:%d\nArgument:%d",kw,id,op,num,sp,ar);
        printf("\nTotal number of Token:%d",count); printf("\n
        Number Of lines:%d",new_line);
}
void keyw(char *p)
{
    int k,flag=0; for(k=0;k<=31;k++)
    {
        if(strcmp(keys[k],p)==0)
        {
            printf("%s is a keyword\n",p);
            kw++;
            count++;
            flag=1; break;
        }
    }
    if(flag==0)
    {
        if(isdigit(p[0]))
        {
            printf("%s is a number\n",p);
            num++;
            count++;
        }
        else
                {
            if(p[0]!='\0')
            {
                printf("%s is an identifier\n",p); id++;
                count++;
            }
        }
    }
    i=-1;
}
```

Output :



```
int is a keyword
( is a Sepretor
main is an identifier
) is a Sepretor
{ is a Sepretor
int is a keyword
= is an operator
a is an identifier
, is a Sepretor
10 is a number
= is an operator
b is an identifier
; is a Sepretor
20 is a number
( is a Sepretor
if is a keyword
> is an operator
a is an identifier
) is a Sepretor
b is an identifier
return is a keyword
; is a Sepretor
a is an identifier
else is a keyword
return is a keyword
; is a Sepretor
b is an identifier
} is a Sepretor

Keywords: 6
Identifiers: 7
Operators: 3
Numbers: 2
Sepretor:10
Argument:0
Total number of Token:28
 Number Of lines:7
--------------------------------
Process exited after 0.6474 seconds with return value 0
Press any key to continue . . .
```

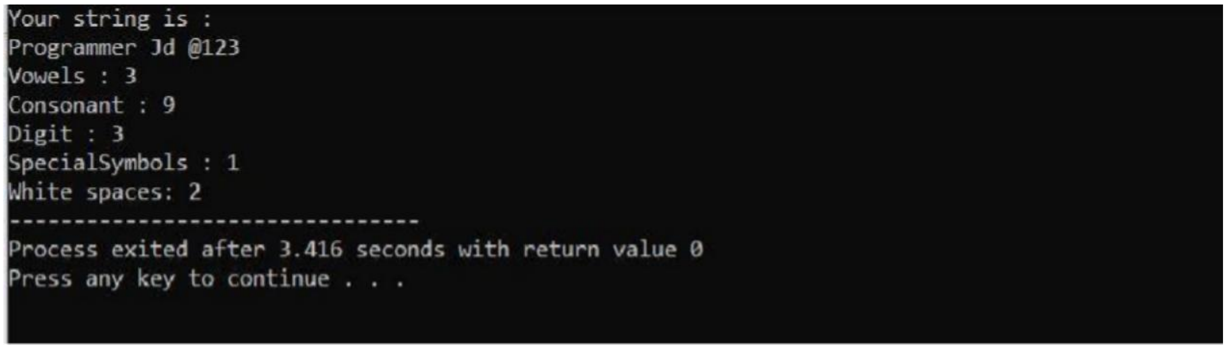
Experiment No: 2

Aim : Program to count digits, vowels and symbols in C.

Program :

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

int main() {
    char str[100];
    int i = 0;
    int vowels = 0, consonant = 0, digit = 0, symbols = 0, spaces = 0;
    FILE *fp;
    char ch;

    fp = fopen("input.txt", "r");
    if (fp == NULL) {
        printf("File not opened ");
        exit(1);
    }

    ch = fgetc(fp);
    printf("Your string is :\n");
    while (!feof(fp)) {
        str[i++] = ch;
        ch = fgetc(fp);
    }
    str[i] = '\0';
    printf("%s", str);
    fclose(fp);
    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == 'a' || str[i] == 'A' || str[i] == 'e' || str[i] == 'E' ||
            str[i] == 'i' || str[i] == 'I' || str[i] == 'o' || str[i] == 'O' ||
            str[i] == 'u' || str[i] == 'U') {
            vowels++;
        }
        else if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')) {
            consonant++;
        }
        else if (str[i] >= '0' && str[i] <= '9') {
            digit++;
        }
        else if (str[i] == ' ') {
            spaces++;
        }
        else {
            symbols++;
        }
    }

    printf("\nVowels : %d", vowels);
    printf("\nConsonant : %d", consonant);
    printf("\nDigit : %d", digit);
    printf("\nSpecial Symbols : %d", symbols);
    printf("\nWhite Spaces : %d\n", spaces);

    return 0;
}
```

Output :



```
Your string is :
Programmer Jd @123
Vowels : 3
Consonant : 9
Digit : 3
SpecialSymbols : 1
White spaces: 2
--------------------------------
Process exited after 3.416 seconds with return value 0
Press any key to continue . . .
```

Experiment No: 3

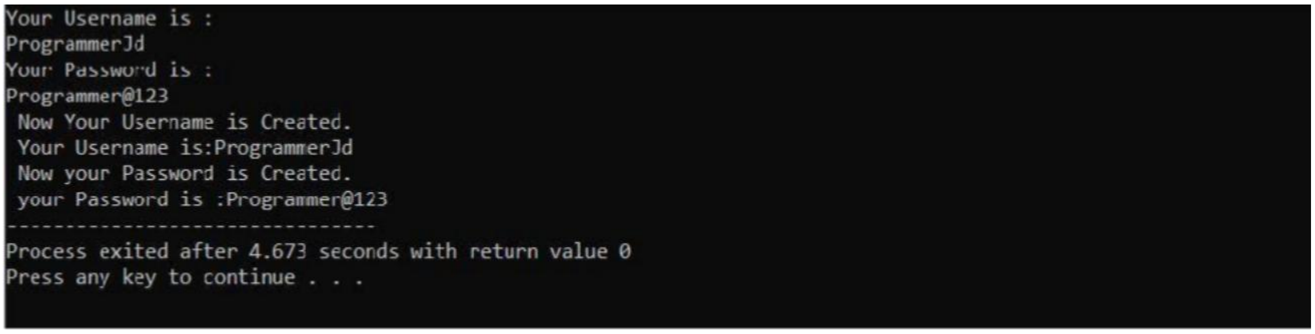
Aim: Program to check validation of User Name and Password in C.

Program:

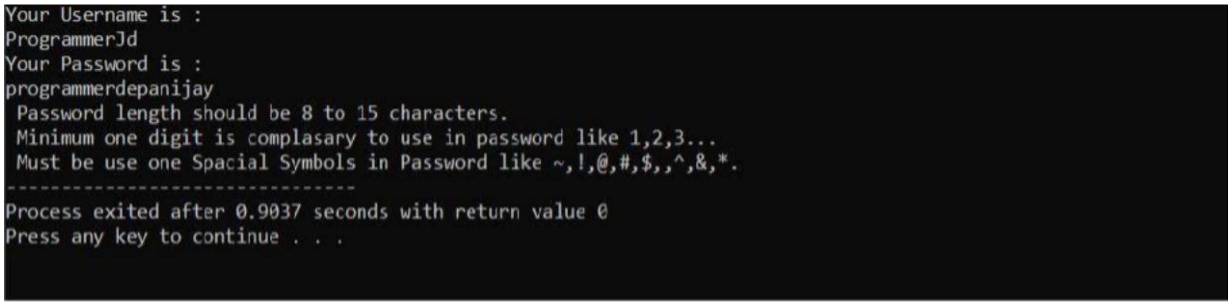
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main() {

    char str[25], pwd[20];
    int i = 0, j = 0;
    int hasDigit = 0, hasSymbol = 0, hasSpace = 0;
    FILE *fp, *fp1;
    char ch, ch1;

    // Reading username from file
    fp = fopen("user.txt", "r");
    if (fp == NULL) {
        printf("File not opened ");
        exit(1);
    }

    printf("Your Username is :\n");
    while ((ch = fgetc(fp)) != EOF) {
        str[i++] = ch;
    }
    str[i] = '\0';
    fclose(fp);
    printf("%s", str);

    // Username validations
    if (strcmp(str, "JayDepani") == 0 || strcmp(str, "DepaniJay") == 0 ||
        strcmp(str, "JayPatel") == 0 || strcmp(str, "PatelJay") == 0) {
        printf("\n This Username is already taken, please try another.");
        return 0;
```

```
}

if (!(str[0] >= 'A' && str[0] <= 'Z')) {
    printf("\n First character of the Username should be capital.");
}

for (i = 0; str[i] != '\0'; i++) {
    if (str[i] >= '0' && str[i] <= '9') {
        printf("\n Digits are not allowed in Username.");
        return 0;
    }
    if (str[i] == '~' || str[i] == '!' || str[i] == '@' || str[i] == '#' || str[i] == '$' ||
        str[i] == '%' || str[i] == '^' || str[i] == '&' || str[i] == '*') {
        printf("\n Special symbols are not allowed in Username.");
        return 0;
    }
}

// Reading password from file
fp1 = fopen("pass.txt", "r");
if (fp1 == NULL) {
    printf("File not opened ");
    exit(1);
}

printf("\nYour Password is :\n");
while ((ch1 = fgetc(fp1)) != EOF) {
    pwd[j++] = ch1;
}
pwd[j] = '\0';
fclose(fp1);
printf("%s", pwd);

// Password length check
if (strlen(pwd) < 8 || strlen(pwd) > 15) {
    printf("\n Password length should be between 8 to 15 characters.");
    return 0;
    }

    // Password digit & symbol check
    for (i = 0; pwd[i] != '\0'; i++) {
        if (pwd[i] >= '0' && pwd[i] <= '9')
            hasDigit = 1;
        if (pwd[i] == '~' || pwd[i] == '!' || pwd[i] == '@' || pwd[i] == '#' || pwd[i] == '$' ||
            pwd[i] == '%' || pwd[i] == '^' || pwd[i] == '&' || pwd[i] == '*')
            hasSymbol = 1;
        if (pwd[i] == ' ')
            hasSpace = 1;
    }

    if (!hasDigit) {
        printf("\n Minimum one digit is compulsory in Password.");
        return 0;
    }

    if (!hasSymbol) {
        printf("\n Must use one special symbol in Password like ~,!,@,#,$,%%,^,&,*.");
        return 0;
    }

    if (hasSpace) {
        printf("\n Password should not contain any space.");
        return 0;
    }

    // Final success output
    printf("\n\n Now Your Username is Created.\n Username: %s", str);
    printf("\n Now Your Password is Created.\n Password: %s", pwd);

    return 0;
}
```

Output :



```
Your Username is :
ProgrammerJd
Your Password is :
Programmer@123
 Now Your Username is Created.
 Your Username is:ProgrammerJd
 Now your Password is Created.
 your Password is :Programmer@123
--------------------------------
Process exited after 4.673 seconds with return value 0
Press any key to continue . . .
```



```
Your Username is :
ProgrammerJd
Your Password is :
programmerdepanijay
 Password length should be 8 to 15 characters.
 Minimum one digit is complasary to use in password like 1,2,3...
 Must be use one Spacial Symbols in Password like ~,!,@,#,$,,^,&,*.
------------------------------
Process exited after 0.9037 seconds with return value 0
Press any key to continue . . .
```

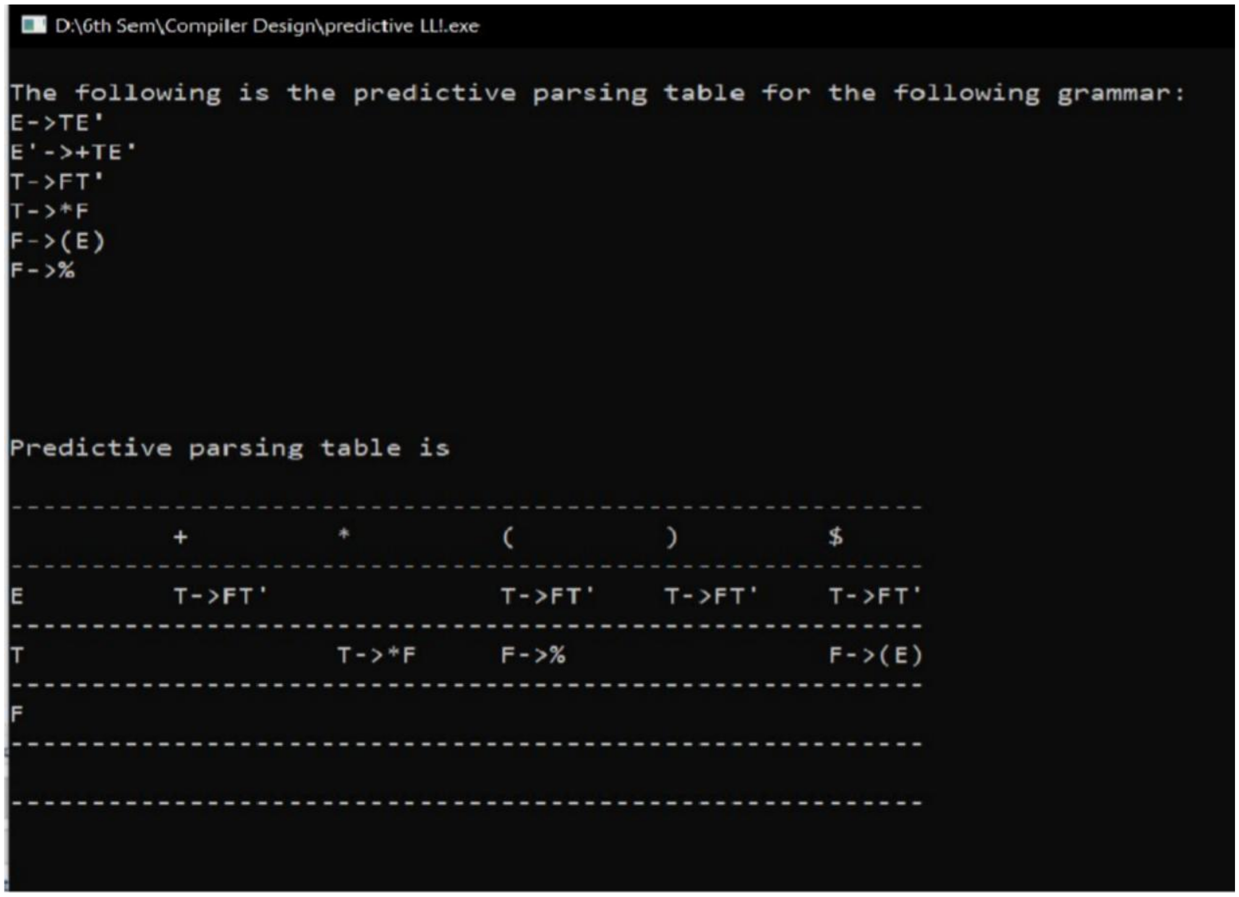
Experiment No: 4

Aim : Program to implement Predictive Parsing LL(1) in C.

Program :

```
#include <stdio.h>

#include <string.h>


char prol[6][10]  = {"E","E'","T","T'","F","F"};

char pror[6][10]  = {"TE'","+TE'","FT'","*FT'","(E)","%"};

char prod[6][15]  = {"E->TE'","E'->+TE'","T->FT'","T'->*FT'","F->(E)","F->%"};


char first[6][10]  = {"(%","+@","(%","*@","(%","%"};

char follow[6][10] = {"$)","$)","+$)","+$)","*+$)","*+$)"};


char table[4][6][20];


int numr(char c) {

    switch(c) {

        case 'E': return 0;

        case 'T': return 1;

        case 'F': return 2;

        case '+': return 0;

        case '*': return 1;

        case '(': return 2;

        case ')':  return 3;

        case '%': return 4;
        case '$': return 5;
    }
    return 2;
}

int main() {
    int i, j, k;

    for(i = 0; i < 4; i++)
        for(j = 0; j < 6; j++)
            strcpy(table[i][j], " ");

    printf("Predictive Parsing Table for Grammar:\n");
    for(i = 0; i < 6; i++)
        printf("%s\n", prod[i]);

    for(i = 0; i < 6; i++) {
        k = strlen(first[i]);
        for(j = 0; j < k; j++) {
            if(first[i][j] != '@')
                strcpy(table[numr(prol[i][0])][numr(first[i][j])], prod[i]);
        }
    }

    for(i = 0; i < 6; i++) {
        if(strstr(pror[i], "@")) {
            k = strlen(follow[i]);

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

                strcpy(table[numr(prol[i][0])][numr(follow[i][j])], prod[i]);

        }

    }


    char term[6] = { '+','*','(',')','%','$' };

    char nonT[3] = { 'E','T','F' };


    printf("\n\n       +          *          (          )          %%          $\n");

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

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

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

            printf("%-10s", table[i][j]);

        printf("\n\n");

    }


    return 0;

}
```

Output :



```
D:\6th Sem\Compiler Design\predictive LL!.exe

The following is the predictive parsing table for the following grammar:
E->TE'
E'->+TE'
T->FT'
T->*F
F->(E)
F->%




Predictive parsing table is

--------------------------------------------------------
          +         *         (         )         $
--------------------------------------------------------
E         T->FT'              T->FT'    T->FT'    T->FT'
--------------------------------------------------------
T                   T->*F     F->%                F->(E)
--------------------------------------------------------
F
--------------------------------------------------------

--------------------------------------------------------
```

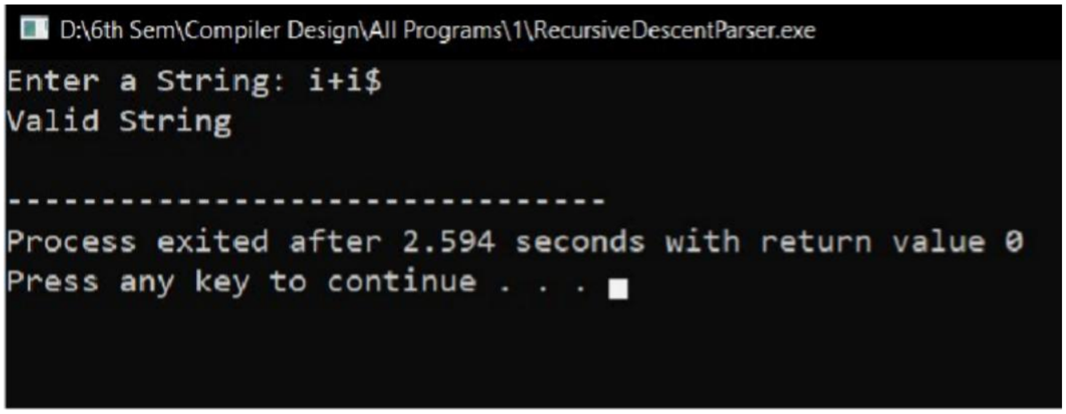
Experiment No. : 5

Aim : Program to implement Recursive Descent Parsing in C.

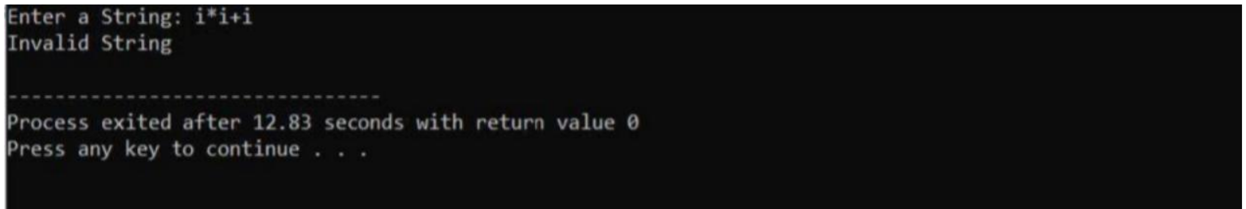
Program :

```
#include <stdio.h>

static char c[20];

int input = 0;     // index pointer for reading the string

void E();

void EPRIME();

int main() {

    printf("Enter a String: ");

    scanf("%s", c);


    E();   // start parsing from start symbol


    if (c[input] == '$') {

        printf("Valid String\n");

    } else {

        printf("Invalid String\n");

    }

    return 0;

}


void E() {

    // E → i E'

    if (c[input] == 'i') {
        input++;
        EPRIME();
    }
}

void EPRIME() {
    // E' → + i E' | ε
    if (c[input] == '+') {
        input++;
        if (c[input] == 'i') {
            input++;
            EPRIME();
        }
    }
}
```

Output :



```
D:\6th Sem\Compiler Design\All Programs\1\RecursiveDescentParser.exe
Enter a String: i+i$
Valid String

--------------------------------
Process exited after 2.594 seconds with return value 0
Press any key to continue . . .
```



```
Enter a String: i*i+i
Invalid String

--------------------------------
Process exited after 12.83 seconds with return value 0
Press any key to continue . . .
```

Experiment No: 6

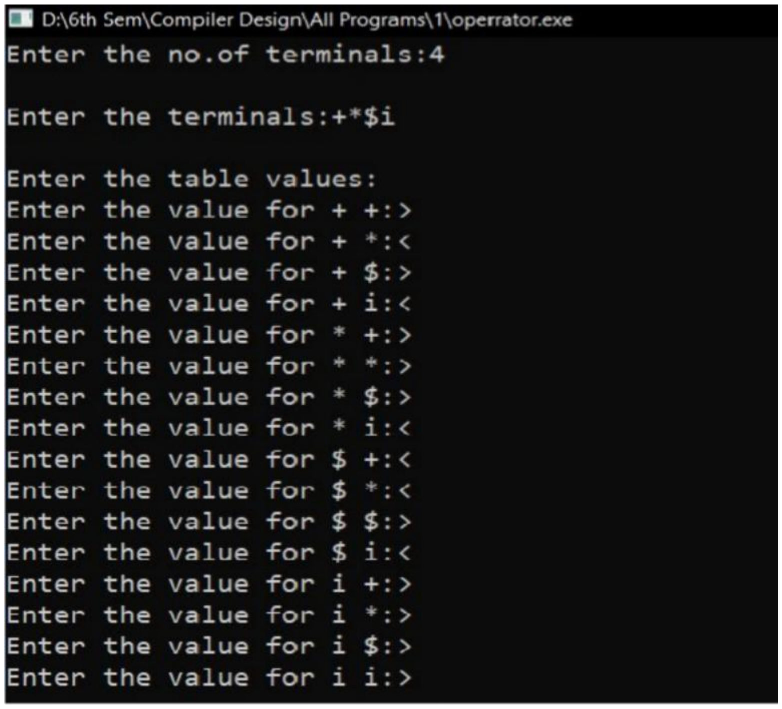
Aim: Program to implement Operator Precedence Parsing in C. Program :

```
#include<stdio.h>

#include<conio.h>

#include<string.h> int

main()

{

char stack[20],ip[20],opt[10][10][1],ter[10]; int

i,j,k,n,top=0,row,col;

int len;

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

{

stack[i]=NULL;

ip[i]=NULL;

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

{ opt[i][j][1]=NUL L;

}

}

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++)
```

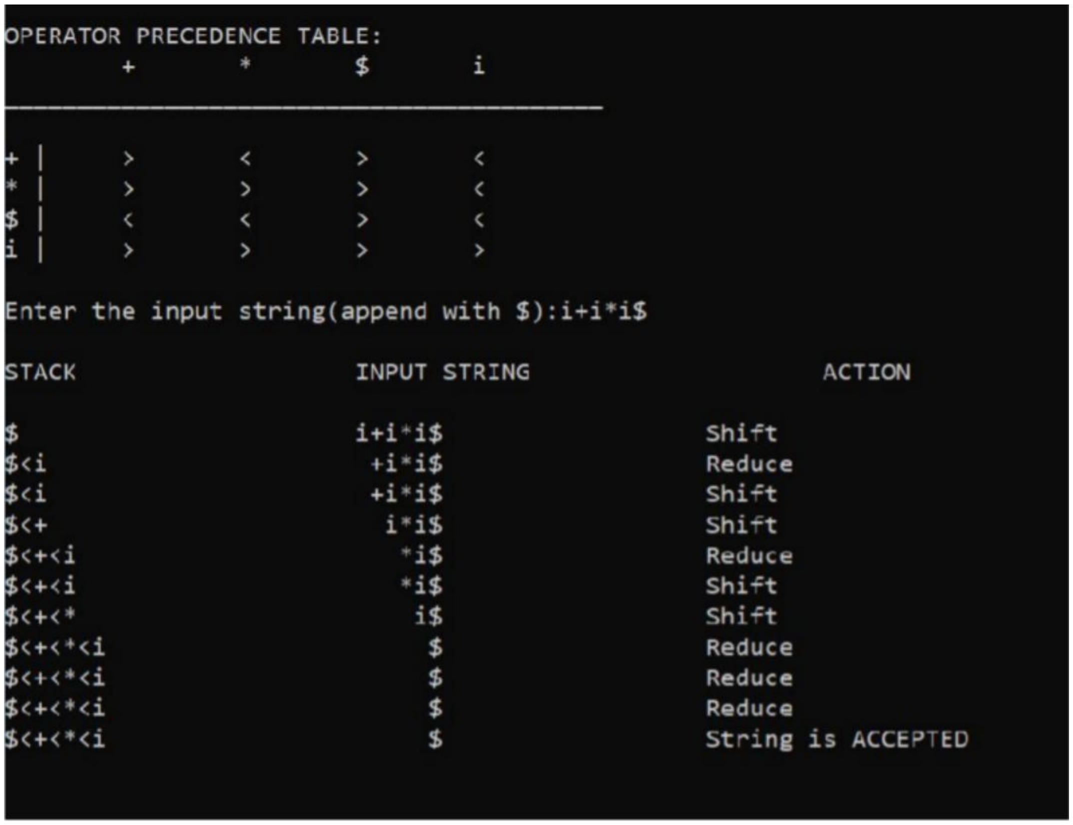
```
{
for(j=0;j<n;j++)
{
printf("Enter the value for %c %c:",ter[i],ter[j]);
scanf("%s",opt[i][j]);
}
}
printf("\nOPERATOR PRECEDENCE TABLE:\n");
for(i=0;i<n;i++)
{
printf("\t%c",ter[i]);
}
printf("\n_____________________________________________");
printf("\n");
for(i=0;i<n;i++)
{
printf("\n%c |",ter[i]);
for(j=0;j<n;j++)
{
printf("\t%c",opt[i][j][0]);
}
}
stack[top]='$';
printf("\n\nEnter the input string(append with $):");
scanf("%s",ip);
i=0;
```

```
printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");

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


len=strlen(ip);

while(i<=len)

{

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

{

if(stack[top]==ter[k])

row=k; if(ip[i]==ter[k])

col=k;

}

if((stack[top]=='$')&&(ip[i]=='$'))

{

printf("String is ACCEPTED");

break;

}

else if((opt[row][col][0]=='<') ||(opt[row][col][0]=='='))

{

stack[++top]=opt[row][col][0];

stack[++top]=ip[i];

ip[i]=' ';

printf("Shift %c",ip[i]); i++;

}
else
{
if(opt[row][col][0]=='>')
{
while(stack[top]!='<')
{
--top;
}
top=top-1;
printf("Reduce");
}
else
{
printf("\nString is not accepted");
break;
}
}
printf("\n");
printf("%s\t\t\t%s\t\t\t",stack,ip);
}
getch();
}
```

Output :



```
D:\6th Sem\Compiler Design\All Programs\1\operrator.exe
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 + $:>
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 $ *:<
Enter the value for $ $:>
Enter the value for $ i:<
Enter the value for i +:>
Enter the value for i *:>
Enter the value for i $:>
Enter the value for i i:>
```



```
OPERATOR PRECEDENCE TABLE:
        +       *       $       i
________________________________________

+ |     >       <       >       <
* |     >       >       >       <
$ |     <       <       >       <
i |     >       >       >       >

Enter the input string(append with $):i+i*i$

STACK                   INPUT STRING                    ACTION

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

Experiment No: 7

Aim : Program to implement LALR Parsing in C.

Program :

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

void push(char *,int *,char);

char stacktop(char *);

void isproduct(char,char); int

ister(char);

int  isnter(char);  int

isstate(char);    void

error();

void isreduce(char,char); char

pop(char *,int *);

void printt(char *,int *,char [],int);

void rep(char [],int);

struct action

{

char row[6][5];

};

const struct action A[12]={

{"sf","emp","emp","se","emp","emp"},

{"emp","sg","emp","emp","emp","acc"},

{"emp","rc","sh","emp","rc","rc"},
{"emp","re","re","emp","re","re"},
{"sf","emp","emp","se","emp","emp"},
{"emp","rg","rg","emp","rg","rg"},
{"sf","emp","emp","se","emp","emp"},
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","sl","emp"},
{"emp","rb","sh","emp","rb","rb"},
{"emp","rb","rd","emp","rd","rd"},
{"emp","rf","rf","emp","rf","rf"}
};
struct gotol
{
char r[3][4];
};
const struct gotol G[12]={
{"b","c","d"},
{"emp","emp","emp"},
{"emp","emp","emp"},
{"emp","emp","emp"},
{"i","c","d"},
{"emp","emp","emp"},
{"emp","j","d"},
{"emp","emp","k"},
{"emp","emp","emp"},
{"emp","emp","emp"},
};
```

```
char ter[6]={'i','+','*',')','(','$'};

char nter[3]={'E','T','F'};

char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};

char stack[100];

int top=-1; char

temp[10]; struct

grammar

{

char left; char

right[5];

};

const struct grammar rl[6]={

{'E',"e+T"},

{'E',"T"},

{'T',"T*F"},

{'T',"F"},

{'F',"(E)"},

{'F',"i"},

};

int main()

{

char inp[80],x,p,dl[80],y,bl='a'; int

i=0,j,k,l,n,m,c,len;

printf(" Enter the input :");

scanf("%s",inp);

len=strlen(inp);
```

```
inp[len]='$';

inp[len+1]='\0';

push(stack,&top,bl); printf("\n

stack \t\t\t input");

printt(stack,&top,inp,i);

do

{

x=inp[i]; p=stacktop(stack);

isproduct(x,p);

if(strcmp(temp,"emp")==0)

error();

if(strcmp(temp,"acc")==0)

break;

else

{

if(temp[0]=='s')

{

push(stack,&top,inp[i]);

push(stack,&top,temp[1]);

i++;

}

else

{

if(temp[0]=='r')

{
```

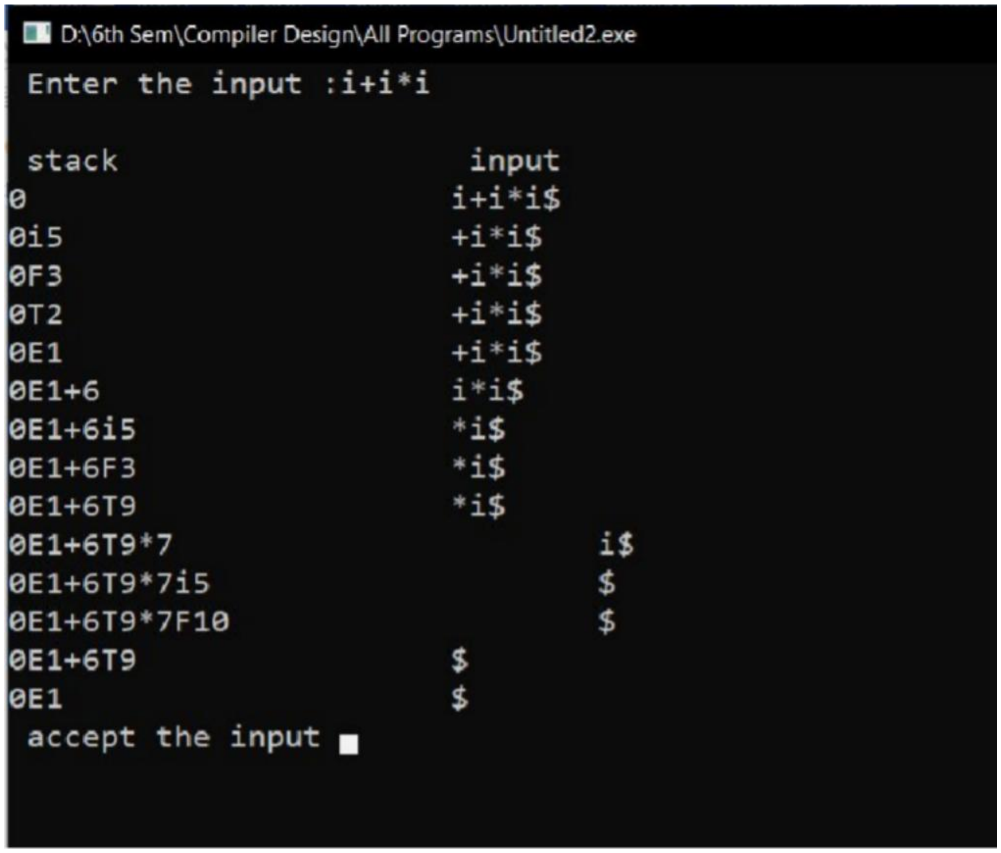
```
j=isstate(temp[1]);

strcpy(temp,rl[j-2].right);

dl[0]=rl[j-2].left;

dl[1]='\0';

n=strlen(temp); for(k=0;k<2*n;k++)

pop(stack,&top);

for(m=0;dl[m]!='\0';m++)

push(stack,&top,dl[m]);

l=top;

y=stack[l-1]; isreduce(y,dl[0]);

for(m=0;temp[m]!='\0';m++)

push(stack,&top,temp[m]);

}

}

}

printt(stack,&top,inp,i);

}while(inp[i]!='\0');

if(strcmp(temp,"acc")==0)

printf(" \n accept the input "); else

printf(" \n do not accept the input ");

getch();

}

void push(char *s,int *sp,char item)
```

```
{
if(*sp==100)
printf(" stack is full ");
else
{
*sp=*sp+1;
s[*sp]=item;
}
}
char stacktop(char *s)
{
char i; i=s[top];
return i;
}
void isproduct(char x,char p)
{
int k,l; k=ister(x);
l=isstate(p);
strcpy(temp,A[l-1].row[k-1]);
}
int ister(char x)
{
int i; for(i=0;i<6;i++)
if(x==ter[i])

return i+1; return

0;

}

int isnter(char x)

{

int i;

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

if(x==nter[i])

return i+1; return 0;

}


int isstate(char p)

{

int i;

for(i=0;i<12;i++)

if(p==states[i])

return i+1; return 0;

}

void error()

{

printf(" error in the input "); exit(0);

}
```

```
void isreduce(char x,char p)

{

int k,l;

k=isstate(x);

l=isnter(p);

strcpy(temp,G[k-1].r[l-1]);

}

char pop(char *s,int *sp)

{

char item;

if(*sp==-1)

printf(" stack is empty ");

else

{

item=s[*sp];

*sp=*sp-1;

}

return item;

}

void printt(char *t,int *p,char inp[],int i)

{

int r; printf("\n");

for(r=0;r<=*p;r++)

rep(t,r); printf("\t\t\t");
```

```
for(r=i;inp[r]!='\0';r++)

printf("%c",inp[r]);

}

void rep(char t[],int r)

{

char c; c=t[r];

switch(c)

{

case 'a': printf("0");

break;

case 'b': printf("1");

break;

case 'c': printf("2");

break;

case 'd': printf("3");

break;

case 'e': printf("4");

break;

case 'f': printf("5");

break;

case 'g': printf("6");

break;

case 'h': printf("7");

break;

case 'm': printf("8");
break;
case 'j': printf("9");
break;
case 'k': printf("10");
break;
case 'l': printf("11");
break;
default :printf("%c",t[r]);
break;
}
}
```

Output :



```
D:\6th Sem\Compiler Design\All Programs\Untitled2.exe
 Enter the input :i+i*i

 stack                   input
0                       i+i*i$
0i5                     +i*i$
0F3                     +i*i$
0T2                     +i*i$
0E1                     +i*i$
0E1+6                   i*i$
0E1+6i5                 *i$
0E1+6F3                 *i$
0E1+6T9                 *i$
0E1+6T9*7                       i$
0E1+6T9*7i5                     $
0E1+6T9*7F10                    $
0E1+6T9                 $
0E1                     $
 accept the input
```

Experiment No: 8

Aim : To Study about Lexical Analyzer Generator (LEX) and Flex (Fast Lexical Analyzer)

Lex - A Lexical Analyzer Generator:-

Lex is a program generator designed for lexical processing of character input streams. It accepts a high-level, problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions.

The grammar in the above diagram is a text file you create with a text edtior. Yacc will read your grammar and generate C code for a syntax analyzer or parser. The syntax analyzer uses grammar rules that allow it to analyze tokens from the lexical analyzer and create a syntax tree. The syntax tree imposes a hierarchical structure the tokens. For example, operator precedence and associativity are apparent in the syntax tree. The next step, code generation, does a depth-first Lexical Analyzer Syntax Analyzer a = b + c * d id1 = id2 + id3 * id4 = + * id1 source code tokens syntax tree id2 id3 id4 load id3 mul id4 add id2 store id1 Code Generator generated code Lex Yacc patterns grammar 5 walk of the syntax tree to generate code. Some compilers produce machine code, while others, as shown above, output assembly language.

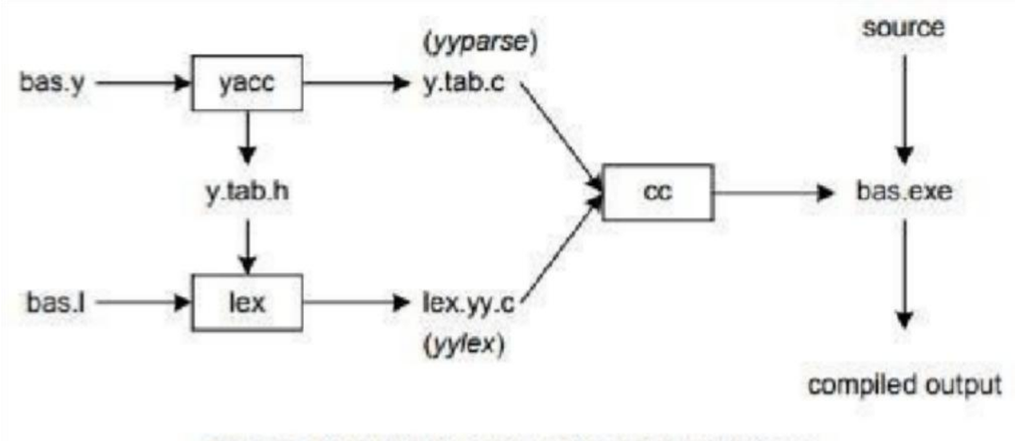


Figure 2: Building a Compiler with Lex/Yacc

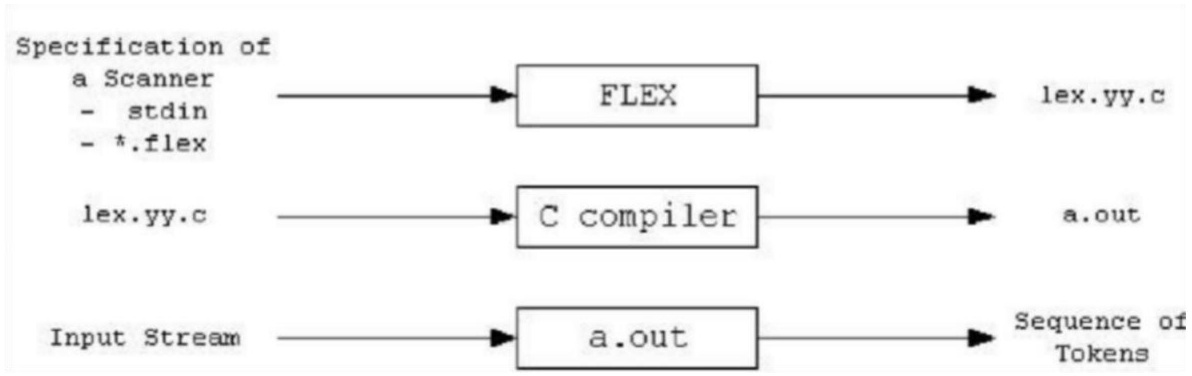
What is Flex?

- Flex is a powerful, open source application framework which allows to build traditional applications for browser, mobile and desktop using the same programming model, tool, and codebase.

- Flex provides FLEX SDK consisting of the Flex class library (ActionScript classes), the Flex compilers, the debugger, the MXML and ActionScript programming languages, and other utilities to build expressive and interactive rich internet applications (RIA)
- Flex takes care of the user interface (UI) or the client-side functionality of a web application. Server-side functionality dependent on server-side components written in a traditional scripting language

How to use FLEX?

FLEX (Fast LEXical analyzer generator) is a tool for generating scanners. In stead of writing a scanner from scratch, you only need to identify the *vocabulary* of a certain language (e.g. Simple), write a specification of patterns using regular expressions (e.g. DIGIT [0-9]), and FLEX will construct a scanner for you. FLEX is generally used in the manner depicted here:



First, FLEX reads a specification of a scanner either from an input file *.lex, or from standard input, and it generates as output a C source file *lex.yy.c*. Then, *lex.yy.c* is compiled and linked with the "-lfl" library to produce an executable *a.out*. Finally, *a.out* analyzes its input stream and transforms it into a sequence of tokens.

▪ ***.lex** is in the form of pairs of regular expressions and C code. (sample1.lex, sample2.lex) ▪ **lex.yy.c** defines a routine *yylex()* that uses the specification to recognize tokens. ▪ **a.out** is actually the scanner!

How to Compile & Run LEX / YACC Programs on Windows ?

If you are installing Ubuntu (or any Linux based OS) on your system either through Virtual Box or by making your system Multi-Bootable, just to execute your Lex & Yacc programs; then you might be wasting your HDD space & your valuable time. You can easily skip this annoying process and run your programs in Windows OS without any hassles.

Here's how you can do it:

Installing Softwares:

1. Download **Flex 2.5.4a**
2. Download **Bison 2.4.1**
3. Download **DevC++**
4. **Install** Flex **at** "C:\GnuWin32"
5. **Install** Bison **at** "C:\GnuWin32"
6. **Install** DevC++ **at** "C:\Dev-Cpp"
7. Open Environment Variables.
8. Add "**C:\GnuWin32\bin;C:\Dev-Cpp\bin;**" to path.

Compilation & Execution of your Program:

1. Open Command prompt and switch to your working directory where you have stored your lex file (".l") and yacc file (".y")
2. Let your lex and yacc files be "hello.l" and "hello.y". Now, follow the preceding steps to compile and run your program.
    1. For Compiling **Lex** file only:
        1. flex hello.l
        2. gcc lex.yy.c
    2. For Compiling **Lex & Yacc** file both:
        1. flex hello.l
        2. bison -dy hello.y
        3. gcc lex.yy.c y.tab.c
3. For **Executing** the Program
    1. a.exe

EXAMPLE:- **HELLO.L FILE**

```
%{

#include "y.tab.h"

int yyerror(char *errormsg);

%}

%%

("hi"|"oi")"\n"      { return HI; }
```

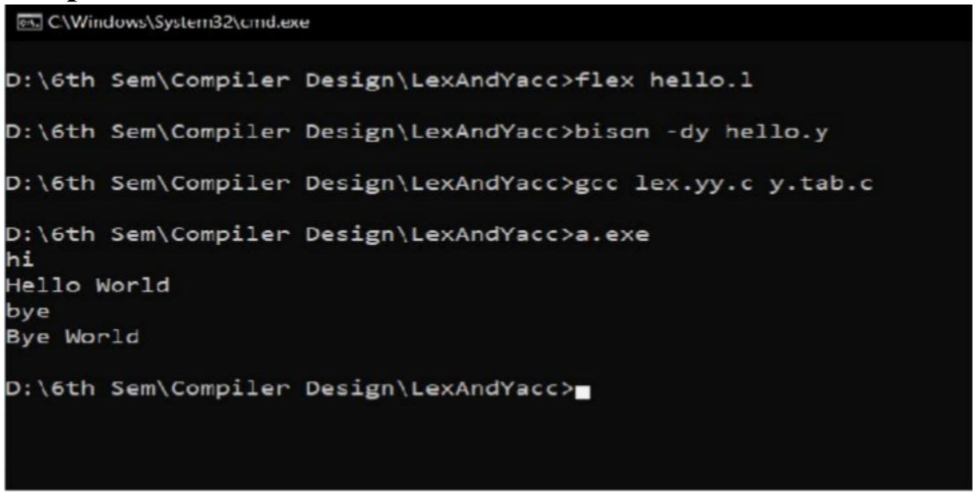
```
("tchau"|"bye")"\n" { return BYE; }

.                   { yyerror("Unknown char"); }

%%

int main(void)

{

  yyparse();

  return 0;

}

int yywrap(void)

{

  return 0;

}


int yyerror(char *errormsg)

{

   fprintf(stderr, "%s\n", errormsg); exit(1);

}
```

<u>HELLO.Y FILE</u>

```
%{

#include       <stdio.h>
#include <stdlib.h> int
yylex(void);
int yyerror(const char *s);

%}

%token HI BYE

%%
```

```
program:
        hi bye
       ;

hi:                                                   HI     { printf("Hello World\n"); }
                                                      ;

bye:                                                  BYE    { printf("Bye World\n"); exit(0); }
                                                       ;
```

Output :



```
C:\Windows\System32\cmd.exe

D:\6th Sem\Compiler Design\LexAndYacc>flex hello.l

D:\6th Sem\Compiler Design\LexAndYacc>bison -dy hello.y

D:\6th Sem\Compiler Design\LexAndYacc>gcc lex.yy.c y.tab.c

D:\6th Sem\Compiler Design\LexAndYacc>a.exe
hi
Hello World
bye
Bye World

D:\6th Sem\Compiler Design\LexAndYacc>
```

Experiment No: 9

Aim : A). Create a Lexer to take input from text file and count no of characters, no. of lines & no. of words.

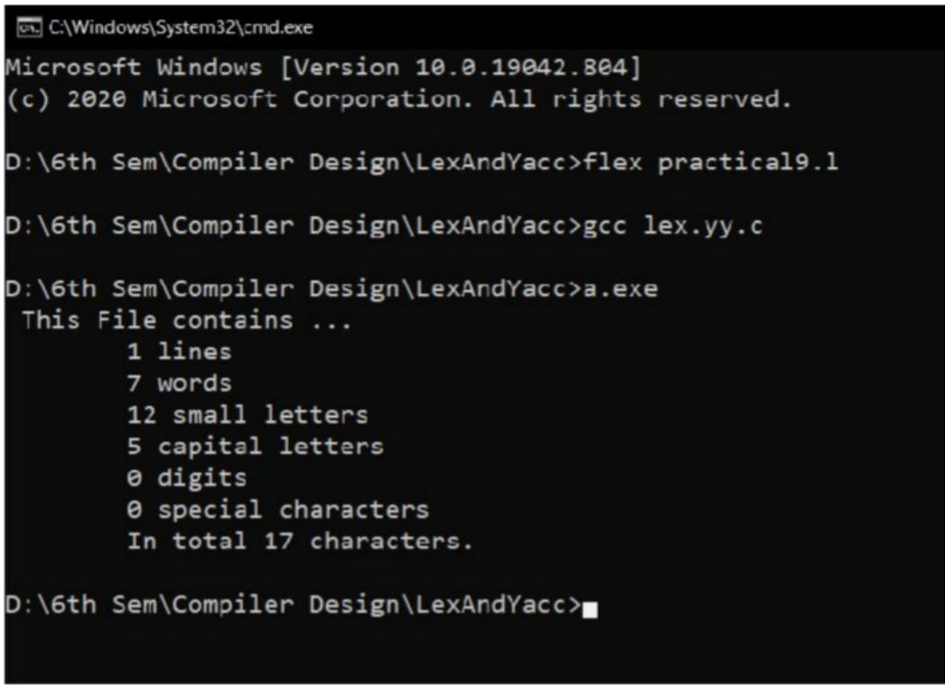
Program :

```
%{
#include<stdio.h>
 int lines=0, words=0,s_letters=0,c_letters=0, num=0, spl_char=0,total=0;
%}
%%
\n { lines++; words++;} [\t '
'] words++;
[A-Z] c_letters++;
[a-z] s_letters++;
[0-9] num++;
. spl_char++;
%%
main(void)
{
yyin= fopen("practical9.txt","r"); yylex();
total=s_letters+c_letters+num+spl_char;
printf(" This File contains ...");
printf("\n\t%d lines", lines); printf("\n\t%d
words",words); printf("\n\t%d small
letters", s_letters); printf("\n\t%d capital
letters",c_letters); printf("\n\t%d digits",
num);
printf("\n\t%d special characters",spl_char);
 printf("\n\tIn total %d characters.\n",total);
}
int yywrap()
{
return(1);
}
```

Practical9.txt FILE :

```
Hii
I'm Dijay From Cse
```

Output :



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\6th Sem\Compiler Design\LexAndYacc>flex practical9.l

D:\6th Sem\Compiler Design\LexAndYacc>gcc lex.yy.c

D:\6th Sem\Compiler Design\LexAndYacc>a.exe
 This File contains ...
        1 lines
        7 words
        12 small letters
        5 capital letters
        0 digits
        0 special characters
        In total 17 characters.

D:\6th Sem\Compiler Design\LexAndYacc>
```

B). Write a Lex program to count number of vowels and consonants in a given input string.
Program :

```
%{
int vow_count = 0;
int const_count = 0;
%}

%%
[aeiouAEIOU]      { vow_count++; }        /* Count vowels */
[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z]   { const_count++; }   /* Count consonants */
.                 { /* ignore other characters */ }
%%

int main() {
   yyin = fopen("practical8.txt", "r");
   if (!yyin) {
      printf("File not found!\n");
      return 1;
   }
```

```
    yylex();

    printf("The number of vowels are: %d\n", vow_count);
    printf("The number of consonants are: %d\n", const_count);
    return 0;
}

int yywrap() {
    return 1;
}
```

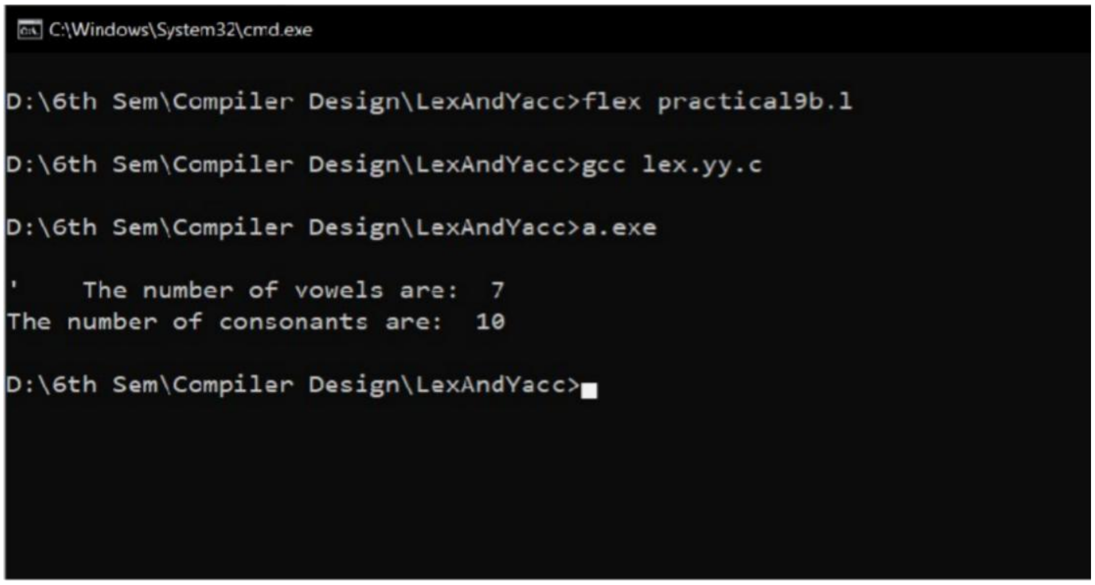
practical8.txt (Input File Example)

```
Hii
I'm Dijay From Cse
```

▶ How to Compile and Run (Linux / Windows with Flex)

```
flex vowel_const.l
gcc lex.yy.c -o count
./count
```

Output :


C:\Windows\System32\cmd.exe

```
D:\6th Sem\Compiler Design\LexAndYacc>flex practical9b.l

D:\6th Sem\Compiler Design\LexAndYacc>gcc lex.yy.c

D:\6th Sem\Compiler Design\LexAndYacc>a.exe

'    The number of vowels are:  7
The number of consonants are:  10

D:\6th Sem\Compiler Design\LexAndYacc>
```

Experiment No: 10

Aim : A). Write a Lex program to print out all numbers from the given file.

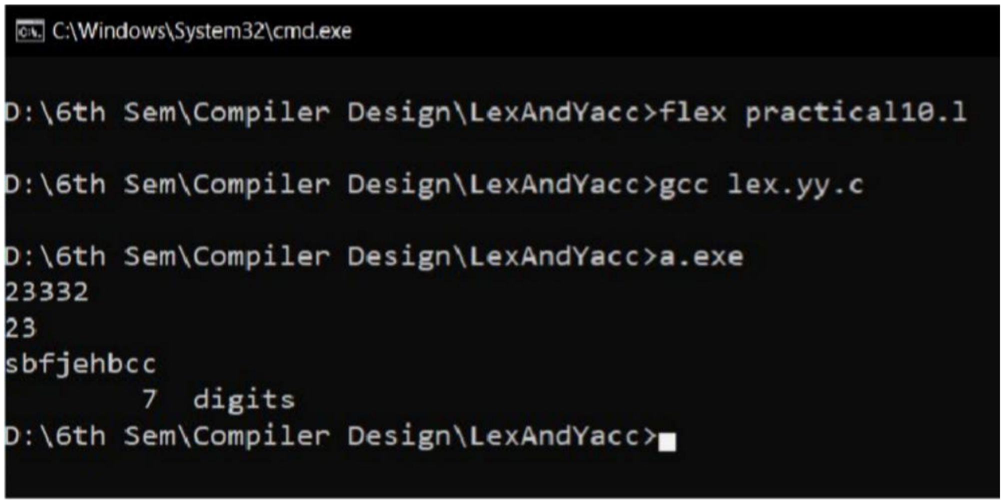
Program :

```
%{
#include<stdio.h> int
num=0;
%}
%%
[0-9] num++; ECHO;
%%
main(void)
{
yyin= fopen("practical10.txt","r");
yylex();
printf("\n\t%d digits", num);
}
int yywrap()
{
return(1);
}
```

Practical10.txt FILE

```
23332
23
Sbfjehbcc
```

Output :



```
C:\Windows\System32\cmd.exe

D:\6th Sem\Compiler Design\LexAndYacc>flex practical10.l

D:\6th Sem\Compiler Design\LexAndYacc>gcc lex.yy.c

D:\6th Sem\Compiler Design\LexAndYacc>a.exe
23332
23
sbfjehbcc
        7  digits
D:\6th Sem\Compiler Design\LexAndYacc>
```

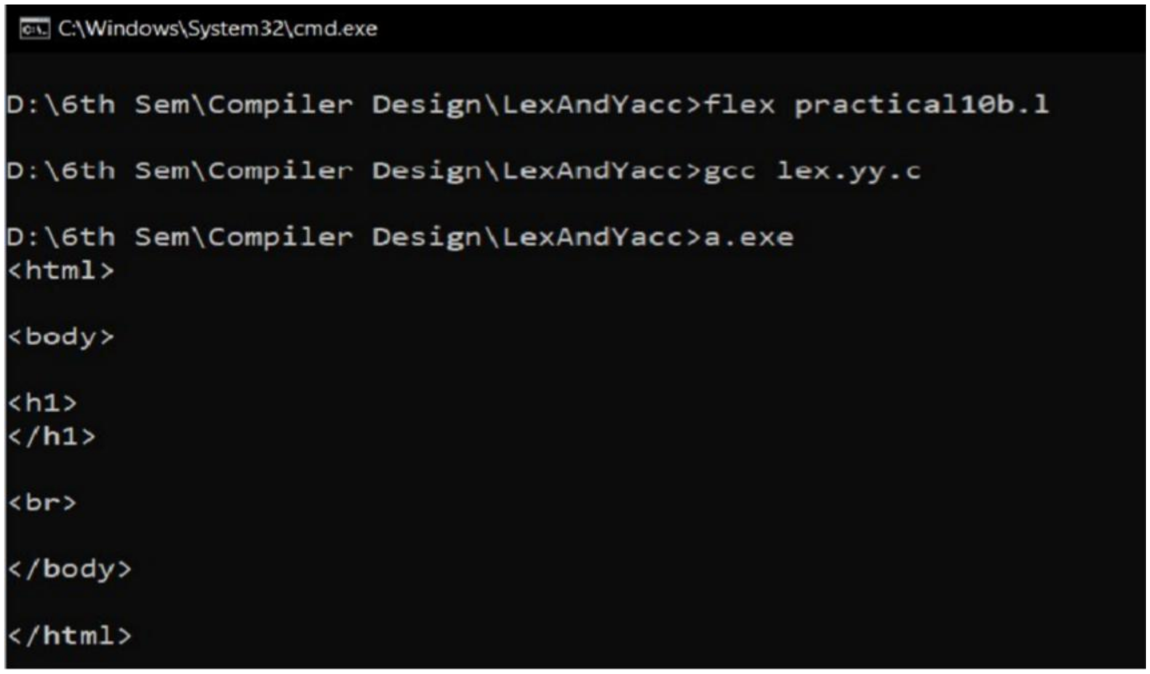
B). Write a Lex program to printout all HTML tags in file. Program :

```
%{
#include<stdio.h>
%}

%%
\<[^>]*\> printf("%s\n",yytext);
.|\n;
%%

int yywrap()
{
return 1;
}

int main()
{
yyin=fopen("practical10b.txt","r");
yylex();
return 0;
}
```

Practical10b.txt FILE

```
<html>
<body>
<h1> Hello</h1>
<br>
</body>
</html>
```

Output :



```
C:\Windows\System32\cmd.exe

D:\6th Sem\Compiler Design\LexAndYacc>flex practical10b.l

D:\6th Sem\Compiler Design\LexAndYacc>gcc lex.yy.c

D:\6th Sem\Compiler Design\LexAndYacc>a.exe
<html>

<body>

<h1>
</h1>

<br>

</body>

</html>
```

C). Write a Lex program which adds line numbers to the given file and display the same onto the standard output.

Program :

```
%{
#include <stdio.h>
int line_number = 1;
%}

line   .*\n

%%
{line}    { printf("%d %s", line_number++, yytext); }
.        { /* ignore remaining characters */ }
%%

int main() {
   extern FILE *yyin;
   yyin = fopen("add.txt", "r");
   if (yyin == NULL) {
      printf("File Not Found\n");
      return 1;
   }
   yylex();
   return 0;
}

int yywrap() {
    return 1;
}
```

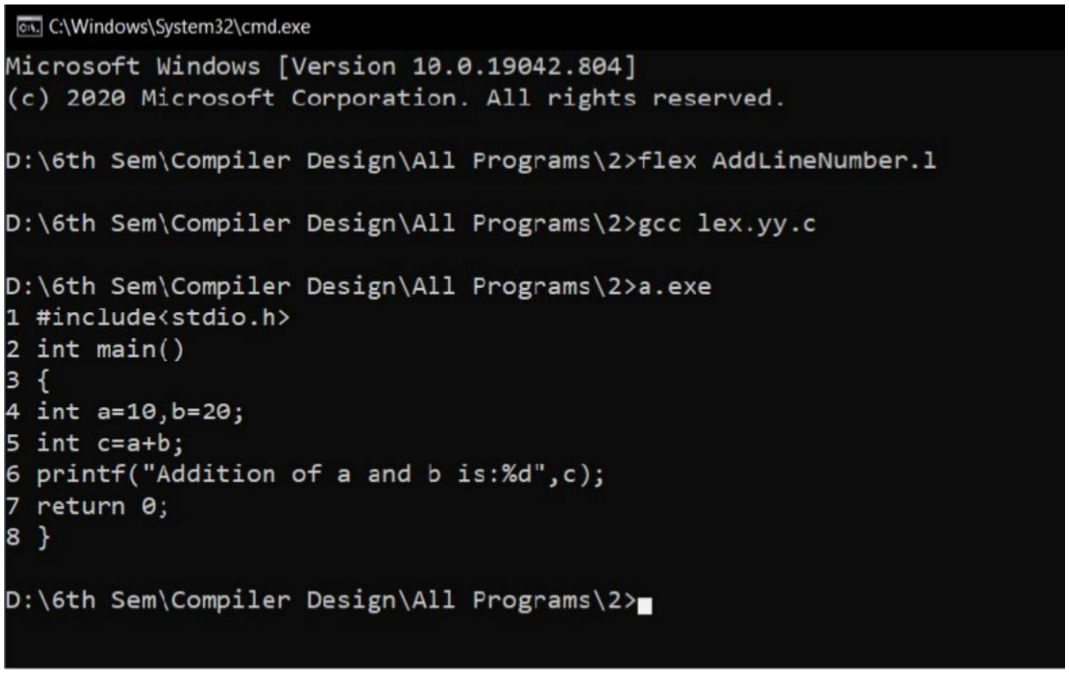
Input File (add.txt)

```
#include<stdio.h>
int main()
{
int a=10,b=20;
int c=a+b;
printf("Addition of a and b is:%d",c);
return 0;
}
```

▶ How to Compile and Run

```
flex add_line_numbers.l
gcc lex.yy.c -o add
./add
```

Output :



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\6th Sem\Compiler Design\All Programs\2>flex AddLineNumber.l

D:\6th Sem\Compiler Design\All Programs\2>gcc lex.yy.c

D:\6th Sem\Compiler Design\All Programs\2>a.exe
1 #include<stdio.h>
2 int main()
3 {
4 int a=10,b=20;
5 int c=a+b;
6 printf("Addition of a and b is:%d",c);
7 return 0;
8 }

D:\6th Sem\Compiler Design\All Programs\2>
```