EXPERIMENT-1

Aim: Design a Lexical analyzer for the given language. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int isKeyword(char *str) {
char k[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;
for (i = 0; i < 32; i++) {
if (strcmp(k[i], str) == 0)
{ return 1;
}
}
return 0;
}
int isFunction(char *str) {
if (strcmp(str, "main") == 0 || strcmp(str, "printf") == 0)
{ return 1;
}
return 0;
}
int main() {
int kc, lno = 1, sno = 0;
char fn[20], c, buf[30];
FILE *fp;
printf("\nEnter the file name: ");
scanf("%s", fn);
fp = fopen(fn, "r");
if (fp == NULL) {
```

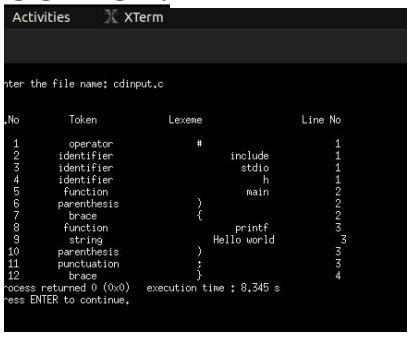
```
printf("Error opening file %s\n", fn);
return 1;
}
printf("\n\nS.No  Token  Lexeme  Line No\n");
while ((c = fgetc(fp)) != EOF) {
if (isalpha(c))
{ buf[kc = 0] = c;
while (isalnum(c = fgetc(fp))) { buf[++kc]
= c;
}
buf[++kc] = '\0';

if (isKeyword(buf))
printf("\n%4d   keyword  %20s  %7d", ++sno, buf, lno);
else if (isFunction(buf))
printf("\n%4d  function  %20s  %7d", ++sno, buf, lno);
else
printf("\n%4d  identifier  %20s   %7d", ++sno, buf, lno);
}
else if (isdigit(c))
{ buf[kc = 0] = c;
while (isdigit(c = fgetc(fp)))
{ buf[++kc] = c;
}
buf[++kc] = '\0';
printf("\n%4d number  %20s  %7d", ++sno, buf, lno);
}
else if (c == '(' || c == ')') {
printf("\n%4d       parenthesis         %6c                  %7d", ++sno, c, lno);
}
else if (c == '{' || c == '}') {
printf("\n%4d         brace              %6c                  %7d", ++sno, c, lno);
}
else if (c == '[' || c == ']') {
printf("\n%4d       array index  %6c   %7d", ++sno, c, lno);
}
else if (c == ',' || c == ';') {
printf("\n%4d  punctuation   %6c   %7d", ++sno, c, lno);
}
else if (c == '"')
{ kc = 0;
while ((c = fgetc(fp)) != '")
{ buf[kc++] = c;
```

```
}
buf[kc] = '\0';
printf("\n%4d string %20s %7d", ++sno, buf, lno);
}
else if (c == ' ' || c == '\t')
{ continue;
}
else if (c == '\n') {
++lno;
}
else {
printf("\n%4d  operator  %6c  %7d", ++sno, c, lno);
}
}
fclose(fp);
return 0;
}
```

INPUT: cdinput.c

```
#include<stdio.h>
main(){
printf("Hello world");
}
```

OUTPUT:



```
Activities        X XTerm

nter the file name: cdinput.c

.No        Token            Lexeme                Line No

 1        operator              #                     1
 2       identifier                   include         1
 3       identifier                     stdio         1
 4       identifier                         h         1
 5        function                       main         2
 6       parenthesis             )                    2
 7         brace                 {                    2
 8        function                     printf         3
 9         string                Hello world          3
10       parenthesis             )                    3
11       punctuation             ;                    3
12         brace                 }                    4
rocess returned 0 (0x0)   execution time : 8.345 s
ress ENTER to continue.
```

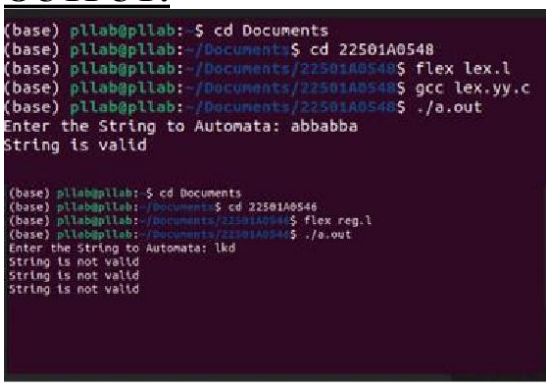
EXPERIMENT-2

Aim :Implement the lexical analyzer using LEX program for the regular expression Re is(a+b)*

PROGRAM:

```
%{
#include<stdio.h>
int result = 0;
%}
pattern a[a|b]*[\n]
%%
{pattern} {printf("String is valid \n "); }
. { printf("String is not valid \n"); }
%%
int yywrap()
{
return 1;
}
int main()
{
printf("Enter the String to Automata: ");
yylex();
result==1?printf("String is valid \n"):
printf("String is not valid \n");
return 0;
}
```

OUTPUT:



EXPERIMENT-3

Aim: Implement the lexical analyzer using JLEX, FLEX or LEX or other lexical analyzer generating stools.

PROGRAM:

```
%{

#include<stdio.h>
char
*word[]={"keyword","identifier","operator","preprocessor","comment","invalid
literal","reserved ", "number","string"};
void display (int);
%}
keyword  "int"|"char"|"short"|"void"|"long"|"if"|"else"|"case"|"for"|"do"|"while"|"
break"|"auto"|"static"|"const"|"enum"|"struct"

reserved   "main"|"FILE"|"printf"|"scanf"|"puts"|"putc"|"getc"|"pow"
comments  "//".|"/"."/"
operator   "."|"{"|"}"|"("|")"|"["|"]"|"->"|"+"|"-"|"*"|"/"|"="|"+="|";"
preprocessor #.*
string "\"".*"\""
identifier  [_][a-zA-Z][a-zA-Z0-9]
number [0-9]+[.][0-9]
%%
{comments} { display(4);}
{preprocessor} { display(3);}
{reserved} { display(6);}
{keyword} { display(0); }
{operator} { display(2);}
{string} { display(8);}
{identifier} { display(1); }
{number} {display(7);}
[\n\t' '] {};
. {display(5); }
%%
void display(int n)
{
printf("\n%s --> %s\n",yytext,word[n]);
}
int yywrap()
{
```

```
return 1;
}
int main(int argc,char **argv)
{
if (argc > 1)
{
yyin = fopen(argv[1],"r");
if(!yyin)
{
printf("could not open %s \n",argv[1]);
exit(0);
}
}
yylex();
return 0;
}
```

OUTPUT:

```
(base) pllab@pllab:~$ cd Desktop
(base) pllab@pllab:~/Desktop$ ls
22501a05b3.sql  exp5.c    lab3_2.l   numberfile.txt  task6_1.java   Util.class
a.out           exp5.o    lexical.l  sum.c           task6_2.class  vennela
exp5            lab3_1.l  lex.yy.c   task6_1.class   task6_2.java
(base) pllab@pllab:~/Desktop$ flex lexical.l
(base) pllab@pllab:~/Desktop$ ls
22501a05b3.sql  exp5.c    lab3_2.l   numberfile.txt  task6_1.java   Util.class
a.out           exp5.o    lexical.l  sum.c           task6_2.class  vennela
exp5            lab3_1.l  lex.yy.c   task6_1.class   task6_2.java
(base) pllab@pllab:~/Desktop$ cc lex.yy.c
(base) pllab@pllab:~/Desktop$ ls
22501a05b3.sql  exp5.c    lab3_2.l   numberfile.txt  task6_1.java   Util.class
a.out           exp5.o    lexical.l  sum.c           task6_2.class  vennela
exp5            lab3_1.l  lex.yy.c   task6_1.class   task6_2.java
(base) pllab@pllab:~/Desktop$ ./a.out
#include<stdio.h>

#include<stdio.h> --> preprocessor
printf("HI mamaya");

printf --> reserved

( --> operator

"HI mamaya" --> string

) --> operator

; --> operator
printf("hi gowthami ");

printf --> reserved

( --> operator

"hi gowthami " --> string
```

(b) Implement the lexical analyzer Program to count no of +ve and –ve integers using LEX.

PROGRAM:

```
%{
#include<stdio.h>
int posint=0, negint=0,posfraction=0, negfraction=0;
%}
%%
[-][0-9]+ {negint++;}
[+]?[0-9]+ {posint++;}
[+]?[0-9]*\.[0-9]+ {posfraction++;}
[-][0-9]*\.[0-9]+ {negfraction++;}
%%
int yywrap()
{
return 1;
}
int main(int argc, char *argv[])
{
if(argc!=2)
{
printf("Usage:<./a.out><sourcefile>\n");
exit(0);
}
yyin=fopen(argv[1],"r");
yylex();
printf("No of +ve integers= %d\n ", posint);
printf("No of –ve integers=%d\n",negint);
printf("No of +ve fractions=%d\n ",posfraction);
printf("No of –ve fractions=%d\n", negfraction);
}
```

Input file : 2
4
-6
-7
0
0
-2
-6
-2
0
0
8
9
10
78
90
-4
23.5
12.5
-12.7

OUTPUT:

```
No of +ve integers= 11
 No of -ve integers=6
No of +ve fractions=2
 No of -ve fractions=1
```

EXPERIMENT-4

Aim:**(a)** Program to count the number of vowels and consonants in a given string.

PROGRAM:

```
%{
#include <stdio.h>
int vowels = 0;
int cons = 0;
%}
%%
[aeiouAEIOU] {vowels++;}
[a-zA-Z] {cons++;}
%%
int yywrap()
{
   return 1;
}
main()
{
   printf("Enter the string... at end press ^d\n");
   yylex();
   printf("No of vowels = %d\nNo of consonants = %d\n", vowels, cons);
}
```

OUTPUT:

```
guest-ZCq0cy@pllab58:~/Desktop$ lex 2b.l
guest-ZCq0cy@pllab58:~/Desktop$ cc lex.yy.c
guest-ZCq0cy@pllab58:~/Desktop$ ./a.out
Enter the string.. at end press ^d
PVPSIT CSE Section A

Compiler Design Lab

No of vowels=12
No of consonants=22
guest-ZCq0cy@pllab58:~/Desktop$
```

(b).Program to count the number of characters, words, spaces, end of lines in a given input file.

PROGRAM:

```
%{
#include <stdio.h>
int c=0, w=0, s=0, l=0;
%}
WORD [^ \t\n,\.:]+
EOL [\n]
BLANK [ ]
%%
{WORD} {w++; c = c + yyleng;}
{BLANK} {s++;}
{EOL} {l++;}
. {c++;}
%%
int
   yywrap(){ retur
   n 1;
}
main(int argc, char
   *argv[]){ if (argc != 2){
      printf("Usage: <./a.out> <sourcefile>\n");
      exit(0);
   }
   yyin = fopen(argv[1], "r");
   yylex();
   printf("No of characters = %d\nNo of words = %d\nNo of spaces =
%d\n No of lines = %d\n", c, w, s, l);
}
```

Input file : input.txt
Prasad V Potluri Siddhartha Institute of Technology
II B.tech CSE Section-1 Students

Compiler Design Lab
Simple Lex programs

OUTPUT:

```
guest-ZCq0cy@pllab58:~$ cd Desktop
guest-ZCq0cy@pllab58:~/Desktop$ lex 2c.l
guest-ZCq0cy@pllab58:~/Desktop$ cc lex.yy.c
guest-ZCq0cy@pllab58:~/Desktop$ ./a.out input.txt
No of characters=107
No of words=19
No of spaces=14
 No of lines=5guest-ZCq0cy@pllab58:~/Desktop$
```

EXPERIMENT-5

5)Implement a _C_ program to calculate First and Follow sets of given grammar.

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;
int main(int argc, char **argv)
{
int jm = 0;
int km = 0;
int i, choice;
char c, ch;
count = 8;
strcpy(production[0], "E=TR");
strcpy(production[1], "R=+TR");
strcpy(production[2], "R=#");
strcpy(production[3], "T=FY");
strcpy(production[4], "Y=*FY");
strcpy(production[5], "Y=#");
strcpy(production[6], "F=(E)");
strcpy(production[7], "F=i");
int kay;
char done[count];
int ptr = -1;
for(k = 0; k < count; k++)
{ for(kay = 0; kay < 100; kay++)
{ calc_first[k][kay] = '!';
```

```
}
}
int point1 = 0, point2, xxx;
for(k = 0; k < count; k++)
{
c = production[k][0];
point2 = 0;
xxx = 0;
for(kay = 0; kay <= ptr; kay++)
if(c == done[kay])
xxx = 1;
if (xxx == 1)
continue;
findfirst(c, 0, 0);
ptr += 1; done[ptr]
= c;
printf("\n First(%c) = { ", c);
calc_first[point1][point2++] = c;
for(i = 0 + jm; i < n; i++) {
int lark = 0, chk = 0;
for(lark = 0; lark < point2; lark++)
{ if (first[i] ==
calc_first[point1][lark])
{
chk = 1;
break;
}
}
if(chk == 0)
{
printf("%c, ", first[i]);
calc_first[point1][point2++] = first[i];
}
}
printf("}\n");
jm = n;
point1++;
}
printf("\n");
printf("-----------------------------------------------------\n\n");
char donee[count];
ptr = -1;
for(k = 0; k < count; k++)
{ for(kay = 0; kay < 100; kay++)
```

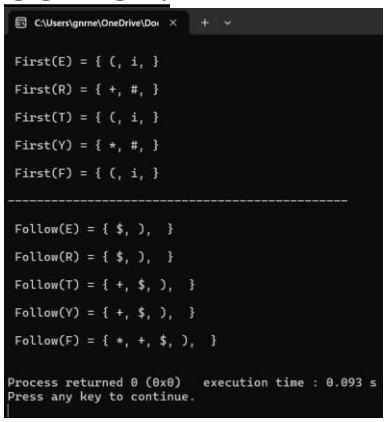
```
calc_follow[k][kay] = '!';
}
}
point1 = 0;
int land = 0;
for(e = 0; e < count; e++)
{
ck = production[e][0];
point2 = 0;
xxx = 0;
for(kay = 0; kay <= ptr; kay++)
if(ck == donee[kay])
xxx = 1;
if (xxx == 1)
continue;
land += 1;
follow(ck);
ptr += 1;
donee[ptr] = ck;
printf(" Follow(%c) = { ", ck);
calc_follow[point1][point2++] = ck;
for(i = 0 + km; i < m; i++) {
int lark = 0, chk = 0;
for(lark = 0; lark < point2; lark++)
{
if (f[i] == calc_follow[point1][lark])
{
chk = 1;
break;
}
}
if(chk == 0)
{
printf("%c, ", f[i]);
calc_follow[point1][point2++] = f[i];
}
}
printf(" }\n\n");
km = m;
point1++;
}
}
void follow(char c)
```

```
{
int i, j;
if(production[0][0] == c)
{ f[m++] = '$';
}
for(i = 0; i < 10; i++)
{
for(j = 2;j < 10; j++)
{
if(production[i][j] == c)
{
if(production[i][j+1] != '\0')
{
followfirst(production[i][j+1], i, (j+2));
}
if(production[i][j+1]=='\0' && c!=production[i][0])
{
follow(production[i][0]);

}
}
}
}
}
void findfirst(char c, int q1, int q2)
{
int j;
if(!(isupper(c)))
{ first[n++] = c;
}
for(j = 0; j < count; j++)
{
if(production[j][0] == c)
{
if(production[j][2] == '#')
{
if(production[q1][q2] == '\0')
first[n++] = '#';
else if(production[q1][q2] != '\0'
&& (q1 != 0 || q2 != 0))
{
findfirst(production[q1][q2], q1, (q2+1));
}
```

```
else
first[n++] = '#';
}
else if(!isupper(production[j][2]))
{
first[n++] = production[j][2];
}
else
{
findfirst(production[j][2], j, 3);
}
}
}
}
void followfirst(char c, int c1, int c2)
{
int k;
if(!(isupper(c)))
f[m++] = c;
else
{
int i = 0, j = 1;
for(i = 0; i < count; i++)
{

if(calc_first[i][0] == c)
break;
}
while(calc_first[i][j] != '!')
{
if(calc_first[i][j] != '#')
{
f[m++] = calc_first[i][j];
}
else
{
if(production[c1][c2] == '\0')
{
follow(production[c1][0]);
}
else
{
followfirst(production[c1][c2], c1, c2+1);
```

```
}
}
j++;
}
}
}
```

OUTPUT:


C:\Users\gnrne\OneDrive\Do
First(E) = { (, i, }
First(R) = { +, #, }
First(T) = { (, i, }
First(Y) = { *, #, }
First(F) = { (, i, }

Follow(E) = { $,), }
Follow(R) = { $,), }
Follow(T) = { +, $,), }
Follow(Y) = { +, $,), }
Follow(F) = { *, +, $,), }
Process returned 0 (0x0) execution time : 0.093 s
Press any key to continue.

EXPERIMENT-6

Aim: Design Predictive parser for the given language.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
char input[20];
int len, ln = 0, err = 0;
void E();
void E1();
void T();
void T1();
void F();
void match(char topChar);
void E()
{
   T();
   E1();
}
void E1()
{
   if (*input == '+')
   {
      match('+');
      T();
      E1();
   }
   else
      return;
}
void T()
{
   F();
   T1();
}
void T1()
{
   if (*input == '*')
   {
      match('*');
      F();
```

```
        T1();
    }
    else
        return;
}
void F()
{
    if (*input == '(')
    {
        match('(');
        E();
        match(')');
    }
    else
        match('i');
}
void match(char topChar)
{
    if (*input == topChar)
    {
        printf("\n%s popped %c", input, topChar);
        ln++;
        strcpy(input, &input[1]); // pops matched input symbol from input
    }
    else
    {
        printf("\nError: %c was not produced by any production at this place",
*input);
        err++;
    }
}

int main()
{
    printf("Enter the Input: ");
    fgets(input, sizeof(input), stdin);

    // Remove trailing newline character if present
    size_t input_length = strlen(input);
    if (input_length > 0 && input[input_length - 1] == '\n')
    {
        input[input_length - 1] = '\0';
        input_length--;
```

```
    }
len = input_length;
    input[len] = '$';
    input[len + 1] = '\0';

    E();
    if (err == 0 && ln == len)
        printf("\n\nString parsed successfully!!!\n");
    else
        printf("\n\nString is not parsed successfully. Errors occurred or input
contains invalid characters.\n\n");

    return 0;
}
```

OUTPUT: