Exp No: 1 Date: 07-12-2024
Name: Pavan Kumar Ch Regd No: 22501A05E0

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 <ctype.h>
#include <string.h>
#define MAX_KEYWORDS 32

char *keywords[MAX_KEYWORDS] = {
    "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 isKeyword(char *str){
    for (int i = 0; i < MAX_KEYWORDS; i++)
        if (strcmp(keywords[i], str) == 0)    return 1;
    return 0;
}

int isFunction(char *str){
    return (strcmp(str, "main") == 0 || strcmp(str, "printf") == 0);
}

int main(){
```

```
FILE *fp;
char filename[20], c, buf[30];
int sno = 0, lno = 1, kc = 0;
printf("Enter the file name: ");
scanf("%s", filename);
fp = fopen(filename, "r");
if (!fp)    {
   printf("Could not open file %s\n", filename);
   return 1;
}
printf("\nS.No \t Token \t\t Lexeme \t\t Line No\n");
while ((c = fgetc(fp)) != EOF) {
   if (isalpha(c)) {  // Check if the character is alphabetic
      buf[kc = 0] = c;
      while (isalnum(c = fgetc(fp)))
         buf[++kc] = c;
      buf[++kc] = '\0';
      if (isKeyword(buf))
         printf("%4d \t keyword \t %20s \t %7d\n", ++sno, buf, lno);
      else if (isFunction(buf))
         printf("%4d \t function \t %20s \t %7d\n", ++sno, buf, lno);
      else
         printf("%4d \t identifier \t %20s \t %7d\n", ++sno, buf, lno);
   }
   else if (isdigit(c)) {
      buf[kc = 0] = c;
      while (isdigit(c = fgetc(fp))){
         buf[++kc] = c;
      }
      buf[++kc] = '\0';
```

```
            printf("%4d \t number \t %20s \t %7d\n", ++sno, buf, lno);
        }
        else if (c == '(' || c == ')')
            printf("%4d \t parenthesis \t %6c \t %7d\n", ++sno, c, lno);
        else if (c == '{' || c == '}')
            printf("%4d \t brace \t %6c \t %7d\n", ++sno, c, lno);
        else if (c == '[' || c == ']')
            printf("%4d \t array \t index \t %6c \t %7d\n", ++sno, c, lno);
        else if (c == ',' || c == ';')
            printf("%4d \t punctuation \t %6c \t %7d\n", ++sno, c, lno);
        else if (c == '\"') // Handle string literals
        {
            kc = -1;
            while ((c = fgetc(fp)) != '\"') {
                buf[++kc] = c;
            }
            buf[++kc] = '\0';
            printf("%4d \t string \t %20s \t %7d\n", ++sno, buf, lno);
        }
        else if (c == ' ')
                continue;
        else if (c == '\n')
            lno++;
        else
            printf("%4d \t operator \t %6c \t %7d\n", ++sno, c, lno);
    }
    fclose(fp);
    return 0;
}
```

Exp No: 1 Date: 07-12-2024
Name: Pavan Kumar Ch Regd No: 22501A05E0

sum.c file:

```
#include<stdio.h>

int main() {

int a = 10, b = 5;

printf("Sum : %d", a + b);

return 0;

}
```

Output:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/01-lexical-analyser 〉main ?1 ./Lexical.out

Enter the file name: input.c


S.No        Token           Lexeme              Line No
   1       keyword            int                    1
   2       function           main                   1
   3      parenthesis           (                    1
   4      parenthesis           )                    1
   5        brace               {                    2
   6       keyword            int                    3
   7      identifier             a                    3
   8      punctuation            ;                    3
   9       function         printf                   4
  10      parenthesis           (                    4
  11        string           Hello                     4
  12      parenthesis           )                    4
  13      punctuation           ;                    4
  14        brace               }                    5%
```

Conclusion: Lexical analyzer for the given language has been implemented successfully.

Exp No: 2 Date: 21-12-2024
Name: Pavan Kumar Ch Regd No: 22501A05E0

Experiment – 2

Aim: (a) Implement the lexical analyzer using LEX program for the regular expression RE's: a(a+b)*.

Program:

```
%{
#include<stdio.h>
int result = 0;
%}
%%
[\n] {
result == 1?printf("Valid string\n"):printf("Invalid string\n");
}
^a[a|b]*$ {result = 1;}
. result = 0;
%%
int main() {
printf("Enter string: ");
yylex();
return 0;
}
```

Output:

Case-1:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/02-regex 〉main ?2 〉 flex 2a.lex
gcc lex.yy.c -lfl
./a.out

Enter string: aababababaaaba
Valid string
```

Case-2:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/02-regex 〉main ?2  flex 2a.lex
gcc lex.yy.c -lfl
./a.out

Enter string: pavan
Invalid string
```

Conclusion: Lexical analyzer using LEX program for the regular expression RE's: a(a+b)* has been implemented successfully.

Exp No: 2 Date: 21-12-2024
Name: Pavan Kumar Ch Regd No: 22501A05E0

Aim: (b) Implement the LEX program to implement RE's: (a+b)*abb(a+b)*

Program:

```
%{
#include<stdio.h>
int result = 0;
%}
%%
[\n] {
result == 1?printf("Valid string\n"):printf("Invalid string\n"); result = 0;
}
^[a|b]*abb[a|b]*$ {result = 1;}
. {}
%%
int main() {
printf("Enter string: ");
yylex();
return 0;
}
```

Output:

Case-1:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/02-regex 〉main ?2 〉 flex 2b.lex
gcc lex.yy.c -lfl
./a.out

Enter string: aabb
Valid string
```

Exp No: 2 Date: 21-12-2024
Name: Pavan Kumar Ch Regd No: 22501A05E0

Case-2:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/02-regex 〉main ?2  flex 2b.lex
gcc lex.yy.c -lfl
./a.out

Enter string: pavan
Invalid string
```

Conclusion: Lexical analyzer using LEX program for the regular expression RE's: (a+b)*abb(a+b)* has been implemented successfully.

Exp No: 3 Date: 08-01-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

Experiment – 3

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

Program:

```
%{
#include <stdio.h>
#include <stdlib.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); }
[ \t\n]         { /* Ignore whitespace */ }
.               { display(5); }

%%

void display(int n) {
   printf("%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;
}
```

Exp No: 3 Date: 08-01-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

sum.c file:

```
#include<stdio.h>

int main() {

int a = 10, b = 5;

printf("Sum : %d", a + b);

return 0;

}
```

Output:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/03-jflex 〉main ?3 〉 ./lexer sum.c

 --> preprocessor
int --> keyword
main --> reserved
( --> invalid literal
) --> invalid literal
{ --> invalid literal
 --> invalid literal
int --> keyword
a --> identifier
= --> invalid literal
10 --> number
, --> invalid literal
b --> identifier
= --> invalid literal
5 --> number
; --> invalid literal
 --> invalid literal
printf --> reserved
( --> invalid literal
" --> invalid literal
Sum --> identifier
: --> invalid literal
% --> invalid literal
d --> identifier
" --> invalid literal
, --> invalid literal
a --> identifier
+ --> invalid literal
b --> identifier
) --> invalid literal
; --> invalid literal
 --> invalid literal
return --> identifier
0 --> number
; --> invalid literal
 --> invalid literal
} --> invalid literal
```

Conclusion: Lexical analyzer using JLEX, FLEX or LEX or other lexical analyzer generating stools has been implemented successfully.

Exp No: 3 Date: 08-01-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

Aim: (b) Implement lexical analyzer program to count no of +ve and –ve integers using LEX.

Program:

```
%{
#include<stdio.h>
int posint=0;
int negint=0;
int posfraction=0;
int negfraction=0;
%}
%%
[-][0-9]+ {negint++;}
[+]?[0-9]+ {posint++;}
[+]?[0-9]*\.[0-9]+ {posfraction++;}
[-][0-9]*\.[0-9]+ {negfraction++;}
[\n\t&#39; &#39;] {}
%%
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\nNo of –ve integers: %d\nNo of +ve fractions: %d\nNo of –ve fractions: %d", posint, negint,posfraction, negfraction);
return 0;
}
```

number.txt file:

-10 1 5.31 28 9 24 12 12.24 0.124 -542.01 -8 -43.0 -1.02865343258 -4628

Output:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/03-jflex 〉main ?3 〉 flex 3b.l

/mnt/c/U/pavan/Doc/G/CompilerDesign/03-jflex 〉main ?3 〉 cc lex.yy.c

/mnt/c/U/pavan/Doc/G/CompilerDesign/03-jflex 〉main ?3 〉 ./a.out number.txt
No of +ve integers: 5
No of -ve integers: 3
No of +ve fractions: 3
No of -ve fractions: 3
```

Conclusion: Lexical analyzer to count no of +ve and –ve integers using LEX has been implemented successfully.

Exp No: 4 Date: 29-01-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

Experiment – 4

Aim: (a) Implement lexical analyzer program to count number of vowels and consonants in given string.

Program:

```
%{
#include <stdio.h>
int vowel_count = 0, consonant_count = 0;
%}
%%
[aAeEiIoOuU][\n] {vowel_count++;}
[a-zA-Z] {consonant_count++;}
. ;
%%
int yywrap() {return 1;}
int main() {
yylex();
printf("Number of vowels: %d\n Number of consonants: %d\n ", vowel_count, consonant_count);
return 0;
}
```

Output:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/04-count 〉 main ?4 〉 flex 4a.l

/mnt/c/U/pavan/Doc/G/CompilerDesign/04-count 〉 main ?4 〉 cc lex.yy.c

/mnt/c/U/pavan/Doc/G/CompilerDesign/04-count 〉 main ?4 〉 ./a.out
Enter a string: HelloWorld
Number of vowels: 3
Number of consonants: 7
```

Conclusion: Lexical analyzer program to count number of vowels and consonants in given string. has been implemented successfully.

Exp No: 4 Date: 29-01-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

Aim: (b) Implement the lexical analyzer 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(){ return 1; }
int main(int argc, char *argv[]) {
if(argc!=2) {
printf("Usage: <./a.out> <sourcefile>\n");

}
yyin=fopen(argv[1],"r");
yylex();
printf("No of characters=%d\nNo of words=%d\nNo of spaces=%d\nNo of lines=%d",c,w,s,l);
return 0;
}
```

input.txt file:

Exp No: 4 Date: 29-01-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

This is a text file used for the experiment 4.

this is used for 4b question.

Output:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/04-count 〉main ?4 〉 flex 4b.l

/mnt/c/U/pavan/Doc/G/CompilerDesign/04-count 〉main ?4 〉 cc lex.yy.c

/mnt/c/U/pavan/Doc/G/CompilerDesign/04-count 〉main ?4 〉 ./a.out input.txt
No of characters = 63
No of words = 17
No of spaces = 14
No of lines = 1
```

Conclusion: Lexical analyzer program to count the number of characters, words, spaces, end of lines in a given input file has been implemented successfully.

Exp No: 5 Date: 05-02-2025
Name: Pavan Kumar Ch Regd No: 22501A05E0

Experiment – 5

Aim: Implement a C program to calculate First and Follow sets of given grammar.

Program:

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

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;


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++;
        }
    }
}
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]);
                }
            }
        }
```

```
    }
}


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] = '!';
            calc_follow[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;
                break;
            }
        }
        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----------------------------------------------\n\n");

    char donee[count];
    ptr = -1;
    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++;
    }
}
```

Output:

```
/mnt/c/U/pavan/Doc/G/CompilerDesign/05-first-follow 〉 main ?4  ./firstfollow

First(E) = { (, i, }

First(R) = { (, i, +, #, }

First(T) = { (, i, +, #, (, i, }

First(Y) = { (, i, +, #, (, i, *, #, }

First(F) = { (, i, +, #, (, i, *, #, (, i, }

-----------------------------------------

Follow(E) = { $, ),  }

Follow(R) = { $, ), $, ),  }

Follow(T) = { $, ), $, ),  }

Follow(Y) = { $, ), $, ),  }

Follow(F) = { $, ), $, ),  }
```

Conclusion: C program to calculate First and Follow sets of given grammar has been implemented successfully.