GHARDA FOUNDATION'S
GHARDA INSTITUTE OF TECHNOLOGY, LAVEL
COMPUTER ENGINEERING DEPARTMENT
A/P: Lavel, Tal.Khed Dist. Ratnagiri

Evaluation Sheet

Class: TE-Computer

Semester: VI

Subject: **System Programming & Compiler Construction**

Experiment No: 7

Title of Experiment: Study and implement experiments on LEX.

Sr. No.	Evaluation Criteria	Max Marks	Marks Obtained
1	Practical Performance	10	
2	Oral	5	
	Total	15	

Signature of Subject Teacher

Theory:

Flex – Lexical Analysis Tool

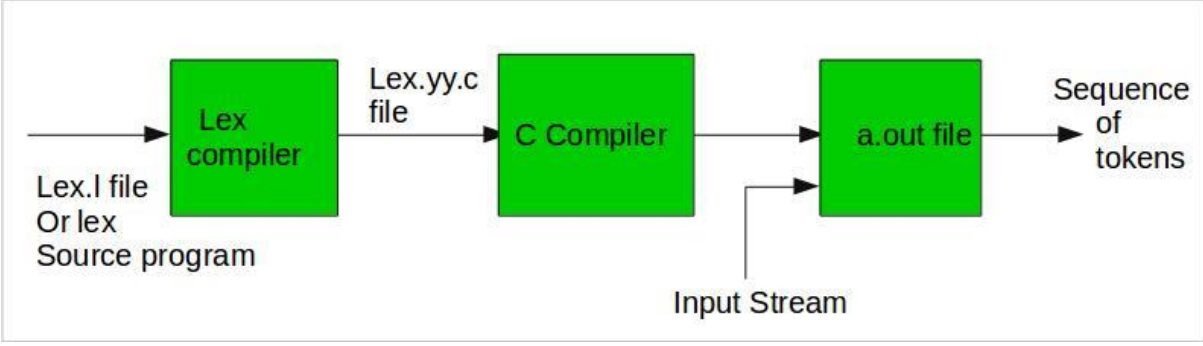
FLEX (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produces faster code.

Bison produces parser from the input file provided by the user. The function yylex() is automatically generated by the flex when it is provided with a .l file and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.

Installing Flex on Ubuntu:

```
sudo apt-get update
sudo apt-get install flex
```

How Flex is used?



Step 1: An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.

Step 2: The C compiler compile lex.yy.c file into an executable file called a.out.

Step 3: The output file a.out take a stream of input characters and produce a stream of tokens.

Program Structure:

In the input file, there are 3 sections:

1. **Definition Section**: The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in "%{ %}" brackets. Anything written in this brackets is copied directly to the file lex.yy.c.

   Syntax:

```
%{
   // Definitions
%}
```

2. **Rules Section**: The rules section contains a series of rules in the form: pattern action and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in "%% %%".

   Syntax:

```
%%
pattern  action
%%
```

3. **User Code Section**: This section contains C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

   Basic Program Structure:

```
%{
// Definitions
%}

%%
Rules
%%

User code section
```

How to run the program:

To run the program, it should be first saved with the extension .l or .lex. Run the below commands on terminal in order to run the program file.

Step 1: flex filename.l or flex filename.lex depending on the extension file is saved with

Step 2: gcc lex.yy.c

Step 3: ./a.out

Step 4: Provide the input to program in case it is required

myfile.txt –

```
void main ()
{
      int a,c;
      float b;
      if a=10;
      b=10.55;
      return (0);
}
```

a.txt –

```
GIT
GIT
GIT
```

count.l

```
%{

#include<stdio.h>

int lines=0, words=0,s_letters=0,c_letters=0, num=0, spl_char=0,total=0;
//Global variable declaration

%}


%%

\n { lines++; words++;}      //\n represent new line plus end of one word

[\t ' ']+ words++;        // 0 or more \t or space represent recognition
of one word

[A-Z] c_letters++;     //any upper case letter recognize upper case letter

[a-z] s_letters++;   //any lower case letter recognize lower case letter

[0-9] num++;       //any digit from 0 to 9 recognize digit

. spl_char++;       //any other character (do not match with above RE)
recognize special character

%%

main(void)

      {

      yyin= fopen("myfile.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);

      }
```

Output –

```
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ flex count.l
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ gcc lex.yy.c -ll
count.l:26:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   26 | main(void)
      | ^~~~
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ ./a.out
 This File contains ...
        8 lines
        21 words
        29 small letters
        0 capital letters
        7 digits
        15 special characters
        In total 51 characters.
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$
```

count1.l -

```
/*This program is going to count total number of lines and total number
of Characters present in input file*/

%{

int num_lines = 0, num_chars = 0;

%}

%%

\n      ++num_lines;

.       ++num_chars;

%%

main(void)

      {

      yyin= fopen("a.txt","r");

      yylex(); //printf(" This File contains ...");

      printf("\n\t%d lines", num_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 Total characters\n",num_chars);

      //printf("\n\tIn total %d characters.\n",total);

      }
```

Output –

```
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ flex count1.l
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ gcc lex.yy.c -ll
count1.l:16:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   16 | main(void)
      | ^~~~
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ ./a.out

        3 lines
        9 Total characters
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$
```

count2.l –

```
/*This program is going to count total number of lines and total number
of Characters present in input file*/

%{

#include <stdlib.h>

%}

KEYWORD  void|int|float|char|return|for|if|else|while

DIGIT    [0-9]

ID       [a-zA-z][a-zA-z0-9]*

INT      {DIGIT}+

REAL     {DIGIT}*"."{DIGIT}+

%%

{KEYWORD}   {
            printf("The keyword :%s\n",yytext);
          }

{INT}       {
            printf( "An integer: %s (%d)\n", yytext,
                    atoi( yytext ) );
            }

{REAL}      {
            printf( "A float: %s (%g)\n", yytext,
                    atof( yytext ) );
```

```
            }

if          {printf( "The keyword: %s\n", yytext );
            }


{ID}        printf( "An identifier: %s\n", yytext );



[ \t\n]+          /* eat up whitespace */

.           printf( "Unrecognized character: %s\n", yytext );

%%

main(void)
      {
      yyin= fopen("myfile.txt","r");
      yylex();
      //printf(" This File contains ...");
      //printf("\n\t%d lines", num_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 Total characters\n",num_chars);
      //printf("\n\tIn total %d characters.\n",total);
      }
```

Output –

```
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ flex count2.l
count2.l:32: warning, rule cannot be matched
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ gcc lex.yy.c -ll
count2.l:46:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   46 | main(void)
      | ^~~~
^[[Aniraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ ./a.out
The keyword :void
An identifier: main
Unrecognized character: (
Unrecognized character: )
Unrecognized character: {
The keyword :int
An identifier: a
Unrecognized character: ,
An identifier: c
Unrecognized character: ;
The keyword :float
An identifier: b
Unrecognized character: ;
The keyword :if
An identifier: a
Unrecognized character: =
An integer: 10 (10)
Unrecognized character: ;
An identifier: b
Unrecognized character: =
A float: 10.55 (10.55)
Unrecognized character: ;
The keyword :return
Unrecognized character: (
An integer: 0 (0)
Unrecognized character: )
Unrecognized character: ;
Unrecognized character: }
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$
```

sample.l

```
%{
int dcount=0,ccount=0;
%}
%%
0|1|2|3|4|5|6|7|8|9     {dcount++; printf("\n\n%s is a Digit",yytext); }
[a-zA-Z]              {ccount++; printf("\n\n%s is a character",yytext);}
.    {}
%%
```

```
main()
	{
	FILE *fp;
	fp=fopen("a.txt","r");
	yyin=fp;
	/*printf("\n %d no if digis are there",dcount);
	printf("\n %d no of Character are there",ccount);*/
	yylex();
	printf("\n %d no if digis are there",dcount);
	printf("\n %d no of Character are there",ccount);
	}
```

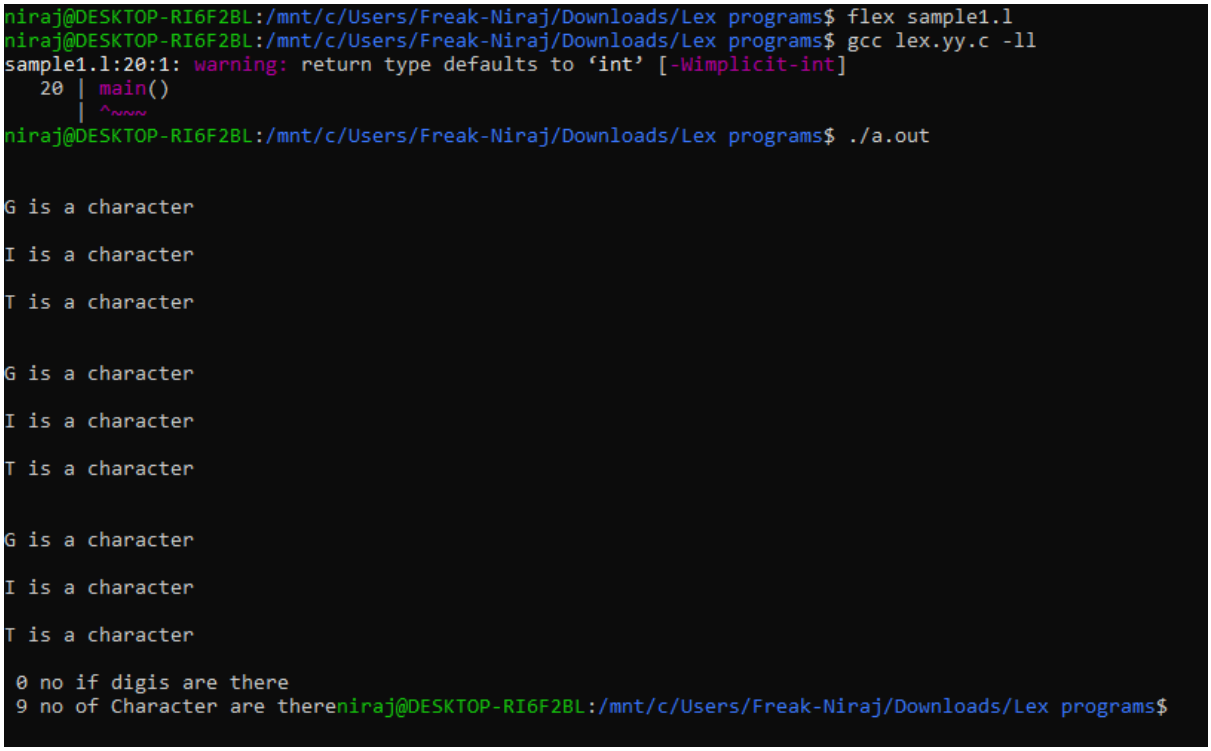
Output –

```
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ flex sample.l
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ gcc lex.yy.c -ll
sample.l:14:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   14 | main()
      | ^~~~
niraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$ ./a.out


G is a character

I is a character

T is a character


G is a character

I is a character

T is a character


G is a character

I is a character

T is a character

 0 no if digis are there
 9 no of Character are thereniraj@DESKTOP-RI6F2BL:/mnt/c/Users/Freak-Niraj/Downloads/Lex programs$
```

Sample1.l

```
%{
int dcount=0,ccount=0;
%}

digit    0|1|2|3|4|5|6|7|8|9

letter   [a-zA-Z]

%%
{digit}     {dcount++; printf("\n\n%s is a Digit",yytext); }
{letter}    {ccount++; printf("\n\n%s is a character",yytext);}
.    {}
%%

main()
      {
      FILE *fp;
      fp=fopen("a.txt","r");
      yyin=fp;
      /*printf("\n %d no if digis are there",dcount);
      printf("\n %d no of Character are there",ccount);*/
      yylex();
      printf("\n %d no if digis are there",dcount);
      printf("\n %d no of Character are there",ccount);
      }
```

Output –

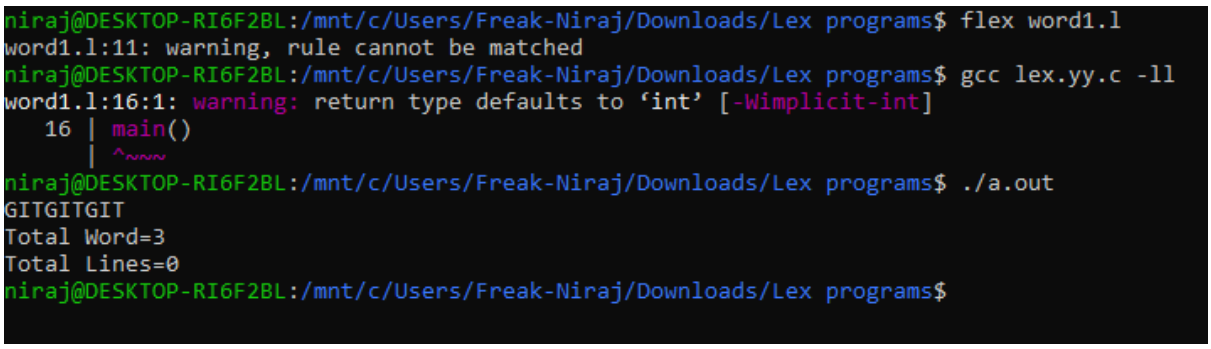


word.l

```
%{
int wcount=0,lcount=0;
%}
word  [^\n\t]+
eol   \n
%%
{word}   {wcount++;}
{eol}    {lcount++;}
%%
main()
      {
      FILE *fp;
      fp=fopen("a.txt","r");
      yyin=fp;
```

```
        /*printf("\n %d no if digis are there",dcount);
        printf("\n %d no of Character are there",ccount);*/
        yylex();
        /*printf("\n %d no if digis are there",dcount);
        printf("\n %d no of Character are there",ccount);*/
        printf("\nTotal Word=%d",wcount);
        printf("\nTotal Lines=%d",lcount);
        printf("\n");
        }
```

Output –

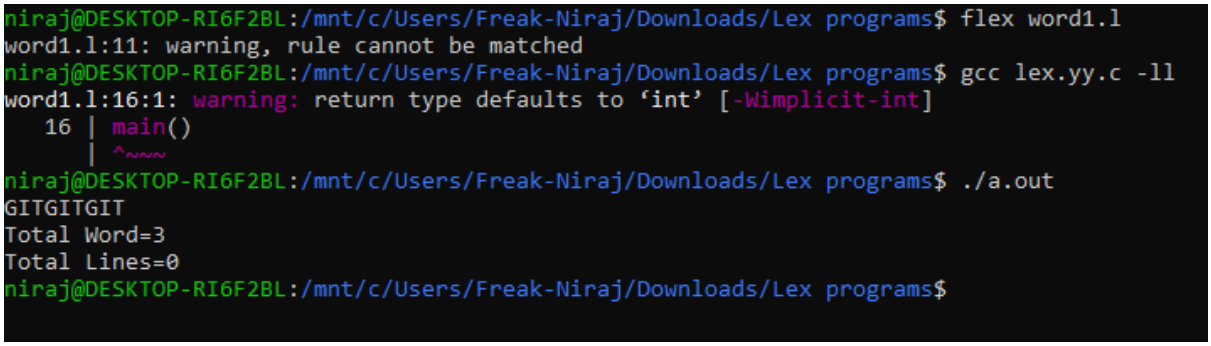


word1.l –

```
%{
int wcount=0,lcount=0;
%}

word  [''\t\n]+
eol   \n

%%
{word}   {wcount++;}
{eol}    {lcount++;}
%%
```

```
main()
	{
	FILE *fp;
	fp=fopen("a.txt","r");
	yyin=fp;
	/*printf("\n %d no if digis are there",dcount);
	printf("\n %d no of Character are there",ccount);*/
	yylex();
	/*printf("\n %d no if digis are there",dcount);
	printf("\n %d no of Character are there",ccount);*/
	printf("\nTotal Word=%d",wcount);
	printf("\nTotal Lines=%d",lcount);
	printf("\n");
	}
```

Output –



Conclusion –

Thus we studied the implementation of Lexical Analysis using Flex Tool.