Laboratory 4

Title of the Laboratory Exercise: Program to count the number of characters, words, spaces, end of lines in a given input file.

1. Introduction and Purpose of Experiment

2. Aim and Objectives

   Aim

   - To write a program to

At the end of this lab, the student will be able to

- Define
-

3. Experimental Procedure

   Students are required to carry out the following steps:

   - Algorithm
   - Write the Lex program
   - Compile and execute the program (steps)
   - Complete the documentation for the given problem

4. Presentation of Results

lab04.l

```
%{

#include <iostream>

extern int charcnt, wordcnt, spacecnt, eolcnt ;

%}

%option noyywrap

white        [ \n\t]+
digit       [0-9]
integer     [digit]+
exponent    [eE][+-]?{integer}
letter      [a-ZA-Z]
eol         [\n]

%x          WORD

%%

[\n]                    { eolcnt++; }
[ \t]                   { spacecnt++; }
[a-zA-Z]                { BEGIN(WORD); charcnt++; }
<WORD>[a-zA-Z]          { charcnt++; }
<WORD>[\n]              { BEGIN(INITIAL); wordcnt++; eolcnt++; charcnt++; }
<WORD>[ \t]             { BEGIN(INITIAL); wordcnt++; spacecnt++; charcnt++; }
<WORD>[^a-zA-Z]         { BEGIN(INITIAL); wordcnt++; charcnt++;}
.                       { charcnt++; }
```

main.cpp

```
#include <stdio.h>

#include <iostream>

extern int yylex();
extern FILE* yyin;

int charcnt = 0, wordcnt = 0, spacecnt = 0, eolcnt = 0;

auto main(int argc, char* argv[]) -> int {
    FILE* file;

    if (argc > 1) {
        file = fopen(argv[1], "r");
        yyin = file;
    }

    if (yylex() == 0)
        std::cout << "parsed successfully" << '\n';
    else
        std::cerr << "error parsing" << '\n';

    std::cout << "character count = " << charcnt << '\n'
              << "word count = " << wordcnt << '\n'
              << "space count = " << spacecnt << '\n'
              << "EOL count = " << eolcnt << '\n';

    return 0;
}
```

Makefile

```
# name of the files and the program
NAME = lab04

# compiler setup
CC = g++
LEX = flex

all: ${NAME}

${NAME}: main.cpp lex.yy.c
    ${CC} main.cpp lex.yy.c -o ${NAME}

lex.yy.c: ${NAME}.l
    ${LEX} ${NAME}.l

clean:
    rm -f lex.yy.c  ${NAME}

run:
    @./${NAME}
```

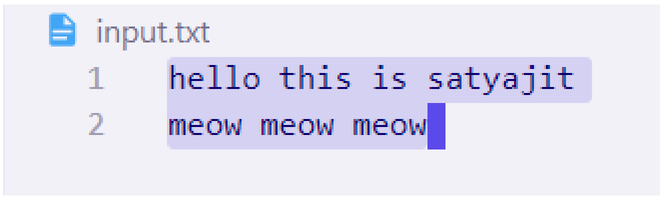
Algorithm:

```
count_context()

1.  for each lexeme
2.      if regex_match [\n]
3.          eolcnt++
4.      if regex_match [ \t]
5.          spacecnt++
6.      if regex_match [a-zA-Z]
7.          state = WORD
8.          charcnt++
9.      if state == WORD and regex_match [a-zA-Z]
10.         charcnt++
11.     if state == WORD and regex_match [\n]
12.         state = INITIAL
13.         wordcnt++; eolcnt++; charcnt++;
14.     if state == WORD and regex_match [ \t]
15.         state = INITIAL
16.         wordcnt++; spacecnt++; charcnt++;
```

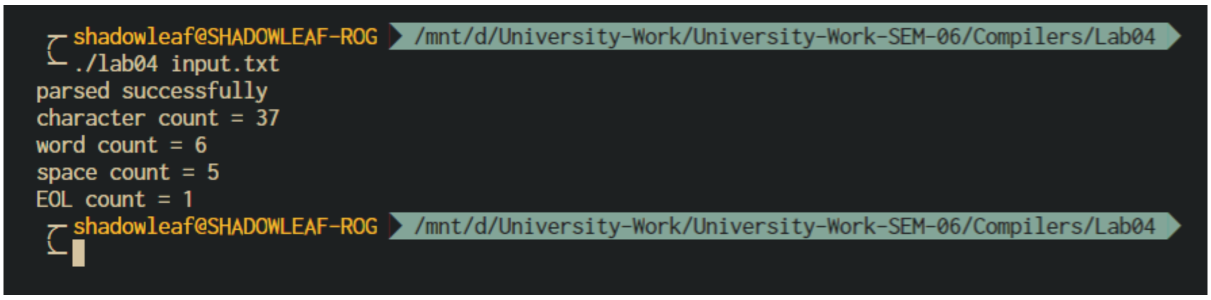
```
17.     if state == WORD and  regex_match [^a-zA-Z]
18.         state = INITIAL
19.         wordcnt++; charcnt++;
20.     else
21.         charcnt++
22. display wordcnt, spacecnt, eolcnt, charcnt
```

5. Analysis and Discussions



```
input.txt
1    hello this is satyajit
2    meow meow meow
```

Figure 0-1 INPUT file



```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-06/Compilers/Lab04
./lab04 input.txt
parsed successfully
character count = 37
word count = 6
space count = 5
EOL count = 1
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-06/Compilers/Lab04
```

Figure 0-2 OUTPUT

BEGIN

The action:

`BEGIN newstate;`

switches the state (start condition) to newstate. If the string newstate has not been declared previously as a start condition in the Definitions section, the results are unspecified. The initial state is indicated by the digit '0' or the token INITIAL.

ECHO

Write the contents of the string yytext on the output.

6. Conclusions

Flex provides a mechanism for conditionally activating rules. Any rule whose pattern is prefixed with '<sc>' will only be active when the scanner is in the start condition named sc. For example,

```
<STRING>[^"]*        { /* eat up the string body ... */
            ...
            }
```

will be active only when the scanner is in the STRING start condition, and

```
<INITIAL,STRING,QUOTE>\.        { /* handle an escape ... */
            ...
            }
```

will be active only when the current start condition is either INITIAL, STRING, or QUOTE.

Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns with no state specified shall be also active; in a %x state, such patterns shall not be active.

The rest of the line, after the first word, shall be considered to be one or more <blank>-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in Regular Expressions in lex.

7. Comments

a. Limitations of Experiments

Lex cannot be used to recognize nested structures such as parentheses. Nested structures are handled by incorporating a stack. Whenever we encounter a "(" we push it on the stack. When a ")" is encountered we match it with the top of the stack and pop the stack. However, lex only has states and transitions between states. Since it has no stack it is not well suited for parsing nested structures.

b. Limitations of Results

The lexical analyzer is not very efficient and its maintenance can be complicated. The number of states, their start and end conditions will increase as we add more context information to the transitions.

c. Learning happened

We learnt how to use Lex states, and use this to count the number of words, lines, spaces and characters.

d. Recommendations

Try not to complicate the states used in lex, and define the problem statement more clearly to make it easy to write the lex rules.

Component	Max Marks	Marks Obtained
Viva	6	
Results	7	
Documentation	7	
Total	20	