Name: Prathamesh Prabhakar Thakare
Class: TY CS D
Batch: 2
Roll number: 58
PRN: 12220016

Assignment number 2: Implement LEX code to count the number of characters, words and lines in the input.

Theory:

Lexical Analysis and Lex:
Lex is a powerful tool used to create lexical analyzers or scanners, which are essential for processing and tokenizing text or source code.

Structure of Lex Code
Lex code is typically organized into three main sections:

Definitions Section (%{ ... %}):

In this section, you can include C code that will be directly incorporated into the generated lexer code. For example, you can define variables and functions here that you want to use throughout your lexer.

Rules Section (%% ... %%):

This section is where you define patterns and their corresponding actions.
These patterns are used to identify and process tokens from the input text.
Actions specify what should be done when a particular pattern is matched.
These patterns and actions together constitute the core logic of the lexer.

User Code Section:

Typically, the int main() function serves as the entry point for the generated lexer code.
In this section, you invoke yylex() to initiate the scanning process on the input text.
The user code section can include any additional code or functions required to interact with the lexer or handle the results.

Definitions Section Example:
- Within the definitions section, you can define variables and functions that assist in the lexer's operation.
- For instance, you might declare variables to keep track of counts or other state information.

Rules Section Example:
- In the rules section, you define regular expressions or patterns that specify the tokens you want to identify.
- For each pattern, you provide an associated action or code snippet that is executed when the pattern is matched.
- Patterns and actions collectively define how the lexer tokenizes and processes the input.

User Code Section Example:
- The main() function in the user code section is where the lexer's execution begins.
- It often involves setting up the input source, invoking yylex() to start scanning, and possibly handling the lexer's output or results.

Maths:

Initialization:

sc (space count), wc (word count), lc (line count), and cc (character count) are all initialized to zero.
These variables will be used to keep track of various counts in the text.

Regular Expressions:

The code uses regular expressions within the %% section to match different elements in the input text:
[\n]: Matches newline characters (counting lines).
[\t]: Matches spaces and tabs (counting spaces).
[^\t\n]+: Matches sequences of characters that are not spaces, tabs, or newlines (counting words).

Counting:

When a newline character is encountered, lc (line count) is incremented.
When a space or tab is encountered, sc (space count) is incremented.
When a sequence of characters that is not a space, tab, or newline is encountered, wc (word count) is incremented.
In all cases, cc (character count) is incremented by yyleng, which represents the length of the matched text.

Algorithm:

```
1. Initialize variables sc (space count), wc (word count), lc (line count), and cc (character count) to zero.

2. Print "Enter the input:" to prompt the user for input.

3. Read input text.

4. For each character in the input text:
   - If the character is a newline:
     - Increment lc (line count) by 1.
     - Increment cc (character count) by the length of the matched text.
   - If the character is a space or tab:
     - Increment sc (space count) by 1.
     - Increment cc (character count) by the length of the matched text.
   - If the character is not a space, tab, or newline:
     - Increment wc (word count) by 1.
     - Increment cc (character count) by the length of the matched text.

5. Print the following statistics:
   - "The number of lines = lc"
   - "The number of spaces = sc"
   - "The number of words = wc"
   - "The number of characters = cc"

6. End the program.
```

Code

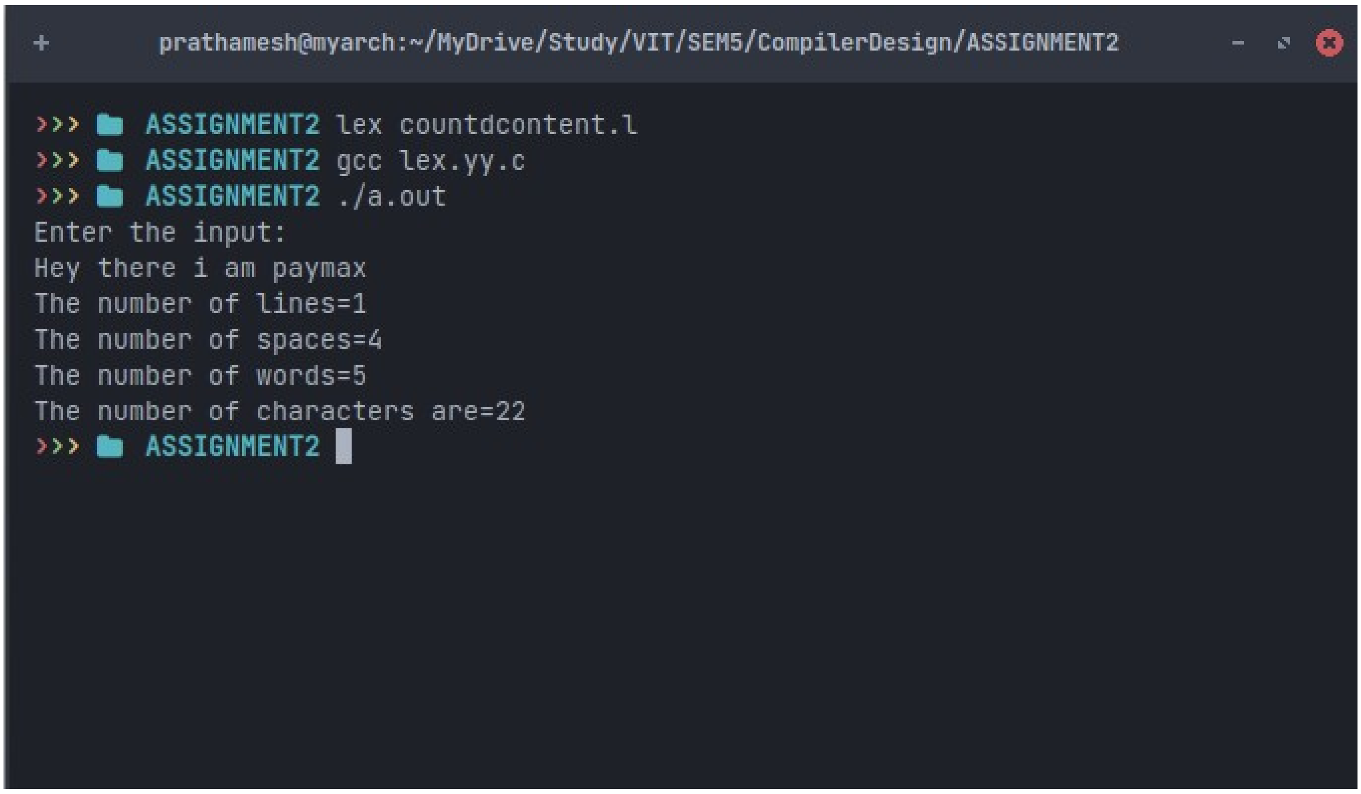
```
%{
#include<stdio.h>
int sc=0,wc=0,lc=0,cc=0;
%}

%%
[\n] { lc++; cc+=yyleng;}
[  \t] { sc++; cc+=yyleng;}
[^\t\n ]+ { wc++;  cc+=yyleng;}
%%

int main(int argc ,char* argv[ ])
{
        printf("Enter the input:\n");
        yylex();
        printf("The number of lines=%d\n",lc);
        printf("The number of spaces=%d\n",sc);
        printf("The number of words=%d\n",wc);
        printf("The number of characters are=%d\n",cc);
}

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

Output



```
prathamesh@myarch:~/MyDrive/Study/VIT/SEM5/CompilerDesign/ASSIGNMENT2

>>> ASSIGNMENT2 lex countdcontent.l
>>> ASSIGNMENT2 gcc lex.yy.c
>>> ASSIGNMENT2 ./a.out
Enter the input:
Hey there i am paymax
The number of lines=1
The number of spaces=4
The number of words=5
The number of characters are=22
>>> ASSIGNMENT2
```

Conclusion

In this assignment, we've developed a straightforward yet insightful lex program for analyzing text content. The program efficiently counts lines, spaces, words, and characters within the input text, offering valuable statistical information about the text's structure.