**Compiler design**

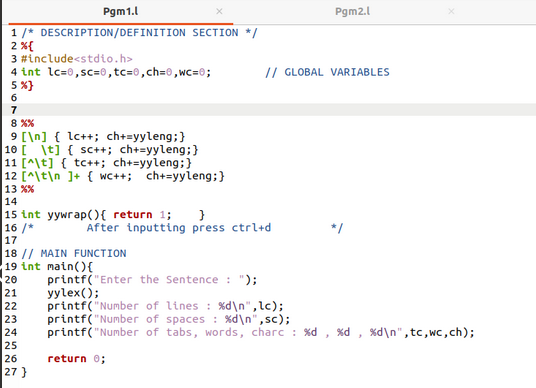
**Keerthi Rohan**

**CH.EN.U4CSE22030**

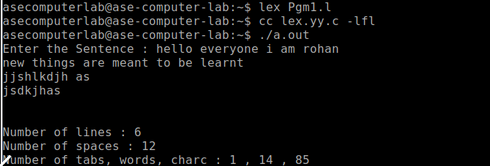
**Exercise 1**

1. Aim : Write a Lex program to count number of lines, spaces and etc.

Code :

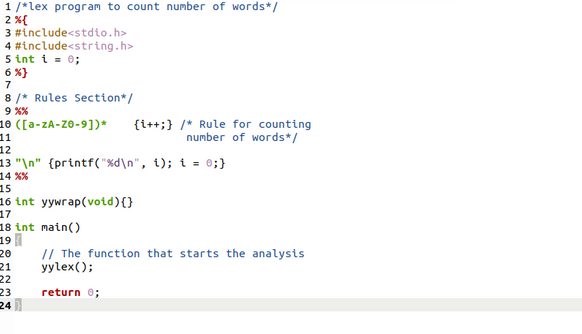


Output :

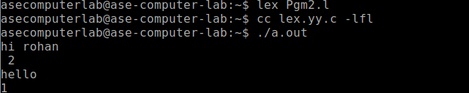


1. Aim : Write a Lex program to count number of words in given sentence.

Code :

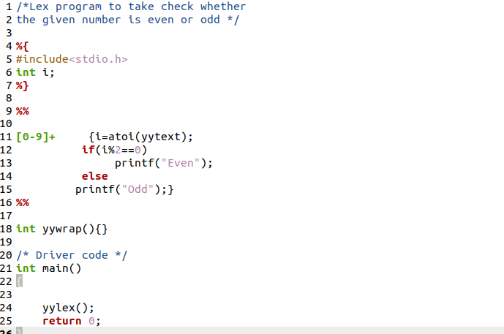


Output :

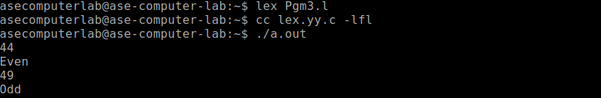


1. Aim : Write a Lex program to check whether the given number is even or odd

Code :



Output :

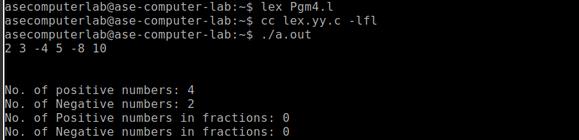


1. Aim : Write a Lex program to count the positive numbers, negative numbers and fractions.

Code :

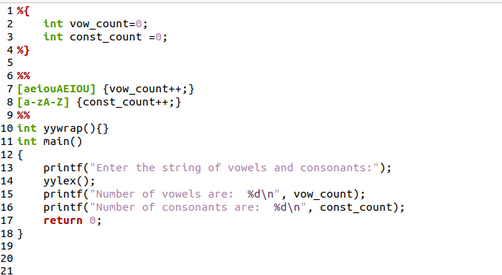


Output :

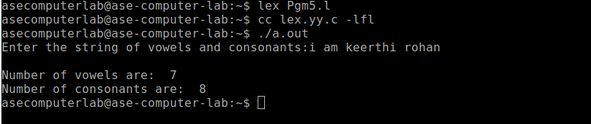


1. Aim : Write a Lex program to count the vowels and consonants in the given string

Code :



Output :



**Exercise 2**

1. Aim: To implement eliminate left recursion and left factoring from the given grammar using C program.

Algorithm:

Left Factoring:

* Start the processes by getting the grammar and assigning it to the appropriate

variables

* Find the common terminal and non-terminal elements and assign them in a

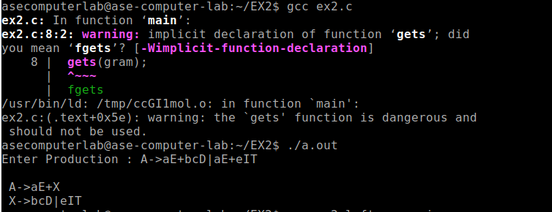
separate grammar

* Display the new and modified grammar.

Code:



Output:



Left recursion:

1. Aim: To implement left recursion using C.

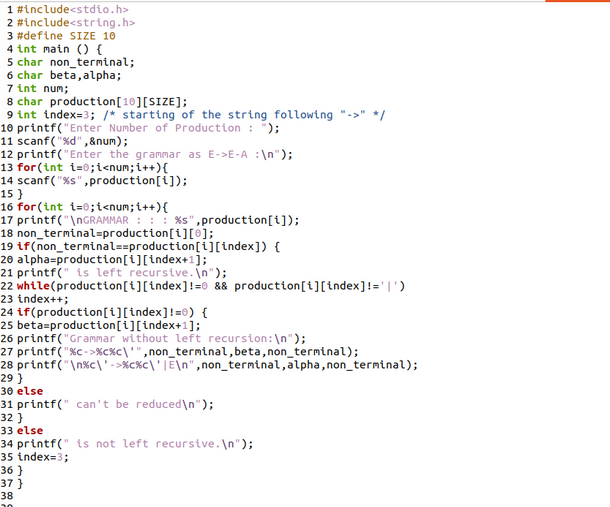
Algorithm:

* Start the processes by getting the grammar and assigning it to the appropriate

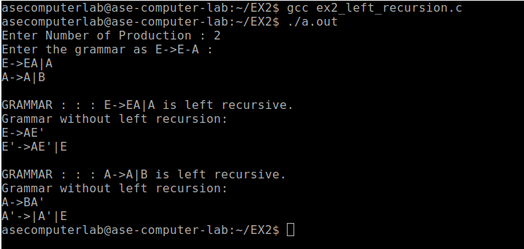
variables.

* Check if the given grammar has left recursion.
* Identify the alpha and beta elements in the production.
* Print the output according to the formula to remove left recursion

Code:



Output:



Result: The program to implement left factoring and left recursion has been successfully executed.

**Exercise 3**

Aim: To implement LL(1) parsing using C program.

Algorithm:

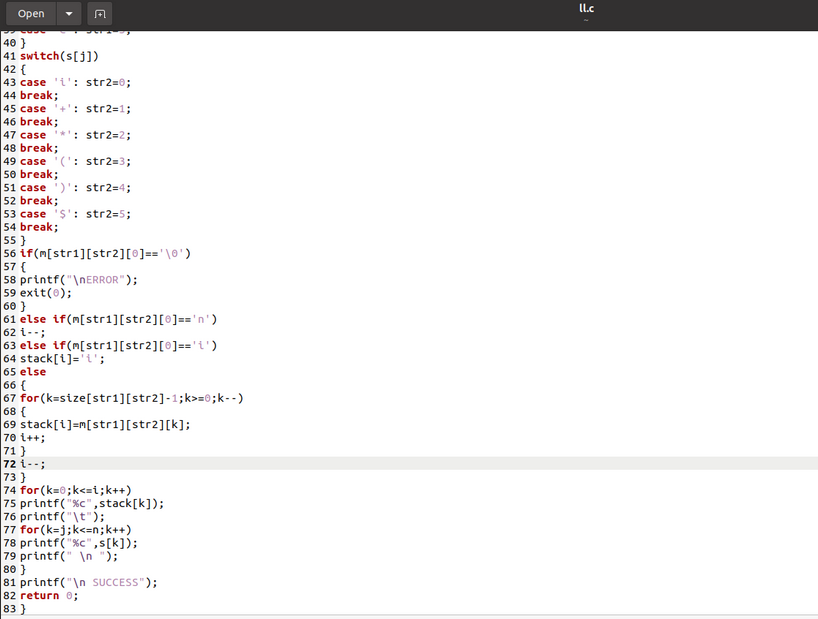
1) Read the input string.

2) Using predictive parsing table parse the given input using stack.

3) If stack [i] matches with token input string pop the token else shift it repeat the

process until it reaches to $.

Code :



Output :

