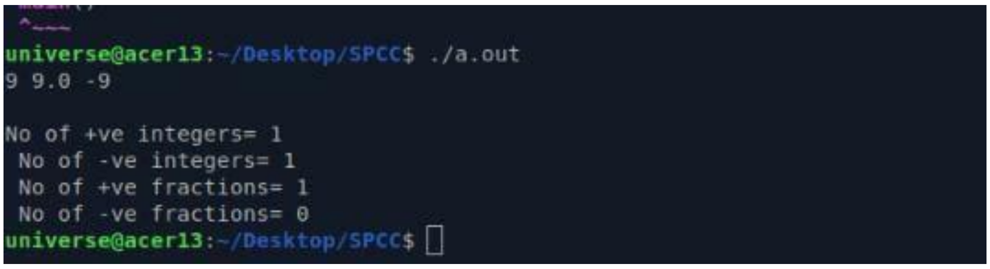
Name: Upmanyu Jha **Roll No: 8875**

Div: A **Batch: B**

Topic: Lex programs

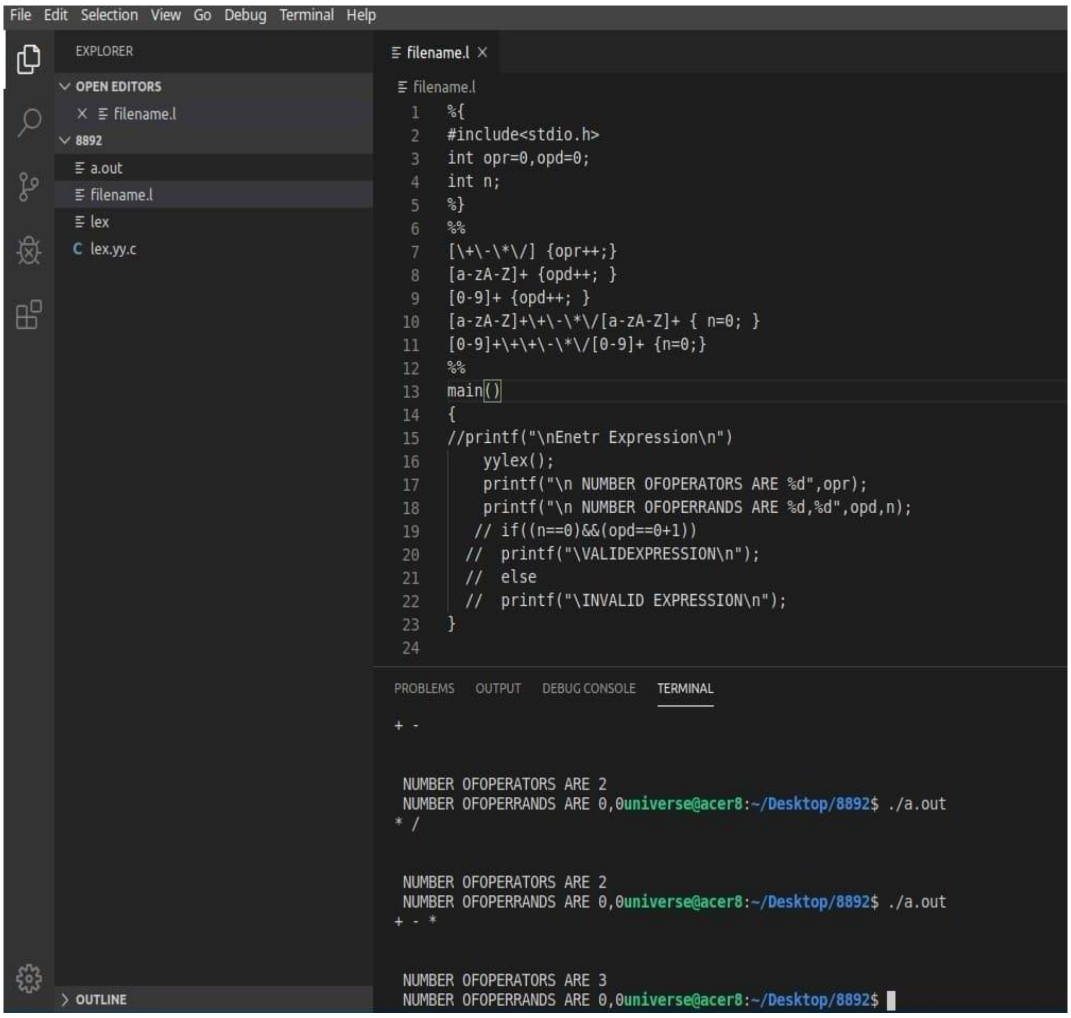
1. Number of positive and negative integers and fractions

```
%{
#include<stdio.h>
int posint=0,negint=0,postfraction=0,negfraction=0;

%}
%%
[-][0-9]+ {negint++;}
[+]?[0-9]+ {posint++;}
[+]?[0-9]*\.[0-9]+ {postfraction++;}
[-]?[0-9]*\.[0-9]+ {negfraction++;}
%%

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



yylex();
printf("No of +ve integers= %d \n No of -ve integers= %d \n No of +ve fractions= %d \n No of -ve 
fractions= %d \n",posint,negint,postfraction,negfraction);



}
```



```
universe@acer13:~/Desktop/SPCC$ ./a.out
9 9.0 -9

No of +ve integers= 1
 No of -ve integers= 1
 No of +ve fractions= 1
 No of -ve fractions= 0
universe@acer13:~/Desktop/SPCC$ 
```

2. Number of operators and operands

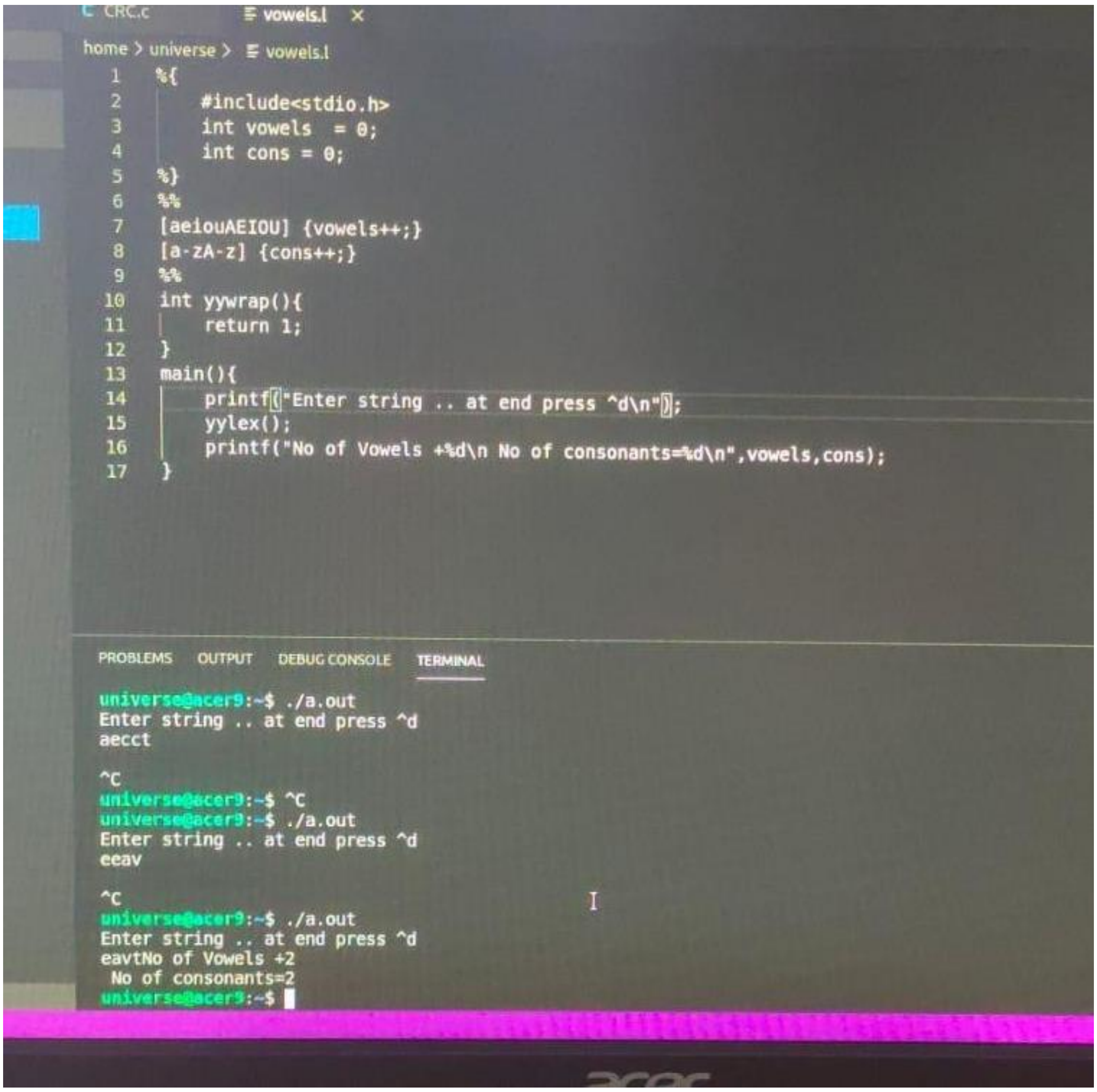


```
%{
#include<stdio.h>
int opr=0,opd=0;
int n;
%}
%%
[\+\-\*\/] {opr++;}
[a-zA-Z]+ {opd++; }
[0-9]+ {opd++; }
[a-zA-Z]+\+\-\*\/[a-zA-Z]+ { n=0; }
[0-9]+\+\+\-\*\/[0-9]+ {n=0;}
%%
main()
{
//printf("\nEnetr Expression\n")
    yylex();
    printf("\n NUMBER OFOPERATORS ARE %d",opr);
    printf("\n NUMBER OFOPERRANDS ARE %d,%d",opd,n);
   // if((n==0)&&(opd==0+1))
  //  printf("\VALIDEXPRESSION\n");
  //  else
  //  printf("\INVALID EXPRESSION\n");
}
```

```
+ -


 NUMBER OFOPERATORS ARE 2
 NUMBER OFOPERRANDS ARE 0,0universe@acer8:~/Desktop/8892$ ./a.out
* /


 NUMBER OFOPERATORS ARE 2
 NUMBER OFOPERRANDS ARE 0,0universe@acer8:~/Desktop/8892$ ./a.out
+ - *


 NUMBER OFOPERATORS ARE 3
 NUMBER OFOPERRANDS ARE 0,0universe@acer8:~/Desktop/8892$ 
```

3. Number of Vowels and consonants

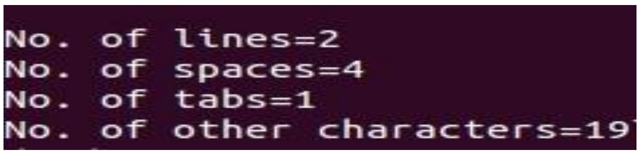
4. Number of words,characters,spaces,end of lines in a given input file.



```
%{
    #include<stdio.h>
    int vowels  = 0;
    int cons = 0;
%}
%%
[aeiouAEIOU] {vowels++;}
[a-zA-z] {cons++;}
%%
int yywrap(){
    return 1;
}
main(){
    printf("Enter string .. at end press ^d\n");
    yylex();
    printf("No of Vowels +%d\n No of consonants=%d\n",vowels,cons);
}
```

```
universe@acer9:~$ ./a.out
Enter string .. at end press ^d
aecct

^C
universe@acer9:~$ ^C
universe@acer9:~$ ./a.out
Enter string .. at end press ^d
eeav

^C
universe@acer9:~$ ./a.out
Enter string .. at end press ^d
eavtNo of Vowels +2
 No of consonants=2
universe@acer9:~$
```

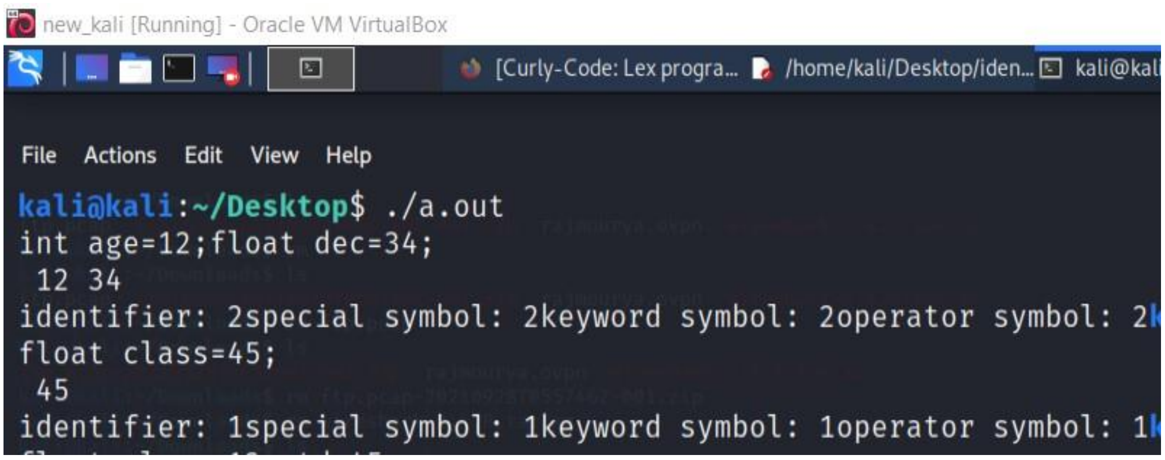
```
/*lex code to count the number of lines,
tabs and spaces used in the input*/
%{
#include<stdio.h> int lc=0, sc=0, tc=0, ch=0;
/*Global variables*/
%}

/*Rule Section*/
%%
\n lc++; //line counter
([ ])+ sc++; //space counter
\t tc++; //tab counter
. ch++;     //characters counter
%%
int main()
{
    // The function that starts the analysis
yylex();



    printf("\nNo. of lines=%d", lc);      printf("\nNo. of spaces=%d", sc);
printf("\nNo. of tabs=%d", tc);     printf("\nNo. of other characters=%d", ch);

}
```



```
No. of lines=2
No. of spaces=4
No. of tabs=1
No. of other characters=19
```

5. Number of variable,keywords,special symbol,contants,operators.

```
%{#include<iostream.h>
int identifier=0; char
ch=0; int special; int
keyword; int operator;
%}
keyword["int","float","double]
digit[0-9] letter[a-zA-Z_]
special[$&,:;?@#|'.^()!]
operator[+=*/]
%%
{keyword}+ {
keyword++;
}
{letter}({letter}|{digit})* {
identifier++;
}
{special}+ {
special++;
}
{operator}+ {







operator++;
} %%d int
main()
{  yylex();  printf("identifier:
%d",identifier);  printf("special
symbol: %d",special); printf("keyword
symbol: %d",keyword); printf("operator
symbol: %d",operator);  return 0;
}
```


new_kali [Running] - Oracle VM VirtualBox
[Curly-Code: Lex progra...
/home/kali/Desktop/iden...
kali@kal
File
Actions
Edit
View
Help

```
kali@kali:~/Desktop$ ./a.out
int age=12;float dec=34;
 12 34
identifier: 2special symbol: 2keyword symbol: 2operator symbol: 2k
float class=45;
 45
identifier: 1special symbol: 1keyword symbol: 1operator symbol: 1k
```

POSTLAB:

Q.1. Write the structure of Lex

A **lex** program consists of three sections: a section containing *definitions*, a section containing *translations*, and a section containing *functions*. The style of this layout is similar to that of **yacc**.

Throughout a **lex** program, you can freely use newlines and C-style comments; they are treated as white space. Lines starting with a blank or tab are copied through to the **lex** output file. Blanks and tabs are usually ignored, except when you use them to separate names from definitions, or expressions from actions.

The definition section is separated from the following section by a line consisting only of **%%**. In this section, named regular expressions can be defined, which means you can use names of regular expressions in the translation section, in place of common subexpressions, to make that section more readable. The definition section can be empty, but the **%%** separator is required.

Q.2 Write the structure of Yacc

Yacc has 3 sections deifintions ,rules and auxiliary routines.

Input File: Definition Part:

- The definition part includes information about the tokens used in the syntax definition:
- Yacc automatically assigns numbers for tokens, but it can be overridden by
- Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should not overlap ASCII codes.
- The definition part can include C code external to the definition of the parser and variable declarations, within **%{** and **%}** in the first column.
- It can also include the specification of the starting symbol in the grammar:

Input File: Rule Part:

- The rules part contains grammar definition in a modified BNF form.
- Actions is C code in { } and can be embedded inside (Translation schemes).

Input File: Auxiliary Routines Part:

- The auxiliary routines part is only C code.
- It includes function definitions for every function needed in rules part.
- It can also contain the main () function definition if the parser is going to be run as a program.
- The main() function must call the function yyparse().