Lab 02

Ques 1: Write a LEX program to recognize the following

Operators: +, -, *, /, |,
Numbers
newline
Any other character apart from the above should be recognized as mystery character

For each of the above mentioned matches (classes of lexeme) in your input, the program should print the following: PLUS, MINUS, MUL, DIV, ABS, NUMBER, NEW LINE, MYSTERY CHAR respectively. Your program should also strip of whitespaces.

```
number[0-9]+


%%

"+" printf("PLUS\n");

"-" printf("MINUS\n");

"*" printf("MULT\n");

"/" printf("DIV\n");

{number} {printf("NUMBER\n");}

. printf("MYSTERY\n");

"\n" printf("NEWLINE\n");

%%
```

There are 7 classes ie number, +, -, *, /, . and newline. The precedence starts from +, -, *, /, number, ., new line. The corresponding action states are written next to the corresponding states. The same is shown in the output given below.

```
[2019A7PS0020U@linuxbpdc1 lab3]$ a.out
+
PLUS
NEWLINE
-
MINUS
NEWLINE
*
MULT
NEWLINE
5
NUMBER
NEWLINE
a
MYSTERY
NEWLINE
[2019A7PS0020U@linuxbpdc1 lab3]$
```

Ques 2: Write a LEX program to print the number of words, characters and lines in a given input.

```
%{
int char_count =0;
int newline=0;
int word_count=0;
%}

char [a-z,A-Z]

%%
{char}+ {++word_count;
char_count+=yyleng;}
"\n" {++newline;}
%%

int main()
{
yylex();
printf("\nNo of characters = %d",char_count);
printf("\nNo of words = %d", word_count);
printf("\nNo of newlines = %d\n", newline);

return 0;
}
```

There's a *char* class that contains letters from a-z and A-Z. When compiler comes across atleast one character, it increases the no of words and stores the count in word_count. In the same action, the length of word is stored in *char_count* using *yyleng*. The number of new lines is counted using *\n* character (as a state). The precedence of counting words and characters is higher than newline. Finally the main program prints the results. *Yylex()* in main helps to take continuous input from user.

```
[2019A7PS0020U@linuxbpdc1 lab3]$ a.out
Dhruv Maheshwari
 CC Lab
 Tuesday

No of characters = 27
No of words = 5
No of newlines = 3
[2019A7PS0020U@linuxbpdc1 lab3]$
```

Ques 3: Modify the above LEX program so that a word and its characters are counted only if its length is greater than or equal to 6.

```
%{
int char_count = 0;
int newline = 0;
int word_count = 0;
%}

char [a-z,A-Z]

%%
{char}+ {if (yyleng > 6) {++word_count;}
char_count += yyleng;}
"\n" {++newline;}
%%

int main()
{
        yylex();
        printf("\nNo of characters = %d",char_count);
        printf("\nNo of words = %d",word_count);
        printf("\nNo of newline = %d\n",newline);

        return 0;
}
```

Same as Q2, only change is that an *if* statement checks the length of the word before incrementing the no of words. However, it stores the no of characters and displays it.

```
[2019A7PS0020U@linuxbpdc1 lab3]$ a.out
Dhruv Maheshwari
 CC Lab
 Tuesday

No of characters = 27
No of words = 2
No of newline = 3
[2019A7PS0020U@linuxbpdc1 lab3]$
```

Ques 4: Write a LEX program to print if the input is an odd number or an even number along with its length. Also, the program should check the correctness of the input (i.e. if the input is one even number and one odd number).

```
%{
int count =0;
%}

even [0-9]*[2|4|6|8|0]
odd [0-9]*[1|3|5|7|9]

%%
{even}""{odd} {printf("Correct\n");
count = yyleng;}
%%

int main()
{
        yylex();
        printf("Length = %d\n", count);
        return 0;
}
```

There are 2 classes defined as *even and odd* as shown in the code. In the rules section, there's a concatenation between even and odd grammar using "", which acts as an epsilon. The output

shown depicts that the compiler matches everything using . operator except even digit followed by odd digit and also gives the length of string.