## Assignment : 2

**Write a LEX program to display word, character and line counts for a sample input text file**

## file1.l

```
%{

   int ccount=0,capcount=0,smalcount=0,lcount=0;
%}

cword [A-Z]+
sword [a-z]+
eol \n
%%

{cword} {capcount++;ccount+=yyleng;}
{sword} {smalcount++;ccount+=yyleng;}
{eol}  {ccount++;lcount++;}
 .  {ccount++;}
%%

int main(void)
{
  yylex();
  printf("\nNumber of characters : %d",ccount);
  printf("\nNumber of Capital Case Words : %d",capcount);
  printf("\nNumber of small Case Words : %d",smalcount);
```

```
  printf("\nNumber of Lines : %d",lcount);
  return 0;
}

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

**Input.txt**

```
sys PROG
op sys
cl TE
br COMP
div B
ES iot
Wt
```

**Output: lex file1.l**

**cc.lex.yy.c**

**./a.out <input.txt**

**Explanation:**

### Definitions Section

```c
%{
   int ccount=0, capcount=0, smalcount=0, lcount=0;
%}
```

- `%{ %}`: This block is for C code that is copied verbatim into the generated C file.
- Inside this block, we declare and initialize four integer variables:
  - `ccount`: Total character count.
  - `capcount`: Count of words consisting entirely of uppercase letters.
  - `smalcount`: Count of words consisting entirely of lowercase letters.
  - `lcount`: Line count.

### Patterns Section

```c
cword [A-Z]+
sword [a-z]+
eol \n
```

This section defines regular expressions for different types of tokens:

- `cword`: Matches one or more uppercase letters.

- `sword`: Matches one or more lowercase letters.

- `eol`: Matches a newline character.

### Rules Section

```c
%%
```

The `%%` marks the beginning of the rules section where each pattern is associated with an action.

```c
{cword} {capcount++; ccount += yyleng;}

{sword} {smalcount++; ccount += yyleng;}

{eol}  {ccount++; lcount++;}

 .  {ccount++;}
```

- `{cword}`: When a sequence of uppercase letters is matched:

  - `capcount++`: Increment the count of capital case words.

- `ccount += yyleng`: Increment the character count by the length of the matched text (`yyleng` is a Flex variable that gives the length of the matched text).

- `{sword}`: When a sequence of lowercase letters is matched:
  - `smalcount++`: Increment the count of small case words.
  - `ccount += yyleng`: Increment the character count by the length of the matched text.

- `{eol}`: When a newline character is matched:
  - `ccount++`: Increment the character count by 1.
  - `lcount++`: Increment the line count by 1.

- `.`: For any single character not matched by the previous patterns (essentially a fallback for any other character):
  - `ccount++`: Increment the character count by 1.

### User Code Section

```c
%%
```

This `%%` marks the end of the rules section. The following code is the user code section, which typically includes the main function and other C code.

```c
```

```
int main(void)
{
  yylex();
  printf("\nNumber of characters : %d", ccount);
  printf("\nNumber of Capital Case Words : %d", capcount);
  printf("\nNumber of small Case Words : %d", smalcount);
  printf("\nNumber of Lines : %d", lcount);
  return 0;
}
```

- `int main(void)`: Defines the main function of the program.
- `yylex()`: This function is generated by Flex and starts the scanning process. It reads input, matches it against the patterns, and executes the corresponding actions.
- After `yylex()` completes, `printf` functions are used to output the total counts:
  - Number of characters.
  - Number of capital case words.
  - Number of small case words.
  - Number of lines.
- `return 0;`: Indicates that the program terminates successfully.

```c
int yywrap()
{
```

```
  return 1;
}
```

- `int yywrap()`: This function is called by `yylex()` when the end of the input is reached. Returning 1 indicates that there is no more input to process and `yylex()` should stop.

### Input File (Input.txt)

The input provided in `Input.txt`:

```
sys PROG
op sys
cl TE
br COMP
div B
ES iot
Wt
```

### Program Execution Explanation

When the program is run and `Input.txt` is processed:

1. `sys` matches `{sword}`: `smalcount` becomes 1, `ccount` increases by 3.
2. `PROG` matches `{cword}`: `capcount` becomes 1, `ccount` increases by 4.
3. `\n` matches `{eol}`: `lcount` becomes 1, `ccount` increases by 1.
4. `op` matches `{sword}`: `smalcount` becomes 2, `ccount` increases by 2.
5. `sys` matches `{sword}`: `smalcount` becomes 3, `ccount` increases by 3.
6. `\n` matches `{eol}`: `lcount` becomes 2, `ccount` increases by 1.
7. `cl` matches `{sword}`: `smalcount` becomes 4, `ccount` increases by 2.
8. `TE` matches `{cword}`: `capcount` becomes 2, `ccount` increases by 2.
9. `\n` matches `{eol}`: `lcount` becomes 3, `ccount` increases by 1.
10. `br` matches `{sword}`: `smalcount` becomes 5, `ccount` increases by 2.
11. `COMP` matches `{cword}`: `capcount` becomes 3, `ccount` increases by 4.
12. `\n` matches `{eol}`: `lcount` becomes 4, `ccount` increases by 1.
13. `div` matches `{sword}`: `smalcount` becomes 6, `ccount` increases by 3.
14. `B` matches `{cword}`: `capcount` becomes 4, `ccount` increases by 1.
15. `\n` matches `{eol}`: `lcount` becomes 5, `ccount` increases by 1.
16. `ES` matches `{cword}`: `capcount` becomes 5, `ccount` increases by 2.
17. `iot` matches `{sword}`: `smalcount` becomes 7, `ccount` increases by 3.
18. `\n` matches `{eol}`: `lcount` becomes 6, `ccount` increases by 1.
19. `Wt` does not match `{cword}` or `{sword}`, but matches `.` for each character: `ccount` increases by 2.

### Summary Output

After processing, the counts will be:

- `ccount`: Total number of characters (including newlines).
- `capcount`: Total number of capital case words.
- `smalcount`: Total number of small case words.
- `lcount`: Total number of lines.

The output will be:

```
Number of characters : 45
Number of Capital Case Words : 5
Number of small Case Words : 7
Number of Lines : 6
```