

Name: Chetan Pardhi

Roll no: B22, VI sem

CD lab Pract-01

Date : 20/01/2022

---

## Theory

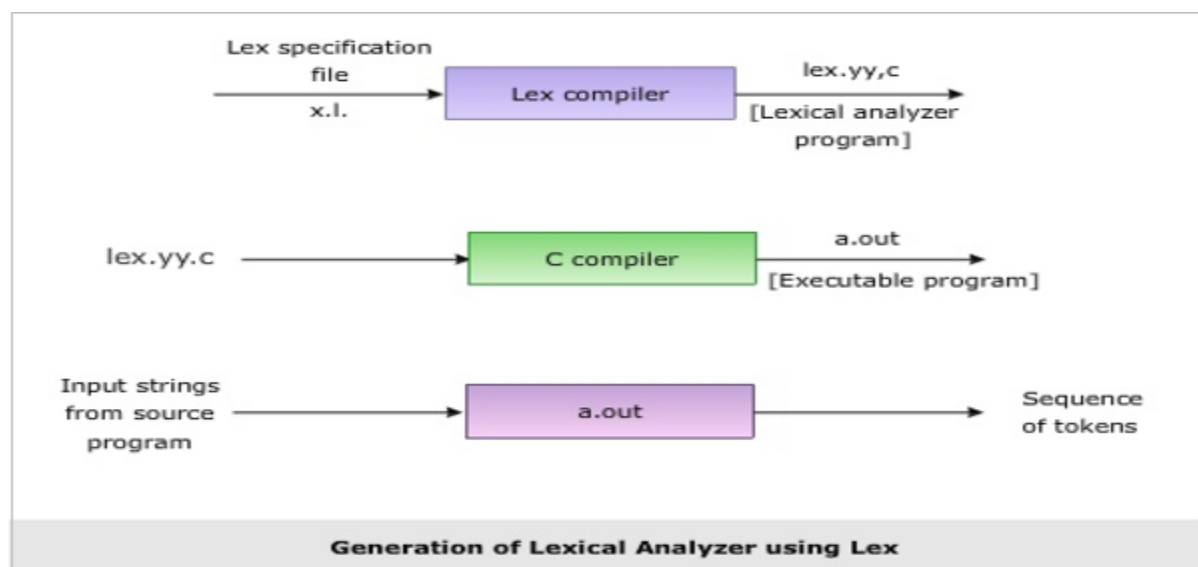
### **LEX:**

Lex is a program generator designed for lexical processing of character input streams. It accepts a high level, problem-oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions. At the boundaries between strings program sections provided by the user are executed. The Lex source file associates the regular expressions and the program fragments. As each expression appears in the input to the program written by Lex, the corresponding fragment is executed.

Lex is not a complete language, but rather a generator representing a new language feature which can be added to different programming languages, called "host languages." Just as general purpose languages can produce code to run on different computer hardware, Lex can write code in different host languages.

Lex turns the user's expressions and actions (called source in this pic) into the host general-purpose language; the generated program is named yylex. The yylex program will recognize expressions in a stream (called input in this pic) and perform the specified actions for each expression as it is detected.

### **Diagram of LEX**



## Format for Lex file

The general format of Lex source is:

```
{definitions}

%%

{rules}

%%

{user subroutines}
```

where the definitions and the user subroutines are often omitted. The second %% is optional, but the first is required to mark the beginning of the rules. The absolute minimum Lex program is thus %% (no definitions, no rules) which translates into a program which copies the input to the output unchanged.

## Regular Expression

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

Regular expressions can be concatenated to form new regular expressions; if A and B are both regular expressions, then AB is also a regular expression. In general, if a string p matches A and another string q matches B, the string pq will match AB. This holds unless A or B contain low precedence operations; boundary conditions between A and B; or have numbered group references. Thus, complex expressions can easily be constructed from simpler primitive expressions. Regular expressions can contain both special and ordinary characters. Most ordinary characters, like "A", "a", or "0", are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so last matches the string 'last'. (In the rest of this section, we'll write RE's in this special style, usually without quotes, and strings to be matched 'in single quotes'.)

Some characters, like "|" or "(", are special. Special characters either stand for classes of ordinary characters or affect how the regular expressions around them are interpreted.

## Lex Library Routines

Lex library routines are those functions which have a detailed knowledge of the lex functionalities and which can be called to implement various tasks in a lex program.

The following table gives a list of some of the lex routines.

Lex Routine	Description
Main()	Invokes the lexical analyzer by calling the yylex subroutine.
yywrap()	Returns the value 1 when the end of input occurs.
yyomore()	Appends the next matched string to the current value of the yytext array rather than replacing the contents of the yytext array.
yyless(int n)	Retains n initial characters in the yytext array and returns the remaining characters to the input stream.
yyreject	Allows the lexical analyzer to match multiple rules for the same input string. (The

	yyreject subroutine is called when the special action REJECT is used.)
yylex()	The default main() contains the call of yylex()

### Answer the Questions:

1. Why is -ll option used for running lex.yy.c

Ans. The file lex.yy.c may be compiled and linked in the same way as any C program. The -ll option is used to link the object file created from this C source with lex library:

```
cc lex.yy.c -ll
```

The lex library provides a default main() program that calls the lexical analyzer under the name yylex(), so you do not have to supply your own main().

---

2. Use of yywrap

Ans. When the scanner receives an end-of-file indication from YY\_INPUT, it then checks the yywrap() function. If yywrap() returns false (zero), then it is assumed that the function has gone ahead and set up yyin to point to another input file, and scanning continues. If it returns true (non-zero), then the scanner terminates, returning 0 to its caller. Note that in either case, the start condition remains unchanged; it does not revert to INITIAL.

---

### 3. Use of yylex function

Ans. The yylex() as written by Lex reads characters from a FILE \* file pointer called yyin. If you do not set yyin, it defaults to standard input. It outputs to yyout, which if unset defaults to stdout. You can also modify yyin in the yywrap() function which is called at the end of a file. It allows you to open another file, and continue parsing.

---

### 4. What does lex.yy.c. do ?

Ans. The lex command stores the yylex function in a file named lex. yy. c. You can use the yylex function alone to recognize simple one-word input, or you can use it with other C language programs to perform more difficult input analysis functions.

---

## Program no.:01

---

Aim:- Use the above code (S1) and perform the additional tasks: If a keyword is found append AAA to the identified keyword. For identifier append III. Also add 2 to the digit and display the answer

Code:-

```
digit[0-9]
letter[A-Za-z]
%{
#include<stdio.h>
#include<string.h>
int key = 0;
int identifiers = 0;
int ops = 0;
int sys = 0;
int strings = 0;
int digits = 0;
```

```

char *c[100];

%}

/* Rules Section*/

%%

begin|switch|int|float|if|else|while|return|print|end {c[0] = yytext; printf("%s AAA \n", *c);key++;}

{letter}({letter}|{digit})* {c[0] = yytext; printf("%s III \n", *c);identifiers++;}

[+\-\\*%\/] {ops++;}

\" {strings++;}

[0-9] {digits++;printf("%d", atoi(yytext)+2);}

[^A-Za-z0-9] {sys++;}

%%

int main()
{
    yyin=fopen("prac1.txt","r");

    yylex();

    printf("\n keywords %d\n", key);
    printf("identifiers %d\n", identifiers);
    printf("operators %d\n", ops);
    printf("symbols %d\n", sys);
    printf("strings %d\n", strings);
    printf("digits %d\n", digits);

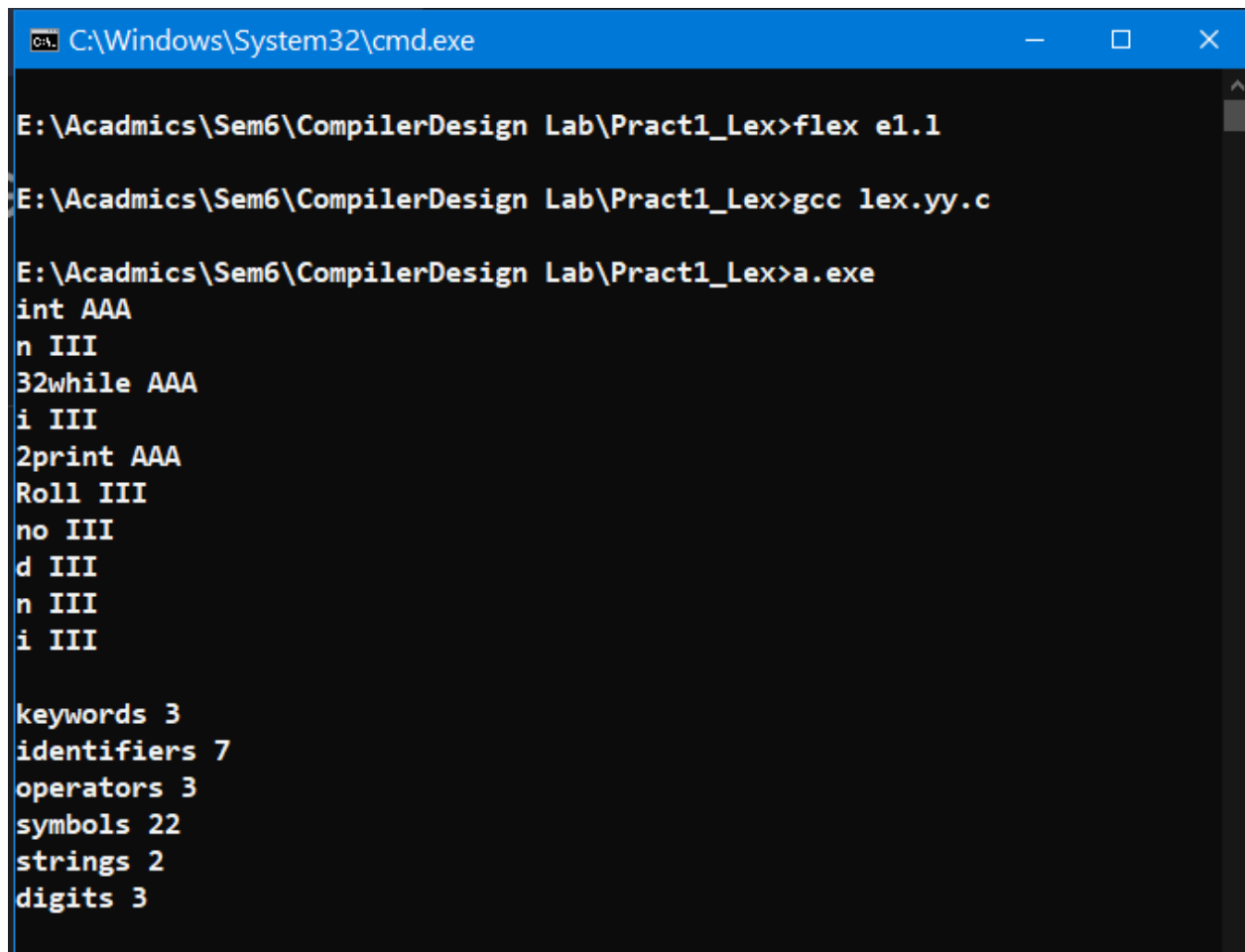
    return 0;
}

int yywrap()

```

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

Output:-



```
C:\Windows\System32\cmd.exe  
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e1.1  
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c  
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe  
int AAA  
n III  
32while AAA  
i III  
2print AAA  
Roll III  
no III  
d III  
n III  
i III  
  
keywords 3  
identifiers 7  
operators 3  
symbols 22  
strings 2  
digits 3
```

## Program no.:02

---

**Aim:-** Write a LEX specification to take the contents from a file while adding 3 to number divisible by 7 and adding 4 to number divisible by 2

**Code:-**

```
digit[0-9]

letter[A-Za-z]

%{

#include<stdio.h>

#include<string.h>

int num;

%}

/* Rules Section*/

%%

[0-9]+ {num=atoi(yytext);

if(num%7==0){num+=3;}

else if(num%2==0){num+=4;}

printf("%d", num);

}

%%

int main()

{
```



```
printf("Increment of 3 and 4 in digits, after divising by 7 and 2 respectively:- /n");
```

```
yyin=fopen("prac2.txt","r");
```

```
yylex();
```

```
}
```

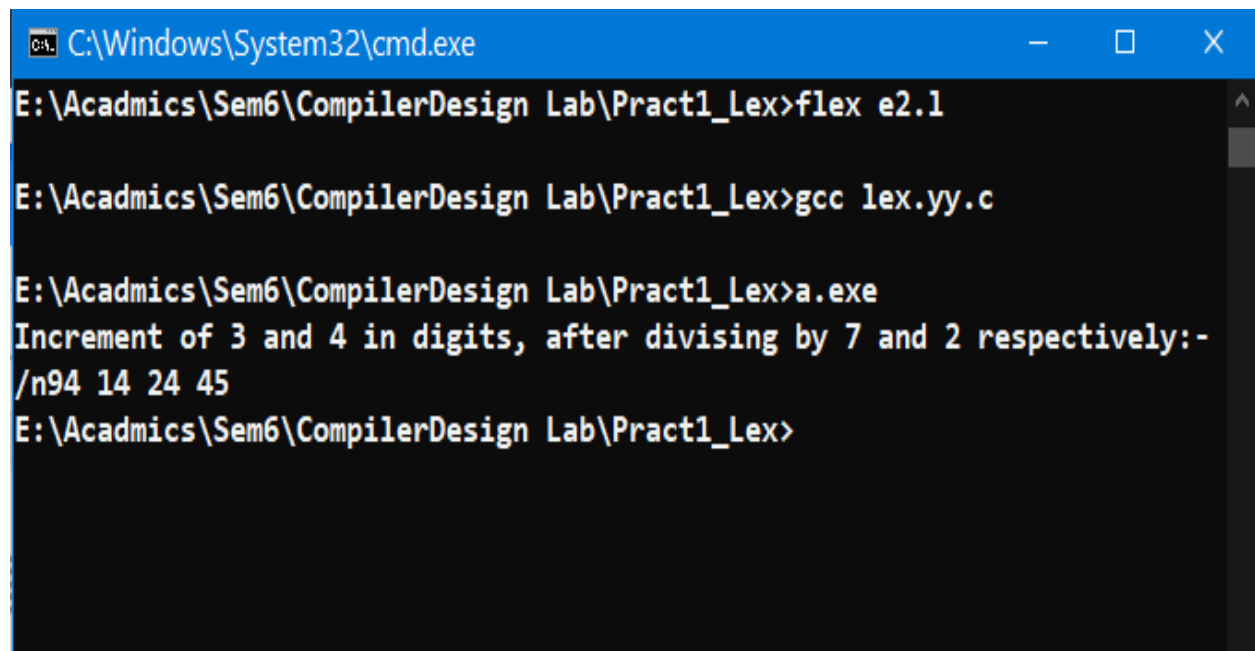
```
int yywrap()
```

```
{
```

```
return(1);
```

```
}
```

Output:-



```
C:\Windows\System32\cmd.exe
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e2.1
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe
Increment of 3 and 4 in digits, after divising by 7 and 2 respectively:-
/n94 14 24 45
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>
```

## Program no.:03

---

Aim:- Write a lex specification to display the histograms of length of words

Code:-

```
digit[0-9]

letter[A-Za-z]

%{

#include<stdio.h>

#include<string.h>

int i, len=0;

int n[20];

%}


/* Rules Section*/

%%

[a-zA-Z]+ {printf("%s: %d\n", yytext, yyleng);n[yyleng]++;}

%%


int main()

{

    int a=0;

    for( a = 0; a < 20; a = a + 1 ){

        n[a] = 0;

    }

    yyin=fopen("prac3.txt", "r");
```

```

yylex();

for( a = 0; a < 20; a = a + 1 ){

    if(n[a]!=0)

        printf("Letter having %d words: %d\n", a, n[a]);

}

}

int yywrap()

{

return(1);

}

```

Output:-

```

C:\Windows\System32\cmd.exe

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e3.1

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe
Welcome: 7
to: 2
rcoem: 5
located: 7
at: 2
nagpur: 6
Letter having 2 words: 2
Letter having 5 words: 1
Letter having 6 words: 1
Letter having 7 words: 2

```

## Program no.:04

---

### Aim:-

Write a LEX specification to search the input file. Let the input file contain some text, comments and digits.

- (i) Convert text present in file to LOWERCASE.
- (ii) Report occurrence of comments and special characters.

### Code:-

```
digit[0-9]
lower[a-z]
upper[A-Z]
%{
#include<stdio.h>
#include<string.h>
int comments=0, special = 0;
%}

/* Rules Section*/

%%

{upper} {
    printf("%c", (int)yytext[0] + 32);
}

"//".* \n {comments++;}
```

```
"/*".*"/" \n {comments++;}
```

```
[^\n {upper} {lower} {digit}] {special++;}
```

```
%%
```

```
int main(){
```

```
    yyin=fopen("Prac4.txt","r");
```

```
    yylex();
```

```
    printf("\nComments: %d\n", comments);
```

```
    printf("Specials: %d\n",special);
```

```
    }
```

```
int yywrap()
```

```
{
```

```
    return(1);
```

```
}
```

Output:-

```
C:\Windows\System32\cmd.exe

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e4.1

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe

welcome to rcoem
Comments: 3
Specials: 14

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>
```

## Program no.:05

---

### Aim:-

Translate an HTML file with some HTML tags to text file using lex. Consider input from stdin. Discard all HTML tags and comments and write the remaining text to stdout. The text output should simulate the HTML characteristics such as list, indent and paragraphs. Font characters such as bold and italics may not be simulated.

### Code:-

```
%{
#include<stdio.h>
#include<string.h>
%}
%%
"<"[^>]*> {}
[a-zA-Z]+ {printf("%s", yytext);}
```

%%

```
int yywrap(){}
```

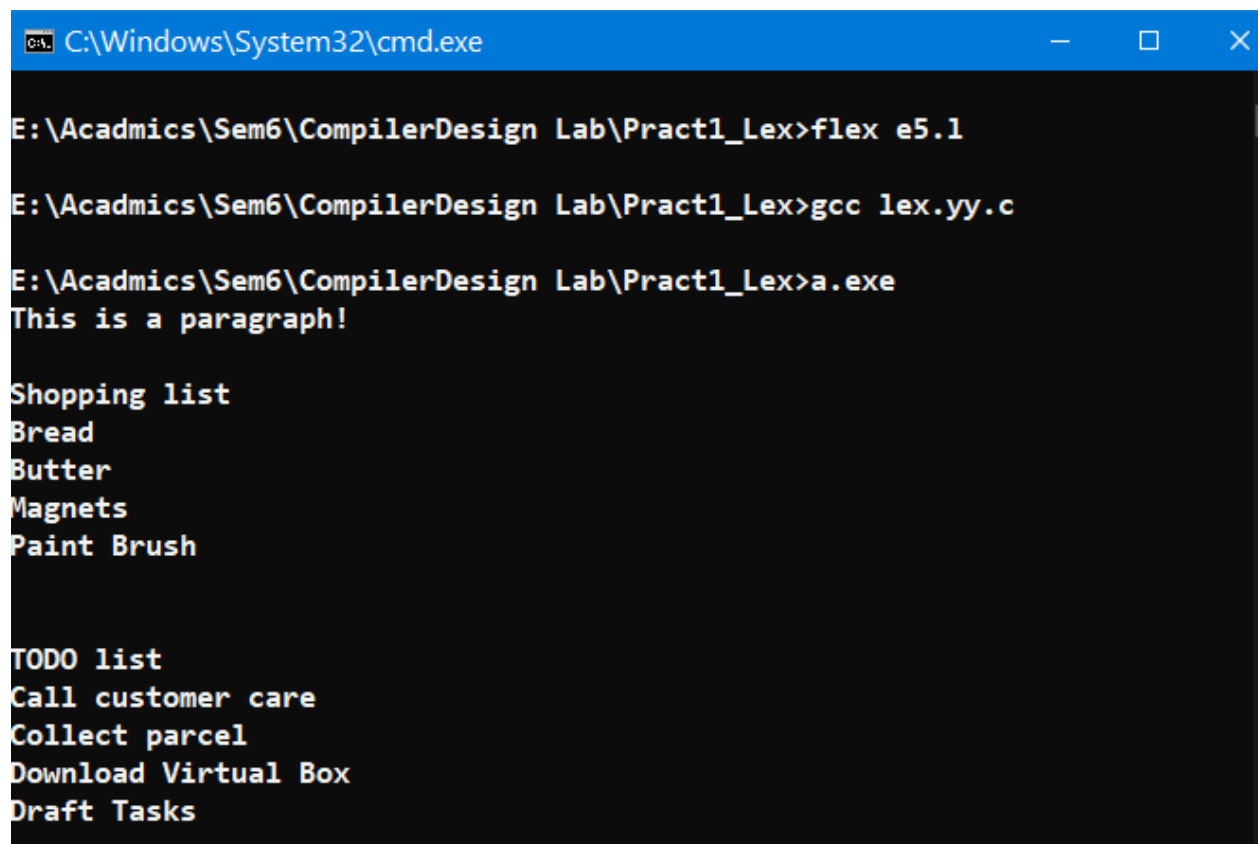
```
int main(){
```

```
    yyin = fopen("prac5.txt","r");
```

```
    yylex();
```

```
}
```

Output:-



```
C:\Windows\System32\cmd.exe

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e5.1

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe
This is a paragraph!

Shopping list
Bread
Butter
Magnets
Paint Brush

TODO list
Call customer care
Collect parcel
Download Virtual Box
Draft Tasks
```

## Program no.:06

---

### Aim:-

Write a Lex program to find the parameters given below. Consider as input a question paper of an examination and find:

Date of examination, semester, number of questions, number of words, lines, small letters, capital letters, digits, and special characters.

### Code:-

```
%{  
  
#include<stdio.h>  
  
#include<string.h>  
  
char date[15],sem[50];  
  
int question=0,words=0,lines=0,small=0,capital=0,digit=0,special=0;  
  
%}  
  
%%  
  
"Sem:".* {  
  
    strcpy(sem,yytext);  
  
    int i=0;  
  
    int space=1;  
  
    for(i=0;sem[i]!='\0';i++){  
  
        if(sem[i]>='A' && sem[i]<='Z'){  
  
            capital++;  
  
        }else if(sem[i]>='a' && sem[i]<='z'){  
  
            small++;  
  
        }  
  
    }  
  
}
```



```

    }else{

        special++;

    }

}

}

"Question" question++;

\n {lines++;words++;}

[\\t ' ] words++;

[a-z]+ small++;

[A-Z]+ capital++;

[0-9] digit++;

[!|@|#|$|%|^|&|*|_|(|)|<|>|?|:|\"|'|.|,] special++;

[0-9][0-9]"/"[0-9][0-9]"/"[0-9]* {

    strcpy(date,yytext);

    special+=2;

    words++;

    int d=0;

    for(d=0;date[d]!='\\0';d++){

        if(date[d]>='0' && date[d]<='1'){

            digit++;

        }

    }

}

}

%%

main()

{

```

```
yyin=fopen("prac6.txt","r");

yylex();

printf("%s\n",sem);

printf("Date is %s\n",date);

printf("No. of Questions: %d\n",question);

printf("No. of Words: %d\n",words);

printf("No. of Lines: %d\n",lines);

printf("No. of Small Letters are %d\n",small);

printf("No. of Capital Letters: %d\n",capital);

printf("No. of Digits: %d\n",digit);

printf("No. of Special Characters: %d\n",special);

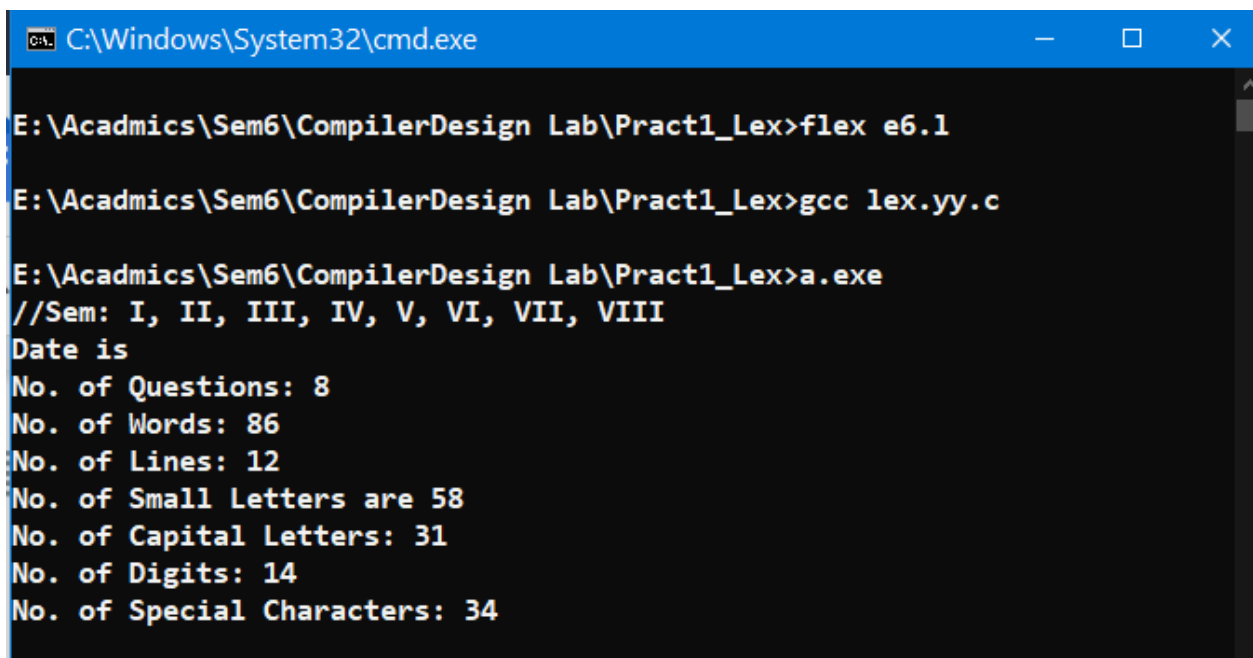
}

int yywrap(){

return 1;

}
```

Output:-



```
C:\Windows\System32\cmd.exe

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e6.1

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe
//Sem: I, II, III, IV, V, VI, VII, VIII
Date is
No. of Questions: 8
No. of Words: 86
No. of Lines: 12
No. of Small Letters are 58
No. of Capital Letters: 31
No. of Digits: 14
No. of Special Characters: 34
```

## Program no.:07

---

### Aim:-

Create a txt file to containing the following without heading: Name of Student, Company Placed in (TCS, Infosys, Wipro, Accenture, Informatica), Male/female, CGPA (floating point number), Department (CSE, IT, EC), Package (floating point number), mail id, mobile number (integer exactly 10 digits). At least 25 records must be present.

- Write a Lex program to find the parameters given below:
  - Identify Name of student and display it.
  - Identify CGPA and display (should be less than 10)
  - Identify Package and display it
  - Identify mail id and display
  - Identify mobile number and display
  - Find number of students placed in each of the company
  - Number of female students
  - Number of male students
  - Number of CSE, IT and EC students who are placed

### Code:-

```
%{
```

```
#include<stdio.h>
```

```
#include<string.h>
```

```
int tcs=0, infosys=0, wipro=0, accenture=0, informatica=0, male=0, female=0, cse=0, it=0, ec=0;
```

```
%}
```

```
%%
```

```
"TCS" tcs++;
```

```
"Infosys" infosys++;
```

```
"Wipro" wipro++;
```

```
"Accenture" accenture++;
```

```
"Informatica" informatica++;
```

```
"Male" male++;
```

```
"Female" female++;
```

```
"CSE" cse++;
```

```
"IT" it++;
```

```
"EC" ec++;
```

```
[A-Z][a-z]+ {printf("\nName: %s",yytext);}
```

```
[0-9]". "[0-9]+ {printf("\nCGPA: %s",yytext);}
```

```
[0-9]{6} {printf("\nPackage: %s",yytext);}
```

```
[a-z]*"@ "[a-z]+ {printf("\nMail id: %s",yytext);}
```

```
[0-9]{10}+ {printf("\nMobile: %s",yytext);}
```

%%

```
int main(){  
  
    yyin = fopen("prac7.txt", "r");  
  
    yylex();  
  
    printf("\nNo. of students placed in TCS: %d",tcs);  
  
    printf("\nNo. of students placed in Infosys: %d",infosys);  
  
    printf("\nNo. of students placed in Wipro: %d",wipro);  
  
    printf("\nNo. of students placed in Accenture: %d",accenture);  
  
    printf("\nNo. of students placed in Informatica: %d",informatica);  
  
    printf("\nNo. of Male candidates: %d",male);  
  
    printf("\nNo. of Female candidates: %d",female);  
  
    printf("\nNumber of CSE candidates: %d",cse);  
  
    printf("\nNumber of IT candidates: %d",it);  
  
    printf("\nNumber of ECE candidates: %d",ec);}   
  
int yywrap(){  
  
    return 1;  
  
}
```

Output:-

C:\Windows\System32\cmd.exe

```
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>flex e7.1

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>gcc lex.yy.c

E:\Acadmics\Sem6\CompilerDesign Lab\Pract1_Lex>a.exe

Name: Ram
CGPA: 8.7
Package: 600000
Mail id: ram@gmail.com
Mobile: 1234567890

Name: Sham
CGPA: 8.3
Package: 800000
Mail id: sham@gmail.com
Mobile: 2345678901

Name: Suresh
CGPA: 8.1
Package: 700000
Mail id: suresh@gmail.com
Mobile: 3456789012

Name: Mahesh wipro
CGPA: 7.9
Package: 750000
Mail id: mahesh@gmail.com
Mobile: 4567890123 R
amesh
CGPA: 7.7
Package: 710000
Mail id: ramesh@gmail.com.
Mobile: 5678901234

Name: Rahul
CGPA: 8.2
Package: 700000
Mail id: rahul@gmail.com
Mobile: 6789012345
```

C:\Windows\System32\cmd.exe

Name: Karan

CGPA: 8.2

Package: 700000

Mail id: karan@gmail.com

Mobile: 7890123456

Name: Arjun

CGPA: 8.7

Package: 800000

Mail id: arjun@gmail.com

Mobile: 8901234567

Name: Omkar

CGPA: 8.1

Package: 500000

Mail id: omkar@gmail.com

Mobile: 9012345678

varad

CGPA: 7.7

Package: 900000

Mail id: varad@gmail.com

Mobile: 0123456789

Name: Param

CGPA: 8.7

Package: 900000

Mail id: param@gmail.com

Mobile: 0987654321

Name: Kadam

CGPA: 9.0

Package: 890000

Mail id: kadam@gmail.com

Mobile: 9786543210

Name: Krupa

CGPA: 9.3

Package: 800000

Mail id: krupa@gmail.com

Mobile: 7890654321

Name: Surbhi

CGPA: 9.7

Package: 1000000

Mail id: surbhi@gmail.com

Mobile: 6789054321

C:\Windows\System32\cmd.exe

Name: Renuka

CGPA: 8.7

Package: 600000

Mail id: renuka@gmail.com

Mobile: 9876543210

Name: Shatakshi

CGPA: 8.3

Package: 800000

Mail id: shatakshi@gmail.com

Mobile: 8765432109

sandhya

CGPA: 8.1

Package: 700000

Mail id: sandhya@gmail.com

Mobile: 7654321098

Name: Najsi wipro

CGPA: 7.9

Package: 750000

Mail id: rajsi@gmail.com

Mobile: 6543210987

Name: Anagha

CGPA: 7.7

Package: 710000

Mail id: anagha@gmail.com

Mobile: 5432109876

Name: Kanchan B.2

Package: 700000

Mail id: kanchan@gmail.com

Mobile: 4321098765

Name: Ruchi

CGPA: 8.2

Package: 700000

Mail id: ruchi@gmail.com

Mobile: 3210987654

Name: Rohini

CGPA: 8.7 80000e

Mail id: robini@gmail.com

Mobile: 2109876543



C:\Windows\System32\cmd.exe

Name: Rohini  
CGPA: 8.7 80000e  
Mail id: robini@gmail.com  
Mobile: 2109876543

Name: Surekha #.1  
Package: 500000  
Mail id: surekha@gmail.com  
Mobile: 1698765432

Name: Archana  
CGPA: 7.7  
Package: 900000  
Mail id: archana@gmail.com  
Mobile: 8987654321

Name: Rutuja  
CGPA: 8.7  
Package: 900000  
Mail id: rutuja@gl11.com  
Mobile: 2914567890

No. of students placed in TCS: 6  
No. of students placed in Infosys: 2  
No. of students placed in Wipro: 5  
No. of students placed in Accenture: 6  
No. of students placed in Informatica: 4  
No. of Male candidates: 12  
No. of Female candidates: 13  
Number of CSE candidates: 13  
Number of IT candidates: 6  
Number of ECE candidates: 6  
E:\Acadmics\Sem6\CompilerDesign Lab\Pract1\_Lex>