

Name: **Bhavesh Kewalramani**
Roll No.: **A-25**

Practical No. 1

Theory

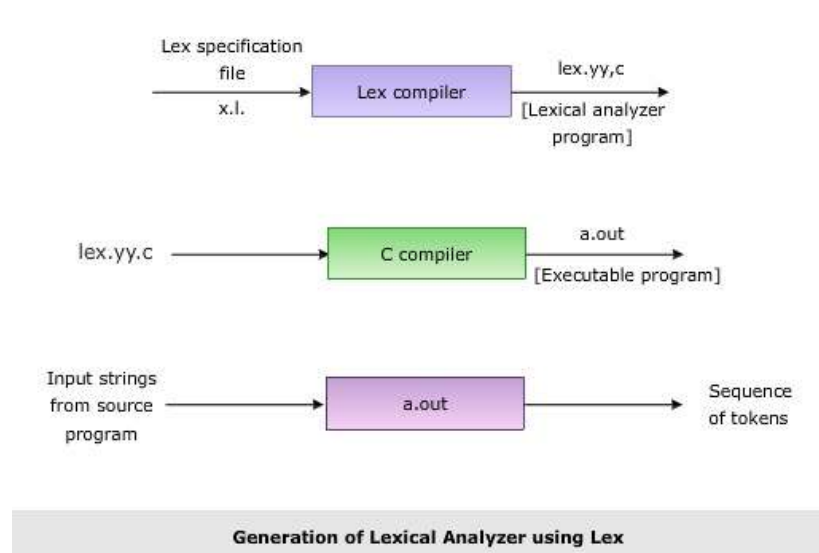
LEX:

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.
yymore()	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

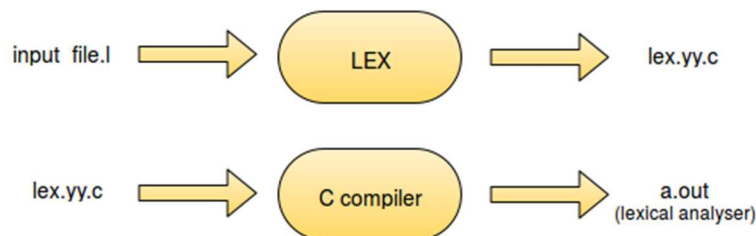
The lex library supplies a default **main()** that calls the function **yylex()**, so you need not supply your own **main()**. The library is accessed by invoking the **-ll** option. The **-ll** option is used to link the object file created from this C source with **lex** library

2. Use of yywrap

Function **yywrap** is called by lex when input is exhausted. Return 1 if you are done or 0 if more processing is required. Every C program requires a **main** function. In this case we simply call **yylex** that is the main entry-point for lex. Some implementations of lex include copies of **main** and **yywrap** in a library thus eliminating the need to code them explicitly. This is why our first example, the shortest lex program, functioned properly.

3. Internal representation of Lex

LEX is a tool used to generate a lexical analyzer. This document is a tutorial for the use of LEX for **ExpL Compiler** development. Technically, LEX translates a set of regular expression specifications (given as input in **input_file.l**) into a C implementation of a corresponding finite state machine (**lex.yy.c**). This C program, when compiled, yields an executable lexical analyzer.



The source ExpL program is fed as the input to the the lexical analyzer which produces a sequence of tokens as output. (Tokens are explained below). Conceptually, a lexical analyzer scans a given source ExpL program and produces an output of tokens.

Each token is specified by a token name. The token name is an abstract symbol representing the kind of lexical unit, e.g., a particular keyword, or a sequence of input

characters denoting an identifier. The token names are the input symbols that the parser processes. For instance integer, boolean, begin, end, if, while etc. are tokens in ExpL.

A lex program consists of three parts: the definition section, the rules section, and the user subroutines.

```
...definition section ...
```

```
%%
```

```
... rules section ...
```

```
%%
```

```
... user subroutines ...
```

The parts are separated by lines consisting of two percent signs. The first two parts are required, although a part may be empty. The third part and the preceding %% line may be omitted. (This structure is the same as that used by yacc, from which it was copied.)

Definition Section

The definition section can include the *literal block*, *definitions*, *internal table declarations*, *start conditions*, and *translations*. (There is a section on each in this reference.) Lines that start with whitespace are copied verbatim to the C file. Typically this is used to include comments enclosed in “/*” and “*/”, preceded by whitespace.

Rules Section

The rules section contains pattern lines and C code. A line that starts with whitespace, or material enclosed in “%{” and “%}” is C code. A line that starts with anything else is a pattern line.

C code lines are copied verbatim to the generated C file.

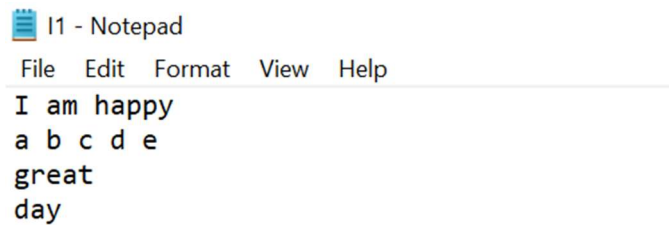
Practicals

Aim (I1): Write a Lex specification to declare whether the entered word starts with a vowel or not.

Program:

```
%{
#include<stdio.h>
%}
%%
[aeiouAEIOU][a-zA-Z0-9]* {printf("Word starts with a Vowel : \t%s\n",yytext);};
[a-zA-Z0-9]+ {printf("Word does not starts with a Vowel : \t%s\n",yytext);};
%%
int main(){
yyin = fopen("I1.txt","r");
yylex();
}
int yywrap() {return 1;}
```

Output:



I1 - Notepad

File Edit Format View Help

I am happy
a b c d e
great
day

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I1>flex I1.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I1>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I1>a.exe
Word starts with a Vowel :      I
Word starts with a Vowel :      am
Word does not starts with a Vowel :    happy

Word starts with a Vowel :      a
Word does not starts with a Vowel :    b
Word does not starts with a Vowel :    c
Word does not starts with a Vowel :    d
Word starts with a Vowel :      e

Word does not starts with a Vowel :    great

Word does not starts with a Vowel :    day

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I1>_
```

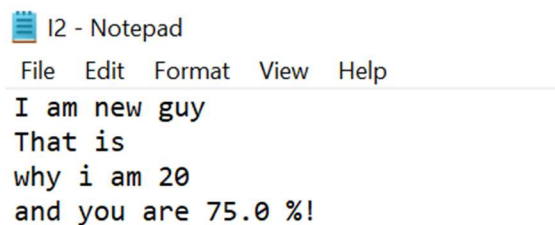
Aim (I2): Write a Lex Specification to count the number of words, lines, small letters, capital letters,digits and special characters in a given input file.

Program:

```
%{
#include<stdio.h>
int lines=0,words=0,smallchar=0,capitalchar=0,digits=0,specialchar=0;
%}
%%
\n {lines++; words++;}
[t' '] words++;
[a-z] smallchar++;
[A-Z] capitalchar++;
[0-9] digits++;
. specialchar++;
%%
void main()
{
yyin = fopen("I2.txt","r");
yylex();
```

```
printf("\n File has %d lines",lines);
printf("\n File has %d words",words);
printf("\n File has %d small characters",smallchar);
printf("\n File has %d capital characters",capitalchar);
printf("\n File has %d digits",digits);
printf("\n File has %d special characters",specialchar);
printf("\n File has %d total characters in all",smallchar+capitalchar+digits+specialchar);
}
int yywrap(){ return 1;}
```

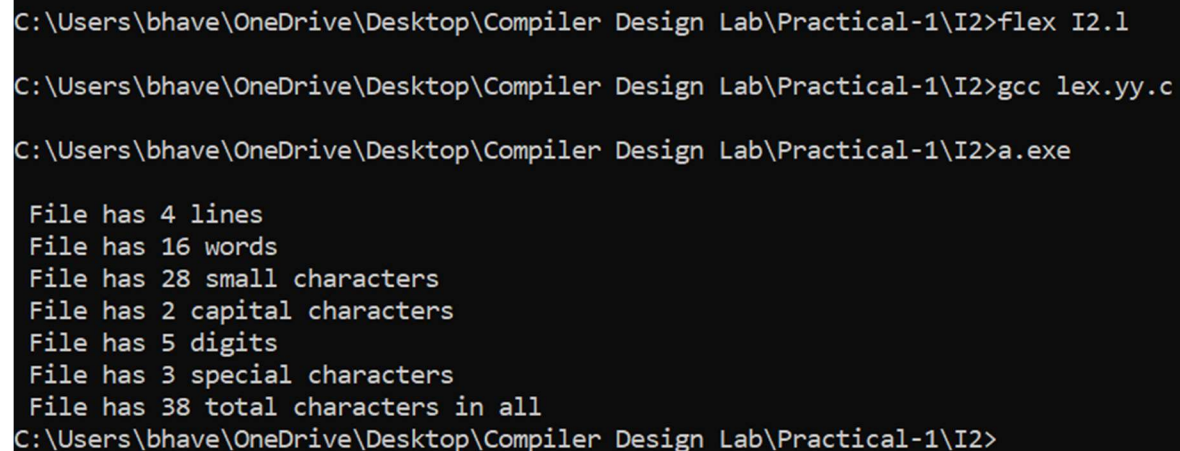
Output:



I2 - Notepad

File Edit Format View Help

I am new guy
That is
why i am 20
and you are 75.0 %!



```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I2>flex I2.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I2>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I2>a.exe

File has 4 lines
File has 16 words
File has 28 small characters
File has 2 capital characters
File has 5 digits
File has 3 special characters
File has 38 total characters in all
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\I2>
```

Aim (S1): Design a lexical analyser to identify the tokens such as keywords, identifiers, operators, symbols and strings for C language using Lex.

Program:

```

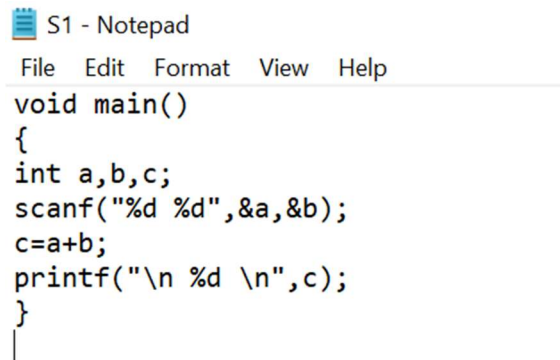
%{
#include<stdio.h>
%}
digit [0-9]
letter[a-zA-Z]
symbol [.,:|&|%( )|'|{|}|<|>]
string (\".*\")
%%

if|else|switch|while|do|void|main|printf|scanf|for {printf("\n %s is a KEYWORD",yytext);}
{letter}({letter}|{digit})* {printf("\n %s is an IDENTIFIER",yytext);}
{string} {printf("\n %s is a STRING",yytext);}
"&&"|"=="|"+"|"+|"-|"*"|"/"|"|"|"!" {printf("\n %s is an OPERATOR",yytext);}
({digit})* {printf("\n %s is a DIGIT",yytext);}
{symbol} {printf("\n %s is a SYMBOL",yytext);}
\n {printf("\n %s is a NEW LINE",yytext);}
\t {printf("\n %s is a TAB SPACE",yytext);}
%%

void main(){
yyin = fopen("S1.txt","r");
yylex();
}
int yywrap(){return 1;}

```

Output:



```

S1 - Notepad
File Edit Format View Help
void main()
{
int a,b,c;
scanf("%d %d",&a,&b);
c=a+b;
printf("\n %d \n",c);
}
|

```



```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\S1>flex S1.1

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\S1>gcc lex.yy.c

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\S1>a.exe

void is a KEYWORD
main is a KEYWORD
( is a SYMBOL
) is a SYMBOL

is a NEW LINE
{ is a SYMBOL

is a NEW LINE
int is an IDENTIFIER
a is an IDENTIFIER
, is a SYMBOL
b is an IDENTIFIER
, is a SYMBOL
c is an IDENTIFIER
; is a SYMBOL

is a NEW LINE
scanf is a KEYWORD
( is a SYMBOL
"%d %d" is a STRING
, is a SYMBOL
& is a SYMBOL
a is an IDENTIFIER
, is a SYMBOL
& is a SYMBOL
b is an IDENTIFIER
) is a SYMBOL
; is a SYMBOL

is a NEW LINE
c is an IDENTIFIER
= is an OPERATOR
```

```

is a NEW LINE
c is an IDENTIFIER
= is an OPERATOR
a is an IDENTIFIER
+ is an OPERATOR
b is an IDENTIFIER
; is a SYMBOL

is a NEW LINE
printf is a KEYWORD
( is a SYMBOL
"\n %d \n" is a STRING
, is a SYMBOL
c is an IDENTIFIER
) is a SYMBOL
; is a SYMBOL

is a NEW LINE
} is a SYMBOL

is a NEW LINE
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\S1>

```

Aim (E1): 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 digit and display the answer.

Program:

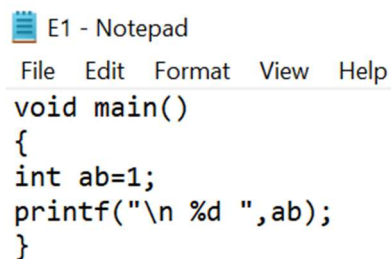
```

%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int val;
char c[100];
%}
digit [0-9]
letter [a-zA-Z]
symbol [.,;|&|%( )|'|"|{}|<|>]
string (".*\\")
%%
if|else|switch|while|do|void|main|printf|scanf|for|int|bool|char|float|double

```

```
{strcpy(c,yytext);strcat(c,"AAA");printf("\n KEYWORD: %s",c);}
{letter}({letter}|{digit})* {strcpy(c,yytext);strcat(c,"III");printf("\n IDENTIFIER: %s",c);}
({digit})* {val = atoi(yytext);printf("\n DIGIT: %d",val+2);}
%%
void main(){
yyin = fopen("E1.txt","r");
yylex();
}
int yywrap(){return 1;}
```

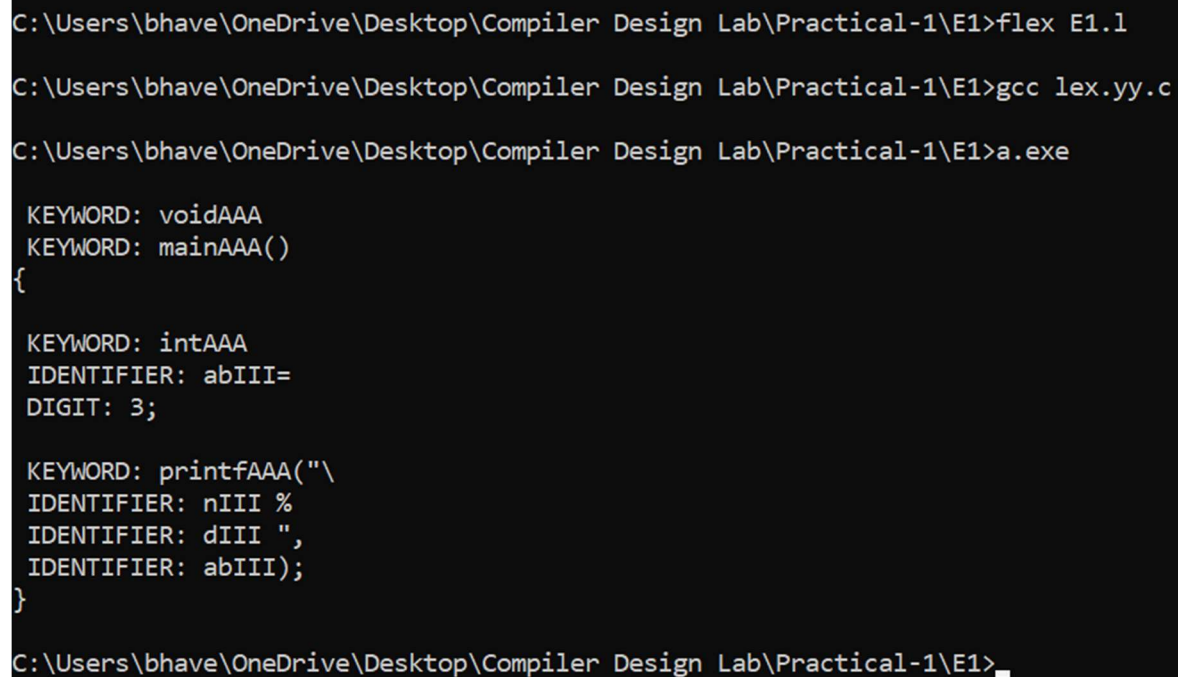
Output:



E1 - Notepad

File Edit Format View Help

```
void main()
{
int ab=1;
printf("\n %d ",ab);
}
```



```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E1>flex E1.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E1>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E1>a.exe

KEYWORD: voidAAA
KEYWORD: mainAAA()
{

KEYWORD: intAAA
IDENTIFIER: abIII=
DIGIT: 3;

KEYWORD: printfAAA("\
IDENTIFIER: nIII %
IDENTIFIER: dIII ",
IDENTIFIER: abIII);
}

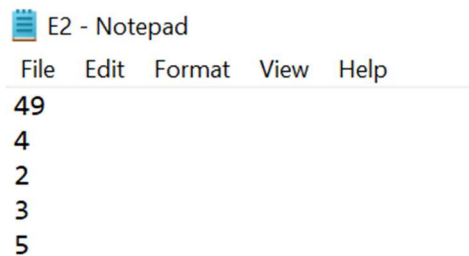
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E1>_
```

Aim (E2): 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.

Program:

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int val;
%}
digit [0-9]
%%
({digit})* {
    val = atoi(yytext);
    if(val%7==0)
        printf("\n %d \t %d",val,val+3);
    else if(val%2==0)
        printf("\n %d \t %d",val,val+4);
    else
        printf("\n %d",val);
}
%%
void main(){
yyin = fopen("E2.txt","r");
yylex();
}
int yywrap(){return 1;}
```

Output:



E2 - Notepad

File Edit Format View Help

49
4
2
3
5

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E2>flex E2.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E2>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E2>a.exe

49      52
4        8
2        6
3
5

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E2>
```

Aim (E3): Write a lex specification to display the histograms of length of words.


Program:

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int val;
char c[100];
%}
%%
[a-zA-Z0-9]* {strcpy(c,yytext);}
%%

void main(){
yyin = fopen("E3.txt","r");
yylex();
char ch;
int i=0,j,s=0,w=0,ndig[10],len=0;
for(i=0;i<10;i++)
    ndig[i]=0;
i=0;
```

```
while(c[i]!=EOF)
{
    if(s==1)
        len++;
    if(c[i]==' '||c[i]=='\n'||c[i]=='\t')
    {
        s=0;
        if(len<10)
            ++ndig[len];
        len=0;
    }
    else if(s==0)
    {
        s=1;
        w++;
    }
    i++;
}
for(i=0;i<10;i++)
{
    printf("%d =\t %d",i,ndig[i]);
    printf("\n");
}
}
int yywrap(){return 1;}
```

Output:

 E3 - Notepad
File Edit Format View Help
Life imposes things on you that you cannot control
but you still have the choice of how you are going to
live through this

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E3>flex E3.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E3>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E3>a.exe

0 =      0
1 =      2
2 =      4
3 =      4
4 =      4
5 =      2
6 =      0
7 =      0
8 =      5
9 =      0

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E3>
```

Aim (E4): 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.


Program:

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
char c[100];
int i;
%}
symbol [,;|&|%|(|)"'|{|}|<|>|!|@|#|$|^]
letter[a-zA-Z]
digit [0-9]
%%
[a-zA-Z] {      strcpy(c,yytext);
               for(i=0;c[i]!='\0';i++)
```

```

        {
            if(c[i]>='A' && c[i]<='Z')
            {
                c[i]=c[i]+32;
            }
        }
        printf("%s",c);
    }
[0-9]* {printf("\n Digit: %s",yytext);}
"//".*\n {    printf("\n Single Line Comment \t");
              strcpy(c,yytext);
              for(i=0;c[i]!='\0';i++)
              {
                  if(c[i]>='A' && c[i]<='Z')
                  {
                      c[i]=c[i]+32;
                  }
                  if(c[i]=='/')
                  {
                      c[i]=' ';
                  }
              }
              printf("%s",c);
          }
"/*"[^*/*]*)" {printf("\n Multi-Line Comment \n");
                strcpy(c,yytext);
                for(i=0;c[i]!='\0';i++)
                {
                    if(c[i]>='A' && c[i]<='Z')
                    {
                        c[i]=c[i]+32;
                    }
                    if(c[i]=='/' || c[i]=='*')
                    {
                        c[i]=' ';
                    }
                }
                printf("%s",c);
            }
{symbol}+ {printf("\n \n Special Character: %s",yytext);}
%%
void main(){
yyin = fopen("E4.txt","r");
yylex();
}
int yywrap(){return 1;}

```


Output: E4 - Notepad

```
File Edit Format View Help
Hello my name is ABC !
//You know : )
/*This is a great day
smile and shine*/
0987
```

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E4>flex E4.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E4>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E4>a.exe
hello my name is abc

Special Character: !

Single Line Comment      you know : )

Multi-Line Comment
    this is a great day
smile and shine

Digit: 0987

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E4>_
```

Aim (E5): 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.

Program:

```
%{
```

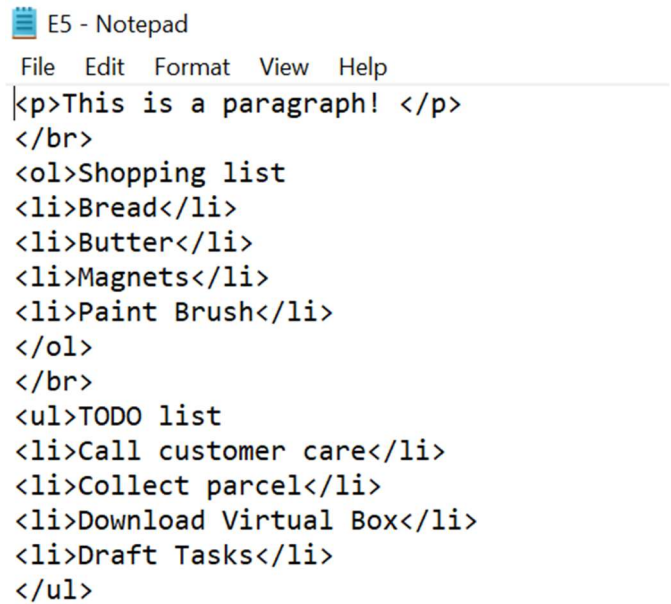
```

#include<stdio.h>
#include<string.h>
char c[100];
int flagOL=0,flagUL=0,i=1,j,k;
%}
%%
"</br>" {printf("\n");}
"<p>".*\n {k=0;for(j=0;j<strlen(yytext)-5;j++)
    {
        if(yytext[j]=='<' && yytext[j+1]=='p')
        {
            j=j+3;
        }
        if(yytext[j]=='<' && yytext[j+1]=='\')
        {
            j=j+4;
        }
        c[k]=yytext[j];
        k++;
    }
    printf("\n %s ",c);
}
"<ol>" {flagOL=1;
        flagUL=0;
}
"</ol>" {i=0,flagOL=0;}
"<li>".*"/li"> { if(flagOL==1 && flagUL==0)
    {
        printf("\n %d \t",i);
        i++;
    }
    if(flagOL==0 && flagUL==1)
    {
        printf("\n # \t");
    }
    for(j=4;j<strlen(yytext)-5;j++)
    {
        printf("%c",yytext[j]);
    }
}
"<ul>" {flagOL=0;
        flagUL=1;
}
"</ul>" {flagUL=0;}
%%

```

```
void main(){
yyin = fopen("E5.txt","r");
yylex();
}
int yywrap(){return 1;}
```

Output:

A screenshot of a Notepad window titled "E5 - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text content is HTML code that has been rendered into a visual format. It starts with a paragraph "This is a paragraph!", followed by a line break. Then there is an ordered list with four items: "Shopping list", "Bread", "Butter", and "Magnets", followed by "Paint Brush". After another line break, there is an unordered list with four items: "TODO list", "Call customer care", "Collect parcel", and "Download Virtual Box", followed by "Draft Tasks".

E5 - Notepad

File Edit Format View Help

<p>This is a paragraph! </p>

</br>

Shopping list

Bread

Butter

Magnets

Paint Brush

</br>

TODO list

Call customer care

Collect parcel

Download Virtual Box

Draft Tasks


```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E5>flex E5.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E5>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E5>a.exe

  This is a paragraph!

Shopping list

1      Bread
2      Butter
3      Magnets
4      Paint Brush

TODO list

#      Call customer care
#      Collect parcel
#      Download Virtual Box
#      Draft Tasks

C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E5>_
```

Aim (E6): 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, numbers of words, lines, small letters, capital letters, digits, and special characters.

Program:

```
%{
#include<stdio.h>
#include<string.h>
```

```

int lines=0,words=0,smallchar=0,capitalchar=0,digits=0,specialchar=0,questions=0;
char ch[100];
int i,j,k;
%}
%%
\n {lines++; words++;}
([0-2][0-9]|3[0-1])\|(0[1-9]|1[0-2])\|([1-2][0-9][0-9][0-9]) {printf("\n Date of Examination :
%s",yytext);}
[a-zA-Z0-9]*"College "[a-zA-Z0-9]* {printf("\n College Name : %s",yytext);
    strcpy(ch,yytext);
    for(i=0;ch[i]!='\0';i++)
    {
        if(ch[i]>='a' && ch[i]<='z')
        {
            smallchar++;
        }
        if(ch[i]>='A' && ch[i]<='Z')
        {
            capitalchar++;
        }
        if(ch[i]>='0' && ch[i]<='9')
        {
            digits++;
        }
    }
    lines++;
}
}
"Sem:".*\n {
    strcpy(ch,yytext);
    for(i=0;ch[i]!='\0';i++)
    {
        if(ch[i]>='a' && ch[i]<='z')
        {
            smallchar++;
        }
        if(ch[i]>='A' && ch[i]<='Z')
        {
            capitalchar++;
        }
        if(ch[i]>='0' && ch[i]<='9')
        {
            digits++;
        }
        if(ch[i]==' ')
        {
            words++;
        }
    }
}

```


```

    }
    if(ch[i]>='S' && ch[i+1]<='e' && ch[i+2]=='m' && ch[i+3]==':')
    {
        ch[i]=' ';
        ch[i+1]=' ';
        ch[i+2]=' ';
        ch[i+3]=' ';
        i=i+3;
    }
}
printf("\n Semester : %s",ch);
lines++;
}
"Question".*\n {questions++;
    strcpy(ch,yytext);
    for(i=0;ch[i]!='\0';i++)
    {
        if(ch[i]>='a' && ch[i]<='z')
        {
            smallchar++;
        }
        if(ch[i]>='A' && ch[i]<='Z')
        {
            capitalchar++;
        }
        if(ch[i]>='0' && ch[i]<='9')
        {
            digits++;
        }
        if(ch[i]==' ')
        {
            words++;
        }
        if(ch[i]>='S' && ch[i+1]<='e' && ch[i+2]=='m' && ch[i+3]==':')
        {
            ch[i]=' ';
            ch[i+1]=' ';
            ch[i+2]=' ';
            ch[i+3]=' ';
            i=i+3;
        }
    }
    lines++;
}
[\t' '] words++;
[a-z] smallchar++;

```

```
[A-Z] capitalchar++;
[0-9] digits++;
. specialchar++;
%%
void main()
{
yyin = fopen("E6.txt","r");
yylex();
printf("\n Number of Questions : %d",questions);
printf("\n File has %d lines",lines);
printf("\n File has %d words",words);
printf("\n File has %d small characters",smallchar);
printf("\n File has %d capital characters",capitalchar);
printf("\n File has %d digits",digits);
printf("\n File has %d special characters",specialchar);
printf("\n File has %d total characters in all",smallchar+capitalchar+digits+specialchar);
}
int yywrap(){ return 1;}
```

Output:

 E6 - Notepad

File Edit Format View Help

ABC College

01/01/2000

Sem: I, II, III, IV, V, VI, VII, VIII

Question1 : What are the benefits of tree plantation?

Question2 : What is water pollution?

Question3 : What should be done to avoid road accidents?

Question4 : What are your view on noise pollution?

Question5 : Why should people adopt pets?

Question6 : What is green gym?

Question7 : What norms must be implemented to minimize the loss from construction to environment?

Question8 : What is air pollution?

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E6>flex E6.1
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E6>gcc lex.yy.c
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E6>a.exe

Date of Examination : 01/01/2000
Semester :      I, II, III, IV, V, VI, VII, VIII

Number of Questions : 8
File has 12 lines
File has 75 words
File has 303 small characters
File has 39 capital characters
File has 8 digits
File has 0 special characters
File has 350 total characters in all
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E6>_
```

Aim (E7): 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

Program:

```
%{
```



```

#include<stdio.h>
#include<string.h>
char ch[100];
int a=0,b=0,c=0,d=0,e=0,nf=0,nm=0,nc=0,ni=0,ne=0;
%}
%%
"TCS"|"Infosys"|"Wipro"|"Accenture"|"Informatica" {strcpy(ch,yytext);
                                printf("\n Company Name : %s",ch);
                                if(strcmp(ch,"TCS")==0) a++;
                                if(strcmp(ch,"Infosys")==0) b++;
                                if(strcmp(ch,"Wipro")==0) c++;
                                if(strcmp(ch,"Accenture")==0) d++;
                                if(strcmp(ch,"Informatica")==0) e++;
}
"CSE"|"IT"|"EC" {strcpy(ch,yytext);
                printf("\n Branch : %s",ch);
                if(strcmp(ch,"CSE")==0) nc++;
                if(strcmp(ch,"IT")==0) ni++;
                if(strcmp(ch,"EC")==0) ne++;
}
"Male"|"Female" {strcpy(ch,yytext);
                printf("\n Gender : %s",ch);
                if(strcmp(ch,"Female")==0) nf++;
                if(strcmp(ch,"Male")==0) nm++;
}
[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9] {printf("\n Mobile : %s",yytext);}
[1-9][0-9][0-9][0-9][0-9][0-9]* {printf("\n Package : %s",yytext);}
[a-z.0-9_]+@[a-z]+".com"|[a-z.0-9_]+@[a-z]+".in"|[a-z.0-9_]+@[a-z]+".edu" {printf("\n Mail id
: %s",yytext);}
[0-9]"."[0-9]* {printf("\n CGPA : %s",yytext);}
[a-zA-Z]* {printf("\n Name of the student : %s",yytext);}
%%
void main()
{
yyin = fopen("E7.txt","r");
yylex();
printf("\n\n\n");
printf("\n Number Of students placed in TCS      : %d ",a);
printf("\n Number of students placed in Infosys   : %d ",b);
printf("\n Number of students placed in Wipro      : %d ",c);
printf("\n Number of students placed in Accenture   : %d ",d);
printf("\n Number of students placed in Informatica : %d ",e);
printf("\n Number of Female students              : %d ",nf);
printf("\n Number of Male students                  : %d ",nm);
printf("\n Number of CSE students placed            : %d ",nc);
printf("\n Number of IT students placed              : %d ",ni);
}

```

```
printf("\n Number of EC students placed      : %d ",ne);
}
int yywrap(){ return 1;}
```

Output:

 E7 - Notepad

File	Edit	Format	View	Help		
aaa	TCS	Female	9.4	CSE 600000	aaa@rknec.edu	9999999999
bbb	Infosys	Male	8.5	IT 750000	bbb@rknec.edu	7897897899
ccc	Wipro	Female	9.9	IT 650000	ccc@rknec.edu	8888888888
ddd	Accenture	Female	9.5	CSE 500000	ddd@rknec.edu	7777777777
eee	Wipro	Male	8.6	CSE 400000	eee1@rknec.edu	7654555555
fff	TCS	Male	8.8	CSE 700000	fff@rknec.edu	8877887788
ggg	TCS	Male	8.7	EC 750000	ggg1@rknec.edu	7766776677
hhh	Informatica	Female	9.4	EC 600000	hhh_1@rknec.edu	8898898898
iii	Wipro	Female	8.9	IT 400000	iii@rknec.edu	9900990099
jjj	Accenture	Female	9.0	CSE 700000	jjj@rknec.edu	9898977799
kkk	Wipro	Female	9.5	CSE 800000	kkk1.1@rknec.edu	9898977799
lll	Wipro	Male	9.6	CSE 900000	lll.m@rknec.edu	9898977799
mmm	Informatica	Female	9.4	EC 1000000	mmm@rknec.edu	9898977799
nnn	Informatica	Male	9.9	EC 1100000	nnn@rknec.edu	9898977799
ooo	Informatica	Male	9.8	IT 1200000	ooo_@rknec.edu	9898977799

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E7>flex E7.1
```

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E7>gcc lex.yy.c
```

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E7>a.exe
```

```
Name of the student : aaa  
Company Name : TCS  
Gender : Female  
CGPA : 9.4  
Branch : CSE  
Package : 600000  
Mail id : aaa@rk nec.edu  
Mobile : 9999999999
```

```
Name of the student : bbb  
Company Name : Infosys  
Gender : Male  
CGPA : 8.5  
Branch : IT  
Package : 750000  
Mail id : bbb@rk nec.edu  
Mobile : 7897897899
```

```
Name of the student : ccc  
Company Name : Wipro  
Gender : Female  
CGPA : 9.9  
Branch : IT  
Package : 650000  
Mail id : ccc@rk nec.edu  
Mobile : 8888888888
```

Name of the student : ddd
Company Name : Accenture
Gender : Female
CGPA : 9.5
Branch : CSE
Package : 500000
Mail id : ddd@rknec.edu
Mobile : 7777777777

Name of the student : eee
Company Name : Wipro
Gender : Male
CGPA : 8.6
Branch : CSE
Package : 400000
Mail id : eee1@rknec.edu
Mobile : 7654555555

Name of the student : fff
Company Name : TCS
Gender : Male
CGPA : 8.8
Branch : CSE
Package : 700000
Mail id : fff@rknec.edu
Mobile : 8877887788

Name of the student : ggg
Company Name : TCS
Gender : Male
CGPA : 8.7
Branch : EC
Package : 750000
Mail id : ggg1@rknec.edu
Mobile : 7766776677

Name of the student : hhh
Company Name : Informatica
Gender : Female
CGPA : 9.4
Branch : EC
Package : 600000
Mail id : hhh_1@rknec.edu
Mobile : 8898898898

Name of the student : iii
Company Name : Wipro
Gender : Female
CGPA : 8.9
Branch : IT
Package : 400000
Mail id : iii@rknec.edu
Mobile : 9900990099

Name of the student : jjj
Company Name : Accenture
Gender : Female
CGPA : 9.0
Branch : CSE
Package : 700000
Mail id : jjj@rknec.edu
Mobile : 9898977799

Name of the student : kkk
Company Name : Wipro
Gender : Female
CGPA : 9.5
Branch : CSE
Package : 800000
Mail id : kkk1.1@rknec.edu
Mobile : 9898977799

Name of the student : lll
Company Name : Wipro
Gender : Male
CGPA : 9.6
Branch : CSE
Package : 900000
Mail id : lll.m@rknec.edu
Mobile : 9898977799

Name of the student : mmm
Company Name : Informatica
Gender : Female
CGPA : 9.4
Branch : EC
Package : 1000000
Mail id : mmm@rknec.edu
Mobile : 9898977799

Name of the student : nnn
Company Name : Informatica
Gender : Male
CGPA : 9.9
Branch : EC
Package : 1100000
Mail id : nnn@rknec.edu
Mobile : 9898977799

Name of the student : ooo
Company Name : Informatica
Gender : Male
CGPA : 9.8
Branch : IT
Package : 1200000
Mail id : ooo_@rknec.edu
Mobile : 9898977799

```
Name of the student : nnn
Company Name : Informatica
Gender : Male
CGPA : 9.9
Branch : EC
Package : 1100000
Mail id : nnn@rknec.edu
Mobile : 9898977799
```

```
Name of the student : ooo
Company Name : Informatica
Gender : Male
CGPA : 9.8
Branch : IT
Package : 1200000
Mail id : ooo_@rknec.edu
Mobile : 9898977799
```

```
Number Of students placed in TCS      : 3
Number of students placed in Infosys  : 1
Number of students placed in Wipro    : 5
Number of students placed in Accenture : 2
Number of students placed in Informatica : 4
Number of Female students             : 8
Number of Male students               : 7
Number of CSE students placed         : 7
Number of IT students placed          : 4
Number of EC students placed          : 4
```

```
C:\Users\bhave\OneDrive\Desktop\Compiler Design Lab\Practical-1\E7>
```