Name: Kush Munot
Roll No. 47

Practical No. 1

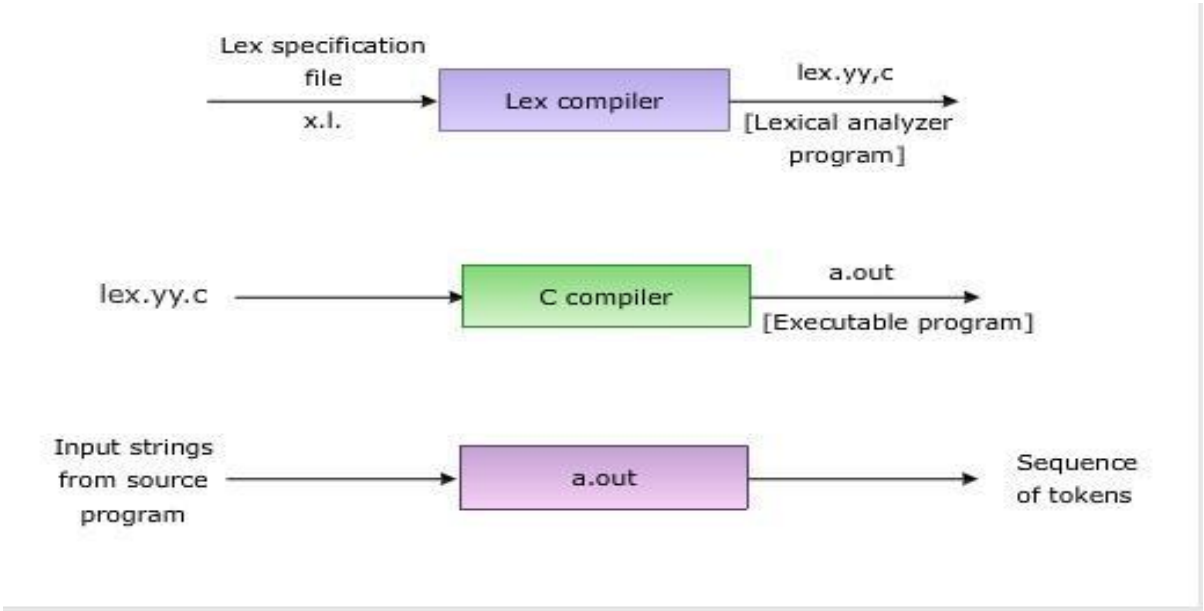
Theory

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 com puter 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



Generation of Lexical Analyzer using 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 pqwill 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. ***Use of yywrap***
   In lex, yywrap is a function that is used to indicate the end of input to the lex scanner. It is called by the scanner when it reaches the end of the input stream or when an error occurs during scanning. The yywrap function returns a non-zero value if there is no more input available to scan, and a zero value otherwise.

   The purpose of yywrap is to allow lex to process multiple input files or input streams. When lex reaches the end of the current input file, it calls yywrap to determine whether there is more input to process. If yywrap returns a non-zero value, lex exits; otherwise, lex continues scanning the next input file or input stream.

2. ***Use of yylex function***
   In lex, yylex is the main function that runs the lexical scanner. It reads input from a file or a standard input stream, applies the rules defined in the lex program to the input, and produces a stream of tokens that can be used by a parser or another program.

   The yylex function returns the next token from the input, according to the rules defined in the lex program. The tokens can be any type of data structure that is appropriate for the particular application. For example, if the lex program is used to scan a programming language, the tokens might include keywords, identifiers, literals, and punctuation symbols

3. ***What does lex.yy.c. do?***
   When you run a lex program, the lex compiler generates a C source code file called lex.yy.c. This file contains the implementation of the lexical scanner that was defined in the lex program.

   The lex.yy.c file can be compiled with a C compiler, such as gcc, to create an executable program that implements the lexical scanner. The resulting program can then be used to scan input files and produce streams of tokens that can be used by a parser or another program.

Practical No. E1

Aim: Design a lexical analyzer to identify the tokens such as keywords, identifiers, operators, constants (Int & float), special symbols and strings for C language using LEX. Use File for the input.

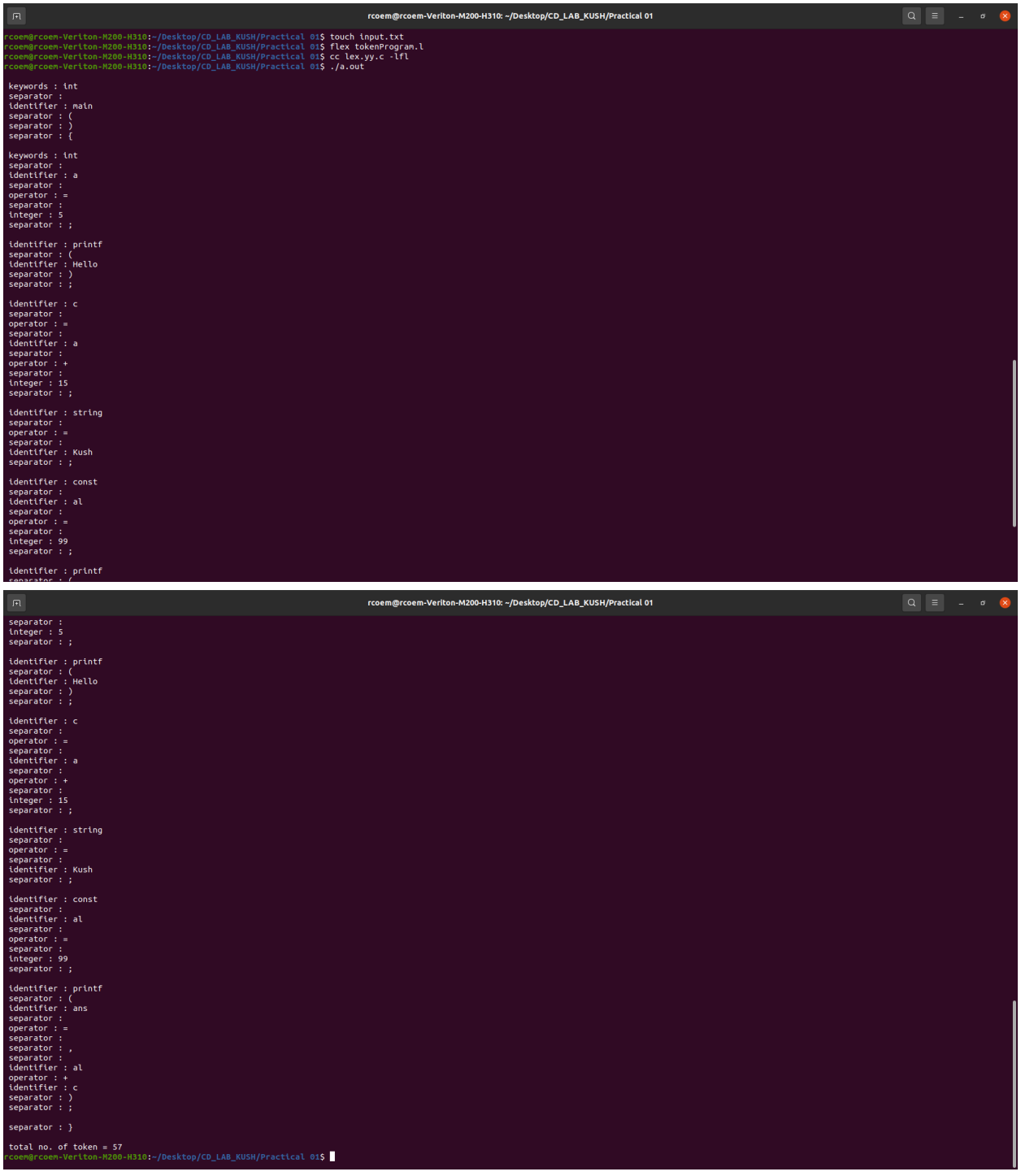
Program:

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

%%
"while"|"if"|"else" {n++;printf("\n keywords : %s", yytext);}
"int"|"float" {n++;printf("\n keywords : %s", yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("\n identifier : %s", yytext);}
"<="|"=="|"="|"++"|"-"|"*"|"+" {n++;printf("\n operator : %s", yytext);}
[(){}|, ;] {n++;printf("\n separator : %s", yytext);}
[0-9]*"."[0-9]+ {n++;printf("\n float : %s", yytext);}
[0-9]+ {n++;printf("\n integer : %s", yytext);}
. ;
%%

int main()
{
    extern FILE* yyin;
       yyin = fopen("input.txt", "r");
       yylex();
    printf("\n total no. of token = %d\n", n);

}
```

Input:

```
int main(){
    int a = 5;
    printf("Hello");
    c = a + 15;
    string = "Kush";
    const al = 99;
    printf("ans = ", al+c);
}
```

Output:


rcoem@rcoem-Veriton-M200-H310: ~/Desktop/CD_LAB_KUSH/Practical 01
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01$ touch input.txt
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01$ flex tokenProgram.l
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01$ cc lex.yy.c -lfl
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01$ ./a.out
keywords : int
separator :
identifier : main
separator : (
separator :)
separator : {
keywords : int
separator :
identifier : a
separator :
operator : =
separator :
integer : 5
separator : ;
identifier : printf
separator : (
identifier : Hello
separator :)
separator : ;
identifier : c
separator :
operator : =
separator :
identifier : a
separator :
operator : +
separator :
integer : 15
separator : ;
identifier : string
separator :
operator : =
separator :
identifier : Kush
separator : ;
identifier : const
separator :
identifier : al
separator :
operator : =
separator :
integer : 99
separator : ;
identifier : printf

rcoem@rcoem-Veriton-M200-H310: ~/Desktop/CD_LAB_KUSH/Practical 01
separator :
integer : 5
separator : ;
identifier : printf
separator : (
identifier : Hello
separator :)
separator : ;
identifier : c
separator :
operator : =
separator :
identifier : a
separator :
operator : +
separator :
integer : 15
separator : ;
identifier : string
separator :
operator : =
separator :
identifier : Kush
separator : ;
identifier : const
separator :
identifier : al
separator :
operator : =
separator :
integer : 99
separator : ;
identifier : printf
separator : (
identifier : ans
separator :
operator : =
separator :
separator : ,
separator :
identifier : al
operator : +
identifier : c
separator :)
separator : ;
separator : }
total no. of token = 57
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01$

Practical No. E2

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, numbers of words, lines, small letters, capital letters, digits, and special characters.

Program:

```
%{
#include<stdio.h>
int lines=0, words=0,s_letters=0,c_letters=0, num=0, spl_char=0,total=0,
questions = 0, sem = 0;
%}
%%
\n { lines++; words++;}
[\t ' '] words++;
[A-Z] c_letters++;
[a-z] s_letters++;
[0-9] num++;
[?] questions++;
"I"|"II"|"III"|"IV"|"V"|"VI"|"VII"|"VIII"|"IX"|"X" {sem++;}
. spl_char++;
%%
int main(void)
{
yyin= fopen("input.txt","r");
yylex();
total=s_letters+c_letters+num+spl_char;
printf(" This File contains ...");
printf("\n\t%d lines", lines);
printf("\n\t%d words",words);
printf("\n\t%d small letters", s_letters);
printf("\n\t%d capital letters",c_letters);
printf("\n\t%d digits", num);
printf("\n\t%d special characters",spl_char);
printf("\n\t%d number of questions",questions);
printf("\n\t%d number of semesters",sem);
printf("\n\tIn total %d characters.\n",total);
}

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

Input:

```
      ABC College

1/1/2000                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 pe implemented to minimize the loss from
construction to
environment?
Question8 : What is air pollution?
```

Output:

```
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01/E2$ flex QuestionPaper.l
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01/E2$ cc lex.yy.c
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01/E2$ ./a.out
 This File contains ...
        12 lines
        113 words
        305 small letters
        23 capital letters
        14 digits
        18 special characters
        8 number of questions
        6 number of semesters
        In total 360 characters.
```

Practical No. E3

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

Program:

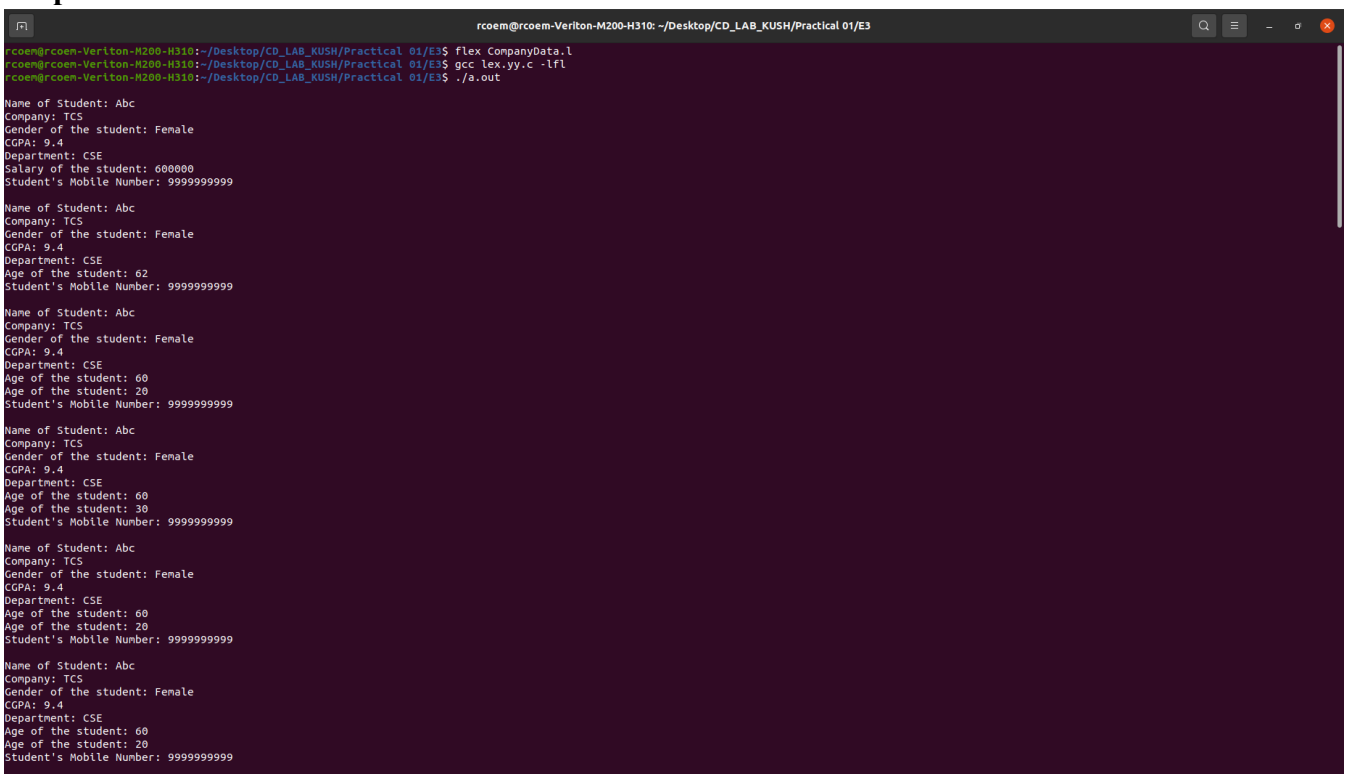
```
%{
int i = 0,count = 0  ;
%}

%%
"TCS"|"Infosys"|"Wipro"|"Accenture"|"Informatica" {i++;printf("Company:
%s\n",yytext);}
[1-9][0-9]{9} {i++;printf("Student's Mobile Number: %s\n",yytext);}
"Female"|"Male"|"female"|"male" {i++;printf("Gender of the student:
%s\n",yytext);}
[0-9]*"."[0-9]+ {i++;printf("CGPA: %s\n", yytext);}
"CSE"|"IT"|"EC" {i++;printf("Department: %s\n",yytext);}
[a-z.0-9]+@[a-z]+(".com"|".in") {i++;printf("Email ID: %s\n",yytext);}
[A-Z]* {i++;printf("College: %s\n",yytext);}
[A-Z]+[a-z]* {i++;printf("\nName of Student: %s\n",yytext);}
[1-9][0-9] {i++;printf("Age of the student: %s\n",yytext);}
[1-9](0000|00000) {i++;printf("Salary of the student: %s\n",yytext);}
[\n] {count++;}
. {;}
%%

int main()
{
    extern FILE* yyin;
       yyin = fopen("input.txt", "r");
       yylex();
    printf("\n total no. of token = %d\n", i);
}
```

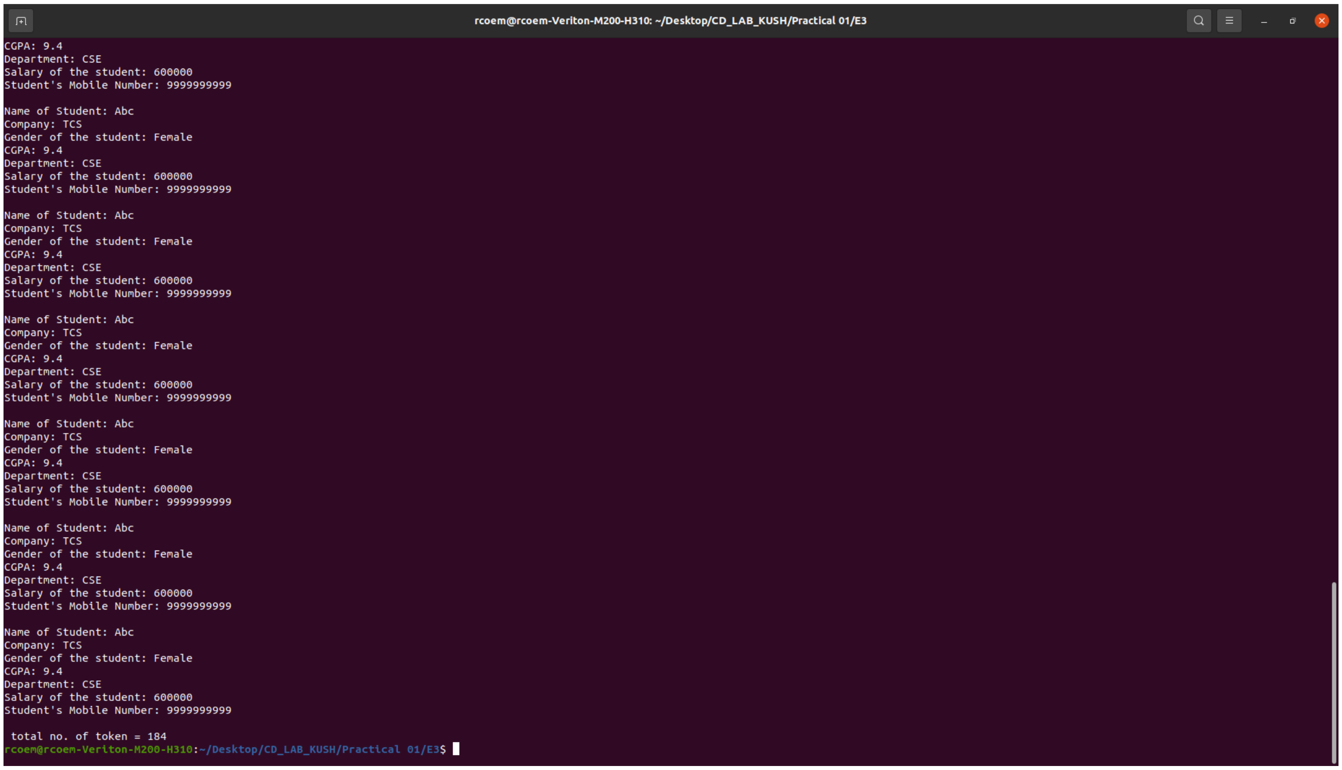
Input:

```
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 620000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 602000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 603000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600200 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600020 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600002 abc@rknec.edu 9999999999
```

```
Abc TCS Female 9.4 CSE 600003 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600300 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 603000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 630000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 640000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 604000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600400 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600040 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600004 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
Abc TCS Female 9.4 CSE 600000 abc@rknec.edu 9999999999
```

Output:




rcoem@rcoem-Veriton-M200-H310: ~/Desktop/CD_LAB_KUSH/Practical 01/E3
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
Name of Student: Abc
Company: TCS
Gender of the student: Female
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
Name of Student: Abc
Company: TCS
Gender of the student: Female
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
Name of Student: Abc
Company: TCS
Gender of the student: Female
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
Name of Student: Abc
Company: TCS
Gender of the student: Female
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
Name of Student: Abc
Company: TCS
Gender of the student: Female
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
Name of Student: Abc
Company: TCS
Gender of the student: Female
CGPA: 9.4
Department: CSE
Salary of the student: 600000
Student's Mobile Number: 9999999999
total no. of token = 184
rcoem@rcoem-Veriton-M200-H310:~/Desktop/CD_LAB_KUSH/Practical 01/E3$