

Name: Himani Dighorikar

Roll No.: 03

Practical No. 1

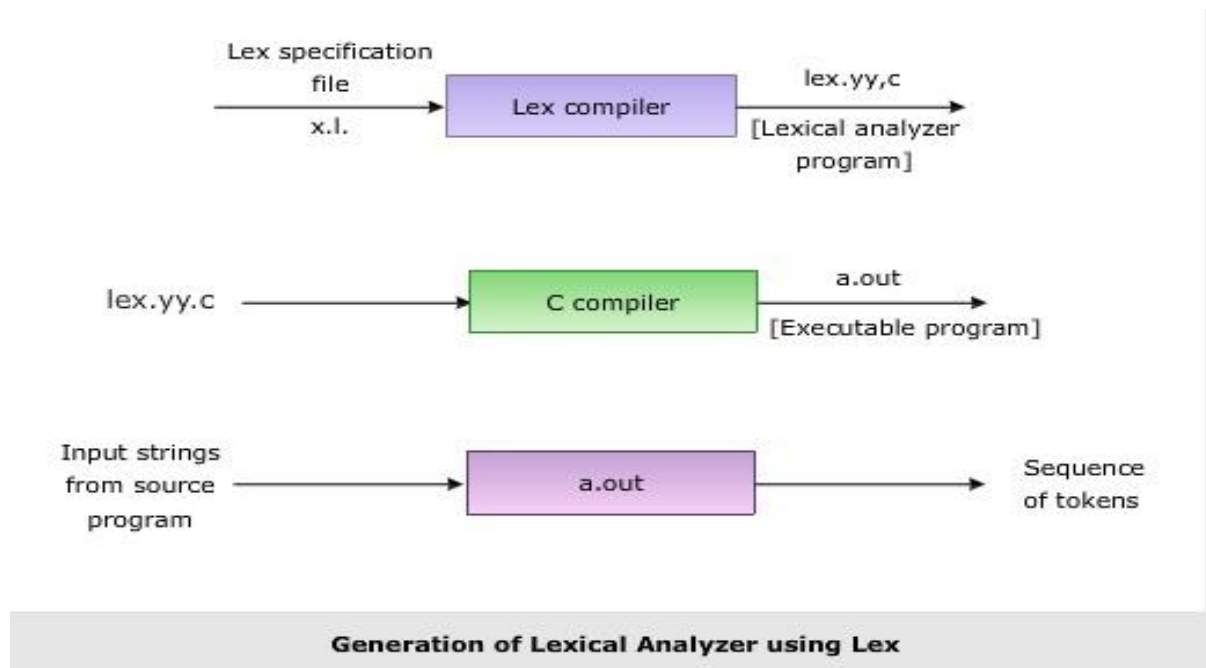
Theory

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



Compiler Design Lab Practical 1: 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 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

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. The default yywrap() always returns 1

3. Use of yylex

Yylex function yytext is of type char* and it contains the lexeme currently found. A lexeme is a sequence of characters in the input stream that matches some pattern in the Rules Section. (In fact, it is the first matching sequence in the input from the position pointed to by yyin.) Each invocation of the function yylex() results in yytext carrying a pointer to the lexeme found in the input stream by yylex(). The value of yytext will be overwritten after the next yylex() invocation.

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

LEX Generates lex.yy.c which defines a routine yylex(), The lex command reads File or standard input, generates a C language program, and writes it to a file named lex.yy.c. This file, lex.yy.c, is a compilable C language program The lex command uses rules and actions contained in File to generate a program, lex.yy.c, which can be compiled with the cc command. The compiled lex.yy.c can then receive input, break the input into the logical pieces defined by the rules in File, and run program fragments contained in the actions in File. The generated program is a C language function called yylex. 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

PROGRAMS

//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

```
% {
#include<stdio.h>
#include<string.h>
int keywords=0,identifiers=0,operators=0,symbols=0, strs=0,digit=0;
char temp[]="";
% }

%%

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

while|for|if|else|void  { keywords++; printf("%sAAA ",yytext);}

[a-zA-Z_][a-zA-Z0-9_]*    { identifiers++; printf("%sIII ",yytext);}

", "|" ;"|"("|")"        { symbols++;printf("%s ",yytext);}

[a-zA-Z/][a-zA-Z_./-]*    { strs++;printf("%s ",yytext);}

%%

int main(){

    yylex();

    printf("\nKeywords : %d",keywords);

    printf("\nIdentifiers : %d",identifiers);

    printf("\nSymbols: %d",symbols);

    printf("\nStrings : %d",strs);

    return 0;

}
```

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

OUTPUT

```
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> gcc lex.yy.c
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
while(a<10)
whileAAA ( aIII <12 )
for ( i=0; i<10;i++)
forAAA ( iIII =2 ; iIII <12 ; iIII ++)
```

/*E2: Write a LEX specification to take the contents from a file while adding 3 to number check if it is divisible by 7 and adding 4 to number check if it is divisible by 2*/

Input file : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

CODE

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

int i;
%}

%%

[0-9]+ {      i=atoi(yytext);
              if(i%7==0)
                  printf("%d",atoi(yytext)+3);
              else if(i%2==0)
                  printf("%d", atoi(yytext)+4);
```

```

        else

        printf("%s", yytext);

}

```

```

%%

```

```

int main(){

    yyin=fopen("E2Input.txt","r");

    yylex();

}

```

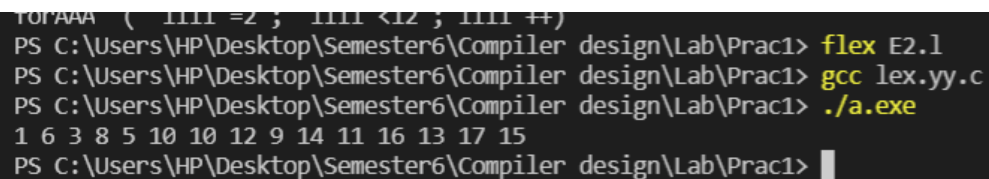
```

int yywrap(){

    return (1);

}

```



```

PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> flex E2.l
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> gcc lex.yy.c
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
1 6 3 8 5 10 10 12 9 14 11 16 13 17 15
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1>

```

/*E3: Write a lex specification to display the histograms of length of words*/

```

%{

#include<stdio.h>

#include<string.h>

```

```
int i;  
int wrdcount=0;  
%}  
  
%%  
[a-zA-Z]+ {  
    printf("%s : %d", yytext, yyleng); wrdcount++;  
}
```

```
%%
```

```
int main(){  
    yyin=fopen("E3Input.txt","r");  
    yylex();  
    printf("Total words : %d",wrdcount);  
  
}  
  
int yywrap(){  
    return (1);  
}
```



```

PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> flex E3.1
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> gcc lex.yy.c
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
Ramdeobaba : 10 College : 7 of : 2 Engineering : 11Total words : 4
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1>

```

/* 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.**

***/**

% {

#include<stdio.h>

#include<string.h>

int spchr=0,comm=0;

% }

%%

[a-z] {printf("%c",yytext[0]);}

[A-Z] {printf("%c", yytext[0]+32);}

"/".*"\n" {comm+=1; printf("Comment %d : %s",comm,yytext);}

"/".*["\n"].*"/" {comm+=1; printf("Comment %d : %s",comm,yytext);}

[&,\$/?~_!=!@#] {spchr++;}

%%

```

int main(){

    yyin=fopen("E4_Input.txt","r");

    yylex();

    printf("\nTotal comments : %d",comm);

    printf("\nSpecial characters : %d\n",spchr);

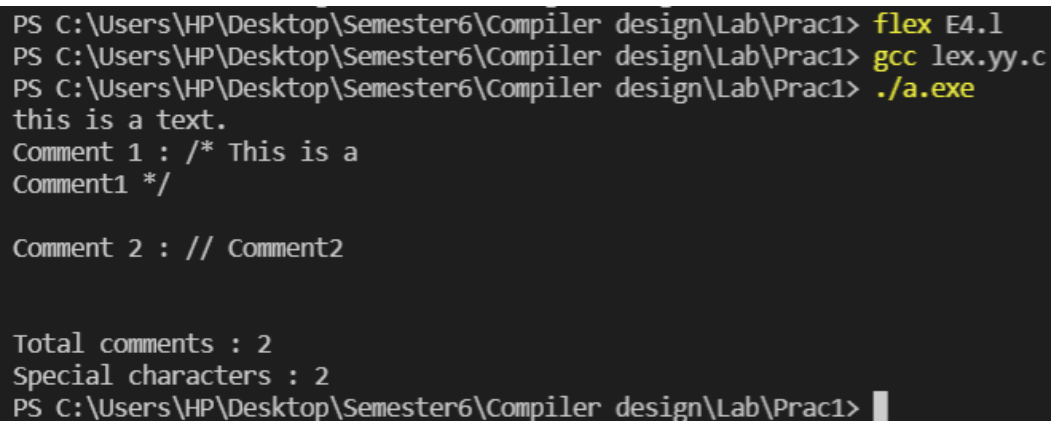
}

int yywrap(){

    return (1);

}

```



```

PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> flex E4.1
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> gcc lex.yy.c
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
this is a text.
Comment 1 : /* This is a
Comment1 */

Comment 2 : // Comment2

Total comments : 2
Special characters : 2
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1>

```

/*E5: Translate an HTML file with some HTML tags to text file using lex.

Consider input from file. 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.

***/**

%{

```
#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<math.h>

int p=0,ul=0,br=0,html=0,head=0,body=0,li=0,h1=0,ol=0;
```

```
% }
```

```
% %
```

```
"<[^>]*">"
```

```
[^<]* {printf("%s",yytext);}
```

```
% %
```

```
int main(){

    yyin=fopen("E5Input.html","r");

    yylex();


    // printf("\n\np tag : %d\n",p);

    // printf("ul tag : %d\n",ul);

    // printf("ol tag : %d\n",ol);

    // printf("br tag : %d\n",br);

    // printf("li tag : %d\n",li);

    // printf("html tag : %d\n",html);

    // printf("head tag : %d\n",head);
```

```
        // printf("body tag : %d\n",body);

    }

    int yywrap(){

        return (1);

    }
```

INPUT FILE

```
<html>
  <head>
    <h1>Shppoing Store</h1>
  </head>
  <body>
    <p>This is a paragraph! </p>
    <br>
    <ol>Shopping list
    <li>Bread</li>
    <li>Butter</li>
    <li>Magnets</li>
    <li>Paint Brush</li>
    </ol>
    <br>
    <ul>TODO list
    <li>Call customer care</li>
    <li>Collect parcel</li>
    <li>Download Virtual Box</li>
    <li>Draft Tasks</li>
    </ul>
  </body>
</html>
```

```
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> flex E5.1
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> gcc lex.yy.c
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
```

Shppoing Store

This is a paragraph!

Shopping list

Bread

Butter

Magnets

Paint Brush

TODO list

Call customer care

Collect parcel

Download Virtual Box

Draft Tasks

```
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> █
```

/* 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. */

```
% {
```

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

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

```
#include<ctype.h>
```

```
int words=0,lines=0,cap=0,small=0,spchr=0,digits=0,qnum=0,sem=0;
```

```
% }
```

```
%%
```

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

```
[\t ' ' ] { words++;}
```

```
[I+V+X]* {sem++;}
```

```
[A-Z] {cap++;}
```

```
[a-z] {small++;}
```

```
[0-9]* {digits+=yyleng;}
```

```
[0-9\0-9\0-9]+ {printf("date : %s",yytext); digits+=yyleng-2;}
```

```
[:?#$% @ !\.,] { spchr++; if(!strcmp(yytext,"?")){qnum++;}}
```

```
%%
```

```
int main(){
```

```
    yyin=fopen("E6Input.txt","r");
```

```
    yylex();
```

```
    printf("\nNo. of Questions : %d",qnum);
```

```
    printf("\nSemesters : %d",sem);
```

```
    printf("\nTotal words : %d",words);
```

```
    printf("\nTotal lines : %d",lines);
```

```
    printf("\nCapital letters : %d",cap);
```

```
    printf("\nSmall letters : %d",small);
```

```
    printf("\nDigits : %d",digits);
```

```
    printf("\nSpecial characters : %d",spchr);
```

```
        return 0;
    }
    int yywrap(){
    return (1);
    }
```

INPUT

ABC College

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 be implemented to minimize the loss from construction to environment?

Question8 : What is air pollution?

OUTPUT

```

PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
date : 1/1/2000
No. of Questions : 8
Semesters : 8
Total words : 86
Total lines : 13
Capital letters : 25
Small letters : 311
Digits : 14
Special characters : 24
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1>

```

/*

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.

*/

{

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<float.h>

int maleCount=0, femaleCount=0, cse=0, it=0, ece=0, TCS=0, Infosys=0, Wipro=0, Accenture=0, Informatica=0;

% }

%%

```
[7-9][0-9]*    {  
    if(yyval==10){  
        printf("Phone No.: %s\n",yytext);  
    }  
    else{  
        printf("Package: %s\n",yytext);  
    }  
}
```

```
[0-9]+ { fprintf(yyout,"Package: %s\n",yytext);}
```

```
[a-z.0-9_]+@[a-z]+".edu"    { printf("Email Id: %s\n",yytext);}
```

```
[A-Z]+ {
```

```
    if(!strcmp("CSE",yytext) || !strcmp("ECE",yytext) || !strcmp  
("IT",yytext)){
```

```
        printf("Department: %s\n",yytext);
```

```
        if(!strcmp("CSE",yytext)) {cse++;}
```

```
        else if(!strcmp("IT",yytext)) {it++;}
```

```
        else if(!strcmp("ECE",yytext)) {ece++;}
```

```
    }
```

```
if(!strcmp("TCS",yytext)){ TCS++;}
```

```
}
```

```

[A-Za-z]* { if(!strcmp("Female",yytext)){

    fprintf(yyout,"Gender: %s\n",yytext);

    femaleCount+=1;

}

else if(!strcmp("Male",yytext)){

    fprintf(yyout,"Gender: %s\n",yytext);

    maleCount+=1;

}

else if(!strcmp("Accenture",yytext)|| !strcmp("TCS",yytext) ||
!strcmp("Infosys",yytext) || !strcmp("Informatica",yytext) || !strcmp("Wipro",yytext)){

    printf("Company : %s\n",yytext);

    if(!strcmp("TCS",yytext))          { TCS++;}

    else if(!strcmp("Accenture",yytext))  { Accenture++; }

    else if(!strcmp("Wipro",yytext))      { Wipro++;}

    else if(!strcmp("Informatica",yytext))  { Informatica++;}

    else if(!strcmp("Infosys",yytext))    { Infosys++;}

}

else{

    printf("\n\nName of the Student: %s\n",yytext);

}

}

```

```

[0-9]*[.][0-9]+ {

```

```

        if(atoi(yytext)<=10.0){
            printf("CGPA: %s\n",yytext);
        }
        else{
            printf("[Invalid] CGPA: %s\n",yytext);
        }
    }
}

```

%%

```

int main(){
    yyin=fopen("E7Input.txt","r");
    yylex();
    printf("\n-----\n");
    printf("No of males : %d\n",maleCount);
    printf("No of females : %d\n",femaleCount);
    printf("-----\n");
    printf("No. of CSE students placed : %d\n",cse);
    printf("No. of IT students placed : %d\n",it);
    printf("No. of ECE students placed : %d\n",ece);
    printf("-----\n");
    printf("No. of students placed in Accenture : %d\n",Accenture);
}

```

```

printf("No. of students placed in TCS : %d\n",TCS);
printf("No. of students placed in Infosys : %d\n",Infosys);
printf("No. of students placed in Wipro : %d\n",Wipro);
printf("No. of students placed in Informatica : %d\n",Informatica);

return 0;
}

int yywrap(){
return (1);
}

```

//Name gender dept email phno cgpa company package

INPUT

Khushboo TCS Female 9.4 CSE 600000 khuranak1@rk nec.edu 9999999999

Tarun Accenture Male 9.1 ECE 500000 tarun@rk nec.edu 987123451

Sonal TCS Female 9.5 IT 3500000 sonal@rk nec.edu

Ayush Infosys Male 9.6 CSE 700000 ayush@rk nec.edu

Aman Wipro Male 9.23 ECE 590000 aman@rk nec.edu

Baman Informatica Male 9.34 IT 345000 baman@rk nec.edu

Riana Wipro Female 9.22 456000 CSE riana@rk nec.edu

Riya Accenture Female 9.0 509000 IT riya@rk nec.edu

Sneha Infosys Female 9.34 400000 ECE snehal@rk nec.edu

Fiya Accenture Female 9.89 800000 CSE fiya@rk nec.edu

OUTPUT

```
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> gcc lex.yy.c
PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> ./a.exe
```

Name of the Student: Khushboo
Gender: Female
CGPA: 9.4
Department: CSE
Package: 600000
Email Id: khuranak1@rknec.edu
Phone No.: 9999999999

Name of the Student: Tarun
Company : Accenture
Gender: Male
CGPA: 9.1
Department: ECE
Package: 500000
Email Id: tarun@rknec.edu
Package: 987123451

Name of the Student: Sonal
Gender: Female
CGPA: 9.5
Department: IT
Package: 3500000
Email Id: sonal@rknec.edu

Name of the Student: Ayush
Company : Infosys
Gender: Male
CGPA: 9.6
Department: CSE
Package: 700000
Email Id: ayush@rknec.edu

Name of the Student: Aman
Company : Wipro
Gender: Male
CGPA: 9.23
Department: ECE
Package: 590000
Email Id: aman@rknec.edu

Name of the Student: Baman
Company : Informatica
Gender: Male
CGPA: 9.34
Department: IT
Package: 345000
Email Id: baman@rknec.edu

Name of the Student: Riana
Company : Wipro
Gender: Female
CGPA: 9.22
Package: 456000
Department: CSE
Email Id: riana@rknec.edu

Name of the Student: Riya
Company : Accenture
Gender: Female
CGPA: 9.0
Package: 509000
Department: IT
Email Id: riya@rknec.edu

Name of the Student: Sneha
Company : Infosys
Gender: Female
CGPA: 9.34

Name of the Student: Sneha
Company : Infosys
Gender: Female
CGPA: 9.34
Package: 400000
Department: ECE
Email Id: snehal@rknec.edu

Name of the Student: Fiya
Company : Accenture
Gender: Female
CGPA: 9.89
Package: 800000
Department: CSE
Email Id: fiya@rknec.edu

No of males : 4

No of females : 6

No. of CSE students placed : 4

No. of IT students placed : 3

No. of ECE students placed : 3

No. of students placed in Accenture : 3

No. of students placed in TCS : 2

No. of students placed in Infosys : 2

No. of students placed in Wipro : 2

No. of students placed in Informatica : 1

PS C:\Users\HP\Desktop\Semester6\Compiler design\Lab\Prac1> █