**Name: Yash Patni**

**Roll No.: 71**

**Practical No. 1**

**Theory**

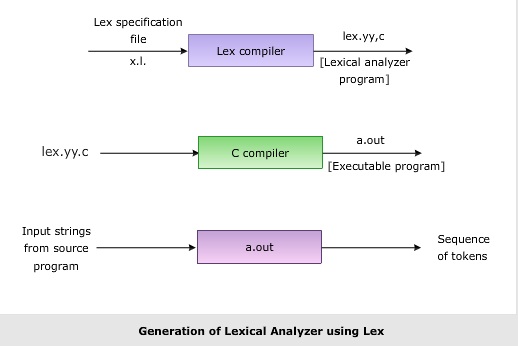
**LEX:**

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**



**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. **Why is –ll option used for running lex.yy.c**

* The -ll option is used for running lex.yy.c to link the Lex-generated lexical analyzer with the Lex library. It includes the Lex library routines needed for the functioning of the lexical analyzer.

1. **Use of yywrap**

* yywrap is a function in Lex programs that allows customization when the end of input occurs. It returns a value (typically 1) indicating that there is no more input. Programmers can define yywrap to handle end-of-input conditions according to their specific needs.

1. **Internal representation of** Lex

* The internal representation of Lex involves defining regular expressions and associated actions within the Lex source file. Lex translates these specifications into a generated program, often named yylex, which recognizes the specified regular expressions in an input stream and executes the corresponding actions. The internal representation essentially comprises the user-defined rules and actions specified in the Lex source file.

**Practicals**

**Aim (E1):**Design a lexical analyzer to identify the tokens such as keywords, identifiers, operators, constants (Int, float &amp; character), special symbols and strings for C language using LEX. UseFile for the input.

**Program:**

%{

#include<stdio.h>

%}

%%

"auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double"|"else"|"enum"|"extern"|"float"|"for"|"goto"|"if"|"int"|"long"|"register"|"return"|"short"|"signed"|"sizeof"|"static"|"struct"|"switch"|"typedef"|"union"|"unsigned"|"void"|"volatile"|"while" {

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

}

\{|\}|\(|\)|\;|\:|\,|\[|\] {printf("Special Symbol: %s\n", yytext);}

[a-zA-Z][\_a-zA-Z0-9]\* {

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

}

[-+\*/%&|^=!<>]=?|<<|>>|\&\&|\|\||\+\+|--|->|\. {

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

}

[-+]?[0-9]+ {

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

}

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

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

}

'.' {

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

}

\".\*\" {

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

}

. {}

%%

int main(void) {

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

yylex();

fclose(yyin);

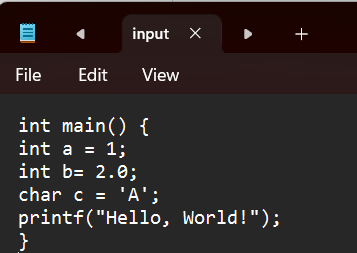
return 0;

}

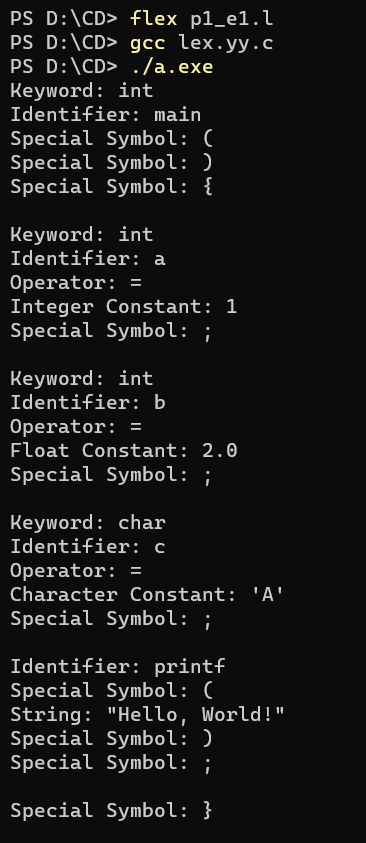
int yywrap(void) {

return (1);

}

****

**Output:**

****

**Aim (E2):**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.

Define following data in a text file.

Input Text file:

ABC College

1/1/2000 Sem: III

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?

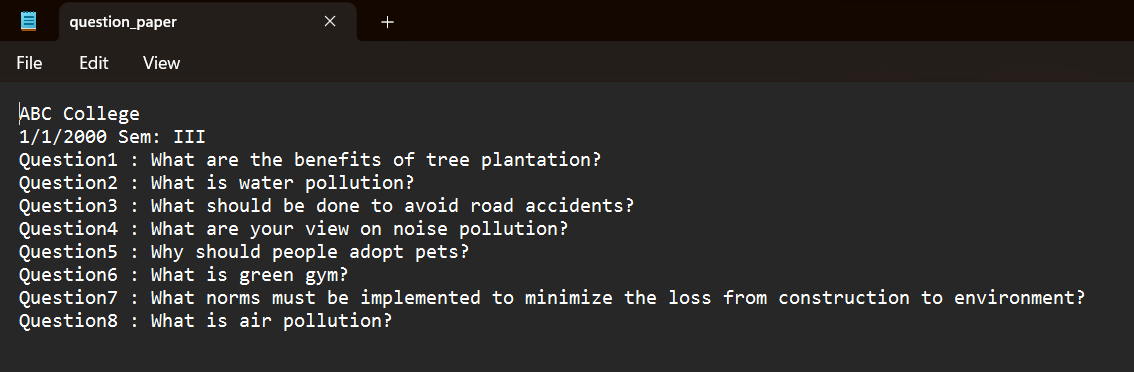
Question5 : Why should people adopt pets?

Question6 : What is green gym?

Question7 : What norms must pe implemented to minimize the loss from construction to

environment?

Question8 : What is air pollution?



**Program:**

%{

#include <stdio.h>

int dc = 0;

int sc = 0;

int qc = 0;

int wc = 0;

int lc = 0;

int slc = 0;

int clc = 0;

int digitc = 0;

int spl\_char = 0;

%}

%%

\n {lc++; wc++;}

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

printf("Date of Examination: %s\n", yytext);

printf("Valid date\n");}

"III" { printf("Semester:%s\n", yytext);}

\? {qc++;}

[\t ' '] { wc++;}

[A-Z] {clc++;}

[a-z] {slc++;}

[0-9]\* {digitc++;}

. {spl\_char++;}

%%

int main(void)

{

yyin= fopen("question\_paper.txt","r");

yylex();

printf("Number of Questions: %d\n", qc);

printf("Number of Words: %d\n", wc);

printf("Number of Lines: %d\n", lc);

printf("Number of Small Letters: %d\n", slc);

printf("Number of Capital Letters: %d\n", clc);

printf("Number of Digits: %d\n", digitc);

printf("Number of Special Characters: %d\n", spl\_char);

}

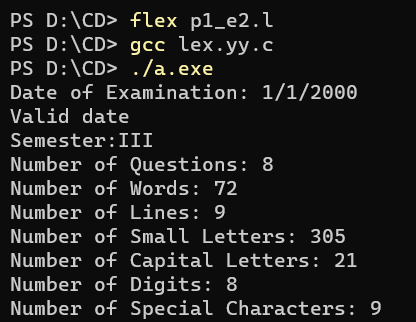
int yywrap(void)

{

return(1);

}

**Output:**

****

**Aim (E3):**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 10 records must be present.

Write a Lex program to find the parameters given below:

o Identify Name of student and display it.

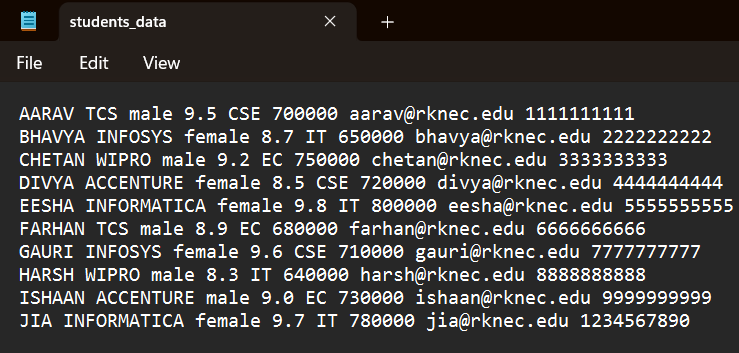
o Identify CGPA and display (should be less than 10)

o Identify Package and display it

o

o Number of male students

o Number of CSE, IT and EC students who are placed



**Program:**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int tcs = 0;

int wip = 0;

int acc = 0;

int inf = 0;

int isys = 0;

int fc = 0;

int mc = 0;

int csec = 0;

int itc = 0;

int ecc = 0;

%}

%%

[A-Z][a-z]+ { printf("Name of Student: %s\n", yytext); }

[0-9]\.[0-9]+ { double cgpa = atof(yytext);

if (cgpa < 10.0) {

printf("CGPA: %.1f\n", cgpa);

} }

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

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

}

[^ \t\n]+@"rknec.edu" { printf("Mail ID: %s\n", yytext); }

[1-9][0-9]+ {

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

}

"TCS" { tcs++; }

"WIPRO" { wip++; }

"ACCENTURE" { acc++; }

"INFOSYS" { isys++; }

"INFORMATICA" { inf++; }

"female" { fc++; }

"male" { mc++; }

"CSE" { csec++; }

"IT" { itc++; }

"EC" { ecc++; }

. { }

%%

int main(void) {

yyin = fopen("students\_data.txt", "r");

yylex();

printf("Number of Students Placed in TCS: %d\n", tcs);

printf("Number of Students Placed in Wipro: %d\n", wip);

printf("Number of Students Placed in Accenture: %d\n", acc);

printf("Number of Students Placed in Infosys: %d\n", isys);

printf("Number of Students Placed in Informatica: %d\n", inf);

printf("Number of Female Students: %d\n", fc);

printf("Number of Male Students: %d\n", mc);

printf("Number of CSE Students: %d\n", csec);

printf("Number of IT Students: %d\n", itc);

printf("Number of EC Students: %d\n", ecc);

fclose(yyin);

return 0;

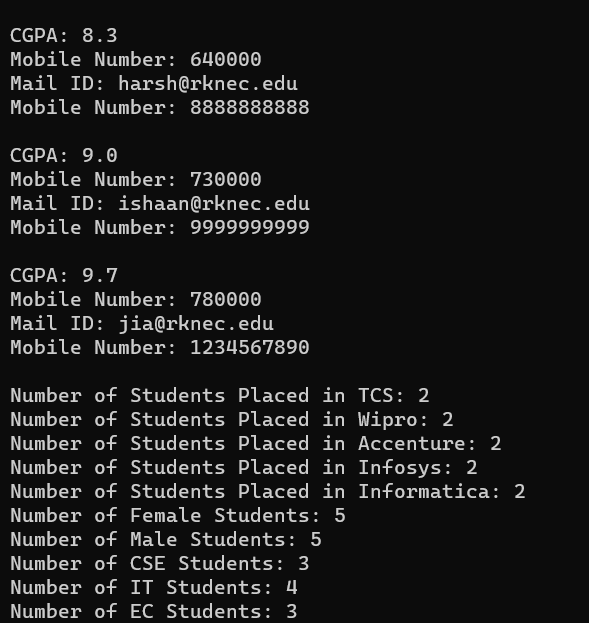
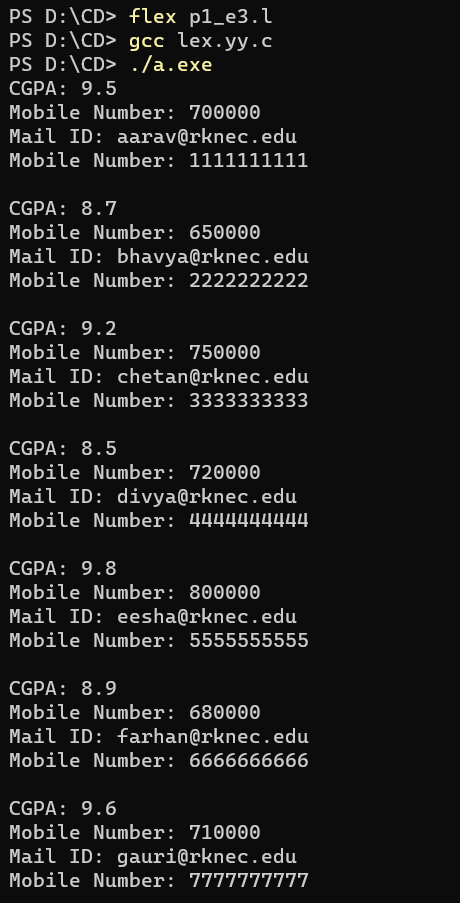
}

int yywrap(void) {

return(1);

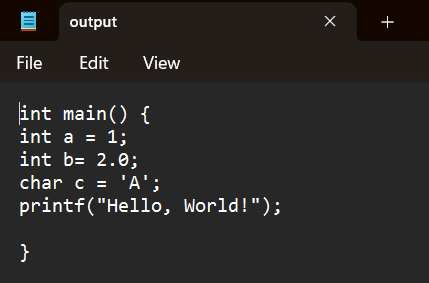
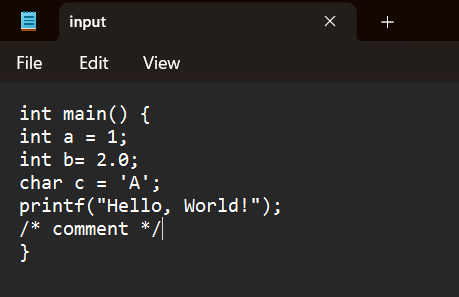
}

**Output:**

****

**Aim (E4):**Write a Lex Program which takes C program from file &amp; write the sane C program in

another file after removing the comments.



**Program:**

%{

#include<stdio.h>

%}

%%

\/\/(.\*) { };

\/\\*(.\*\n)\*.\*\\*\/ { };

%%

int main()

{

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

yyout=fopen("output.txt","w");

yylex();

return 0;

}

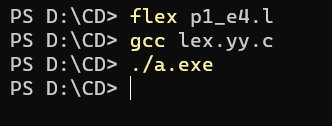
int yywrap()

{

return 1;

}

**Output:**

****