Name : Janak Mandavgade
Roll No. : 43

Practical No. 1

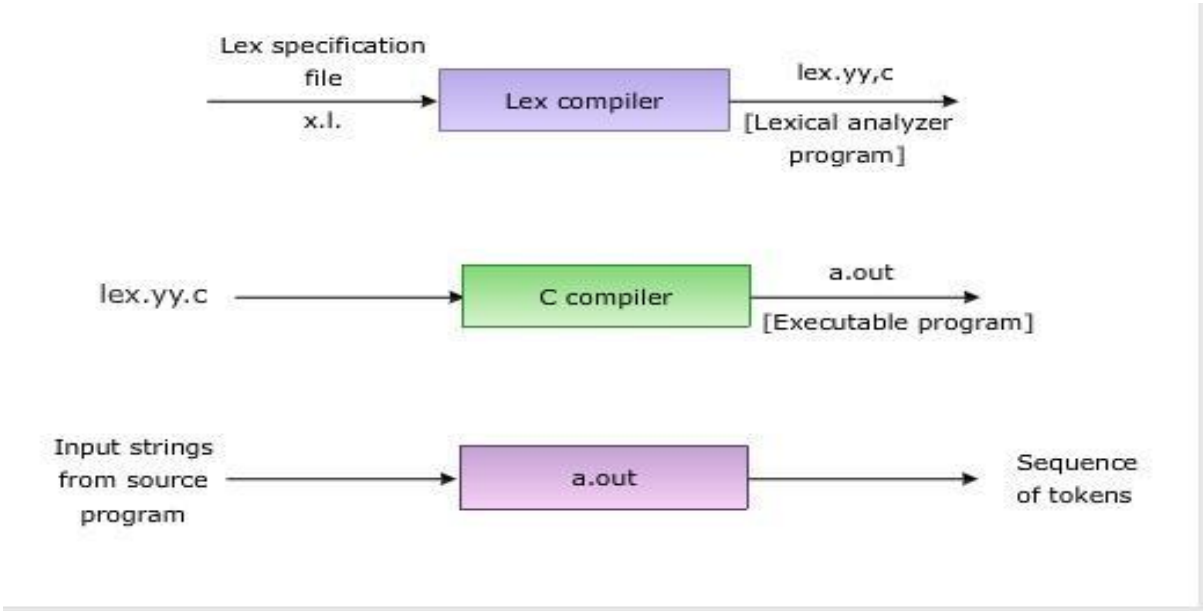
Theory

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

   A lex library routine that you can redefine is yywrap() , which is called whenever the scanner reaches the end of file. If yywrap() returns 1, the scanner continues with normal wrapup on the end of input.
   Function yywrap is called by lex when input is exhausted. Return 1 if you are done or 0 if more processing is required.

2. Use of yylex function
   yylex() returns a value indicating the type of token that has been obtained. If the token has an actual value, this value (or some representation of the value, for example, a pointer to a string containing the value) is returned in an external variable named yylval.

3. What does lex.yy.c. do?
   The table is translated to a C program (lex.yy.c) which reads an input stream partitioning the input into strings which match the given expressions and copying it to an output stream if necessary

---

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:

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

int statements=0,ids=0,assign=0,rel=0,keywords=0,integers=0;
%}
DIGIT [0-9]
LETTER [A-Za-Z]
TYPE int|char|bool|float|void|for|do|while|if|else|return|void
%option yylineno
%option noyywrap
%%
```

```
\n {statements++;}
{TYPE} {printf("\t %s is a keyword\n",yytext);keywords++;}
(<|>|<=|>=|==) {printf("\t %s is a special character\n",yytext);rel++;}
'#'/[a-zA-Z0-9]*    {;}
[a-zA-Z]+[a-zA-Z0-9]* {printf("\t %s is an identifier\n",yytext);ids++;}
= {assign++;}
\".*\"     printf("%s is a string\n", yytext);
[0-9]+ {printf("\t %s is an integer\n",yytext);integers++;}
.      {;}

%%

main(void)
{
yyin= fopen("c1.txt","r");
yylex();


printf("statements = %d ids = %d assign = %d rel = %d keywords = %d integers = %d
\n",statements,ids,assign,rel,keywords,integers);
}
```

Input:

```
#include<stdio.h>
void main(){
   float a123;
   char a;
   char b123;
   char c;
   int ab[5];
   int bc[2];
   int ca[7];
   int ds[4];
   for( a = 0; a < 5 ;a++)
    printf("%d ", a);
   return 0;
}
```

Output:

```
include is an identifier
```

```
        < is a special character
        stdio is an identifier
        h is an identifier
        > is a special character
        void is a keyword
        main is an identifier
        float is a keyword
        a123 is an identifier
        char is a keyword
        a is an identifier
        char is a keyword
        b123 is an identifier
        char is a keyword
        c is an identifier
        int is a keyword
        ab is an identifier
        5 is an integer
        int is a keyword
        bc is an identifier
        2 is an integer
        int is a keyword
        ca is an identifier
        7 is an integer
        int is a keyword
        ds is an identifier
        4 is an integer
        for is a keyword
        a is an identifier
        0 is an integer
        a is an identifier
        < is a special character
        5 is an integer
        a is an identifier
        printf is an identifier
"%d " is a string
        a is an identifier
        return is a keyword
        0 is an integer
statements = 13 ids = 17 assign = 1 rel = 3 keywords = 11 integers = 7
```

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<stdlib.h>
#include<string.h>
int
Qcount=0,numLines=0,sLetter=0,cLetter=0,digit=0,specChar=0,numWord=0,date=0,day=0,mon
th=0,year=0;
char word[]= "Question";
%}

%option noyywrap

%%
\n  {numLines++,numWord++;}
[0-9]  {digit++;}
^(0?[1-9]|[1][0-9]|3[01])[- /.]([1-9]|0[1-9]|1[012])[- /.](19|20)\d\d$  {printf("Date of examination:
%s",yytext);}
(I|II|III|IV|V|VI|VII|VIII)  {printf("Semester: %s",yytext);}
[$&+,:;=?@#|'<>.^*()%!-] {specChar++;}
[a-z]  {sLetter++;}
[A-Z]  {cLetter++;}
[a-zA-Z]+    { if(strcmp(yytext, word)==0)
                Qcount++; }

[\t ' ']  {numWord++;}
.  {;}
%%

int main(void)
{
extern FILE *yyin;
yyin= fopen("question.txt","r");
yylex();
printf("\nNumber of lines: %d\nNumber of questions: %d\nNumber of small letters:
%d\n",numLines,Qcount,sLetter);
printf("Number of capital letters: %d\nNumber of digits: %d\nNumber of special characters:
%d\n",cLetter,digit,specChar);
printf("Number of words: %d\n",numWord);
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:

Semester: VI
Number of lines: 10
Number of questions: 8
Number of small letters: 0
Number of capital letters: 0
Number of digits: 15
Number of special characters: 20
Number of words: 86

Practical No. E3

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:
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:

```
%{
#include<stdio.h>
int i=0;
%}

%option noyywrap

%%
"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);}
. {;}
%%

int main(void){
yyin=fopen("trial.txt","r");
yylex();
printf("---------------------------------\n");
printf("Number of tokens: %d\n",i);
```

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

Input:

```
Harsh Wipro male 8.11 CSE 600000 harshs@gmail.com 9284871465 42 RCOEM
Rajesh TCS male 9.89 CSE 2000 razzc@gmail.com 9284871465 52 IIMb
Gunjan Wipro male 8.89 CSE 600000 gxu@gmail.com 9284871465 57 RCOEM
```

Output:

```
Name of Student: Harsh
Company: Wipro
Gender of the student: male
CGPA: 8.11
Department: CSE
Salary of the student: 600000
Email ID: harshs@gmail.com
Student's Mobile Number: 9284871465
Age of the student: 42
College: RCOEM

Name of Student: Rajesh
Company: TCS
Gender of the student: male
CGPA: 9.89
Department: CSE
Age of the student: 20
Email ID: razzc@gmail.com
Student's Mobile Number: 9284871465
Age of the student: 52

Name of Student: IIMb

Name of Student: Gunjan
Company: Wipro
Gender of the student: male
CGPA: 8.89
Department: CSE
Salary of the student: 600000
Email ID: gxu@gmail.com
Student's Mobile Number: 9284871465
Age of the student: 57
```

College: RCOEM

Number of tokens: 30