Name: Shantanu Mane
Roll No.: E63

Practical No. 1

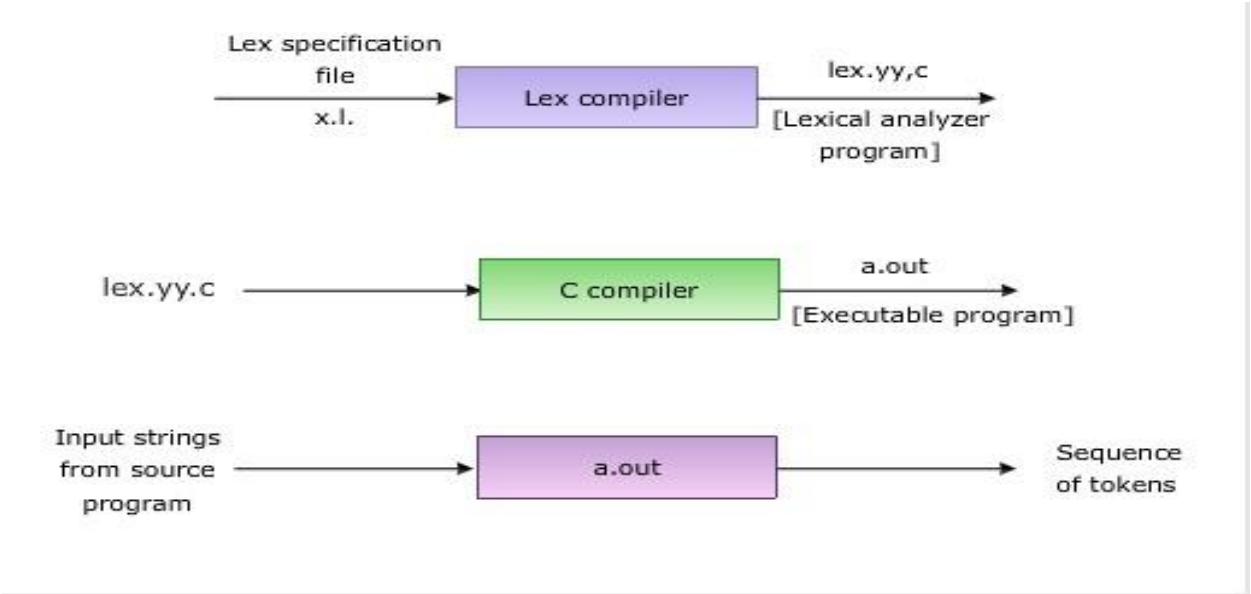
Theory

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
2. Use of yylex function
3. What does lex.yy.c. do?

Practical No. 1

Aim: Design a lexical analyzer to identify the tokens such as keywords, identifiers, operators, symbols and strings for C language.

Program:

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

%%
int | float | printf {n++; printf("%s is a keyword!\n", yytext);}
[a-zA-Z][a-zA-Z0-9]* {n++; printf("%s is an identifier!\n", yytext);}
[(|)|{|}|,|;|#|&] {n++; printf("%s is a symbol!\n", yytext);}
[=|++|-|*|+] {n++; printf("%s is an operator!\n", yytext);}
\".*\" {n++; printf("%s is a string!\n", yytext);}
\<.*\> {n++; printf("%s is a preprocessor directive!\n", yytext);}
%%

int main(){
    yyin = fopen("c-file.txt", "r");
    yylex();

    printf("\nTotal no. of tokens = %d\n", n);
    return 0;
}

int yywrap(void){}
```

Input:

```
#include <stdio.h>
void main(){
    int a, b, c;
    c = a + b;
    printf("%d", c);
}
```

Output:

$SHANTANU > CountTokens > ♥ 10:30 > flex .\CountTokens.l
$SHANTANU > CountTokens > ♥ 10:31 > gcc .\lex.yy.c
$SHANTANU > CountTokens > ♥ 10:31 > .\a.exe
is a symbol!
include is an identifier!
 <stdio.h> is a preprocessor directive!

void is an identifier!
 main is an identifier!
(is a symbol!
) is a symbol!
{ is a symbol!

    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!

    c is an identifier!
 = is an operator!
 a is an identifier!
 + is an operator!
 b is an identifier!
; is a symbol!

    printf is an identifier!
(is a symbol!
"%d" is a string!
, is a symbol!
 c is an identifier!
) is a symbol!
; is a symbol!

} is a symbol!

Total no. of tokens = 29

Practical No. 2

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 words = 0;
    int lines = 0;
    int digits = 0;
    int questions = 0;
    int specialChar = 0;
    int smallChar = 0;
    int capitalChar = 0;
%}

%%
[0-9]"/"[0-9]"/"[2000-2100]+ {printf("Date : %s\n", yytext);}
["\n"] {lines++;}
[I|II|III|IV|VII|VIII|IX|X]+ {printf("Semesters : %s\n", yytext);}
[0-9]+ {digits++;}
[!@#$%^&*()?] {specialChar++;}
[a-z] {smallChar++;}
[A-Z] {capitalChar++;}
[" "] {words++;}▩₹₪▰℃
"Question"[1-9]+ {questions++;}
%%

int yywrap(void){

}

int main(){
    yyin = fopen("questions.txt", "r");
    yylex();

    printf("\nNum Digits : %d\n", digits);
    printf("\nNum Letters : %d\n", smallChar);
    printf("\nNum Capital Letters : %d\n", capitalChar);
    printf("\nNum Words : %d\n", words);
    printf("\nNum Lines : %d\n", lines);
    printf("\nNum Questions : %d\n", questions);

    return 0;
}
```

Input:

ABC College

Date: 8/9/2022 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:

```
$SHANTANU > CountParams > ♥ 10:39 > flex .\CountParams.l
$SHANTANU > CountParams > ♥ 10:39 > gcc .\lex.yy.c
$SHANTANU > CountParams > ♥ 10:40 > .\a.exe
:Date : 8/9/2022
:Semesters : I
,Semesters : II
,Semesters : III
,Semesters : IV
,Semesters : V
,Semesters : VI
,Semesters : VII
,Semesters : VIII
::::::::
Num Digits : 0

Num Letters : 252

Num Capital Letters : 14

Num Words : 70

Num Lines : 11

Num Questions : 8
```