Subject: System Programming and Operating System

Class: CSE Sem IV Div I & II

EXP 3

Title: Lexical Analysis

1) **Implement Lexical Analyzer for simple arithmetic equation to separate identifiers and operators.**
2) **Use of FLEX Tool: Write a program using Lex Specifications to implement lexical analysis phase of compiler to find whether given letter is a vowel or not.**
3) **Use of FLEX Tool: Lex Program to count numbers of lines, words, spaces and characters.**

Theory:

I) Lexical Analysis

Lexical analysis is the first phase of a compiler, responsible for converting a sequence of characters from the source code into a sequence of tokens. It simplifies the parsing process by categorizing the input stream into manageable chunks. Here's a deep dive into the theory behind lexical analysis:

Key Concepts

1. Token: A token is a sequence of characters that represent a unit of meaning in the source code, such as keywords, identifiers, operators, and literals. Each token has a type and a value.
    - Examples: int (keyword), x (identifier), + (operator), 42 (literal).
2. Lexeme: A lexeme is the actual sequence of characters that form a token. For example, in the token int, the lexeme is the string "int".
3. Pattern: A pattern is a rule that describes the structure of lexemes. Patterns are often specified using regular expressions.
4. Symbol Table: A data structure used to store information about identifiers, such as variable names and function names, along with their attributes.

II) Use of FLEX Tool

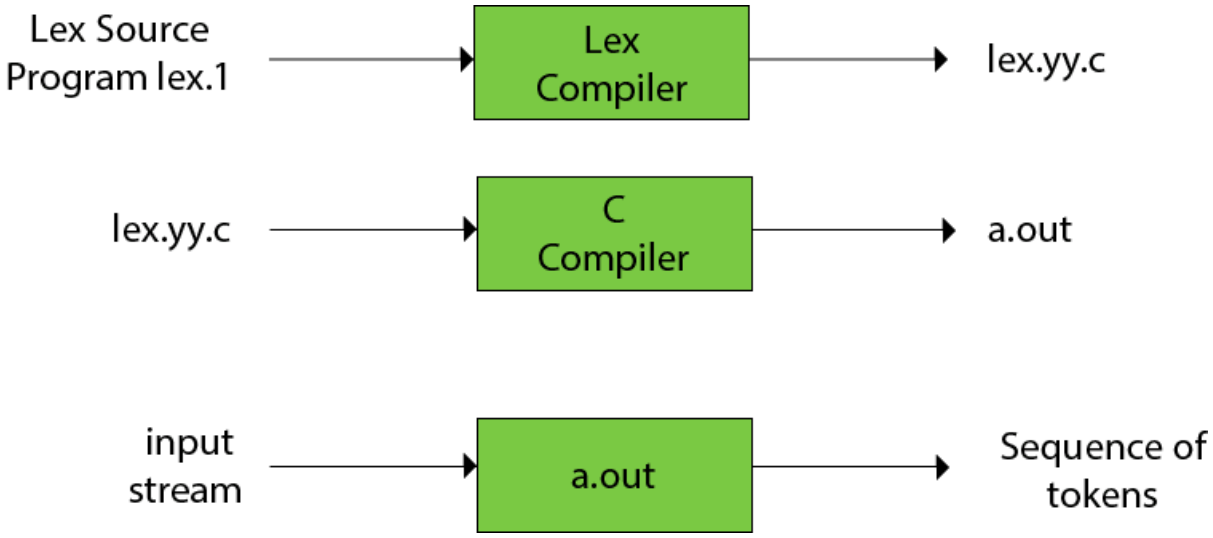
Flex (short for "Fast Lexical Analyzer Generator") is a tool used to generate lexical analyzers, also known as scanners or tokenizers. These analyzers are used in compilers and interpreters to convert a sequence of characters from the source code into a sequence of tokens. Flex is particularly useful for creating the lexical analysis phase of a compiler.

How Flex Works

Flex takes a set of rules defined by the user, written in a specific syntax, and generates a C source file that implements a lexical analyzer. This file contains code to recognize patterns in the input and produce corresponding tokens.

The function of Lex is as follows:

- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.



Lex file format

A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:

1. { definitions }
2. %%
3. { rules }
4. %%
5. { user subroutines }

Definitions include declarations of constant, variable and regular definitions.

Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.

Where **pi** describes the regular expression and **action1** describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

Expression	Description
\x	x, if x is a lex operator
"xy"	xy, even if x or y is a lex operator (except \)
[xy]	x or y
[a-z]	a to z
[^x]	Any character but x
.	Any character but newline
^x	x at the beginning of a line
<y>x	x when lex is in start condition y
x$	x at the end of a line
x?	Optional x
x*	0, 1, 2, ... instances of x
x+	1, 2, 3, ... instances of x
x{m,n}	m through n occurrences of x
xx|yy	Either xx or yy
x |	The action on x is the action for the next rule
(x)	x
x/y	x but only if followed by y
{xx}	The translation of xx from the definitions section

Sample Program:

```
%{
        /*To find whether given letter is a vowel or not*/
#undef yywrap
#define yywrap() 1
        void display(int);
%}
%%
[a|e|i|o|u|]{
                int flag=1;
                display(flag);
                return;
                }
.+ {
        int flag=0;
        display(flag);
        return;
        }
%%
void display(int flag)
{
        if(flag==1)
                printf("The given letter [%s] is a vowel",yytext);
        else
                printf("The given letter [%s] is NOT a vowel",yytext);
}
main()
{       printf("Enter a letter to check if it is a vowel or not");
        yylex();
}
```

//Attach Print of Program and Output

Compilation and Execution:

Create a folder on Desktop

flex_programs or whichever name you like Open notepad type in a flex program Save it inside the folder like filename.l

/*Make sure while saving save it as all files rather than as a text document*/

Goto to Command Prompt(cmd)

Goto the directory where you have saved the program

Type in command :- flex filename.l

Type in command :- gcc lex.yy.c

Execute/Run for windows command promt :- a.exe

Key Flex/Lex Concepts:

- **Regular Expressions:** Essential for defining patterns. Learn the syntax for regular expressions (e.g., +, *, ?, [], (), etc.).
- **Actions:** The C code executed when a pattern is matched.
- **yytext:** A character array containing the matched text.
- **yyleng:** The length of the matched text.
- **yylex():** The function that starts the lexical analysis.
- **Start States:** Used for handling different contexts within the input (more advanced).

Print of Programs and its output

Conclusion