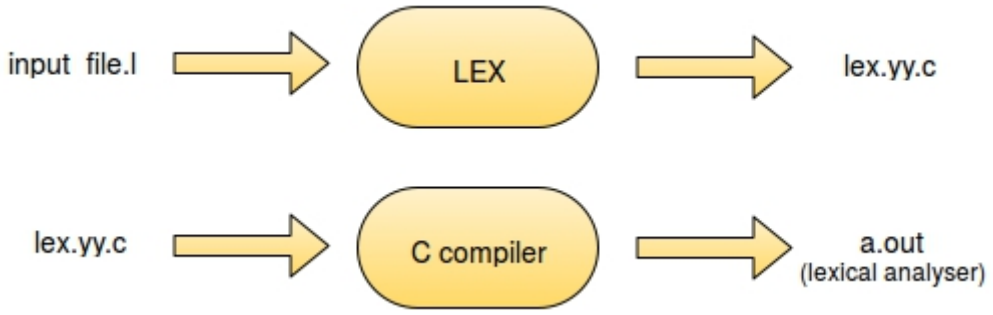
Experiment No: 2

AIM: To install LEX tool and Implement lexical analysis phase.

THEORY:

A. **FLEX (fast lexical analyzer generator)** is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produces faster code.

Bison produces parser from the input file provided by the user. The function **yylex()** is automatically generated by the flex when it is provided with a **.l file** and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.



Installing Flex on Ubuntu:

sudo apt-get update

sudo apt-get install flex

Step 1: An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.

Step 2: The C compiler compile lex.yy.c file into an executable file called a.out.

Step 3: The output file a.out take a stream of input characters and produce a stream of tokens.

Program Structure:

In the input file, there are 3 sections:

1. Definition Section: The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in **" %{ %} "** brackets. Anything written in this brackets is copied directly to the file **lex.yy.c**

Syntax:

```
%{
```

// Definitions

%}

2. Rules Section: The rules section contains a series of rules in the form: *pattern action* and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in **"%% %%"**.

Syntax:

%%

pattern action

%%

3. User Code Section: This section contains C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

Basic Program Structure:

%{

// Definitions

%}

%%

Rules

%%

User code section

How to run the program:
To run the program, it should be first saved with the extension **.l or .lex**. Run the below commands on terminal in order to run the program file.

Step 1: lex filename.l or lex filename.lex depending on the extension file is saved with
Step 2: gcc lex.yy.c
Step 3: ./a.out
Step 4: Provide the input to program in case it is required

COMPUTING ENVRONMENT:

Platform: ubuntu
Tool: FLEX

Sample Example: Count the number of characters in a string

Write the program in text editer and save the file with .l extention.

For writing program:

Algorithm:

a. **Definition Section has one variable which can be accessed inside yylex() and main()**
b. **Rule Section has three rules, first rule matches with capital letters, second rule**

matches with any character except newline and third rule does not take input after the enter.

c. Code Section prints the number of capital letter present in the given input.

Input: Any string of Characters and numbers
Output:Number of capital letters in given input

B. To demonstrate the concept of how to separates the tokens into lexical analysis phase.

In a compiler linear analysis is called as lexical analysis or scanning. In this the Stream of chars making to right & group into tokens, The blanks separating the chars of these tokenswould normally be eliminated during lexical analysis Here the sequence of char, have a collective meaning.

To know the concept of lexical analyzer and how it works.

Consider the statement

a:=b+c*d

Tokens:

A, b, c, d - identifiers

:= - Assignment Operator

+ - Addition operator

*- Multiplication operator

Algorithm:

Input : LEX specification files for the token Output : Produces the source code for the Lexical Analyzer with the name lex.yy.c and displays the tokens from an input file.

1. Start
2. Open a file in text editor
3. Create a Lex specifications file to accept keywords, identifiers, constants, operators and relational operators in the following format. a) %{ Definition of constant /header files %} b) Regular Expressions %% Transition rules %% c) Auxiliary Procedure (main( ) function)
4. Save file with .l extension e.g. mylex.l
5. Call lex tool on the terminal e.g. [root@localhost]# lex mylex.l. This lex tool will convert ‚.l' file into ‚.c' language code file i.e., lex.yy.c
6. Compile the file lex.yy.c using C / C++ compiler. e.g. gcc lex.yy.c. After compilation the file lex.yy.c, the output file is in a.out
7. Run the file a.out giving an input(text/file) e.g. ./a.out.
8. Upon processing, the sequence of tokens will be displayed as output.
9. Stop

Expected Output

Input: Any string of Characters,numbers and keywords
Output:The identifier, number and keywords from input.

For example the input string **int a=9**

Expected Output is

int is a small Keyword
a is an identifier
9 is a number

Coclusion: Thus the FLEX tool is installed and the example of Lexical analysis phase is implemented on FLEX tool.

Viva Voce Questions:

1. What is the use of Lex tool?

Answer: Lex is a program designed to generate scanners, also known as tokenizers, which recognize lexical patterns in text. Lex is an acronym that stands for "lexical analyzer generator." It is intended primarily for Unix-based systems.

2. What are the components of Lex?

Answer:A lex program consists of three sections: a section containing definitions, a section containing translations, and a section containing functions.

3. Why is Lex required?

Answer:It is generally used to declare functions, include header files, or define global variables and constants. LEX allows the use of short-hands and extensions to regular expressions for the regular definitions. A regular definition in LEX is of the form : D R where D is the symbol representing the regular expression R.

4. What does the 'Definition section' contain?

Answer: Substitutions code and start states. This section will be copied into lex.yy.c.

5. What does the 'Rule section' contain?

Answer: Defines how to scan and what action to take for ach token

6. What does the last section contain?

Answer: C auxiliary subroutines: any user code and scanning function yylex()