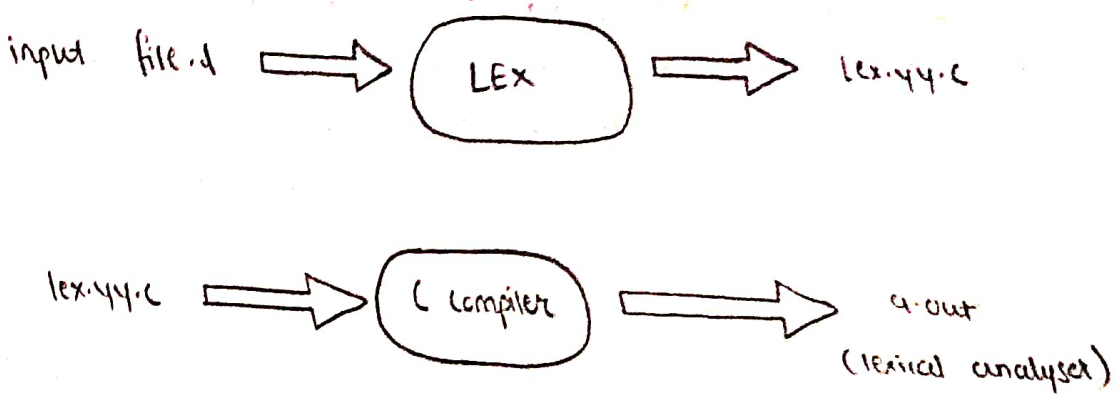
Practical No.2

Aim: To install LEX tool and implement lexical analysis phase.

Diagram:



Date :

Practical 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 analyzer (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 describing 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, a regular definitions, manifest constants. In the

Date :

definition sections, 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 sections contains a series of rules in the form: pattern action and pattern must be unintented and action begin on the same line in {} brackets. The rule section is enclosed in "%% %%".

Syntax:

```
%%
pattern action
%%
```

3. User code section: This section contain C statements and additional function. We can also compile these functions seperately and load with the lexical analyzer.

Basic program structure -

```
%{
    // definitions
%}

%%
Rules
%%

User code section
```

Date :

How to run the program :

To run the program, it should be first saved with the extension .l or .lex . Run the below command 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 Environment :
Platform : ubuntu.
Tool : FLEX .

Sample Example : Count the number of characters in a string.
Write the program in text editor and save the file with .l extension.
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 newlines and third rule does not take input after the enter.
c. Code section print 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.

Date :

8. To demonstrate the concept of how to separate 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 to and group into tokens, the blanks separating the chars of these tokens would 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 display the tokens from an input file.

1. Start
2. Open a file in text editor.
3. Create a lex specification file to accept keywords, identifiers, constants, operator and relational operator in the following format .a) %{ Definition of constant (header file %} b) Regular Expression %% Transition %% c) Auxillary procedure (main() function).
4. Save file with .l extension eg. mylex.l.
5. Call lex tool on the terminal eg. [root@localhost] # lex mylex.l. This lex tool will convert .l file into .c language code file ie. lex.yy.c

Conclusion:

Thus the Flex tool is installed and the example of Lexical analysis phase is implemented on Flex Tool.

Date :

6. Compile the file lex.yy.c using c/c++ compiler. eg. 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) eg. ./a.out.
8. Upon processing, the sequence of token will be displayed as o/p.
9. Stop.

Expected output:
Input : Any string of characters, numbers and keywords.
Output : The identifier, number and keyword 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.

Conclusion:
Thus the FLEX tool is installed and the example of lexical analysis phase is implemented on FLEX tool.

Viva Voce Question:
① what is the use of lex tool?
→ Lex is a program designed to generated scanners, also known as tokenizers, which recognize lexical pattern 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?
→ A lex program consists of three sections : a section containing definitions, a section containing translations, and a section containing functions.

Date :

③ Why is Lex required?

→ 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 expression for the regular definitions. A regular definitions 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?

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

⑤ What does the 'Rule section' contain?

→ Defines how to scan and what action to take for each token.

⑥ What does the last section contains?

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