SRES's Sanjivani College of Engineering, Kopargaon
(*An Autonomous Institute*)
Department of Computer Engineering

SPOS Lab Manual

Assignment No. 05

AIM:
Write a program using Lex specifications to implement lexical analysis phase of compiler to generate tokens of subset of 'C' program.

PROBLEM DEFINITION:
Write a program using Lex specifications to implement lexical analysis phase of compiler to generate tokens of subset of 'C' program.

OBJECTIVES:
1. To understand lexical analyzer
2. To use the lex tool

INPUT:
Sample C program consisting of different types of tokens

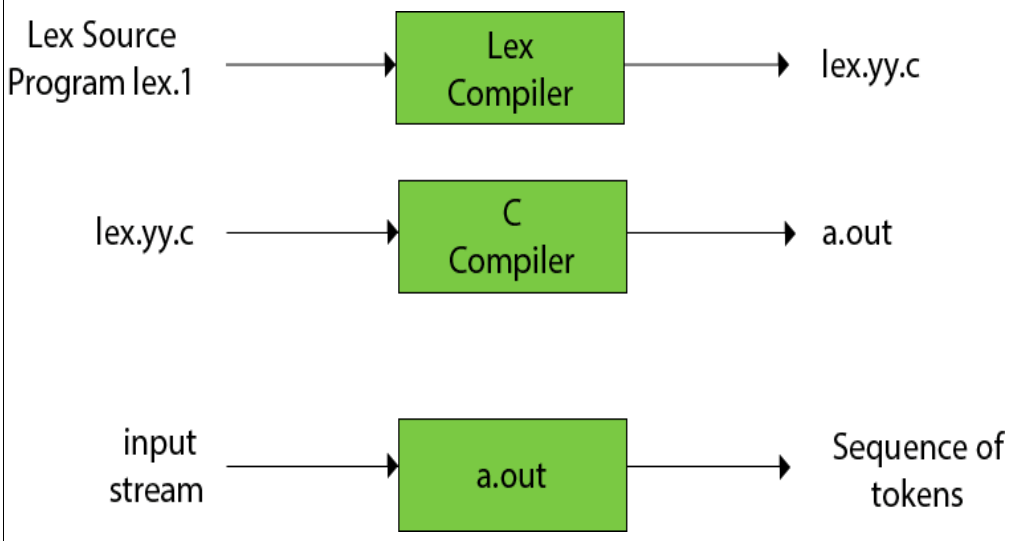
OUTPUT:
Set of generated tokens

THEORY:

The function of Lex is as follows:

- Firstly lexical analyzer creates a program lex.l in the Lex language. Then Lex compiler runs the lex.l 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 is a tool used to generate a lexical analyzer. LEX translates a set of regular expression specifications (given as input in input_file.l) into a C implementation of a corresponding finite state machine (lex.yy.c). This C program, when compiled, yields an executable lexical analyzer.



The source program is fed as the input to the lexical analyzer which produces a sequence of tokens as output. Conceptually, a lexical analyzer scans a given source program and produces an output of tokens.

Each token is specified by a token name. The token name is an abstract symbol representing the kind of lexical unit, e.g., a particular keyword, or a sequence of input characters denoting an identifier. The token names are the input symbols that the parser processes. For instance, integer, float, boolean, begin, end, if, while etc. are tokens in source program.

"float" {return KEYWORD;}

This example demonstrates the specification of a **rule** in LEX. The rule in this example specifies that the lexical analyzer must return the token named KEYWORD when the lexeme "float" is found in the input file. A rule in a LEX program comprises of a 'pattern' part (specified by a regular expression) and a corresponding (semantic) 'action' part (a sequence of C statements). In the above example, "float" is the pattern and {return KEYWORD;} is the corresponding action. The statements in the action part will be executed when the pattern is detected in the input.

The structure of LEX programs

A LEX program consists of three sections: **Declarations, Rules and Auxiliary functions**

DECLARATIONS

%%

RULES

%%

AUXILIARY FUNCTIONS

A] Declarations

The declarations section consists of two parts, **auxiliary declarations** and **regular definitions**.

The auxiliary declarations are copied as such by LEX to the output *lex.yy.c* file. This C code consists of instructions to the C compiler and are not processed by the LEX tool. The auxiliary declarations (which are optional) are written in C language and are enclosed within ' %{ ' and ' %} ' . 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.

B] Rules

define the statement of form

```
p1      {action1}
p2      {action2}
.
.
.
.
pn      {action n}.
```

Rules in a LEX program consists of two parts :

1. The pattern to be matched
2. The action the lexical analyzer should take when pattern pi matches a lexeme.

C] Auxiliary functions

User subroutines are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately.

LEX generates C code for the rules specified in the Rules section and places this code into a single function called *yylex()*. In addition to this LEX generated code, the programmer may wish to add his own code to the *lex.yy.c* file. The auxiliary functions section allows the programmer to achieve this.

The auxiliary declarations and auxiliary functions are copied as such to the lex.yy.c file

Once the code is written, *lex.yy.c* maybe generated using the command *lex "filename.l"* and compiled as *gcc lex.yy.c*

ALGORITHM:

Step1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user_subroutines

Step2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..%}. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step3: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step5: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

Step7: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

Execution Steps:

1. **Compile lex program**

   lex file.l ----→ lex.yy.c

2. **Compile c program lex.yy.c**

   cc lex.yy.c ---→ a.out

3. **Execute object program**

   ./a.out

Patterns for C Tokens

letter [A-za-z]

digit [0-9]

identifier {letter}({letter}|{digit})*

number {digit}+(\.{digit}+)?

punctuation [; , " # ( ) { }]

operators "+" | "-" | "=" | "*" | "<" | ">"

keywords "if" | "else" | "for" |"int" | "while" | "do" | "void"

literal \".*\"

comment "//".*

headerfile "#".*

multicomment "/*".*"*/"

OBSERVATION:

It is observed that all the tokens from the input file are get displayed on the screen

CONCLUSION:

LEX tool is used to generate the lexical analyzer for the given input program

References:

Lex & Yacc Book by Doug Brown, John R. Levine, and Tony Mason

Prepared by
Prof.N.G.Pardeshi
Subject Teacher

Approved by
Dr. D.B.Kshirsagar
HOD