SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Engineering

Name: _	Date of Performance: _ _ /_ _/20_ _
Class: B.E. Computer	Date of Completion: _ _ /_ _/20_ _
Division :	Grade:
Batch:	
Roll No:	Sign. of Teacher with Date:
Subject: Compiler Design Lab	

Experiment No. 1

Aim: Implement a lexical analyzer for a subset of C using LEX Implementation should support Error handling

1. Objective: To able implement a lexical analyzer for a subset of C using LEX Implementation.

2. Background:

As the first phase of compiler, the main task of the lexical analyzer is to read the input characters of the source program group them into lexemes and produce as output a sequence of tokens for each lexeme in the source program. When the lexical analyzer discovers a lexeme constituting an identifier, it needs to enter that lexeme into the symbol table. The lexical analyzer not only identifies the lexemes but also pre-processes the source text like removing comments, white spaces, etc.

Lexical analyzers are divided into a cascade of two processes:

a) Scanning - It consists of simple processes that do not require the tokenization of the input such as deletion of comments, compaction of consecutive white space characters into one.
b) Lexical Analysis- This is the more complex portion where the scanner produces sequence of tokens as output.
c) A Token is pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing the kind of lexical unit, eg., a particular keyword or an identifier.
d) A pattern is a description of the form that the lexemes of a token may take. In case of a keyword as a token the pattern is just a sequence of characters that form the keyword.

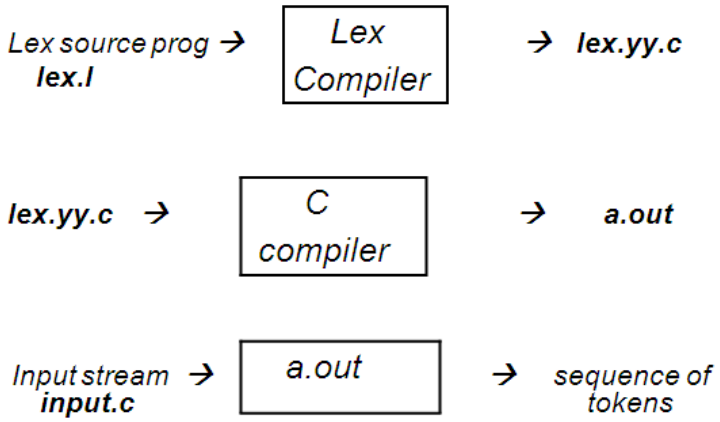
A Lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

3. Pre-lab Task:

- **LEX :** A tool widely used to specify lexical analyzers for a variety of languages, refer to the tool as *Lex compiler ,* and to its input specification as the *Lex language.*
- The main job of a *lexical analyzer (scanner)* is to break up an input stream into more usable elements (*tokens*)

  EX: a = b + c * d ;

  ID ASSIGN ID PLUS ID MULT ID SEMI
- Lex is an utility to help you rapidly generate your scanners



4. In-lab Task:

1. First, write down the **lexical specification** using **regular expression** to specify the lexical structure:
   identifier = letter (letter | digit | underscore)*
   letter = a | ... | z | A | ... | Z
   digit = 0 | 1 | ... | 9
2. based on the above **lexical specification**, build the lexical analyzer (to recognize tokens) by hand,
   Regular Expression Spec ==> NFA ==> DFA ==>Transition Table ==> Lexical Analyzer

Sample Program Template:

```
digit    [0-9]
letter   [a-zA-Z]
%%
{letter}({letter}|{digit})*      printf("id: %s\n", yytext);
\n                               printf("new line\n");
%%
main()
 {
        yylex();  }
```

5. Post-lab Task:

Outcomes:

. . . . . .
. . . . . .
. . . . . .

Questions:

1. What is lexical analyzer?
2. Which compiler is used for lexical analyzer?
3. What is the output of Lexical analyzer?
4. What is LEX source Program?