**Experiment No. 2: Study of LEX and YACC tools**

**LEX - A Lexical Analyzer Generator:** Lex is a computer program that generates lexical analyzers (also called "scanners" or "lexers"). Lex is commonly used with the YACC parser generator. It reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language.

A lexer or scanner performs lexical analysis, breaking up an input stream into meaningful units or tokens. For example, it can break a text file into individual words. Lex is a tool for automatically generating a lexer or scanner given a lex specification (.l file).

**Structure of a Lex File:** The structure of a Lex file is intentionally similar to that of a YACC file. Lex files are divided into three sections, separated by lines containing only two percent signs (%%), as follows:

**Definition section**:

%%

**Rules section:**

%%

**C code section:**

<statements>

* The **definition section** is used to define macros and import header files written in C. You can also write any C code here, which will be copied verbatim into the generated source file.
* The **rules section** is the most important section; it associates patterns with C statements. Patterns are regular expressions. When the lexer sees some text in the input matching a given pattern, it executes the associated C code, which is how Lex operates.
* The **C code section** contains C statements and functions that are copied verbatim into the generated source file. In large programs, it is more convenient to place this code in a separate file and link it at compile time.

**Description:** The Lex command reads a file or standard input, generates a C language program, and writes it to a file named lex.yy.c. This file is a compilable C program. The Lex command can also generate C++ programs by using the -C flag, which renames the output file to lex.yy.C. The C++ program can use either stdio or iostreams. If the cpp define \_CPP\_IOSTREAMS is true during a C++ compilation, the program will use iostreams for I/O, otherwise, it will use stdio.

Lex uses rules and actions in a file to generate a program, lex.yy.c, which can be compiled. The compiled program can then receive input, break it into logical pieces defined by the rules, and execute program fragments in the actions specified in the file.

The generated program is a C function called yylex, which can be used to recognize simple one-word input or combined with other C programs to perform more complex input analysis functions. The yylex function analyzes the input stream using a finite state machine, which allows the program to exist in only one state at a time.

**Regular Expression Basics:**

* . matches any single character except \n
* \* matches 0 or more instances of the preceding regular expression
* + matches 1 or more instances of the preceding regular expression
* ? matches 0 or 1 of the preceding regular expression
* | matches the preceding or following regular expression
* [] defines a character class
* () groups enclosed regular expressions into a new regular expression
* "..." matches everything within the quotes (“”) literally

**Special Functions:**

* yytext: stores the text most recently matched
* yyleng: number of characters in the text most recently matched
* yylval: associated value of the current token
* yymore(): appends the next string matched to the current contents of yytext
* yyless(n): removes all but the first n characters
* unput(c): returns character c to the input stream

The yywrap() function can be replaced by the user. It is called when the lexical analyzer encounters EOF as the first character while trying to match a regular expression.

**Files:**

* y.output: contains a readable description of the parsing tables and a report on conflicts generated by grammar ambiguities
* y.tab.c: contains an output file
* y.tab.h: contains definitions for token names
* yacc.tmp: temporary file
* yacc.debug: temporary file
* yacc.acts: temporary file
* /usr/ccs/lib/yaccpar: contains parser prototype for C programs
* /usr/ccs/lib/liby.a: contains a runtime library

**YACC:** YACC stands for "Yet Another Compiler Compiler." YACC provides a tool to produce a parser for a given grammar. It is designed to compile a LALR(1) grammar and is used to produce the source code of the syntactic analyzer for the language produced by LALR(1) grammar. The input to YACC is a set of rules or grammar, and the output is a C program.