**Practical 1**

**Aim:** Theory assignment for writing details about LEX and YACC compilation.

**Theory:** LEX [Lexical Analyzer Generator] and YACC [Yet Another Compiler Compiler] are tools used in the frontend phase of a compiler.

1. Definition:
2. Lex is responsible for performing lexical analysis, which means scanning the source code and converting it into a sequence of tokens. Tokens are the smallest meaningful units such as keywords, identifiers, operators and symbol. LEX uses regular expressions to match patterns in the input and triggers corresponding actions written in C.
3. YACC is responsible for syntax analysis, where it uses context free grammar to define valid constructors of the programming language. It takes the stream of tokens generated by LEX and checks whether the sequence follows the grammatical rules defined in the parser. If it does, YACC builds a parse tree or abstract tree/abstract syntax tree for further processing. Together they help in building a language processor or compiler frontend that can analyse input programs and prepare them for further compilation or interpretation.
4. Structure:
5. LEX program structure:

%{

// C Declarations

%}

%%

// Pattern and Action Section/Rule Section

%%

// Additional C Functions/User Subroutine Section

1. YACC program structure:

%{

// C Declarations

%}

%%

// Rule Section

%%

// User Subroutine Section

1. Pseudo Code:
2. Write LEX file to tokenize input
3. Write YACC file to define grammar and actions
4. Compile both the files
5. Inbuilt functions:
6. LEX

|  |  |
| --- | --- |
| Functions | Description |
| yylex() | Main lexical analyser function called by YACC to get tokens |
| yytext | Stores the current matched string |
| yyleng | Length of the current token in yytext |
| yyin | File pointer to input source. Default is stdin |
| ECHO | Prints the matched text to output |
| BEGIN | Changes the state of the LEXER [used for stateful lexing] |
| unput() | Pushes character C back into the input stream |
| input() | Reads the next character from file |

1. YACC

|  |  |
| --- | --- |
| Function | Description |
| yyparse() | Main parsing function automatically generated by YACC |
| yylex() | Called by yyparse() to get the next token from LEX |
| yyerror(msg) | Reports syntax errors, msg is a custom error message |
| $$ | Represents the value of the left-hand side non-terminal |
| $1, $2, … | Values of symbols on the right-hand side of a production rule |
| %token | Declares a token to be returned by LEX |
| %left, %right, %nonassoc | Declares associativity and precedence of operators |
| %start | Specifies the starting rule for parsing |

**LEX Code:**

%{

#include <stdio.h>

%}

%%

zero|ZERO|Zero printf("0\n");

one|ONE|One printf("1\n");

two|TWO|Two printf("2\n");

three|THREE|Three printf("3\n");

four|FOUR|Four printf("4\n");

five|FIVE|Five printf("5\n");

six|SIX|Six printf("6\n");

seven|SEVEN|Seven printf("7\n");

eight|EIGHT|Eight printf("8\n");

nine|NINE|Nine printf("9\n");

%%

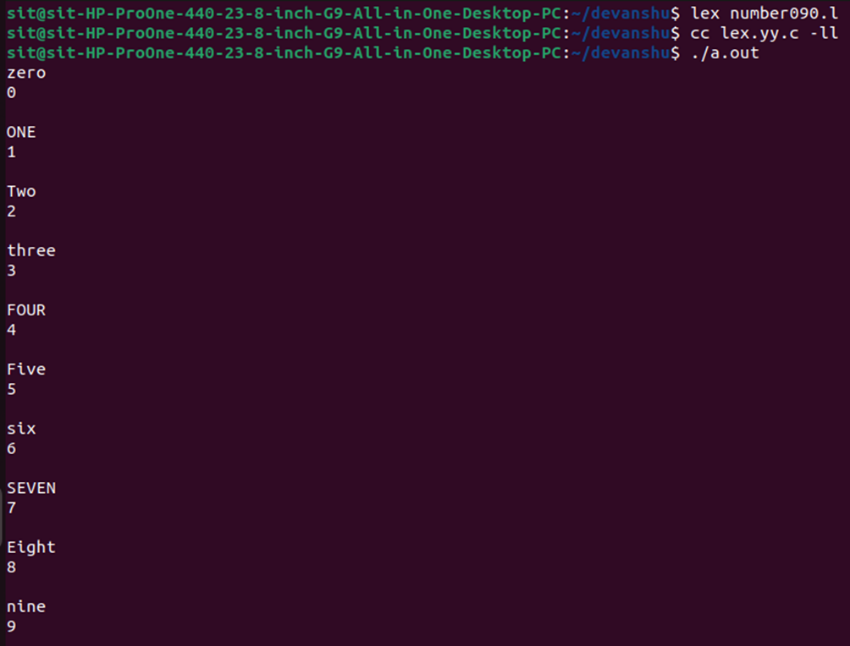
int main() {

yylex();

return 0;

}

**Output:**



**Conclusion:** LEX and YACC are essential tools in compiler construction LEX handles tokenization. YACC performs syntax parsing using grammar rules. This lab helps understand the integration of both tools and how they form the foundation of compiler.