Date: 07/08/2024

Experiment 2.1

AIM

To understand LEX and YACC tools.

THEORY

LEX is a computer program that generates lexical analysers ("scanner" or "lexers"). It is commonly used with the YACC parser generator and is the standard lexical analyser generator on many Unix and Unix-like systems. LEX reads an input stream specifying the lexical analyser and writes source code which implements the lexical analyser in the C programming language.

Structure of Lex Programs:

A LEX program is organized into three main sections: Declarations, Rules, and Auxiliary Functions. Each section serves a distinct purpose in defining the behaviour of the lexical analyser.

- **Declarations**

  The Declarations section comprises two parts:

  - Regular definitions: Define macros and shorthand notations for regular expressions to simplify rule definitions.
  - Auxiliary Declarations: Contains C code such as header file inclusions, function prototypes, and global variable declarations. This code is enclosed within %{ and %} and is copied verbatim into the generated lex.yy.c file.

- **Rules**

  The Rules section consists of pattern-action pairs, where each pair defines a regular expression pattern to match and the corresponding C code to execute upon a successful match.

- **Auxiliary Functions**

  The Auxiliary Functions section includes additional C code that is not part of the rules. This typically contains the main function and any helper functions required by the lexer. This code is placed after the second %% and is directly copied into the lex.yy.c file.

yylex()

The 'yylex()' function, defined by Lex in the 'lex.yy.c' file, reads the input stream, matches it against regular expressions, and executes the corresponding actions for each match. It also generates tokens for further processing by parsers or other components, though the programmer must explicitly invoke 'yylex()' within the auxiliary functions of the LEX program.

yywrap()

The function yywrap() is called by yylex() when the end of an input file is reached. If yywrap() returns 0, scanning continues; if it returns a non-zero value, yylex() terminates and returns 0.

Compilation Steps:

Step-1: Start

Step-2: Create a file with a .l and write your LEX.

Step-3: Give the command ‘lex simple_calc.l’. This command generates a C source file named lex.yy.c.

Step-4: ‘gcc -o simple_calc lex.yy.c -lfl’ to compile the generated C code.

Step-5: Running the Executable Execute the compiled program using: ‘./simple_calc’

YACC (Yet Another Compiler Compiler) generates LALR parsers from formal grammar specifications. It is often used with Lex for building compilers and interpreters, handling the syntactic analysis phase. YACC reads a grammar file and produces a C source parser that processes token sequences (typically from LEX) to check if they follow the grammar. The modern counterpart of YACC is Bison.

Step 1: Create a file with a .y and write your YACC.

Step 2: Give the command ‘yacc -d simple_calc.y’. This command generates a C source file named y.tab.c.

Step 3: Use ‘gcc -o simple_calc lex.yy.c -lfl’ to compile the generated C code.

Step 4: Running the Executable Execute the compiled program using: ‘./simple_calc’

Structure of YACC Program

%{

C Declarations

%}

Yacc Declarations

%%

Grammar Rules

%%

Additional Code (Comments enclosed in /*....*/ may appear in any section)

RESULT

Familiarized ourselves with the working of YACC and LEX tools for compiler design.