**JAVA COMPILER MANUAL**

**UNDER GUIDANCE OF:**

**Prof. Anupam Agarwal and**

**Dr. T. Pant**

**SUBMITTED BY :**

Prakhar Jain ( RIT2009053 )

Naveen Shukla( RIT2009054 )

Mayank Gupta( RIT2009055)

Anshul Jain(RIT2009060)

Shashank Mittal( RIT2009071)

**Introduction**

Our goal is to write a BASIC java compiler. First, we need to specify all pattern matching rules for lex (java.l) and grammar rules for yacc (java.y).

Commands to create our compiler, java\_compiler, are listed below:

yacc -d java.y # create y.tab.h, y.tab.c

lex java.l # create lex.yy.c

gcc lex.yy.c y.tab.c -o java\_compiler # compile/link

Yacc reads the grammar descriptions in java.y and generates a syntax analyzer (parser), that includes function yyparse, in file y.tab.c. Included in file bas.y are token declarations. The â€“d option causes yacc to generate definitions for tokens and place them in file y.tab.h. Lex reads the pattern descriptions in java.l, includes file y.tab.h, and generates a lexical analyzer, that includes function yylex, in file lex.yy.c.

Finally, the lexer and parser are compiled and linked together to create executable java\_compiler. From main we call yyparse to run the compiler. Function yyparse automatically calls yylex to obtain each token.

We have also included make file which contains all these commands.

This is running on Linux machine so user has to grant the executable permission to make.

This can be done by typing the following command

chmod +x make

after doing it user has to run the following command

./make

**Parsing Method used by yacc**

An LALR parser (or look-ahead LR parser) is a type of LR parser based on a finite-state-automata concept.[dubious â€“ discuss] The data structure[dubious â€“ discuss] used by an LALR parser is a pushdown automaton (PDA). A deterministic PDA is a deterministic finite automaton (DFA) with the addition of a stack for a memory, indicating which states the parser has passed through to arrive at the current state. Because of the stack, a PDA can recognize grammars that would be impossible with a DFA; for example, a PDA can determine whether an expression has any unmatched parentheses, whereas an automaton with no stack would require an infinite number of states due to unlimited nesting of parentheses.

LALR parsers are driven by a parser table in a finite-state machine (FSM) format. An FSM is tedious enough for humans to construct by hand that it is more convenient to use a software tool called an LALR parser generator to generate a parser table automatically from a grammar in Backusâ€“Naur Form which defines the syntax of the computer language the parser will process.[1] The parser table is often generated in source code format in a computer language (such as C++ or Java). When the parser (with parser table) is compiled and/or executed, it will recognize files written in the language defined by the BNF grammar.

Our java compiler detects the error for following error:

1- if exp ) statement;

2- if ( exp statement;

3- if exp statement;

4- classname block }

5- classname { block

6- class implements {block}

7- field variable semicolon missing;

8- function parameter comma missing error

9- function arguement comma missing error;

10- for initialization semicolon missing

11- for condition semicolon missing

14- while exp ) statement;

15- while ( exp statement;

16- while exp statement;