Implementation Of LALR Parser Using Lex and Yacc

Aayush Nirwan
1601001

Abhinav Jha
1601002

I. Introduction

Yacc (Yet Another Compiler-Compiler) is a computer program for the Unix operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus Naur Form (BNF). Lex is a computer program that generates lexical analyzers ("scanners" or "lexers"). Lex is commonly used with the yacc parser generator. Lex, originally written by Mike Lesk and Eric Schmidt and described in 1975, is the standard lexical analyzer generator on many Unix systems, and an equivalent tool is specified as part of the POSIX standard. Lex and yacc help us write programs that transform structured input. Lex turns these regular expressions into a form that the lexer can use to scan the input text extremely fast.

II. Context Free Grammar

The grammar which we have implemented is shown below:

(1) Start $\rightarrow$ Stmt_list END

(2) Stmt_list $\rightarrow$ Stmt | Stmt_list END Stmt

(3) Stmt $\rightarrow$ Variable ASSIGN Expression | F
| SelectStmt | Decleration END Stmt
| ReturnStmt END Stmt | FORStmt
| WHILEStmt

(4) ReturnStmt $\rightarrow$ Return Z

(5) Z $\rightarrow$ ID | NUM

(6) Decleration $\rightarrow$ TYPE X

(7) X $\rightarrow$ ID X | empty | COMM X

(8) empty $\rightarrow$ NULL

(9) SelectStmt $\rightarrow$ IF STRBO Expression STRBC CO Stmts CC C Stmt

(10) FORStmt $\rightarrow$ FOR STRBO ID ASSIGN Expression END ID RELOP Expression END ID Inr STRBC CO Stmts CC Stmt

(11) Inr $\rightarrow$ ++ | --

(12) WHILEStmt $\rightarrow$ WHILE STRBO Expression STRBC CO

(13) Stmts $\rightarrow$ Stmt END Stmts | empty

(14) C $\rightarrow$ ELSE CO Stmts CC | empty

(15) F $\rightarrow$ TYPE MAIN STRBO STRBC CO Stmts CC

(16) Variable $\rightarrow$ ID | ID SQRBO Expression SQRBC

(17) Expression $\rightarrow$ Simple_expression
| Simple_expression RELOP Simple_expression

(18) Simple_expression $\rightarrow$ Term
| Simple_expression ADDOP Term

(19) Term $\rightarrow$ Factor | Term MULOP Factor

(20) Factor $\rightarrow$ ID | NUM | STRBO Expression STRBC
| ID SQRBO Expression SQRBC

III. Implementation

Using lex file("assg3.l"), we are generating tokens. These tokens are printed in a file "token.txt". Symbol table is also implemented in the lex file using structure. Symbol table is printed in the file "symbolTable.txt". Variables will only be inserted in the symbol table if they are present in the declaration statements. Operator precedence and balancing of parenthesis are implemented in the grammar itself.

IV. Code Analysis

A. Robustness

Robustness describes how reliable our program is, especially under extreme conditions such as extreme workload or unpredictable user inputs. This program would never crash no matter what the user inputs. Even if the user enters invalid data, program will handle it properly. Unpredictable user inputs such as nested for and if loops are even taken care of in the grammar. Even a huge C code will be parsed according to the grammar, generating appropriate errors if any.

B. Correctness

Correctness of an algorithm is asserted when it is said that the algorithm is correct with respect to a specification. Correctness refers to the input-output behaviour of the algorithm. This program might render some of test cases but it will work properly in most of test cases.

V. Error Handling

Errors are handled in a special manner. Along with printing "error" in the output, program also prints the type of the error. Parsing will halt once an error has occurred.

VI. Prerequisite

Lex and yacc should be installed before hand. DO "chmod +x script.sh" before running