David Nguyen

Professor Le

CPSC 323-04

19 November 2019

Project 2: Syntax Analyzer

1. Problem Statement

Write a syntax analyzer using a top-down parser like, RDP, predictive recursive descent parser or a table-driven predictive parser. Next, we have to rewrite the grammar if there is left recursion or factorization issues. Then we use the lexer() function that we made before to get the tokens so that the parser check if its correct. The parser should print to an output file the tokens, lexemes and the set of production rules. Next, create an error handling mechanism to gracefully terminate and print an exception to the client.

2. How to use your program

Extract the ZIP files. The program is written in C++ using CLion (IDE), so visual studio will not work. To compile and run the program use Clion or create a new project with an IDE of choice. Another method is to go into the cmake-build-debug folder and find the CP323.exe file. Next run that file and the program will parse the syntax_input.txt and write the parsed input into the syntax_output.txt. If you do not wish to use these txt files, then just go into the main change the arguments in the read and write functions.

3. Design of your program

As the programmer, I was tasked to build a syntax analyzer from scratch. The professor gave us three choices of top-down parser: RDP, predictive recursive descent parser or table-driven predictive parser. Fortunately, the professor also gave us a set of production rules so that

we use for our program. The set of production rules is not perfect, there are left recursion in the set of production rules. For example, <Expression> -> <Expression> + <Term> | <Expression> - <Term>. This production needs to be rewritten so that the production does not have left recursion. With the given production, I was able to create predictive recursive descent part of the project. I used the First and Follow so that I can make a table so that the Push Down Automata can traverse through input line and traverse through the table. After sketching my designs for the project, I was able to write a program having the PDA, stack, and the table in mind. After reading the "table driven predictive" part of the book I was able to understand the pseudocode for the PDA part. First, I implemented my table with the terminal and nonterminal. Then, I created the stack that will accept strings of either terminal or nonterminal. Next, I created the PDA function using the pseudocode from the book and lecture. The PDA function is very crucial because it is used to push down the nonterminal when it encounters an input that is not like an operator, separator, or any characters that is simplified. The stack will be then evaluated to check if top of the stack matches with the token. If it does match, pop the stack and that token will have its own set of production that was used to figure out its nature. I have implemented error-handling mechanisms like a try-catch block in the client that will catch any thrown exceptions from the PDA. I also split up the functions in the PDA function so that it convenient for anyone trying to understand my code. For the I/O, the lexer() function will parse the input string to check for tokens and pushed into a vector that has a tuple of a pair of token and lexeme and a vector of productions. It gets a bit more complicated because that vector is for one line of code, so I have to make another vector that will store vector that contains the tuple. That vector that store a vector of tuples will be able to input and output each line of input. The output file should contain the token and lexeme with its set of production rules.

4. Any Limitation

None

5. Any shortcomings

I was not able to properly implement the declarative and conditional statements. I am very upset because maybe there is something that I am not looking at correctly. After looking back into the assignment prompt, I think I did not left factor a production rule. Due to time constraint, I could not solve it in time. In my spare time, I will look more into this because I feel like there is something missing.