**CS323 Assignment 2 Documentation**

1. **Problem Statement**

The purpose of this assignment is to create a syntax analyzer using top-down parser RDP as the algorithm. There are 20 rules that must be followed for the RatSU21 language we are creating. This program will utilize the Lexical Analyzer completed from assignment one to see if the composition of the lexemes matches the 20 rules of RatSU21.

1. **How to use your program**
2. Unzip the file for Assignment2.
3. If you have your own test file, the easiest way to analyze it would be to replace the contents of one of the test files with your own source code.
4. Double click the Assignment2.exe. If this works, skip to step 7. If not, continue on to the next step.
5. In windows cmd, enter into Assignment2 folder.
6. Type dir to see if you are in the right folder and Assignment2.exe is available.
7. Type Assignment2.exe in terminal and program should run.
8. Test output files (ex. SA\_output\_test1.txt) will show the Syntactical analysis of the corresponding test file (ex. test1.txt).
9. **Design of your program**

The Syntactical Analyzer was created into an SA class, with all the rules, except the initial rule, as private functions. This is to make sure no one is disrupting the analysis once it starts. Each time a terminal symbol is reached, the lexer() function is called to return the next lexeme to be analyzed. Lexemes will be printed, followed by the rules that are followed to reach the terminal symbol of the lexeme. Example shown below.

Token: KEYWORD Lexeme: integer

<Declaration List Prime> -> <Declaration List>

<Declaration List> -> <Declaration>; <Declaration List Prime>

<Declaration> -> <Qualifier> <identifier>

<Qualifier> -> integer

If all rules are followed, the file will be completely analyzed with the final analysis comment in the output file as “<Rat21SU> end”. If a rule is broken, the analyzer will exit, and an error will be shown in the output file explaining why the analysis could not be completed.

Though there are 20 rules in the RatSU21 language, we needed to implement 25 functionalities to eliminate left-recursion and backtracking. Below are the rules that had to be reanalyzed for implementation.

<Declaration List> ::= <Declaration> ; <Declaration List Prime>

<Declaration List Prime> ::= <Declaration List> | ε

<Statement List> ::= <Statement> <Statement List Prime>

<Statement List Prime> ::= <Statement List> | ε

<If> ::= if ( <Condition> ) <Statement> <If Prime>

<If Prime> ::= endif | else <Statement> endif

<Expression> ::= <Term> <Expression Prime>

<Expression Prime> ::= + <Term> <Expression Prime> | - <Term> <Expression Prime> | ε

<Term> ::= <Factor> <Term Prime>

<Term Prime> ::= \* <Factor><Term Prime> | / <Factor> <Term Prime> | ε

1. **Any Limitations**

The comments will be ignored, but there must be a space after the beginning of the comment symbol (/\* ) and a space before the end of comment symbol. ( \*/).

1. **Any shortcomings**

Because of the way our initial Assignment 1 was completed, a buffer had to be added to hold our lexemes. Lexer function does work as needed, returning the next lexeme when lexer() is called. However, the exact programming process does not match the initial programming requirements of Assignment 1. We attempted to fix this after the due date of Assignment 1, but found it would be very difficult to do a complete rework of the lexer() and complete the Syntactical Analyzer in time. We ultimately decided to focus on the Syntactical Analyzer and work with what we have.