**CS 323 Assignment 2 Documentation**

by James Vu

1. **Problem Statement**

Write a syntax analyzer using any top-down parser such as a RDP, a predictive recursive descent parser, or a table driven predictive parser.

1. **How to use program**

1. Open up command prompt

2. Navigate to folder containing source code

3. Compile using “g++ -o syntax syntax.cpp”

4. Run the program using “lexer.exe”

5. Input file name and press enter

1. **Design of program**

This program first uses a lexer to break down the code into tokens and lexemes. The tokens and lexemes are stored in a vector of a pair of string and string. It prints out the list of token and lexemes in a separate file from the syntax. The syntax analyzer uses the vector of tokens and lexemes to parse the code. When running multiple statements, the analyzer assumes the start of the program is “begin” and the end of program is “end”; otherwise you can do single assignment statement, declarative statement, if statement, or while statement. The syntax analyzer uses the following rules:

<Statement> -> <Assignment> | <Declarative > | if <Conditional> then <Statement> else <Statement> endif | if <Conditional> then <Statement> endif | while <Conditional> do <Statement> whileend | begin <Statement> <More Statements> end

<Assignment> -> <Identifier> = <Expression> ;

<Expression> -> <Term> <Expression Prime>

<Expression Prime> -> + <Expression> | - <Expression> | epsilon

<Term> -> <Factor> <Term Prime>

<Term Prime> -> \* <Term> | / <Term> | epsilon

<Factor> -> ( <Expression> ) | <Identifier> | <Number>

<Declarative> -> <Type> <Identifier> <More IDs>

<Type> -> int | float | bool

<More IDs> -> , <Identifier> <More IDs> | epsilon

<More Statements> -> ; <Statement> <More Statements> | epsilon

<Conditional> -> <Expression> <Relational Operator> <Expression>

<Relational Operator> -> < | <= | == | <> | >= | >

The syntax analyzer uses a counter to iterate through the vector of token and lexemes. The analyzer calls functions to check if each rule is valid or not. If the rule is valid, then the functions return a non-negative number. If the rule is invalid, then the functions return -1. The syntax analyzer uses a predictive recursive descent parser. For printing out the syntax, it uses a temporary vector of strings and a main vector of string. While the syntax analyzer is running, it stores all the output into the temporary vector. If the rule is correct, it moves the values of the temporary vector to the main vector. At the end of the syntax analyzer, it prints out the vector of strings into an output file.

1. **Limitations**

Source code size is limited to string.maxsize(). Program will not work on empty files.

1. **Shortcomings**

Program does not contain error checking, but you can check the general location where the error occurs in the syntax output by looking at the last token and lexeme outputted.