**CS323 Documentation**

**Assignment 2**

1. **Problem Statement**

For this assignment we are supposed to write a syntax analyzer. The main purpose of the project is to get the tokens from the previous assignment which was a lexical analyzer and parse them through each function using top-down parsing. In this assignment we also need to distinguish the tokens from each other. So we basically build upon our code for the Assignment 1 and just add the production rules that are produced by top down parser. Therefore, the output would be tokens, lexemes and production rules.

1. **How to use your program**

This program can be used by running the parser file. You first would have to open up the executable folder where you will find the file named **parser.** The way to execute this file is by opening up the terminal and navigating to the this directory. After you navigate to this directory you will have to type in **./parser** . In order to run the program, the parser file needs to have a file named **sample.RAT18S** in the same folder/directory as the parser executable file named parser. If the project cannot find the file, it will print out **ERROR - cannot open sample.RAT18S**. If properly ran the program should print out a list of the functions it has visited before getting to the end of the file.

1. **Design of your program**

The program starts off in main.cpp which has a Parser object that is causing the parser to start. In program we created a new class named Parser which has two attributes one a Token object and the other is a Lexer object. The Token attribute to store the current token of where the lexer is pointing. To be able to run the program without any ambiguity we had to get rid of left recursion and back-tracking.

**BACK-TRACKING**

<Function Definitions> ::= <Function> <Function Definitions Prime>

<Function Definitions Prime> ::= <Function Definitions> | ɛ

<Parameter List> ::= <Parameter> <Parameter List Prime>

<Parameter List Prime> ::= , <Parameter List> | ɛ

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

<Declaration List Prime> := <Declaration List>

<IDs> ::= <Identifier> <IDs Prime> | <Identifier>

<IDs Prime> ::= , <IDs> | ɛ

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

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

**LEFT-RECURSION**

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

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

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

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

1. **Any Limitation**

Doesn’t handle errors

1. **Any shortcomings**

None