CS323 Assignment 2 Documentation

1. Problem Statement

The primary objective of this assignment is to develop a syntax analyzer that is able to parse a token list written in Rat24S. It should verify the correctness of the syntax, ensuring compliance with the language's grammar rules and structure. The analyzer should identify and report any syntactic errors in the source code, providing informative error messages to assist developers in debugging their code effectively.

2. Program Usage

Instructions for Executing

- Download and unzip Assignment2.zip to a local destination on your machine.
- Open a terminal and navigate to the location of the assignment2 folder.
    - For example if the folder is located in your desktop folder, you enter the commands for macOS/Linux, 'cd Desktop/assignment2', or if on Windows, 'C:\Users\YourUsername\Desktop\Assignment2'.
- On the terminal use the following command:

```
python3 assignment2.py <input_file> <output_file>
```

    - For example " python3 assignment2.py test1.rat output1.txt "
- The output file will contain the production rules used as well as the tokens.

3. Design

Overview

- Major Components:
    - Lexer: a class that can hold the current token, its type and value, currently being processed by the syntax analyzer. It uses its setNext() function to go to the next token.
- Algorithm used:
    - Recursive Top Down Parsing
        - Process of calling parsing functions recursively to handle nested structures within the source code. When the parser encounters a non-terminal symbol (such as a production rule), it calls a parsing function corresponding to that symbol. If the parsing function for that non-terminal symbol encounters another non-terminal symbol within the grammar rule being parsed, it calls itself recursively to parse that nested structure.This recursive callback mechanism continues until all elements of the input string are parsed, or until an error condition is encountered.
        - For RTD Parsing to work, we must first refactor rules that have backtracking productions and rules that have a left recursion production.

Left Recursion:

Expression = Expression + Term | Expression - Term | Term

- Expression = Term * Expression'
- Expression' = +Term * Expression' | -Term * Expression' | ε

Term = Term * Factor | Term / Factor | Factor

- Term = Factor * Term'
- Term' = * Factor * Term' | / Factor * Term' | ε

Backtracking:

1. Function Definitions = <Function> | <Function> Function Definitions
    a. FunctionDefinitions = <Function> FunctionDefintions'
    b. FunctionDefinitions' = ε | <Function> FunctionDefinitions
2. Parameter List = <Parameter> | <Parameter> , <Parameter List>
    a. ParameterList = <Parameter> ParameterList'
    b. ParameterList' = ε | , <Parameter> ParameterList
3. Declaration List = <Declaration> ; | <Declaration> ; <Declaration List>
    a. DeclarationList = <Declaration> ; DeclarationList'
    b. DeclarationList' = ε | DeclarationList
4. Statement List = <Statement> | <Statement> <Statement List>
    a. StatementList = <Statement> StatementList'
    b. StatementList' = ε | StatementList
5. If = if (<Condition>) <Statement> endif | if (<Condition>) <Statement> else <Statement> endif
    a. If = if (<Condition>) <Statement> If'
    b. If' = endif | else <Statement> endif
6. Return = return ; | return <Expression> ;
    a. Return = return Return'
    b. Return' = ; | <Expression> ;

4. Shortcoming

- Our syntax analyzer lacks a complex error handling mechanism that is able to elaborate the cause of error when it occurs. Such as identifying the line number of the token and printing out parts of the expected output up 'til to the point where the error occured.