**CS323 Documentation**

About 2 pages

1. **Problem Statement**

*<write the problem statement here. You can mostly get it from the*

*assignment itself>*

In this assignment, we are tasked with writing a syntax analyzer using RDP. We are to rewrite the grammar Rat24F to remove any left recursion and use left factorization if necessary. The parser should be able to read a file containing the source code of Rat24F to generate all the production rules used for analyzing the tokens from the lexer function and write out the results to an output file.

1. **How to use your program**

*<write detailed steps how to execute your program>*

1. Download the zip folder and ensure there is the “main.py” file and there are 6 text files named “input(2-3).txt” and “output(2-3).txt”
2. Execute the main.py file, and it will read the “input.txt” run the lexer function and syntax analyzer, and store the results in “output.txt”
3. To test the other input text files, change the file input and output in the source code
   * + 1. Variables labeled Input\_file and Outout\_file change the input and outfile files written to and from
4. Open the output file to see your results
5. **Design of your program**

*< write major components of your program. Also, data structures you are utilizing, particular algorithms you have chosen etc. >*

When we were creating this program. We had to take into consideration left factorization and backtracking. In order to produce the expected output with the correct production rules, we had to remove left factorization and backtracking by creating prime functions of the rule that were affected by this. Furthermore, we had to create a total of 38 functions for each rule and a prime function for left recursion. The main algorithm we had to implement was recursion and calling the correct production rule. The lexer function is what parses through the list of tokens written in from an input file and writes out to another file the token, lexeme and production rules used for that token. It also keeps track of the line number that the parser is on.

1. **Any Limitation**

*<All features are running according to the assignment but you limit your program due to resource limitations, such as*

*Maximum number of lines in the source code, size of the identifier, integer etc.* ***Say ‘None’ if there is no limitation****>*

None

1. **Any shortcomings**

*<Anything you could NOT implement although that is required by the*

*Assignment.* ***Say ‘None’ if there is no shortcoming****>*

Any missing character at the very end of a line that would cause an error in the program (missing ; or closing bracket/parenthesis) is not correctly identified by its line actual line number and instead shows an error for the following line number.

Example - if line 2 was missing a ; at the end of the line, the program will say line 3 is missing a semicolon.

The error handler also does not correctly account for line number if a comment takes up an entire line by itself, and some very specific errors will not display an accurate error message but will still stop the program and display a different error message instead.