**COVER PAGE**

**CS323 Programming Assignment 2 Documentation**

Fill out all entries 1 - 7. If not, there will be deductions!

1. Names [ 1. Elena Marquez ] Section [ 2 ]

2. Assignment Number [ 2 ]

3. Due Date [ 11/10/24 ]

4. Submission Date [ 12/15/24 ] (emailed about this, I know it’s so late, but please see email from elena-marquez@csu.fullerton.edu

5. Executable File name [ MyAssignment2.py ]

(A file that can be executed without compilation by the instructor, such as .exe, .jar,

etc - NOT a source file such as .cpp )

6. Names of the testcase files - input test file output test file

test 1. [ test1.txt ] [ test1\_syntax\_output.txt ]

test 2. [ test2.txt ] [ test2\_syntax\_output.txt ]

test 3. [ test3.txt ] [ test3\_syntax\_output.txt ]

7. Operating System [ Windows ]

(Window – preferred)

**To be filled out by the Instructor:**

Comments and Grade:

**1. Problem Statement:**

The goal of this project is to develop a syntax analyzer for the RAT24F programming language using the Recursive Descent Parsing (RDP) technique. The syntax analyzer ensures that a given RAT24F source code adheres to the language's grammar rules by performing a top-down, recursive analysis. This parser identifies syntactic errors and generates an output file with the token and lexeme found, as well as all the production rules used for analyzing this token

**2. How to use your program:**

First, create a source code file containing RAT24F code or use a test case I provided. Place the input file in the program’s directory. Then, open a terminal or command prompt and navigate to the program folder. Execute the program using this command: ***python MyAssignment2.py***. Also, make sure that the python library is installed. It will prompt you to write the input file name and then will output what it is supposed to output.

**3. Design of your program:**

Lexer: The lexer component is adapted from Assignment 1, which performs lexical analysis by breaking the source code into a stream of tokens and lexemes. These tokens represent the fundamental building blocks of the RAT24F language

Parser: The Parser class is the heart of the syntax analyzer, designed to process the token stream from the lexer and determine if the source code adheres to the RAT24F grammar. It is implemented using Recursive Descent Parsing (RDP), a top-down parsing technique where each grammar rule corresponds to a recursive function. Each non-terminal in the RAT24F grammar has a dedicated function in the parser. For instance: <Expression> is handled by the expression() function, which further calls term() and expression\_prime() based on the grammar rules. <Statement> is handled by the statement() function, which branches to specific functions like while\_statement() or if\_statement() depending on the token type. The parser writes each production rule encountered to an output file, which provides a trace of the parsing process.

Error Handling: My program provides descriptive error messages for syntax violations.

Data Structures: The main data structure of my program is a list of Token objects that have token\_type and lexeme attributes.

**4. Any Limitation:**

There are no limitations that I know of.

**5. Any shortcomings:**

There are no shortcomings to the program that I know of.