**COVER PAGE**

**CS323 Programming Assignment 3 Documentation**

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

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

2. Assignment Number [ 3 ]

3. Due Date [ 12/13/24 ]

4. Submission Date [ 12/19/24 ] (emailed about this, I know it’s late, but please see email from elena-marquez@csu.fullerton.edu

5. Executable File name [ main.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. [ test.txt ] [ test\_output.txt ]

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

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

7. Operating System [ Windows ]

(Window – preferred)

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

Comments and Grade:

**1. Problem Statement:**

This assignment involves creating a parser for a simplified version of the Rat24F language. The parser must handle a symbol table to manage declared identifiers, check for errors like undeclared variables or type mismatches, and ensure semantic correctness. Additionally, the parser will generate assembly code using a stack-based virtual machine instruction set, outputting both the final assembly code listing and the symbol table.

**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 main.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 syntax analyzer designed to process the token stream from the lexer and determine if the source code adheres to the RAT24F grammar, mostly taken from Assignment 2. It is implemented using Recursive Descent Parsing (RDP), a top-down parsing technique where each grammar rule corresponds to a recursive function. Additional logic was added to maintain a symbol table, add assembly instructions, and perform logic checks for conditions.

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:**

I don’t know of any limitations.

**5. Any shortcomings:**

The one thing I couldn’t figure out was how to get the JUMP and LABEL instructions to output correctly, they kept giving me issues, and I couldn’t understand it. Everything else works, though.