Cover Page

CPSC 323- Programming Assignments

1. **Names and Section:**
    a. Riya Jain - 02 (riyajain@csu.fullerton.edu)
    b. Jim Alvarez - 02 (jimalvarez566@csu.fullerton.edu)
    c. Johnathan Fagoaga - 02 (foggy1365@csu.fullerton.edu)
2. **Assignment number:** 2
3. **Due Date:** 11/10/2024
4. **Submission Date:** 11/14/2024
5. **Executable File Name:** Project_1_335 -> syntax_analyzer.py
6. **Names of test case files:** input test file | output test file
    a. test 1. [source_code_input.txt] [output_file1.txt]
    b. test 2. [source_code_input2.txt] [output_file2.txt]
    c. test 3. [source_code_input3.txt] [output_file3.txt]
7. **Operating Systems:** MacOS

1. Problem statement:

a. The main objective of this project is to create a parser (syntax analyzer) for the RAT24F language. The parser will analyze tokens created to ensure they conform to the defined syntax, outputting the results to a dedicated output file. Along with the parsing output, it will also return the tokens.

2. How do you use our program:

a. Ensure that you have the lexer.py file, which is in our zip file as we decided to use our project1 github for project2, for easy access of the previous assignment (lexer.py).
b. Run the command: "python3 syntax_analyzer.py"
c. You will see the output for:
   i. "source_code_input.txt" in "output_file1.txt"
   ii. "source_code_input2.txt" in "output_file2.txt"
   iii. "source_code_input3.txt" in "output_file3.txt"

3. Design of our program:

a. Major Components:

i. Lexer (Lexical Analyzer):

1. The lexer scans the source code, identifying lexemes (e.g., keywords, identifiers, numbers, operators, separators) and categorizes them as tokens.
2. Using pattern matching, it processes the code and classifies lexemes, such as detecting keywords (function, if), integers, and separators.
3. The lexer outputs a structured list of tokens for the parser to analyze.

ii. Syntax Analyzer (Parser):

1. The parser applies recursive descent parsing on the token stream from the lexer, validating the syntax based on RAT24F grammar.
2. Each production rule is represented as a function, calling other rule functions recursively to verify syntax structure.
3. If syntax errors are found, the parser generates descriptive error messages with token type and lexeme information, logging them in the respective output file.

b. Data Structures:

- **i. Tokens:**
  - **1.** Tokens contain information such as type (e.g., keyword, identifier, number), lexeme, and sometimes line number to assist with error reporting.
- **ii. Token List:**
  - 1. The lexer produces an ordered list of tokens passed sequentially to the parser.
- **iii. Output Files:**
  - **1.** Each input file has an associated output file, recording tokenization and parsing logs, syntax matches, and any errors.

c. Algorithm Chosen:

- i. Regular Expressions (for Tokenization)
  1. The lexer utilizes regular expressions to classify lexemes, including:
     - a. Identifiers: *
     - b. Integers: d+
     - c. Real numbers: d+.d+
  2. Keywords are matched by exact string comparison.
- ii. Recursive Descent Parsing (for Syntax Analysis)
  1. The parser uses recursive descent parsing, implementing each production rule in RAT24F grammar as a function.
  2. Functions are recursively called to verify grammar structure, handling sub-rules where necessary.
- iii. Error handling and token Matching
  1. Errors are logged with detailed messages on expected and actual tokens.
  2. Token matching is performed via a `match()` function, advancing the token list only on successful matches.

4. Limitations

- a. Our program currently does not impose specific limits on the lengths of identifiers, integers, or real numbers. While Python supports large files and strings, memory constraints on the user’s machine may impact performance for very large inputs.

5. **Shortcomings**
    a. The code fully meets project requirements, producing a lexer and parser for RAT24F source files. Syntax errors are documented, and output files contain a comprehensive record of the analysis, fulfilling all specifications in the instructions.