Rat25F Syntax Analyzer – Base Documentation

1. Problem Statement

The objective of this project is to develop a syntax analyzer for the Rat25F programming language. The parser verifies whether the input source code follows the grammatical structure of Rat25F. It uses the lexer from Assignment 1 to obtain tokens and lexemes, printing both the recognized tokens and the production rules applied. If a syntax error occurs, the parser will later display an informative message including the token, lexeme, line number, and error type.

2. How To Use the Program

Compilation: Navigate to the project directory and compile using:

```
g++ -std=c++17 lexer.cpp parser.cpp main.cpp -o rat25f
```

Execution: Run the program with:

```
./rat25f
```

Input File: Create a text file named test_input.txt containing Rat25F source code or run the already existing small, medium or large files

Output: The program will create three output files (for the preexisting small, medium, large sizes):

- A test_small_output_parser.txt which shows tokens, lexemes and production rules, as well as test_medium_output_parser.txt and test_large_output_parser.txt.

Example Input File: (test_input.txt):

```
function convert (fahr integer)

{

  return 5 * (fahr - 32) / 9;

}
```

3. Design of your program

Components:

1. **Lexer Class:** The main component that processes input character by character
   - Private members: input, position, curr_line
   - Core method: getNextToken()
2. **Token Structure:** Simple data structure that contains:
   - TokenType type: Enum value representing each token's category

- std::string value: The actual text
- int line: Line number where token was found

3. TokenType Enum: enumeration of all possible token types in Rat25F

Data Structures:

- **std::string sourceCode:** Holds the entire source code for sequential scanning by the parser.
- **Token structure:** Defines a token's type, value, and line number for communication between the lexer and parser.
- **TokenType enum:** Enumerates all possible token categories (keywords, identifiers, operators, separators, etc.).
- **Lexer object reference:** Provides token data to the parser during analysis.
- **Parser class:** Functional component that manages recursive calls and token matching, built using the above data structures.

Key Functions:

- parse(): Starts the parsing process by calling the root grammar rule (Rat25F()) and manages error handling.
- nextToken(): Retrieves the next token from the lexer to continue syntactic analysis.
- match(TokenType expected): Checks whether the current token matches the expected type and advances if correct.
- error(std::string message): Displays an error message with the token, its lexeme, and line number, then stops the execution.
- Rat25F(): Entry point of the Rat25F grammar; represents the full program structure.

4. Any Limitations:

- **None**

5. Shortcomings:

- **None**