CS323
RAT25F Lexical Analyzer Documentation

1. Problem Statement

The objective of this project is implementing a LA for the Rat25F programming language. The lexer will:

- Read Rat25F source code from an input file
- Tokenize the input into lexical units including keywords, identifiers, integers, real numbers, operators, and separators
- Handle comments in double quotes by ignoring them
- Classify each token according to the Rat25F language
- Output a list of tokens with their types, lexemes, and line numbers
- Follow the lexical conventions where identifiers must start with a letter and can contain letters, digits, or '$' symbols
- Recognize all Rat25F keywords: function, integer, boolean, real, if, fi, else, return, get, put, while, true, false
- Handle both integer literals (sequences of digits) and real literals (containing decimal points)

2. How to use our program

To compile and run the lexical analyzer:

1. **Compilation**: Navigate to the project directory and compile using:

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

2. **Input File**: Create a text file named test_input.txt containing Rat25F source code or run the already existing small, medium or large files.
3. **Execution**: Run the program with:

   ./lexer

4. **Output**: The program will display a table that shows:
   - Line number where each token was found
   - Token type
   - The text of the token
5. **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::unordered_set<std::string**:> Used for efficient keyword lookup in O(1) time
- **std::vector**: Stores the complete list of generated tokens
- **std::string**: Holds the entire source code for parsing character by character

Algorithms:

- **Character-by-character scanning**: Linear traversal through input string
- **Finite State Machine approach**: Used for recognizing different token patterns
- **Keyword lookup**: Hash-based lookup for distinguishing keywords from identifiers
- **Pattern recognition**: Different recognition logic for identifiers, numbers, operators, and separators

Key Functions:

- getCurrentChar(): Retrieves current character WITHOUT advancing
- advance(): Moves to next character and handles line counting
- isLetter(), isDigit(): Character classification, self-explanatory
- isKeyword(): Determines if a word is in our reserved keywords list

4. Any Limitation

- **Maximum input file size**: Limited by available system memory as entire file is loaded into a string
- **Identifier length**: No explicit limit mentioned, constrained only by potential memory usage
- **Integer size**: Limited to values that fit in integer notation
- **Nesting depth**: No limit on brace or parenthesis nesting levels
- **Comment handling**: Comments must be closed with matching quotes
- **Line length**: No restriction on individual line length, vague

5. Any shortcomings

None