CS 424 - Compiler Construction

Assignment # 1

Abdullah Farooq - 2020023

This report analyzes the provided MiniLang tokenizer code, explaining its design decisions, structure, and how to run it. It also includes test cases demonstrating its capabilities and edge case handling.

Design Decisions:

1. **Regular Expressions:** The code leverages multiple regular expressions to match keywords, data types, operators, numbers, identifiers, and other relevant patterns. This approach simplifies tokenization and avoids complex conditional logic.
2. **Combined Pattern:** Instead of using separate searches for each pattern, the code combines them into a single regular expression r'//.*?$|/*.*?*/|\b\w+\b|[^\w\s]. This optimizes tokenization by performing a single scan instead of multiple.
3. **Categorization:** Each token is categorized based on the matching regular expression using an if-elif structure. This assigns meaningful labels to each token, aiding further analysis or compilation.

Structure:

1. **Imports:** The code imports the re module for regular expression processing
2. **Regular Expressions:** It defines separate regular expressions for each token type.
3. **Combined Pattern:** The combined pattern is created using the re.findall function with appropriate flags (re.MULTILINE and re.DOTALL).
4. **Tokenized Program:** The program is tokenized, and the results are printed.
5. **Categorization:** Each token is categorized based on its matching pattern and printed with its label.

Running the Program:

1. Save the code as a Python file (e.g., 2020023_A1.py).
2. Ensure the ‘MiniLang.code’ file exists and contains MiniLang code.
3. Execute the script using Python: python 2020023_A1.py.
4. The output will display the list of tokens along with their categorizations.

Test Cases:

1. **Basic Syntax:** Ensure the code correctly identifies keywords, data types, operators, numbers, identifiers, and comments.
2. **Edge Cases:** Test with strings containing special characters, empty lines, and mixed data types.
3. **Numbers:** Confirm proper categorization of integers and non-integers (e.g., floating-point).
4. **Identifiers:** Verify handling of valid and invalid identifiers (e.g., reserved words, keywords).

Limitations:

1. This basic tokenizer does not handle nested comments or multi-line strings.
2. Token categorization could be expanded to include additional types (e.g., keywords for specific languages).

Future Improvements:

1. Enhance error handling to provide informative messages for unexpected input.
2. Implement lexer state management to handle complex structures like nested comments or strings.
3. Expand token categorization for languages with more features or data types.

Conclusion:

This basic MiniLang tokenizer provides a foundation for lexical analysis. It effectively tokenizes the code and assigns meaningful categories using regular expressions. Further improvements can add error handling, support complex structures, and cater to specific language features.

Output: