GHULAM ISHAQ KHAN INSTITUTE OF SCIENCE AND TECHNOLOGY

Assignment 1

CS424

Afeef Ejaz 2020040

Scanner Structure:

Token Class:

- Represents a token with attributes for token type and lexeme.

Scanner Class:

- Implements the scanner functionality.
- Initializes with the input filename.

Defines token patterns and corresponding token types.

- Scans the input file, tokenizes the code, and handles errors.
- Outputs the list of tokens.

Design Decisions:

Token Types and Regular Expressions:

- Defined token types based on MiniLang specifications.
- Used regular expressions to match token patterns.

Finite State Machine (FSM):

- Implemented a simple FSM to tokenize MiniLang source code.
- Used regular expressions to define transitions between states.

Error Handling:

- Detected and reported lexical errors, such as invalid symbols or malformed identifiers.
- Errors are reported with line numbers for easy debugging.

Input/Output:

- Input is taken from a file containing MiniLang source code.
- Output is a list of tokens, each with its type and lexeme.

How to Run the Program:

Save the Script:

- Save the provided Python script in a file (e.g., scanner.py).

Run the Script:

- Open a terminal or command prompt.
- Navigate to the directory containing the script.

Execute the Script:

- Run the script using the command: python scanner.py.
- Enter the filename of the MiniLang source code when prompted.

View Output:

- The scanner will tokenize the MiniLang code and output the list of tokens.
- Lexical errors, if any, will be reported with line numbers.

Test Cases:

Basic MiniLang Code:

- Includes variable assignments, if-else conditions, arithmetic operations, and valid keywords.
- Tests the basic functionality of the scanner.

Edge Cases:

- Intentional errors such as invalid symbols, malformed identifiers, and unexpected characters.
- Tests error handling and reporting capabilities of the scanner.

Large File:

- Test the scanner with a larger MiniLang source code file to ensure efficient tokenization.

Example test cases

1.

```
x = 10

if x == 10

   print "x is 10"

else

   print "x is not 10"
```

2.

```
x = 10
!if x == 10
    print "x is 10"
```

3.

```
x = 10
y = true

if x == 10
    print "x is 10"
else
    print "x is not 10"

z = x + 5
```

Report Conclusion:

The scanner for MiniLang is designed to efficiently tokenize source code while handling lexical errors effectively. It follows a structured approach with clear design decisions to ensure ease of use and maintainability. The provided test cases demonstrate its capabilities across various scenarios, ensuring robustness and reliability in tokenizing MiniLang code.