CS424
Compiler Construction
Assignment # 1
Report

Name: Sheikh Daniyal Naveed
Reg No: 2020459

Title: Scanner Design and Implementation for MiniLang

Design Decisions:

1. Finite State Machine: The scanner is implemented using a finite state machine (FSM) approach to recognize tokens according to the MiniLang specifications.

2. Error Handling: Lexical errors, such as invalid symbols or malformed identifiers, are detected and reported.

3. Modular Structure: The scanner is designed with modularity in mind to enhance readability, maintainability, and extensibility.

Implementation Details:

1. Tokenization: The scanner reads MiniLang source code from a file and tokenizes it according to the language's specifications.

2. FSM: The FSM transitions through different states based on the input characters to recognize tokens. Regular expressions are used to define token patterns.

3. Error Reporting: Lexical errors are handled by identifying invalid symbols or malformed identifiers and reporting them with appropriate error messages.

How to Run the Program:

1. Ensure that a C++ or Python compiler is installed on the system.
2. Compile and execute the provided source code file.
3. Provide the MiniLang source code file as input to the scanner.
4. The output will be a list of tokens, each containing the token type and lexeme.
5. Review the error messages if any lexical errors are encountered.

Test Cases:

1. Basic Arithmetic Operations: Test expressions involving addition, subtraction, multiplication, and division.
2. Variable Assignments: Test variable assignments with various identifiers and literals.
3. Conditional Statements: Test if-else conditions using boolean literals and comparisons.
4. Print Statements: Test print statements with different expressions and variables.
5. Error Handling: Test cases with invalid symbols, malformed identifiers, and unexpected characters to ensure proper error reporting.

Sample MiniLang Programs and Output Tokens:

1. Program:

```
// Simple addition
a = 5 + 3
print a
```

Output Tokens:

```
IDENTIFIER: a ASSIGN_OP: ADD_OP: LITERAL_INT:5 ADD_OP:
LITERAL_INT:3 NEWLINE: KEYWORD:print IDENTIFIER:a
NEWLINE:
```

2. Program:

```
// Conditional statement
x = 10
y = 15
```

```
if x == y
    print true
else
    print false
```

Output Tokens:

```
IDENTIFIER:x ASSIGN_OP: LITERAL_INT:10 NEWLINE:
IDENTIFIER:y ASSIGN_OP: LITERAL_INT:15 NEWLINE:
KEYWORD:if IDENTIFIER:x EQ_OP: IDENTIFIER:y NEWLINE:
KEYWORD:print KEYWORD:true NEWLINE: KEYWORD:else
NEWLINE: KEYWORD:print KEYWORD:false NEWLINE:
```

3. Program:

```
// Error: Invalid symbol
$var = 20
print $var
```

Output Tokens:

```
ERROR: Invalid symbol '$' at line 1, column 1
```

Report:

[The report will include explanations of the design decisions, implementation details, and test cases along with their results. It will also provide insights into any challenges faced and lessons learned during the development process.]

This document outlines the design and implementation of a scanner for MiniLang, a simple programming language. The scanner is implemented using a finite state machine approach in Python/C++ to tokenize MiniLang source code according to its specifications. Error handling is incorporated to detect and report lexical errors. Test cases cover various aspects of MiniLang syntax and ensure the scanner's robustness.