CS-424 Compiler Construction

Assignment #2

Report Author: [Ehsan Elahi].

Registration Number: [2020111].

Introduction:

This report documents the design, implementation specifications, and testing of a scanner and parser for MiniLang, a lightweight instructional language. MiniLang supports basic arithmetic functions, variable functions, if-else statements, and printing.

Scanner Design and Implementation:

The scanner utilizes regular expressions to identify different token types in MiniLang source code. These token types include:

- Integers (e.g., 123, -45)
- Identifiers (variable names starting with a letter followed by alphanumeric characters)
- Operators (+, -, *, /, =, ==, !=)
- Keywords (if, else, print)
- Comments (lines starting with "//")
- Whitespace (spaces, tabs)

The scanner iterates through the source code in rows, trying to match each spectrum to the default regular expressions for each token type. On a successful match, a Token object is created with the token type, the lexeme (the literally matching string), and the line number where it was found.

Parser Design and Implementation:

The parser employs a **top-down** approach to analyze the structure of MiniLang code based on predefined grammar rules. Here's a simplified overview:

- The parser receives a list of tokens from the scanner.
- It iterates through the tokens and attempts to match them against grammar rules for different statements and expressions.
- Grammar rules are implemented using functions that handle specific language constructs.
- For example, a function might parse an assignment statement by checking for an identifier followed by an equal sign and an expression.

Link to Repo:

https://github.com/RaoEhsanElahi/MiniLang_Parser.git

Running the Program

The specific instructions for running the program will depend on the implementation. However, a general outline might involve:

- Setting up the environment: Install any required dependencies (mentioned in the README.md).
- Executing the script: Locate the main script (e.g., compiler_Assignment_2.ipynb) and run it.
- Providing input: The program might prompt a MiniLang source code file or accept it as a command-line argument.

Test Cases:

Here are some test cases demonstrating the parser's capabilities and edge cases:

Basic Functionality:

- **Test 1:** x = 5; print(x); (Valid assignment and print statement)
- **Test 2:** if (a > b) { print("a is bigger"); } else { print("b is bigger"); }

  (Valid if-else statement with comparison).

Edge Cases:

- **Test 3:** x = y + (Missing operand after operator - syntax error).
- **Test 4:** print(undefined_variable); (Using an undeclared variable - potential error).
- **Test 5:** if true { print("always true"); } (Valid but redundant condition).

Note:

The test cases can be in addition accelerated to cover diverse situations like complicated expressions, nested statements, and different combos of language features. The parser has to ideally take care of these part cases by throwing exceptions or providing informative error messages.

Conclusion:

This venture efficaciously developed a simple scanner and parser for MiniLang, an academic language. The implemented scanner successfully identifies tokens like integers, identifiers, operators, and key phrases. The parser utilizes a top-down technique to analyze MiniLang code structure, managing essential statements like assignments, if-else constructs, and printing.