Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi

CS 424 (Compiler Construction)

Assignment 1

Submitted by

Zartaj Asim 2020526

Submitted to

Mr. Usama Arshad

1. Introduction

Assignment 1 requires making a MiniLang Scanner by implementing a finite state machine in C++ or python. For my task ive dome the implementation using Python to recognize tokens defined in MiniLang' s specifications.

2. Design

The implementation utilizes a finite state machine to recognize and tokenize MiniLang source code. The states include whitespace and comment skipping, identifier matching, literal matching, and operator matching.

2.1. Scanner Specifications

The scanner is designed to recognize the following elements in MiniLang.

- Data Types: Integer, Boolean.
- Operators: + (addition), - (subtraction), * (multiplication), / (division), = (assignment), == (equality), != (inequality).
- Keywords: if, else, print, true, false.
- Identifiers: Variable names starting with a letter followed by any combination of letters and digits.
- Literals: Integer literals, Boolean literals (true, false).
- Comments: Single-line comments starting with //.

2.2. Tokenization

The scanner tokenizes the source code into the following token types:

- Keyword
- Identifier
- Integer Literal
- Boolean Literal
- String Literal
- Operator

3. Implementation details

Simply run the. ipynb file on Collab or Jupyter Notebook. Make sure all the test cases files are also in the same directory.

3.1. Language Choice

The MiniLang scanner is implemented in Python.

3.2. Design

Regular expressions are used to match identifiers, literals, operators, and to skip whitespace and comments.

- The use of a set (self.keywords) is employed to quickly check if an identifier is a keyword.
- The scanner uses a list (self.tokens) to store the identified tokens.
- The _skip_whitespace_and_comments function efficiently skips over whitespaces and comments.

3.3. Tokenization

The scan method is responsible for tokenizing the MiniLang source code. It iterates through the source code, skipping whitespaces and comments using skip_whitespace_and_comments.

3.4. Token Types

Token types such as keywords, identifiers, literals, and operators are determined using regular expressions in methods like _match_identifier, _match_literal, and _match_operator.

3.5. Error handling

The scanner can recognize and report lexical errors, such as invalid symbols or malformed identifiers. The _report_error method is designed to raise a ValueError when an invalid token is encountered. Errors include details about the error type and position

4. Test Cases

- **Case 1: Testing IF-Else Code**

  Tested the scanner's ability to tokenize a basic if-else code structure in MiniLang.
- **Case 2: Testing != (not equal to) operator**

  Tested the scanner's handling of the != (not equal) operator within an if statement.
- **Case 3: Testing Boolean Expression**

  Tested the scanner's ability to handle Boolean expressions within an if statement.
- **Case 4: Testing Lexical Error**

  Tested the scanner's ability to detect and report lexical errors in the MiniLang code.

5. Edge Cases

- **Edge Case 1: Empty File**

Input: An empty file.

Expected Output: No tokens.

- **Edge Case 2: File with Only Whitespace**

  Input: File with only whitespace characters.

  Expected Output: No tokens.
- **Edge Case 3: File with Comments Only**

  Input: File with only comments.

  Expected Output: No tokens.
- **Edge Case 4: Long Identifier File**

  Input: File with a long identifier.

  Expected Output:

  Identifier: a1234567890123456789012345678901234567890

  Operator: =

  Integer Literal: 42

6. Conclusion

The MiniLang scanner demonstrates correct tokenization for various test cases, including basic code structures, Boolean expressions, and handling lexical errors. The design and implementation ensure the scanner's accuracy and robustness across different scenarios.