CS-424 Compiler Construction

Assignment #1 Report

Author: [Ehsan Elahi].

Registration Number: [2020111]

1. Design Decisions:

Language Specifications:

The scanner was designed to adhere closely to the language specifications provided for MiniLang, encompassing data types, operators, keywords, identifiers, literals, and comments.

Finite State Machine (FSM):

Employing a Finite State Machine (FSM) approach, the scanner transitions between states based on regular expressions matching the input, with each state corresponding to a specific token type.

Error Handling:

The scanner is equipped to detect and report lexical errors such as invalid symbols or malformed identifiers, providing informative messages for debugging purposes.

C Implementation:

C language was chosen for its efficiency and suitability for system-level programming.

2. Scanner Structure:

The scanner is implemented using Flex, a lexical analyzer generator. It consists of rules defined in a Flex file (**scanner.l**). Regular expressions are used to match different token types, including integers, identifiers, operators, keywords, and comments. Each rule returns the corresponding token type.

3. How to Run the Program:

To execute the scanner:

4. Test Cases:

Valid Input:

Test the scanner with MiniLang source code containing valid tokens, encompassing various data types, operators, keywords, identifiers, literals, and comments.

Invalid Input:

Test the scanner with input files containing invalid symbols, malformed identifiers, and other lexical errors. Ensure appropriate error messages are displayed.

Edge Cases:

Include test cases with edge scenarios such as empty input files, files with single-line and multi-line comments, and files with a mix of different token types in various orders.

5. Conclusion:

The scanner for MiniLang has been successfully designed and implemented according to the provided specifications. It showcases the capabilities of lexical analysis and serves as a foundational component for subsequent stages of compiler design.

By adhering to the provided instructions, we can gain valuable insights into the complexities of lexical analysis, explore tokenization processes, and develop a deeper understanding of compiler construction principles.