**CS-424(Compiler Construction)**

**Assignment#2**

**Syed Ammar Bin Farrukh**

**Reg#2020468**

1. **Design Decisions:**

**Language Specifications**:

The parser adheres closely to the grammar and language specifications provided for MiniLang, encompassing statements, expressions, conditional structures, and print functionality.

**Finite State Machine (FSM)**:

The parser includes basic error handling mechanisms to detect and report syntax errors, such as incomplete statements, invalid keywords, and incorrect usage of language constructs.

**Error Handling**:

The parser is structured with distinct methods for parsing different elements of the language, promoting modularity and ease of maintenance.

**Modularity:**

The parser is structured with distinct methods for parsing different elements of the language, promoting modularity and ease of maintenance.

**Extensibility:**

The design allows for easy extension and modification to accommodate future language enhancements or changes in the grammar.

**Python Implementation**:

Python was chosen for its simplicity and readability, making it suitable for implementing a parser efficiently.

1. **Parser Structure:**

**Constructor (\_\_init\_\_):**

Initializes the parser with a list of tokens generated by the scanner.

Sets the current token index to zero.

**parse**():

Entry point for parsing. Iterates through the token list and calls parse\_statement() to process each statement.

**match(token\_type):**

Checks if the current token matches the specified type. Advances the token index if a match is found.

**parse\_expression():**

Placeholder for parsing expressions. Requires implementation based on the grammar of the language.

**parse\_statement():**

Parses different types of statements based on keywords and identifiers. Handles if-else conditions and print statements.

**parse\_if\_else():**

Handles parsing of if-else statements, including condition evaluation and nested statements.

**parse\_print():**

Parses print statements, validating the identifiers to be printed.

1. **How to Run the Program:**

* Save the provided Python code in a file named scanner.py.
* Ensure you have Python installed on your system.
* Run the program by executing the command python scanner.py.
* Enter the filename of the MiniLang source code file when prompted.
* The parser will tokenize the input file using the scanner and attempt to parse the tokens.

1. **Test Cases:**

**Example Test Cases:**

**Valid Input**:

* Test the parser with MiniLang source code containing valid statements, expressions, and control structures.
* Ensure the parser executes without errors and produces the expected output.

**Invalid Input**:

* Test the parser with input files containing syntax errors, incomplete statements, or invalid constructs.
* Verify that the parser detects and reports syntax errors accurately.

**Edge Cases**:

* Include test cases with edge scenarios such as empty input files, files with single-line and multi-line comments, and complex nested statements.
* Assess the parser's ability to handle diverse code structures and unexpected inputs gracefully.

1. **Conclusion:**

The parser for MiniLang has been designed and implemented according to the provided specifications, demonstrating the fundamental principles of parsing and syntax analysis. By following the instructions provided, users can evaluate the parser's correctness, robustness, and adherence to the language grammar. Additionally, the parser serves as a crucial component in the compiler construction process, laying the groundwork for subsequent stages of code generation and optimization.