**Sameer Arif Khan 2020430**

**Compiler Construction CS-424**

**Assignment 2**

**MiniLang Scanner Documentation**

**Design and Implementation of a Parser for MiniLang**

**Design Decisions:**

In designing the parser for MiniLang, several decisions were made to ensure its effectiveness and adherence to the language specifications:

1.**Parser Type:** A top-down recursive descent parser was chosen for its simplicity and suitability for MiniLang's grammar complexity.

2.**Grammar Rules:** The grammar rules were defined to cover all language constructs, including arithmetic expressions, variable assignments, conditional statements, and print statements.

3.**Abstract Syntax Tree (AST):** The parser constructs an AST representing the hierarchical structure of the MiniLang source code. Each node in the tree corresponds to a language construct, facilitating further analysis or interpretation.

4.**Error Handling:** Error handling mechanisms were implemented to detect and report syntax errors encountered during parsing. Meaningful error messages are provided, aiding developers in identifying and correcting issues in their MiniLang code.

**Parser Implementation:**

The parser is implemented in Python using a class-based approach. The main components of the parser include:

1.**Token Class:** Represents individual tokens with attributes for type and lexeme.

2.**Parser Class:** Contains methods corresponding to grammar rules, such as `program`, `statement\_list`, `assignment`, `conditional`, `print\_statement`, `expression`, `term`, `factor`, `unary`, and `primary`. These methods recursively parse the input tokens and construct the AST.

3.**Error Handling:** The `error` method raises a `SyntaxError` with a meaningful message whenever a syntax error is encountered during parsing.

**How to Run the Program:**

1. Ensure the Python file containing the parser implementation is saved (e.g., `minilang\_parser.py`).

2. Prepare MiniLang source code in a file (e.g., `example.minilang`).

3. Import the parser module into your Python script or interactive session.

4. Call the parser's `parse` method with the MiniLang source code tokens obtained from the scanner.

5. Handle any syntax errors raised by the parser, providing feedback to the user.

**Test Cases**

The scanner is tested with the following test cases:

1. **tc\_1.minilang**: Tests simple arithmetic and variable assignment.

2. **tc\_2.minilang**: Tests invalid syntax.

**Conclusion:**

By executing the parser with these test cases, we validate its correctness and ensure it accurately parses MiniLang source code according to the defined grammar rules.

This parser serves as a fundamental component in the compilation process, enabling further stages such as semantic analysis, optimization, and code generation for MiniLang programs.