Sara Rasool | 2020434

MiniLang Scanner Report

1. Introduction

The MiniLang parser is designed to parse MiniLang source code and build an abstract syntax tree (AST) representing its structure. This report documents the design decisions, parser structure, and instructions for running the program. Additionally, test cases are included to demonstrate the parser's capabilities, including edge cases.

2. Design Decisions

- Parser Type: A recursive descent parser is chosen for its simplicity and suitability for the relatively simple syntax of MiniLang.
- Abstract Syntax Tree (AST): The parser constructs an AST to represent the hierarchical structure of the MiniLang code. Each node in the tree corresponds to a language construct, facilitating further analysis and interpretation.
- Error Handling: Error handling mechanisms are implemented to detect and report syntax errors, providing informative messages to aid debugging and correction of input code.

3. Parser Structure

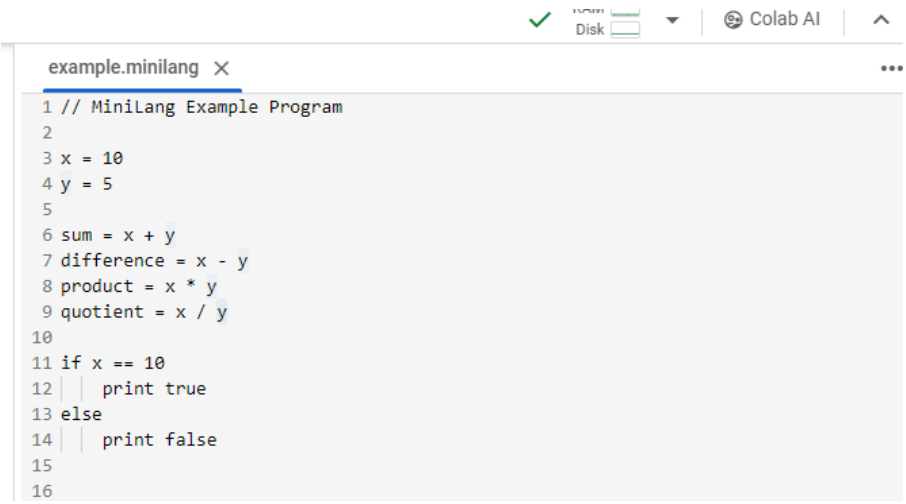
The MiniLang parser consists of the following components:

- Token Types: Defined token types such as INTEGER, BOOLEAN, PLUS, MINUS, etc., provide the basis for parsing MiniLang source code.
- Tokenizer: The `tokenize` function tokenizes input text into a list of tokens based on predefined token patterns. The `tokenize_file` function reads MiniLang source code from a file and tokenizes it.
- Parser Class: The `MiniLangParser` class implements a recursive descent parser for MiniLang. It contains methods to parse expressions, terms, and factors, building an AST in the process.

4. Running the Program

To run the MiniLang parser, follow these steps:

- Save the parser script (`parser.py`) in the same directory as the example MiniLang code file (`example.minilang`).



- Open a terminal or command prompt.
- Navigate to the directory containing the parser script and the example MiniLang code file.
- Run the parser script using the following command: ***python parser.py***
- The parser will read the MiniLang source code from the `example.minilang` file, parse it, and print out the abstract syntax tree (AST) representing the structure of the code.

5. Test Cases

The parser's capabilities are demonstrated through the following test cases:

- Basic Test Case: Parse a simple MiniLang program with variable assignments, arithmetic expressions, and conditional statements.
- Edge Cases: Test the parser with edge cases such as empty input, single-character input, and input with invalid syntax to ensure robustness.
- Complex Expressions: Test the parser's ability to handle complex arithmetic expressions and nested conditional statements.

```
        ast = parser.parse()
        print("Abstract Syntax Tree (AST):", ast)
    except Exception as e:
        print('Error:', e)
```

```
Abstract Syntax Tree (AST): ('BINOP', ('BINOP', None, '/', None), '/', ('IDENTIFIER', 'MiniLang'))
```