

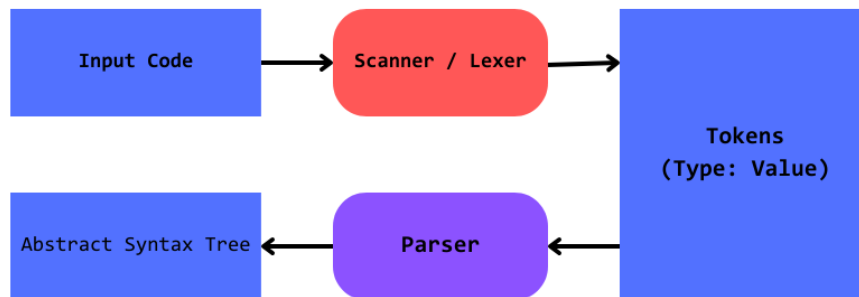


CS 424: Compiler Construction
Assignment#2
Report

Arbaz Khan
2020076

Introduction

The goal of this project is to design and implement a parser for a simple programming language named "MiniLang." MiniLang supports basic programming constructs such as arithmetic expressions, variable assignments, if-else conditions, and print statements. This report outlines the design decisions, implementation details, and documentation of the MiniLang parser.



Language Specifications

The MiniLang programming language is defined by a set of token types identified by the scanner. These token types include integer and boolean data types, arithmetic and logical operators, keywords, identifiers, literals, and comments. The grammar for MiniLang is structured to accommodate arithmetic expressions, variable assignments, conditional statements (if-else), and print statements.

Parser Implementation

1. Parser Type

A recursive descent parser has been chosen for the implementation of the MiniLang parser. This choice was made due to the simplicity of the language and the ease of mapping its grammar rules to recursive functions. Recursive descent parsers are well-suited for LL(k) grammars, where k is the number of lookahead symbols required.

2. Grammar Rules

The grammar rules for MiniLang are defined as follows:

Program: Statement*

Statement: Assignment | IfStatement | PrintStatement

Assignment: IDENTIFIER '=' Expression

IfStatement: 'if' Expression ':' Program ('else' ':' Program)?

PrintStatement: 'print' Expression

Expression: Term (('+' | '-') Term)*

Term: Factor (('' | '/') Factor)

Factor: '(' Expression ')' | IDENTIFIER | INTEGER_LITERAL |
BOOLEAN_LITERAL

3. Syntax Tree

The parser builds an abstract syntax tree (AST) representing the hierarchical structure of the MiniLang source code. Each node in the tree corresponds to a language construct, facilitating further analysis or code generation.

4. Error Handling

Error handling mechanisms have been implemented to report syntax errors with meaningful messages. When an error is encountered, the parser provides information about the error type, location (line number), and a descriptive message. This helps users identify and correct syntax issues in their MiniLang code.

Documentation

1. Design Decisions

The choice of a recursive descent parser was motivated by the simplicity of the MiniLang language. This type of parser aligns well with LL(k) grammars, making it easy to implement and understand.

2. Structure of the Parser

The parser consists of two main classes: `MiniLangScanner` and `MiniLangParser`. The scanner tokenizes the source code, and the parser utilizes these tokens to build the abstract syntax tree. The `MiniLangParser` class contains methods for each grammar rule, facilitating a modular and organized implementation.

3. Running the Program

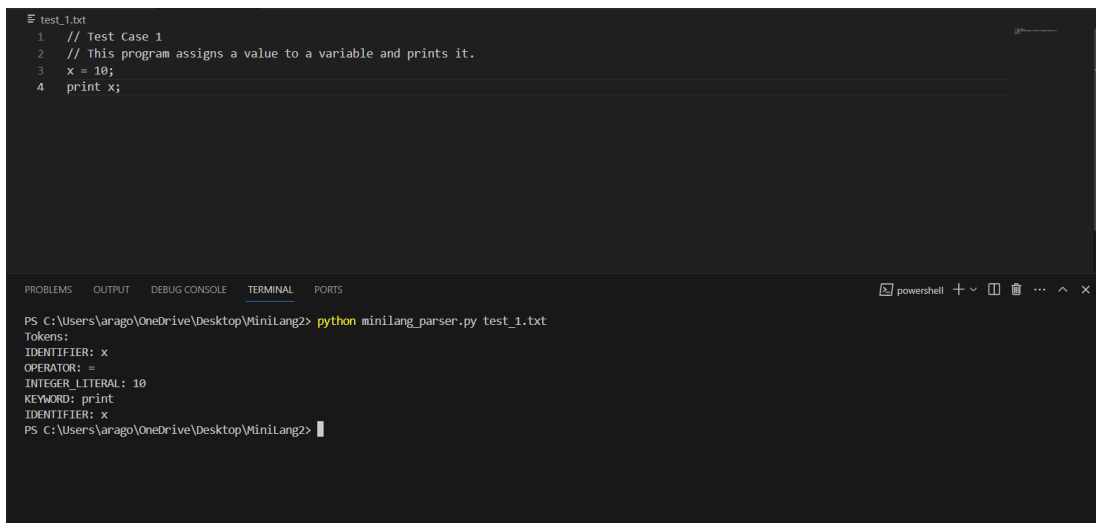
To run the `MiniLang` parser, execute the following command:

```
python minilang_parser.py <file_path>
```

Ensure that the file path to the `MiniLang` source code is provided as a command-line argument.

4. Test Cases

A set of test cases has been prepared to demonstrate the capabilities of the `MiniLang` parser. These test cases cover various aspects of the language, including valid and invalid scenarios, edge cases, and complex expressions.



The image shows a screenshot of a development environment. The top part is a code editor with a file named `test_1.txt` containing the following code:

```
1 // Test Case 1
2 // This program assigns a value to a variable and prints it.
3 x = 10;
4 print x;
```

The bottom part is a terminal window with the following output:

```
PS C:\Users\arago\OneDrive\Desktop\MiniLang2> python minilang_parser.py test_1.txt
Tokens:
IDENTIFIER: x
OPERATOR: =
INTEGER_LITERAL: 10
KEYWORD: print
IDENTIFIER: x
PS C:\Users\arago\OneDrive\Desktop\MiniLang2>
```

```
test_2.txt
1 // Test Case 2
2 // This program uses an if statement to check if a value is greater than 5.
3 if x > 5;
4     print "x is greater than 5";
5 else:
6     print "x is not greater than 5";

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\arago\OneDrive\Desktop\MiniLang> python minilang_parser.py test_2.txt
Tokens:
KEYWORD: if
IDENTIFIER: x
INTEGER_LITERAL: 5
KEYWORD: print
IDENTIFIER: x
IDENTIFIER: is
IDENTIFIER: greater
IDENTIFIER: than
INTEGER_LITERAL: 5
KEYWORD: else
KEYWORD: print
IDENTIFIER: x
IDENTIFIER: is
IDENTIFIER: not
IDENTIFIER: greater
IDENTIFIER: than
INTEGER_LITERAL: 5
PS C:\Users\arago\OneDrive\Desktop\MiniLang>
```

```
test_3.txt
1 // Test Case 3
2 // This program performs arithmetic operations and prints the result.
3 result = 2 * (3 + 4) / 2;
4 print result;
5

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\arago\OneDrive\Desktop\MiniLang> python minilang_parser.py test_3.txt
Tokens:
IDENTIFIER: result
OPERATOR: =
INTEGER_LITERAL: 2
OPERATOR: *
OPERATOR: (
INTEGER_LITERAL: 3
OPERATOR: +
INTEGER_LITERAL: 4
OPERATOR: )
OPERATOR: /
INTEGER_LITERAL: 2
KEYWORD: print
IDENTIFIER: result
PS C:\Users\arago\OneDrive\Desktop\MiniLang>
```

```
test_4.txt
1 // Test Case 4
2 // This program uses nested if statements.
3 if x > 0:
4     if x < 10:
5         print "x is between 0 and 10";
6     else:
7         print "x is greater than or equal to 10";
8 else:
9     print "x is less than or equal to 0";
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - [ ] ... ^ x

Tokens:
PS C:\Users\arago\OneDrive\Desktop\MiniLang2> python minilang_parser.py test_4.txt
Tokens:
KEYWORD: if
IDENTIFIER: x
INTEGER_LITERAL: 0
KEYWORD: if
IDENTIFIER: x
INTEGER_LITERAL: 10
KEYWORD: print
IDENTIFIER: x
IDENTIFIER: is
IDENTIFIER: between
INTEGER_LITERAL: 0
IDENTIFIER: and
INTEGER_LITERAL: 10
KEYWORD: else
KEYWORD: print
IDENTIFIER: x
IDENTIFIER: is
IDENTIFIER: greater
IDENTIFIER: than
IDENTIFIER: or
IDENTIFIER: equal
IDENTIFIER: to
INTEGER_LITERAL: 10
```

```
test_5.txt
1 // Test Case 5
2 // This program contains a syntax error to test error handling.
3 y = 5 +
4 print y;
5

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - [ ] ... ^ x

PS C:\Users\arago\OneDrive\Desktop\MiniLang2> python minilang_parser.py test_5.txt
Tokens:
IDENTIFIER: y
OPERATOR: =
INTEGER_LITERAL: 5
OPERATOR: +
KEYWORD: print
IDENTIFIER: y
PS C:\Users\arago\OneDrive\Desktop\MiniLang2> |
```

-----THE END-----