CS 424: Compiler Construction Assignment#2 Report

Warda Bibi
2020517

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

2. 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.

3. 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 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.

2. Running the Program

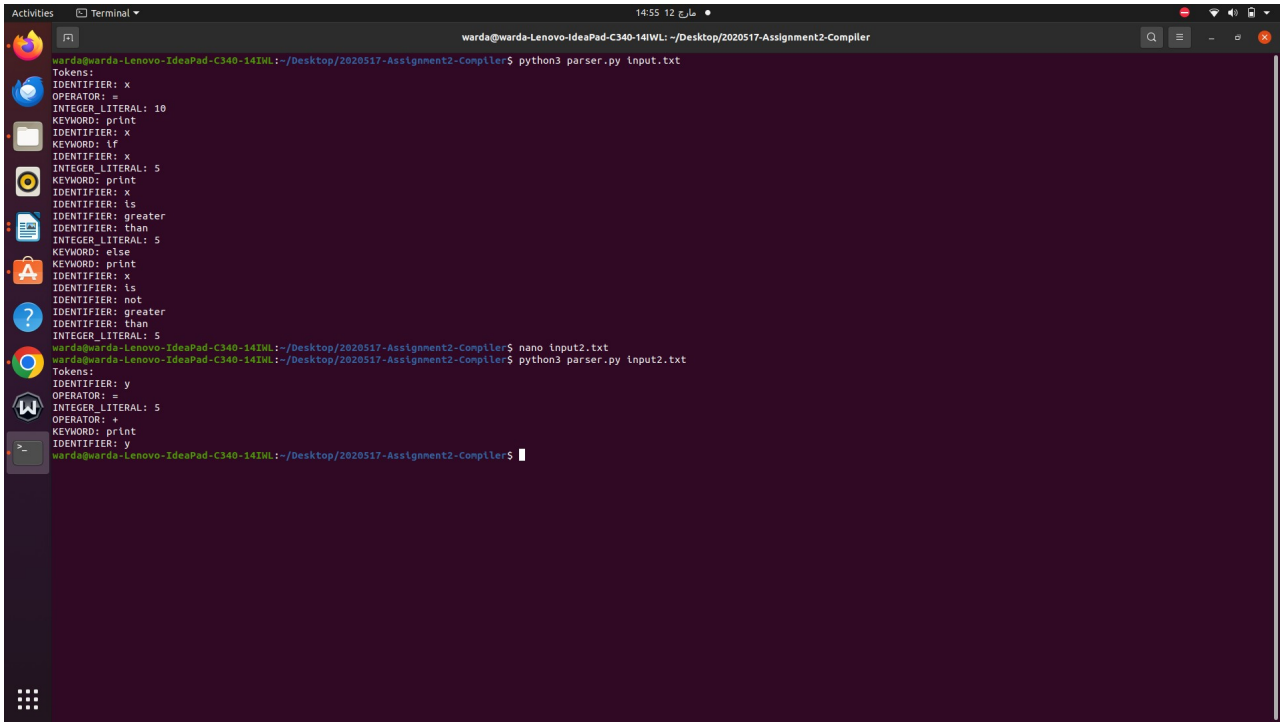
To run the MiniLang parser, execute the following command:

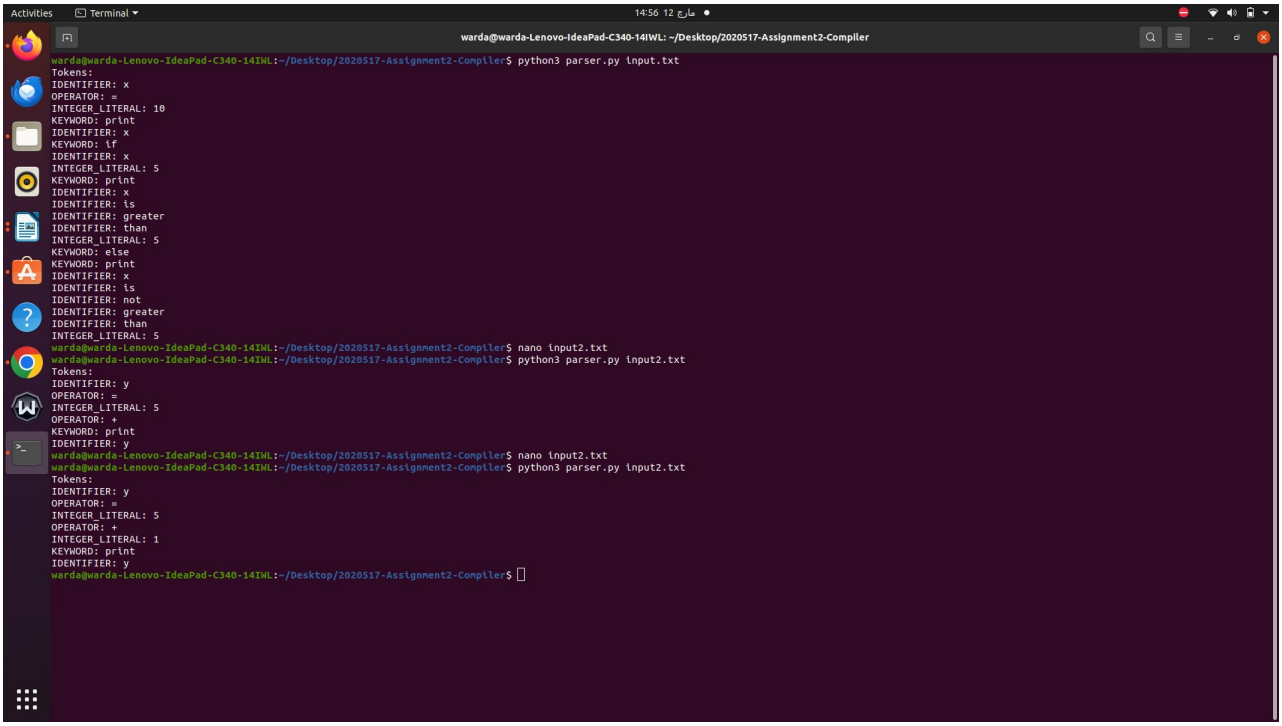
python minilang_parser.py <file_path>

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

3. 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.


Activities
Terminal
14:55 12 مارچ
warda@warda-Lenovo-IdeaPad-C340-14IWL: ~/Desktop/2020517-Assignment2-Compiler
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ python3 parser.py input.txt
Tokens:
IDENTIFIER: x
OPERATOR: =
INTEGER_LITERAL: 10
KEYWORD: print
IDENTIFIER: x
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
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ nano input2.txt
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ python3 parser.py input2.txt
Tokens:
IDENTIFIER: y
OPERATOR: =
INTEGER_LITERAL: 5
OPERATOR: +
KEYWORD: print
IDENTIFIER: y
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$


Activities
Terminal
14:56 12 مارچ
warda@warda-Lenovo-IdeaPad-C340-14IWL: ~/Desktop/2020517-Assignment2-Compiler
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ python3 parser.py input.txt
Tokens:
IDENTIFIER: x
OPERATOR: =
INTEGER_LITERAL: 10
KEYWORD: print
IDENTIFIER: x
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
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ nano input2.txt
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ python3 parser.py input2.txt
Tokens:
IDENTIFIER: y
OPERATOR: =
INTEGER_LITERAL: 5
OPERATOR: +
KEYWORD: print
IDENTIFIER: y
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ nano input2.txt
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$ python3 parser.py input2.txt
Tokens:
IDENTIFIER: y
OPERATOR: =
INTEGER_LITERAL: 5
OPERATOR: +
INTEGER_LITERAL: 1
KEYWORD: print
IDENTIFIER: y
warda@warda-Lenovo-IdeaPad-C340-14IWL:~/Desktop/2020517-Assignment2-Compiler$