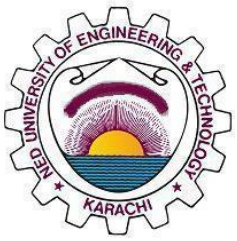
Department of Computer Science and Information

Technology NED University of Engineering and Technology



Complier Design
Project Report

Course Code: CT-463

Submitted to: Dr. Saman Hina

Members:

Muhammad Uzair Khan CT-051
Muhammad Daniyal Saleem CT-063
Faseeh Ur Rehman CT-068
Ahsan Omerjee CT-091

Table of Contents

1. Introduction:

Brief Overview:

The CalcLang Interpreter is a project designed to implement a simple arithmetic compiler that parses and evaluates arithmetic expressions. It provides functionalities for lexical, syntactic, and semantic analysis and is presented through a graphical user interface (GUI) to enhance user interaction.

Purpose and Objectives:

The primary objective of this project is to create a custom-designed language (CalcLang) interpreter that performs basic arithmetic operations. The interpreter aims to demonstrate the workings of lexical and semantic analysis and provide detailed error messages for educational purposes.

2. Language Description and Grammar:

Description of CalcLang:

CalcLang is a simple, custom-designed arithmetic language that supports basic arithmetic operations including addition, subtraction, multiplication, and division. It also supports the use of parentheses for grouping and handling of negative numbers. The language is designed to provide clear and concise syntax to facilitate easy parsing and interpretation.

Grammar Definition:

The grammar for CalcLang is defined using Extended Backus-Naur Form (EBNF):

```
<expression>    ::=   <term>        ( ( '+' | '-' ) <term> )*
<term>          ::=   <factor>      ( ( '*' | '/' ) <factor> )*
<factor>        ::=   <integer>  |  '(' <expression> ')' | '-' <factor>
<integer>       ::=                 [0-9]+
```

3. Selected Functionalities:

- Parsing arithmetic expressions
- Handling integers, operators, and parentheses
- Performing lexical analysis
- Generating tokens with type and position attributes
- Syntax analysis using recursive descent parsing
- Semantic checks (e.g., division by zero)
- GUI for input, output, and analysis logs
- Detailed error reporting

4. Lexical Analysis:

Regular Expressions/Rules

- Integer: [0-9]+
- Plus: \+
- Minus: \-
- Multiplication: *
- Division: \/
- Left Parenthesis: \(
- Right Parenthesis: \)
- Whitespace: \s+

Lexical Analysis Process

The lexical analysis process involves reading the input string character by character and identifying tokens based on the defined regular expressions. The Lexer class is responsible for generating tokens with type and position information, skipping any whitespace in the process.

Token Categories

Keywords: Not applicable (CalcLang has no reserved keywords)

Identifiers: Not applicable (CalcLang does not use variable names)

Literals: Integers

Operators: +, -, *, /

Delimiters: (,)

5. Parsing Techniques:

Syntax Analysis:

The syntax analysis is performed using recursive descent parsing, a top-down parsing technique that involves a set of mutually recursive procedures to process the input based on the grammar rules.

Parsing Process:

- **expr:** Processes terms and addition/subtraction operators.
- **term:** Processes factors and multiplication/division operators.
- **factor:** Processes integers, parentheses, and unary minus.

6. Semantic Analysis:

Semantic Checks:

The interpreter performs semantic checks to ensure the validity of operations. For example, it checks for division by zero during the parsing process and raises a semantic error if encountered.

Error Handling

The interpreter provides detailed error messages that include the position in the input string and a description of the error. This helps users identify and correct issues effectively. Error messages are categorized as follows:

- **Syntax Errors:** Indicate issues with the structure of the input (e.g., missing operators).
- **Semantic Errors:** Indicate logical issues in the operations (e.g., division by zero).

7. Graphical User Interface (GUI):

Features

- **Input Field:** Area to input arithmetic expressions.
- **Output Field:** Displays results or error messages.
- **Buttons:**
  - **Run:** Executes the input expression.
  - **Clear:** Clears the input and output fields.
  - **Save:** Saves the input and output to a text file with a timestamp.
  - **Show Lexical Analysis:** Displays the steps of lexical analysis in a separate window.
  - **Show Semantic Analysis:** Displays the steps of semantic analysis in a separate window.

User Guide

- **Running the Program:** Launch the executable or run the Python script.
- **Entering Expressions:** Type arithmetic expressions in the input field.
- **Executing Expressions:** Click the 'Run' button to evaluate the expression.
- **Viewing Results:** The output field displays the result or error messages.
- **Clearing the Screen:** Click the 'Clear' button to reset input and output fields.
- **Saving Logs:** Click the 'Save' button to save the current input and output.
- **Viewing Analysis:** Click the 'Show Lexical Analysis' or 'Show Semantic Analysis' buttons to view detailed analysis steps.

8. Source Code:

The detailed source code with execution file is included in the same folder named as calc_lang.py and calc_lang.exe in dist folder.

High-Level Overview:

- **Lexer Class:** Implements lexical analysis, generating tokens from the input string.
- **Parser Class:** Implements syntax analysis using recursive descent parsing.
- **Interpreter Class:** Combines parsing and semantic checks to evaluate expressions.
- **CalcLangApp Class:** Implements the GUI using Tkinter.

9. Testing

Test Cases

1. **Correct Expressions:**
   - 2 + 2 → 4
   - 4 - 4 → 0
   - 4 - 8 → -4
2. **Syntax Errors:**
   - 4- → Error: Syntax error at position 2
   - 4++ → Error: Syntax error at position 2
   - -5+7 (→ Error: Syntax error at position 5
3. **Semantic Errors:**
   - 4 / 0 → Error: Division by zero at position 2

Results

The interpreter correctly evaluates valid expressions and provides detailed error messages for invalid input.

10. Limitations

Identified Limitations

- Limited to basic arithmetic operations.
- Does not support floating-point numbers.
- Limited error recovery; stops at the first error encountered.
- The grammar does not support more complex constructs like functions or variables.

11. Conclusion

Summary of the Project

The CalcLang Interpreter successfully parses and evaluates arithmetic expressions, providing detailed lexical, syntactic, and semantic analysis through a user-friendly GUI. The project demonstrates key concepts in compiler design, including error handling and recursive descent parsing. Future enhancements could include support for more complex expressions, floating-point numbers, and improved error recovery mechanisms.

12. Appendices

Installation Instructions

1. **Prerequisites:** Ensure Python and Tkinter are installed.
2. **Running the Python Script:**
   - Run the script with python calc_lang.py.
3. **Creating the Executable:**
   a) **Use PyInstaller:**
      - pip install pyinstaller
      - pyinstaller --onefile calc_lang.py
   b) **Run the generated executable.**

Additional Test Cases

1. **Complex Expressions:**
   - 2 * (3 + 4) → 14
   - (2 + 3) * (4 - 1) → 15
2. **Edge Cases:**
   - -2 + 5 → 3
   - 2 + → Error: Syntax error at position 2