Gavin Ebel

Professor Richard Gayler

Concepts of Programming Languages Section W01

06 October 2025

Project 2: Recursive Descent Parsing - Report

Problem Statement

Implement a parser for lexemes including “(,), +, -, *, /, int_lit, and EOS) using recursive descent parsing.

Summary/Purpose

This project serves as a continuation from the first project. The purpose is to make use of our previously built lexical analyzer to feed the parser tokens and check for syntax errors as well as teaching us how parsers function.

Detailed Solution

Only one class was necessary to make for this project, the “Parser” class contained in “Parser.java”. This class implements the lexical analyzer, tokens, and token types from the first project. My solution is a combination of what I found in our textbook, the given pseudo code for the project, and some of my own handy work and refactoring of the old classes. Much of the refactoring was fixing problems which were stated in the feedback for project 1. The parser class utilizes a lexical analyzer object to contain the contents of the requested file and keep track of the row and column of the token currently being parsed. It also uses a token object to hold the token value at the current

row/column. Aside from that, the implementation of the class is basically the same as the pseudo code with debug information printed to the console.

Input Data

File Name: tst.txt

65+8*9/2++(3*4)
2*2+(8/4)
(3+(4*3) / ((9-1))

Results

Only .txt file will work. Do not include a file extension!

```
Input name of file to parse (Enter "exit" to exit): tst
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 65
Next lexeme is: +
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 8
Next lexeme is: *
Exit <number>
Exit <factor>
Enter <term_prime>
Enter <factor>
Enter <number>
Next token is: 9
Next lexeme is: /
Exit <number>
Exit <factor>
Enter <term_prime>
Enter <factor>
Enter <number>
Next token is: 2
Next lexeme is: +
```

Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Enter <term>
Enter <factor>
Enter <number>
Syntax error on token +, "INT_LITERAL" expected [Line 1, Col 10]
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr_prime>
Exit <expr_prime>
Exit <expr>
Enter <expr>
Enter <term>
Enter <factor>
Next lexeme is: (
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 3
Next lexeme is: *
Exit <number>
Exit <factor>
Enter <term_prime>
Enter <factor>
Enter <number>
Next token is: 4
Next lexeme is:)
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr>
Exit <factor>
Enter <term_prime>

Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr>
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 2
Next lexeme is: *
Exit <number>
Exit <factor>
Enter <term_prime>
Enter <factor>
Enter <number>
Next token is: 2
Next lexeme is: +
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Enter <term>
Enter <factor>
Next lexeme is: (
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 8
Next lexeme is: /
Exit <number>
Exit <factor>
Enter <term_prime>
Enter <factor>
Enter <number>
Next token is: 4
Next lexeme is:)
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr>

Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr_prime>
Exit <expr>
Enter <expr>
Enter <term>
Enter <factor>
Next lexeme is: (
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 3
Next lexeme is: +
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Enter <term>
Enter <factor>
Next lexeme is: (
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 4
Next lexeme is: *
Exit <number>
Exit <factor>
Enter <term_prime>
Enter <factor>
Enter <number>
Next token is: 3
Next lexeme is:)
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr>
Exit <factor>

```
Enter <term_prime>
Enter <factor>
Next lexeme is: (
Enter <expr>
Enter <term>
Enter <factor>
Next lexeme is: (
Enter <expr>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 9
Next lexeme is: -
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Enter <term>
Enter <factor>
Enter <number>
Next token is: 1
Next lexeme is: )
Exit <number>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr_prime>
Exit <expr>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr>
Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr_prime>
Exit <expr>
Syntax error on token null, "R_PAREN" expected [Line 3, Col 19]
```

Exit <factor>
Enter <term_prime>
Exit <term_prime>
Exit <term>
Enter <expr_prime>
Exit <expr_prime>
Exit <expr>

Limitations

what should happen

Limitations of the parser include limitations of the lexical analyzer in that only tokens that can be recognized can be parsed. For example, strings cannot be parsed because it is outside of the scope of the project currently. Accurate syntax error findings are also somewhat erratic after finding the first syntax error in an expression. This is because in the recursive descent implementation, the tree functions based off of assumptions of what it will receive next. A syntax error will not stop the parser from going through the entire file, but it will reduce accuracy further down the same expression.

Improvement

Improvements were made to previous implementations based on the feedback for project 1. For this project, the syntax error findings could likely be improved to be more accurate within one expression when an error is found.

References

- Used for improving my javadoc documentation;
  - OpenAI. “ChatGPT.” *ChatGPT*, OpenAI, 7 Sept. 2025, chatgpt.com/.
- Used for reminders of java syntax and libraries;
  - w3schools. “Java Tutorial.” *W3schools.com*, 2019, www.w3schools.com/java/.
- Sebesta, R. W. (2018). *Concepts of programming languages*.

Usage of AI

Artificial Intelligence was used in the improvement of my documentation through javadoc formatting. This documentation already existed, but usage of javadoc was insufficient.

KSU Academic Integrity Policy

I, Gavin Ebel, state that the content in this assignment adheres to the Kennesaw State University Academic Integrity Policy.