Building a LL(1) Parser for VC

Nguyen Tuan Anh - 20021292 Tran Trong Thinh - 22028073
Hoang Duc Minh - 22028051

May 24, 2024

1 Introduction

This is a report for the project *Building a LL(1) Parser for VC*

2 Self-evaluation

2.1 Identifying and Comprehending the Current Grammar

We have analyzed the current grammar of the VC language to understand its structure and rules. After careful examination, we determined that the language is not LL(1) compatible due to left recursion and ambiguities.
Example:

$$
\begin{aligned}
\text{rel-expr} \rightarrow\ & \text{additive-expr} \\
|\ & \text{rel-expr} < \text{additive-expr} \\
|\ & \text{rel-expr} \leq \text{additive-expr} \\
|\ & \text{rel-expr} > \text{additive-expr} \\
|\ & \text{rel-expr} \geq \text{additive-expr}
\end{aligned}
$$

$$
\begin{aligned}
\text{stmt} \rightarrow\ & \text{compound-stmt} \\
|\ & \text{if-stmt} \\
|\ & \text{for-stmt} \\
|\ & \text{while-stmt} \\
|\ & \text{break-stmt} \\
|\ & \text{continue-stmt} \\
|\ & \text{return-stmt} \\
|\ & \text{expr-stmt}
\end{aligned}
$$

2.2 Converting the Grammar to an Appropriate Variant

To make the grammar LL(1) compatible, we performed the necessary transformations to eliminate the conflicts using the FIRST and FOLLOW sets.

2.3 Building and using the external State Table

Using the modified grammar, we constructed an appropriate state table in an external format. After that, we implement a parser with a readGrammar class to put that file into use.

3 How to run

- Go to the folder containing the lexer and parser folders
- Put the file path without extension into the file variable in main.py function
- In terminal enter "python main.py"

4 Conclusion

The parser can detect errors, but when printing ast tree some expressions are still not in the correct format as required.