CE204 Lab 9: Parsing

David Richerby

Week 25, 2020–21

The lab exercises are not assessed and you are not required to complete all of them, though I recommend that you attempt them all. Feel free to work with others and talk about your answers.

Solutions will be released on Moodle, on the Friday after the labs.

1 Recursive descent parsing

Based on the code from slides 254, 261 and 266–268, write a recursive descent parser for arithmetic expressions involving the operators $+$, $-$, $*$, / and ^ (exponentiation).

The `Token` class already includes exponentiation, and the `ParseTree` class can already include arbitrary operators. You will need to modify the `Parser` class to create nodes for exponentiation.

You will also need to write the `Lexer` class. Its constructor should take a `String` parameter, which is the string to be parsed. It should have a method `Token nextToken()` which, when called repeatedly, returns the tokens one by one. If you want, to simplify the lexer, you can assume that tokens are separated by spaces.

Finally, write code to evaluate the arithmetic expression encoded by the parse tree.

2 Parsing with look-ahead (optional)

In the lecture, I skipped over slides 270–273. Study these and implement a parser for the grammar on slide 270.

You will need to modify your lexer to include a `lookAhead()` method, as described on slide 271, and implement the `parseTERM()` method along with the other methods shown on the slides.

Since this parser also produces a `ParseTree` object, your evaluator from the previous exercise should be able to evaluate the output of your parser without modification.