CSC488 Week 4 Tutorial

Introduction

This exercise is intended to help you get some practice with the yacc (parser) portion of PLY and familiarize yourselves with BNF grammar. The parser will behave similarly to a simple calculator. Lexer portion of the calculator is provided under `calculatorLexer.py` file. It should look familiar, as structure of the code is similar to that of week 2 tutorial.

Requirements

You will be required to modify `calculatorParser.py` so that the program functions as a simple calculator. The calculator must be able to perform addition, subtraction, multiplication, and division. The calculator should also the standard order of operation. Additionally, the user should be able to use brackets to override order of operation.

Refer to `calculatorLexer.py` for all the tokens defined for the calculator. Afterwards, also observe the function `p_expr` within `calculatorParser.py`. It is a simple grammar rule, which specifies that `expr` can be a single `NUMBER` token. You should create additional grammar rules, in order for the calculator to function properly.

Note that PLY is sensitive to ambiguous language. For instance, grammar rule such as "`expr : expr + expr`" is ambiguous, since given an input such as "1 + 2 + 3", PLY will not be able to whether to parse it as "(1 + 2) + 3" or "1 + (2 + 3)", since they are both valid according to the grammar rule. You could create additional grammar rules to remove ambiguity manually, or use precedence rule.

Testing Your Code

You can test your by simply running `calculatorParser.py`. Note that `calculatorLexer.py` must be present in the same directory for the parser to function. Note that running the starter code without any modification will lead to the program outputting bunch of warnings. This is expected, since the starter code does not utilize all the tokens defined in `calculatorLexer.py`. However, you should notice the prompt at the end:

```
calc >
```

You may enter any number (since the grammar for single number is defined), but anything more than a single number would lead to syntax error. Once the parser is fully written, you should be able to enter simple calculations and the parser should return a correct result:

```
calc > 5 + 4
9
calc > 3 * 4
```

```
12
calc > 2 + 3 * 4
14
calc > (2 + 3) * 4
20
```