1. **[11 points total] (a) 4 points.** Show a stratified LL(1) grammar for arithmetic expressions over the three terminals \{num, +, -\} so that “−” can be used as a binary operator in subtraction and as a unary operator in arithmetic negation. The grammar should yield a concrete syntax tree reflecting that unary negation is right-associative and binds more tightly than addition and subtraction. Addition and subtraction have the same precedence and are left-associative. For example, the expression 2 − 3 + -4 - - -5 should have a concrete syntax tree that reflects the ordering (((2-3)+(-4))-(-(-5))).

(b) **4 points.** Describe how you would modify the simpleAST example on the class website to build Expr ASTs using BinExpr, UnaryExpr and NumExpr nodes. Describe your extensions to the scanner, parser, AbstractSyntaxTrees, and visitors. It’s not necessary to write out complete code, but if you are interested you can extend the simpleAST example to try it out.
(c) 3 points. Draw the AST for \( 2 - 3 + -4 - - -5 \) using `BinExpr`, `UnaryExpr` and `NumExpr` nodes and show the spelling of the tokens in the AST.

2. (3) points. Using the class constructors from the miniJava AbstractSyntaxTrees package, show how to build the `AssignStmt` AST for the miniJava statement \( x = 1 - -a[3] \);