Assignment 1

- Academic integrity is taken seriously in this course, which has a zero tolerance policy for cheating. All discovered incidents of cheating will be reported and result in an E in the course.
- You ARE allowed to:
  - discuss high level solutions with other students and
  - ask questions in the class slack channel and during office hours
  - use code presented in the lectures.
- You are NOT allowed to
  - Share your code with anyone else (except your official project partner).
  - Receive code from others (except your official project partner), including students in previous semesters.
- All work submitted should be the work of the submitting student(s), created this semester for this assignment.
- Submissions by groups must have approximately equal contributions from both members.

In this assignment, you will implement a parser for a subset of the programming language we will implement this semester. Your parser will use your lexer from Assignment 0 to check for legal phrase structure. For legal sentences, an abstract syntax tree should be returned. If the parser detects an error, a SyntaxException should be thrown. If the lexer detects an error, a LexicalException should be thrown. (Assuming your lexer is implemented properly, you should not need to do anything at all in this assignment for the latter requirement.)

Classes for the AST nodes have been provided in the package edu.ufl.cise.cop4020fa23.ast. AST is the abstract (direct or indirect) superclass of all AST nodes. Expr is an abstract superclass of all the other nodes that correspond to expressions. Generally, the AST class corresponding to a grammar production should be obvious from the name. You may need to look at the class to find out what parameters need to be given to the constructor.

1. Fix any problems with your lexer
2. Examine the grammar and determine if it is LL(1). If it is not, make a note that you will need to deal with this.
3. Develop code for a recursive decent parser following the procedure described in class. Generally, methods corresponding to grammar productions should return an AST node corresponding to that subexpression.

4. Each AST node class has a field IToken firstToken inherited from AST which is set in the constructor. This should be a reference to the first token that makes up that subtree.
5. Your expression parser should recognize a legal expression and return its AST. If there are still tokens remaining, they should be ignored for this assignment. For example, the input "a + b c" should return the BinaryExpr representing a + b and ignore the remaining IdentToken representing c.

Provided Code

- Modified version of edu.ufl.cise.cop4020fa23.ComponentFactory (new methods to create an expression parser)
- Package edu.ufl.cise.cop4020fa23.ast containing classes to use for the AST nodes. You may modify the toString methods in these classes if you want, but do not make any other modifications in this assignment.
- edu.ufl.cise.cop4020fa23.exceptions.SyntaxException
- edu.ufl.cise.cop4020fa23.ExpressionParser.java -- incorrect and incomplete implementation of the assignment. The provided version should compile and run the test cases, although they will fail until you have provided a correct implementation.
- edu.ufl.cise.cop4020fa23.ExpressionParserTest_starter.java -- a few test cases and useful methods.

Hints

- The only class you must change in this assignment is ExpressionParser.java. You should also create additional test cases, either by adding to ExpressionParserTest_starter or creating a new test case. You may, but are not required to change the toString method in the provided ast classes. No other changes should be made to the ast classes in this assignment.
- Work systematically, from the inside out and test as you go.
- Use git like a professional would.
- Arrange for not yet implemented branches in your code to throw a distinct exception (I use UnsupportedOperationException) and make sure that these are all gone in a complete implementation.
- It is more difficult to create tests for this assignment than assignment 0. The provided tests, which do NOT provide complete coverage, were automatically generated from the output of a reference implementation and are ugly and more detailed than you probably want to bother with when writing your own tests.

- The classes have been instrumented to support the Visitor Pattern. This will be explained later and used in assignments 3,4, and 5. For the time being, just ignore ASTVisitor.java, and the visit methods in the AST node classes.
- Since a token contains its position in the source code, the saving the firstToken of each subexpression will allow error message found in later assignments, such as during type checking, to report errors with their location. This is easy to implement, just define a local variable for firstToken in each method in your parser and save the current token before you do anything else in that method.
- In this assignment, the abstract syntax tree has the same structure as the grammar, except that it omits punctuation. For example, look at the production PixelSelector ::= **[** Expr **,** Expr **]**. Parsing will result in ASTs for the two subexpressions. Use those ASTs to create a PixelSelector object. The name of the class matches the name of the rule; there are fields for the relevant parts of the expression. The punctuation symbols are not saved. (However, firstToken should be the LSQUARE token)
- Tag your lexer submission in git, and continue development in the same repository.

To turn in

- A jar file containing the source code (i.e .java files) of all classes necessary to test your parser with the appropriate directory structure. Do NOT include class files.
- Be aware that your IDE may provide a way to generate jar files that defaults to including .class files instead of .java files. Double check the contents of your jar file before submission.
- A zip file containing your git repository WITH history. To obtain this file, go to the parent of your local git repository and create a zip file. Your zip file should contain a directory called .git (dot git). Do NOT use the "download zip file" option on github--it does not include the history.