Assignment 1

In this assignment, you will finish the parser from assignment 1 so that it handles the complete grammar.

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 the abstract superclass of all the other nodes that correspond to expressions. Statement is the abstract superclass of all the other nodes that correspond to statements.   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, and parser from assignment 1.
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 program fragment.
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 parser should recognize a legal expression and return its AST. If there are still tokens remaining, this should result in a SyntaxException.   See test14 in the ParserTest_starter.java**

New provided code

- Modified version of edu.ufl.cise.cop4020fa23.ComponentFactory (new methods to create a 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. Parser.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. ParserTest_starter.java   -- a few test cases and useful helper methods.

Hints

- The only class you must change in this assignment is Parser.java. You should also create additional test cases, either by adding to ParserTest_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 software developer would. (If you don't know how, go through a git tutorial)
- 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.

- Tag your assignment 1 submission in git, and continue development in the same repository.   You may want to make a copy of your ExpressionParser from assignment 1 named Parser.java and add/modify that to create the parser.

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.