**50.001 Problem Set 3 (Part C)**

**Multi-Unit Calculator [100 points]**

**Overview**

It's often convenient to use different units in the same computation. For example, to figure out how many lines of twelve-point type fit in a six-inch column, it would be nice to have a calculator that accepted the expression "6in/12pt". Construct two grammars for such a language: first, a lexical grammar that breaks the sequence of characters into numbers, unit specifiers, operators, and left/right parentheses, filtering out spaces and tabs (but not requiring them as delimiters); and second, a syntactic grammar that groups expressions appropriately.

Your grammar should be able to recognize the following types of expressions:

* Arithmetic expressions with +, -, \*, and /. You can assume that the order of operations will always be made explicit with parentheses.
* Expressions with integer and decimal operands, both as scalars and inches and points. Point operands will be the number followed by 'pt,' and inches will be followed by 'in.' The result of evaluating the expression should be in appropriate units also.
  + inches/scalar = inches
  + inches\*scalar = inches
  + inches/inches = scalar
  + inches/points = scalar

Assume similar combinations for substitutions of inches and points.

The following combinations are not as intuitive, so we've supplied this specification to allow for consistency in grading:

* + scalar/inches = inches
  + inches\*inches = inches
  + scalar+inches = inches
  + inches+points = inches (use units of the first operand)
* Unit conversion expressions (also made explicit by parentheses), represented by 'in' or 'pt' following an expression.
* As said above, whitespace should be ignored.

You will be expected to be able evaluate all the following expressions, with the result presented after the '=':

* 3+2.4 = 5.4
* 3 + 2.4 = 5.4
* (3 + 4)\*2.4 = 16.8
* 3in \* 2.4 = 7.2in
* 4pt+(3in\*2.4) = 522.4pt
* (3 + 2.4) in = 5.4in
* (3in \* 2.4) pt = 518.4pt

You will NOT be expected to evaluate the following expressions:

* 3+2+1 (order of operations not made explicit)
* 2+(3+4)+2 (order still not explicit)
* (3+4)^2 (do not need to support ^)
* 2+3in pt (order not explicit)

If your calculator receives an expression that it does not know how to handle, you should display some sort of error message, ideally with some information about why the input could not be evaluated, such as "Order of operations not made explicit," or "Cannot divide by 0."

We have provided you with some skeleton code for this calculator. It consists of a simple prompt loop, in the main method of MultiUnitCalculator, for users to enter expressions. There is also a Lexer class along with a Type class, and a Parser class for interpreting the input expression String and evaluating it. This problem set will walk you through the implementation of these classes.

**Multi-unit Calculator**

Implement Lexer.java and Parser.java

This is the only requirement of this homework.

**Here are some suggested steps of how to do this homework**

**Part 1: Specify a Grammar**

Before jumping into the implementation, we recommend you to specify a grammar for your calculator. Your grammar should be able to recognize the expressions described in the Overview section above.

Start by deciding what symbols or terminals your grammar will contain. This should include things like '+,' '6,' and 'pt.' List these as a comment at the top of Type.java.

Next, group your symbols into Types that your Lexer will recognize, based on their function in an expression. Indicate your groupings in your comment in Type.java.

Finally, write out your grammar in a comment at the top of Parser.java

**Part 2: Implement the Lexer**

The first step for evaluating an expression will be converting the user input String into a set of tokens that your parser will be able to recognize. These tokens will be parts of the String paired with the Types that you defined in Type.java. We've provided you with Lexer.java, which defines an internal Token class for you to utilize when implementing your Lexer. To actual recognize the tokens in the input Strings, you may want to use regular expressions. Eventually you will pass the Lexer to the Parser constructor, but exactly how the Lexer presents the Tokens to the Parser is up to you.

*Hint: You can think about the Lexer consuming the expression one character at a time and passing the resulting Tokens to the Parser, but this is only one way of doing things.*

We suggest you to write some tests for testing Lexer.java. Then implement Lexer.java.

**Part 3: Implement the Parser**

Your parser should use the tokenization produced by the Lexer to parse the expression according to the grammar you defined. Exactly how you do this is up to you; the only thing that we enforce is that the Lexer is passed as an argument to the Parser constructor.

We suggest you to write some tests for testing Parser.java. Then implement Parser.java

**Part 4: Put it all together**

Then, MultiUnitCalculator.java should work without any further changes.