In-class Exercise: Adding "repeat" functionality to the calculator

command	comment
`java -cp cup.jar:. JFlex.Main example.lex`	creates Yylex.java
`java -cp cup.jar:. java_cup.Main example.cup`	creates sym.java and parser.java
`javac -cp cup.jar:. parser.java`	compiles everything
`java -cp cup.jar:. parser < input.txt`	runs the parser

Table 1: Commands needed to compile and run a CUP parser with a JFlex scanner

This exercise asks you to add some slightly silly functionality to the calculator, but it will demonstrate many of the concepts you need to understand.

The task is to add functionality that repeats the digits of a given integer before evaluating the integer as a number. The repeat functionality will be triggered by the letter "r". For example, if the calculator encounters `r235` in an expression, this will be interpreted as the numerical value 235235. As a further example, a complete input consisting of "`3.5 + r5;`" would evaluate to 58.5.

The steps required to complete this exercise are as follows:

1. Download the required files (they are all in `calculator.zip`) and run all the commands in Table 1. Edit input.txt and verify that the calculator can handle expressions like "`3*(2.5+1.5);`".

2. Add a new regular definition in `example.lex`. Specifically, define a new regular expression called "REPEAT" immediately after the line beginning "NUM".

3. Add a new action in `example.lex`. Specifically, add a line after the line beginning "{NUM}", returning a new symbol for matches to the REPEAT regular expression. I recommend using a `String` as the datatype of the value for this symbol.

4. You have finished modifying `example.lex`, so now run JFlex to check there are no errors.

5. Add a new terminal called REPEAT, in `example.cup`. Do this by adding a line after the line "terminal Double NUM;".

6. Add a new rule to the grammar in `example.cup`. Specifically, add a new line after the line containing "NUM:n". The line should start with "| REPEAT". Hints: the Java code that appears between the "{:" and ":}" will need to use (i) the `substring` method of the String class; (ii) string concatenation; and (iii) the constructor for `Double` that takes a single `String` argument.

7. Run all the remaining commands (see Table 1 above). Check that inputs such as "`r235;`" and "`3.5 + r5;`" are evaluated correctly.