COMP 356 Homework Assignment 5, Version 2
(25 points; submit to Moodle)

Acknowledgment. This assignment was written by Prof. Tim Wahls, with minor changes by John MacCormick.

Modify the "calculator" example (files `example.lex` and `example.cup` from the example code on the course web pages) by adding the following kinds of expressions:

- applications of the trigonometric function sine to any expression. For example:

  ```
  sine(3.14159 / 2);
  ```

  would then be a legal expression that evaluates to a number close to 1. Note that the Java library function `Math.sin` can be used to compute the sine of a number.

- exponentiation - taking any expression to the power of the result of another expression. The exponentiation operator (`^`) is right associative and has highest precedence in this language, so that:

  ```
  1 + 2 ^ 3 ^ 2;
  ```

  is a legal expression that evaluates to 513.0. Note that the Java library function `Math.pow` takes 2 numbers as arguments and returns the first number to the power of the second number.

- assignment statements. An assignment statement has the form:

  `id :=` <expr>

  where `id` is a token for variable names, and `:=` is the assignment operator. Legal variable names consist of a letter (upper or lowercase) followed by any number of letters or digits. The value of an assignment statement is the value of the expression on the right hand side. After an assignment statement, the id on the left hand side is a valid expression in its own right and it has the value that was assigned to it.

- variables - i.e. the token `id` described above.

  Hint: add the line

  ```
  static java.util.HashMap<String, Double> env = new java.util.HashMap<String, Double>();
  ```

  just before the definition of `main` in your CUP specification file. You can then use `parser.env` to refer to this HashMap in your rules.

- if expressions. An if expression has the form:

  if (<bexpr>) then <expr> else <expr>

  where <bexpr> is defined as:

  <bexpr> → <expr> RELOP <expr>

  Notes:

  - The legal lexemes for token RELOP are <, > and =. These symbols have their normal meanings from mathematics.
  - These relational operators (RELOPs) are not associative and have lowest precedence in this language.
  - If the condition (<bexpr>) used in an if expression is true, then the value of the if expression is the value of the first <expr> argument. Otherwise, the value of the if expression is the value of the second <expr> argument.

– You may find the built-in class `Boolean` and its `booleanValue()` method useful for dealing with boolean expressions.

As a test case, on input:

```
if (4 > 3 ^ 1.5) then 3.4 / 2 else 9/5 + 2;
2 * 2 ^ 3 ^ 2;
sine(3.14159 / 2);
x := 2 ^ 2.0 + 3;
x;
x + 2;
```

your parser should output:

```
3.8
1024.0
0.9999999999991198
7.0
7.0
9.0
```

Submit your modified .lex and .cup files to Moodle before class time on the due date, as a single zip or tar archive.