COP4020 Programming Assignment 7

Download the CalcAST.java and AST.java sources from:

http://www.cs.fsu.edu/~engelen/courses/COP4020/CalcAST.java
http://www.cs.fsu.edu/~engelen/courses/COP4020/AST.java

The CalcAST program constructs an abstract syntax tree (AST) representation of arithmetic expressions. For example, when the expression that you input is 1+2; the program constructs the following AST:

```
+  /
1  2
```

This tree structure is constructed with the AST class, which has a tree node structure that contains an optional operator (e.g. +), an optional value (e.g. 1), and optional left and right subnodes for the operands to monadic and dyadic operators. The AST class has a toLisp method. When invoked it will output the expression in Lisp form, such as (+ 1 2) for example.

Compile the sources with:

```
javac CalcAST.java
```

And run this program:

```
java CalcAST
```

The program will wait for input from the command line, so type 1+2; for example. The program output will be the Lisp equivalent of this expression (+ 1 2).

Modify the AST.java source (and CalcAST.java program if necessary) to evaluate parts of expressions. That is, arithmetic operations are performed by the AST class when the operands are numeric. When one of the operands is non-numeric (symbolic), an AST node is created as defined by the AST class. The output of the program will be partially evaluated expressions translated into Lisp.

In addition, add productions and code to implement the ^ power operator (see also Programming Assignment 5). For the implementation, you need to use the static Math.pow method of class Math computes powers. This operator must be evaluated when possible:

Example:

```
java CalcAST
2*(1+3)-2^3+x;
x
```

The output x is a simplified Lisp expression.

Note that the AST node structure includes a val member that can be used to store a node’s value and to pass values as part of the AST instances that returned from methods (as a synthesized attribute values) and passed to methods (as an inherited attribute values). The type of val is Object, so to create an AST node with an integer value, say 7, you need: new AST(new Integer(7)).
Your program should **at least** be able to compute the following values:

- \(1+2+3;\) 
  - 6
- \(1*2*3;\)
  - 6
- \(1*-2*(3-6);\)
  - 6
- \(1+2+x+3;\)
  - \((+ (+ 3 x) 3)\)
- \(x+1+2;\)
  - \((+ (+ x 1) 2)\)
- \(x+0;\)
  - \(x\)
- \(1*x;\)
  - \(x\)
- \(x^1;\)
  - \(x\)

Note that the semantic rules of the grammar enforce associativity, so \(1+2+x+3\) is evaluated from the left. The evaluation process does not consider commutativity, so this expression does **not** need to be simplified down to \(x+6\).