C M S I 4 8 8

Homework #3

Due: 2014-03-18

As before, turn in hardcopy solutions for all work, in addition to keeping your work in your github repository.

1. Here's a grammar:

```
S -> A M
M -> S?
A -> 'a' E | 'b' A A
E -> ('a' B | 'b' A)?
B -> 'b' E | 'a' B B
```

   a. Describe in English, the language of this grammar.
   b. Draw a parse tree for the string "abaa"
   c. Prove or disprove: "This grammar is LL(1)."
   d. Prove or disprove: "This grammar is ambiguous."

2. Here's a grammar that's trying to capture the usual expressions, terms, and factors, while considering assignment to be an expression.

```
EXP         -> ID ":=" EXP | TERM TERM_TAIL
TERM_TAIL   -> ("+" TERM TERM_TAIL)?
TERM        -> FACTOR FACTOR_TAIL
FACTOR_TAIL -> ("*" FACTOR FACTOR_TAIL)?
FACTOR      -> "(" EXP ")" | ID
```

   a. Prove that this grammar is not LL(1).
   b. Rewrite it so that it is LL(1).

3. Write an attribute grammar for the grammar in the previous problem. Your attribute grammar should describe the "obvious" run-time semantics.

4. Write an attribute grammar for evaluation (using the notation introduced in this class), whose underlying grammar is amenable to LL(1) parsing, for polynomials whose sole variable is x and for which all coefficients are integers, and all exponents are non-negative integers. The following strings must be accepted.
   - `2x`
   - `2x^3+7x+5`
   - `3x^8-x+x^2`
   - `3x-x^3+2`
   - `-9x^5-0+4x^100`
   - `-3x^1+8x^0`

5. Write a command-line application in Ruby, Clojure, JavaScript, or Python that evaluates polynomials from the language you defined above. The first argument should be the polynomial and the second is the value at which to evaluate the polynomial. Here are some example runs:

```
$ ruby evalpoly.rb "2x" 10
  20.000000
$ ruby evalpoly.rb "2x^3+7x+5" 2
  35.000000
$ ruby evalpoly.rb "3x^8-x+x^2" 1
  3.000000
$ ruby evalpoly.rb "3x-x^3+2" 0
  2.000000
```

Note that for this problem it is not necessary for you to build a seriously structured project with separate modules for scanning, parsing, error handling, abstract syntax tree construction and evaluation. Instead, make a very short and sweet script. There are no spaces in the polynomial strings so lexical analysis is no big deal; just bang out the code as a simple script.

6. Complete the parser for your compiler project, with professional unit tests. You should be able to output a complete abstract syntax tree.