C M S I 4 8 8

Homework #1

Due: 2014-02-06

Turn in a hardcopy of the solutions to the problems below. Make sure all answers are given, typeset neatly, including code. Answers involving code should *also* have available softcopy solutions on github, ideone, codepad, or similar location; please include the links on your hardcopy solution. **Make sure that you do not save any tab characters in your source code, and that there is no line wrapping in your printed submission.** Students that are too lazy or too incompetent to turn in hardcopy solutions, or who try to email solutions, will receive a zero.

1. Write regular expressions for:
   a. Canadian Postal Codes
   b. Legal Visa® Card Numbers, *not including checksums*
   c. Legal MasterCard® Numbers, *not including checksums*
   d. Ada 95 numeric literals
   e. Strings of letters and numbers beginning with a letter, EXCEPT those strings that are exactly three letters ending with two Latin letter ohs, of any case.

2. We saw in the course notes, on the page entitled "Syntax", an example of an abstract syntax tree with five concrete syntaxes. Show a sixth version of this "same" program, this time using a concrete syntax that also passes for a JSON object.

3. In the Ada language comments are started with "`--`" and go to the end of the line. Therefore the designers decided not to make the unary negation operator have the highest precedence. Instead, expressions are defined as follows:

```
EXP  → EXP1 ('and' EXP1)*  |  EXP1 ('or' EXP1)*
EXP1 → EXP2 (RELOP EXP2)?
EXP2 → '-'? EXP3 (ADDOP EXP3)*
EXP3 → EXP4 (MULOP EXP4)*
EXP4 → EXP5 ('**'  EXP5)?  |  'not' EXP5  |  'abs' EXP5
```

   Explain why this choice was made. Also, give an abstract syntax tree for the expression `-8 * 5` and explain how this is similar to and how it is different from the alternative of dropping the negation from EXP2 and adding `- EXP5` to EXP4.

4. Here is a description of a language. Programs in this language are made up of a non-empty sequence of function declarations, followed by a single expression. Each function declaration starts with the keyword **fun** followed by the function's name (an identifier), then a parenthesized list of zero or more parameters (also identifiers) separated by commas, then the body, which is a sequence of one or more expressions terminated by semicolons with the sequence enclosed in curly braces. Expressions can be numeric literals, string literals, identifiers, function calls, or can be made up of other expressions with the usual binary arithmetic operators (plus, minus, times, divide) and a unary prefix negation and a unary postfix factorial (`!`). There's a conditional expression with the syntax `x if y else z`. Factorial has the highest precedence, followed by negation, the multiplicative operators, the additive operators, and finally the conditional. Parentheses are used, as in most other languages, to group subexpressions. Numeric literals are non-empty sequences of decimal digits with an optional fractional part and an optional exponent part. String literals delimited with

double quotes with the escape sequences \', \", \r, \n, \\, and \u followed by four hexadecimal digits. Identifiers are non-empty sequences of letters, decimal digits, underscores, at-signs, and dollar signs, beginning with a letter or dollar sign, that are not also reserved words. Function calls are formed with an identifier followed by a comma-separated list of expressions bracketed by parentheses. There are no comments in this language, and whitespace can be used liberally between tokens.

Write the micro and macrosyntax of this language.

5. Give an abstract syntax tree for the following Java code fragment:

```
if (x > 2 || !String.matches(f(x))) {
    write(- 3*q);
} else if (! here || there) {
    do {
        while (close) tryHarder();
        x = x >>> 3 & 2 * x;
    } while (false);
    q[4].g(6) = person.list[2];
} else {
    throw up;
}
```

Please note that the question asked for an abstract syntax tree and not a parse tree. If you give a parse tree, you will get a zero on the problem.

6. Complete the first pass of the design of the language that you will be writing a compiler for during this term. Give your language description on the README.md file of a public github repository you will be setting up for the project. Your README should blend the best parts of each of the following READMEs done by the students from last year:
   - Spitfire, which is very good overall, and includes both macro and microsyntax (A syntax is *required* for you; last year the other students seemed not to take that part of the assignment seriously.)
   - Yoda, which has a nice mascot and theme song, and a playful introduction
   - Koan.