There are six problems labeled A-F. The first four problems are intended to help you practice basic skills with CFGs. If you feel that you need more practice, then do more examples from the textbook.

A. Problem 2.4b (about starting and ending with the same symbol).

B. Problem 2.4c (about odd-length strings).

C. Problem 2.4f (about the empty set).

D. Problem 2.6b (about the complement of our canonical non-regular language).

Here is our CFG from class for fully parenthesized algebraic expressions:

```
<expr> ::=  <var> | <num> | (<expr><op><expr>)
<var>  ::=  <char> | <char><var>
<char> ::=  a | ... | z | A | ... | Z | _
<num>  ::=  <dig> | <dig><num>
<dig>  ::=  0 | ... | 9
<op>   ::=  + | * | ^ | - | /
```

E. Based on the CFG above, draw the parse tree for the expression

```
((3 * (4 / temp)) - ((x + 50) * (rate ^ pow)))
```

Mimic the textbook's examples in Section 2.1, with variables labeling branch nodes and terminals labeling leaf nodes. For example, the root node of your parse tree will be labeled `<expr>`. Do not take shortcuts; draw the entire tree. (For example, just the subtree corresponding to `pow` requires nine nodes.)

F. Prove that every regular language is context-free, by showing that any regular expression can be converted into a context-free grammar that describes the same language. Hint: Use structural induction, as we did in converting regular expressions to NFAs.