CMSC 330 Exam #1 practice questions #2 Fall 2012

Do not open this exam until you are told. Read these instructions:

1. This is a closed book exam. **No notes or other aids are allowed.**
2. **You must turn in your exam immediately when time is called at the end.**
3. This exam contains 6 pages, including this one. **Make sure you have all the pages.** Each question's point value is next to its number. **Write your name on the top of all pages before starting the exam.**
4. In order to be eligible for as much partial credit as possible, show all of your work for each problem, and **clearly indicate** your answers. Credit **cannot** be given for illegible answers.
5. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.
6. If you have a question, raise your hand. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn't need to do this at all, so be careful when making assumptions.
7. If you need scratch paper during the exam, please raise your hand. Scratch paper must be turned in with your exam, with your name and ID number written on it. Scratch paper **will not** be graded.
8. Small syntax errors will be ignored in any code you have to write on this exam, as long as the concepts are correct.
9. The Campus Senate has adopted a policy asking students to include the following handwritten statement on each examination and assignment in every course: "*I pledge on my honor that I have not given or received any unauthorized assistance on this examination*." Therefore, **just before turning in your exam**, you are requested to write this pledge **in full** and **sign it** below:

_____

_____

Good luck!

1. [20 pts.] **Short Answer.**

   a. [5 pts.] **Briefly** explain the difference between a *formal parameter* and an *actual parameter*.

   b. [5 pts.] **Briefly** explain what a *control statement* is and give examples of two control statements in Ruby.

   c. [10 pts.] Write down the type of each of the following OCaml expression, or "error" if the expression has no valid type.

      i. `(1, [3; 4; 5], 'a')`

      ii. `[(1, 2, 3); (4, 5); (6, 7, 8)]`

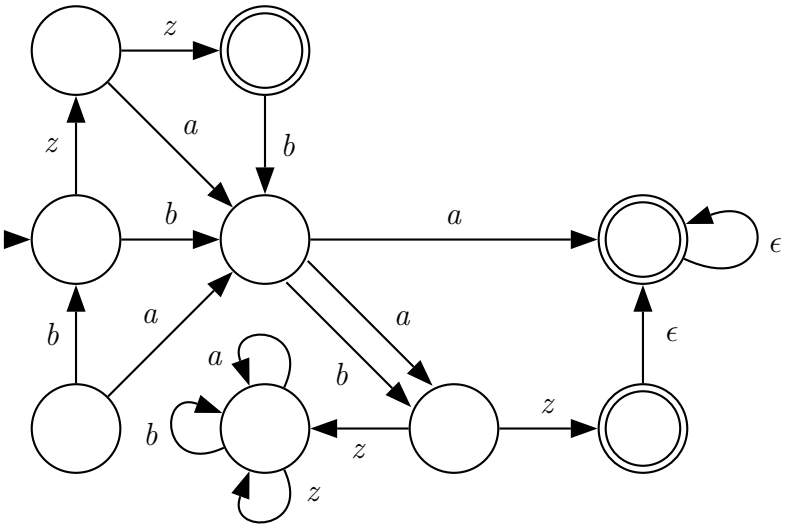
      iii. `if true then (1, 2) else [3; 4]`

      iv. `((1, 2), [3; 4])`

      v. `[(fun x -> x + 1); (fun y -> -y); (fun z -> z * z)]`

2. [50 pts.] **Regular languages from Planet Zorg.** Aliens from the planet Zorg have taken control of Friendly Computer Corporation and are requiring that all software be written in the programming language "Zorg" (the aliens are not very original). Zorg has a number of peculiarities.

   a. [10 pts.] Keywords in Zorg are all the strings recognized by the following NFA. Write down the set of Zorg keywords. (Your answer should just be a list of strings or words.)



   b. [10 pts.] Identifiers in Zorg are made up of a sequence of a's, b's, and z's, must be an even length, and must contain an occurrence of the string az. Write a regular expression for Zorg identifiers. You should use **only** the formal regular expression operations concatenation, alternation, and Kleene closure, which were defined in class. These operations can be nested, and terminal symbols, parentheses, and ϵ may be used as well, but do **not** use any other regular expression operations, and do **not** write a Ruby regular expression.

c. [10 pts.] Class names in Zorg (it's an object–oriented language) are strings of z's such that the length of the string is *not* divisible by three. Write a regular expression for Zorg class names, again using only the notation allowed in part (b). *Hint: Start by writing down the strings whose length is divisible by three and work from there.*

d. [20 pts.] In Zorg, strings begin with $<$ and end with $>$, and may contain occurrences of $<$, z, and $>$. Inside of a string, $>$ may appear but only if it is preceded by z (i.e., z is the "escape" character). z may appear without a following $>$. Moreover, there must be an even number of characters between $<$ and $>$, where the pair $z>$ is counted as a single character. Note that a string cannot end in $z>$, because the pair $z>$ counts as a single character, so the string would lack a terminating $>$. Construct a DFA that accepts valid Zorg strings. Be sure to create a DFA, **not** an NFA. Note: **do not** use the notational shortcuts for DFAs that were given in lecture.

3. [30 pts.] **Linked lists in Ruby.** Write code for a complete Ruby class `List` that implements singly–linked lists. Below is a class `Empty` that is a subclass of your `List` class. Instances of `Empty` represent the empty list.

```
class Empty < List

  def length
    return 0
  end

  # appending any list l to the empty list results in the list l
  def append(l)
    return l
  end
end
```

Your class `List` must be a complete Ruby class, and must include definitions of the following methods:

`length`	Return the length of this list. Your method **must** be recursive.
`append(l)`	Return a new list containing this list followed by list `l`, **without** changing self. Your method **must** be recursive. Your class should allow appending a `List` to an `Empty List`

Hint: Recall that `self` *refers to the current instance within a method. The methods we've given you in* `Empty` *represent the base cases of your recursive functions.* **Note:** your implementation must create a linked structure, and not use any library classes (such as `Array`). You can use the space below, plus the next page if necessary.