CMSC 330 Exam #1 practice questions #2– solutions Fall 2012

1. [20 pts.] Short answer.

   a. [5 pts.] A formal parameter is the name for a parameter as used in the body of a method or function. An actual parameter is the argument that is passed in at a method or function call.

   b. [5 pts.] A control statement is one that changes what the next statement to be executed is. Examples are `if`, `unless`, `while`, and `until`. Method invocation and return can also be considered control statements.

   c. [10 pts.]

      i. `int * int list * char`

      ii. error (can't mix tuples of different sizes)

      iii. error (both branches of an `if` must have the same type)

      iv. `(int * int) * int list`

      v. `(int -> int) list`

2. [50 pts.] Regular languages from Planet Zorg.

   a. [10 pts.] zz, $zzba$, zaa, ba, $zzbaz$, $zzbbz$, $zaaz$, $zabz$, baz, bbz

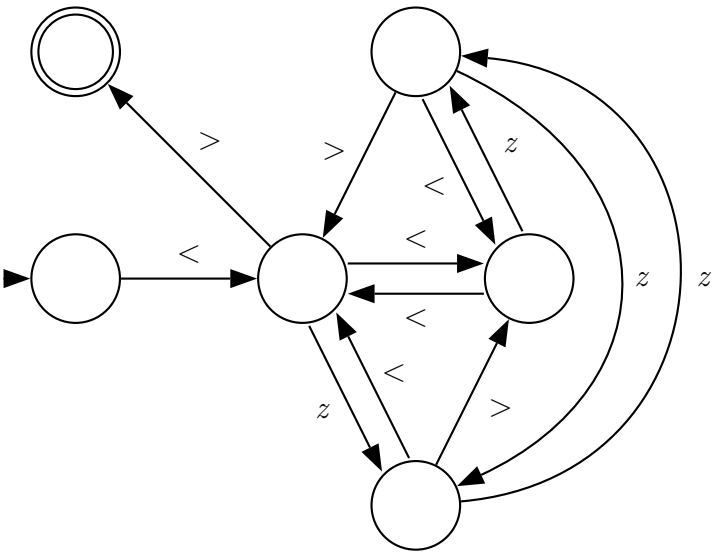
   b. [10 pts.] $((a|b|z)(a|b|z))^*$ is the set of strings of even length, and $(a|b|z)((a|b|z)(a|b|z))^*$ is the set of strings of odd length. Thus one correct answer is:

   $$\left(((a|b|z)(a|b|z))^* az((a|b|z)(a|b|z))^* \;\mid\; (a|b|z)((a|b|z)(a|b|z))^* az(a|b|z)((a|b|z)(a|b|z))^* \right)$$

   c. [10 pts.] The set of strings of z's whose length is divisible by three is $(zzz)^*$. To get strings whose length is not divisible by three we can add one or two more z's to it. Thus two correct answers are:

   $$(z \,|\, zz)\,(zzz)^*$$
   $$\left(z\,(zzz)^* \;\mid\; zz\,(zzz)^* \right)$$

   d. [20 pts.] Just for ease of drawing the DFA the notational shortcut is used of assuming that all omitted states go to a dead state that is not shown. (You were not supposed to use this shortcut, but it's easier to draw DFAs by hand than to typeset them...)



3. [30 pts.] Linked lists in Ruby.

   Here's the `Empty` class again:

```
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
```

   A correct `List` class would be:

```
class List

  def initialize(h, t)
    @head = h
    @tail = t
  end

  def length
    return 1 + (@tail.length)
  end

  def append(l)
    return List.new(@head, @tail.append(l))
  end

end
```