# Midterm #1

**CMSC 330: Organization of Programming Languages**

March 5, 2013

Name _______________________________________

## Instructions

Do not start until told to do so!

- This exam has 10 pages (including this one); make sure you have them all
- You have 75 minutes to complete the exam
- The exam is worth 100 points. Allocate your time wisely: some hard questions are worth only a few points, and some easy questions are worth a lot of points.
- If you have a question, please raise your hand and wait for the instructor.
- You may use the back of the exam sheets if you need extra space.
- In order to be eligible for partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

<table>
<thead>
<tr>
<th></th>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PL concepts</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ruby execution</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Regular expressions</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Finite automata (FA) concepts</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Non-deterministic FAs</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Deterministic FAs</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Context-free grammars</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Ruby programming</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. (Programming language concepts, 6 points total, 1 point each) Indicate whether the following are true (“T”) or false (“F”):

(a) ___ Ruby is object oriented

(b) ___ Ruby uses dynamic type checking

(c) ___ Ruby has syntax for method overloading

(d) ___ Ruby supports both physical and structural equality

(e) ___ A context-free grammar is ambiguous when it describes multiple languages

(f) ___ A regular expression can be used to define the language $\{a^n b^n \mid 0 \leq n \leq 3\}$
2. (Ruby execution, 12 points total, each is worth 3 points) What is the output of the following Ruby programs? If the execution fails at run-time, write the output up until the failure happens, then write FAIL. If multiple outputs are possible, write them all.

(a)  
```ruby
h = { "yes" => [1,2] }
h["no"] = [3,4]
h.values.each { |v| print v[0] }
```

Answer:

```
Multiple outputs:
3
1
```

(b)  
```ruby
s = "abba is a band, is abba"
a = s.scan(/a[bn]|ba/)
puts a.length
```

Answer:

```
5
```

(c)  
```ruby
a = []
b = [1,2]
a[0] = b
a[1] = [1,2]
puts (a[0] == a[1])
```

Answer:

```
true
```

(d)  
```ruby
class D
  def initialize
    @n = 1
    @s = 3
  end
  def next
    @s = @s + @n
    @n = @n + 1
  end
  def state
    @s
  end
end
d = D.new
d.next
d.next
puts d.state
```

Answer:

```
3
6
```
3. (Regular expressions, 12 points)

(a) (4 points) Indicate whether or not the following strings are matched by the regular expression \( aa^*bb^*|b^+|(bb|aa)^* \).
(Write “A” if accepted, and “N” if not accepted.)

i. aabb  
ii. bbb  
iii. aaa  
iv. aaaa

(b) (4 points) Give a Ruby-compliant regular expression that defines the language containing all strings that have an even (but nonzero) number of \( a \) characters followed one, two, or three \( b \) characters. For example, aab, aabb, aaaab, and aaaabbb are all strings in this language.

\( (aa)^+b | (aa)^+bb | (aa)^+bbb \)

(c) (4 points) Give a Ruby-compliant regular expression that accepts the same strings as regular expression \( a(b|c)^*d^+ \) but not strings abd or acd.

\( ad^+ | a(b|c)d^+ | a(b|c)(b|c)^+d^+ \)
4. (Finite automata concepts, 8 points)

(a) (5 points total, 1 point each) Indicate whether the following are true (“T”) or false (“F”):

i. ___ Finite automata require at least one final state.

ii. ___ A finite automaton with an $\varepsilon$-transition must be an NFA.

iii. ___ DFAs are always larger than their equivalent NFAs.

iv. ___ The finite automaton above accepts the string abbb.

v. ___ The finite automaton above accepts the string ababab.

(b) (3 points) Provide a regular expression for the following DFA:
5. (Nondeterministic finite automata, 12 points total, 3 points each)
Reduce the following regular expressions to NFAs.

(a) ab

(b) c*d

(c) ab*c*d

(d) (ab|ba)*
6. (Deterministic finite automata, 12 points total, 6 points each) Reduce the NFAs to DFAs.

(a)

(b)
7. (Context-free grammars, 8 points)

(a) (5 points) Consider the following grammar (where S is the start symbol):

\[
\begin{align*}
S & \rightarrow T \mid U \\
T & \rightarrow xSy \mid xy \mid \epsilon \\
U & \rightarrow yT
\end{align*}
\]

It matches one of these strings. Circle the one it matches, and draw a parse tree for it.

- xxyx
- xyxxxy
- yxxxy
- yxxxxy

(b) (3 points) The above grammar is ambiguous. Modify it to be unambiguous.
8. (Ruby programming, 30 points)

Write a Ruby class Multiset that implements a multiset. A multiset (a.k.a. a bag) is just a set in which the same element may appear multiple times. The skeleton code for the class is given on the next page. In brief, you will have to implement the following methods:

- add(v) adds an element v to the multiset.
- union(m) adds all the elements in multiset m to the current multiset.
- ==(m) is structural equality method on multisets; returns true if the invoked multiset contains the same elements as argument m. (Note that the call x == y ends up invoking x.==(y), so that in the definition of == the variable self plays the role of x.)
- to_a returns all the elements of this multiset as an array.
- to_h returns all the elements of this multiset as a hash. The returned hash will map each element to a non-zero count, where the count is the number of times the element appears in the multiset.

The code on the next page assumes you will store the contents of your multiset as an array, in the instance variable @contents. The constructor initialize takes a single argument arr and initializes the multiset to contain all the elements in the array arr by repeatedly invoking the add method that you will implement. Here are some “public tests” to help confirm the how a Multiset object behaves. (Note that the inspect method is like the to_s method in that it produces a string that represents the current object, but does so in a way that is more informative.)

```ruby
m = Multiset.new [1,2,3]
m.add(3)
puts m.to_a.sort.inspect # prints [1, 2, 3, 3]
puts m.to_h.inspect # prints {1=>1, 2=>1, 3=>2}

x = m.to_a
x.pop
puts (m.to_a == [1,2,3,3]) # prints true

m.union(Multiset.new [1,2,3])
puts m.to_a.sort.inspect # prints [1, 1, 2, 2, 3, 3, 3]
puts m.to_h.inspect # prints {1=>2, 2=>2, 3=>3}

p = Multiset.new [1,1,2,2,3,3]
puts (p == m) # prints false
p.add(3)
puts (p == m) # prints true
puts (m == m) # prints true
```

Other methods you may find useful:

- << adds an element to an array, e.g., x << 3 adds 3 to the (end of the) array x.
- dup copies an object, e.g., x.dup returns a copy of the array x. The sort method duplicates the original array and then sorts the duplicate before returning it.
- keys returns the keys in a hash; values returns the values.
- each, when invoked on an array, invokes the argument code block for each element in the array.
```ruby
class Multiset
  def initialize(arr)
    @contents = []
    arr.each { |v| add v }
  end

  def add(v)
    ## FILL IN
    return self
  end

  def to_h
    ## FILL IN
    return self
  end

  def ==(m)
    if m.class == Multiset then
      ## FILL IN
    else return false
    end
    return false
  end

  def union(m)
    ## FILL IN
    return self
  end

  def to_a
    ## FILL IN
    return self
  end
end
```