CMSC330 Summer 2010—Midterm #1
June 23, 2010

Name ________________________________

Do not start this exam until you are told to do so!

Instructions

• You have 80 minutes for this 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.

• This is a closed book exam. No notes or other aids are allowed.

• Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.

• 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>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Programming languages</td>
<td>/6</td>
</tr>
<tr>
<td>2</td>
<td>Regular Expressions</td>
<td>/4</td>
</tr>
<tr>
<td>3</td>
<td>Finite Automata</td>
<td>/10</td>
</tr>
<tr>
<td>4</td>
<td>NFA to DFA</td>
<td>/15</td>
</tr>
<tr>
<td>5</td>
<td>DFA Minimization</td>
<td>/12</td>
</tr>
<tr>
<td>6</td>
<td>Context Free Grammars</td>
<td>/15</td>
</tr>
<tr>
<td>7</td>
<td>Ruby Features</td>
<td>/8</td>
</tr>
<tr>
<td>8</td>
<td>Ruby Programming</td>
<td>/30</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>/100</td>
</tr>
</tbody>
</table>
1. (Programming Languages, 6 pts)
List and (briefly) describe the main steps of the compilation process (including input and resulting output). How does interpreted execution differ?

2. (Regular Expressions, 4 pts)

(a) (2pts) Write a regular expression that accepts the language:
\[ L = \{ s \mid s \text{ contains at least one 0 and at least one 1} \}. \Sigma = \{0, 1\}. \]

(b) (2pts) Describe the language accepted by the regular expression \((0|1)(0|1)0(0|1)^*\).
3. (Finite Automata, 10 pts)

(a) (4 pts) Give a DFA that accepts the language **all strings containing at least one 0 and at least one 1** from question 2a.

Consider the following NFA:

(b) (2 pts) Give a regular expression for the language accepted by this NFA. $\Sigma = \{0, 1\}$.

(c) (4 pts) For each of the strings below, indicate whether it is accepted by the NFA. Write **YES** if the string matches, **NO** if not.

   i. 0
   ii. 01011
   iii. $\epsilon$
   iv. 1010
4. (NFA to DFA, 15pts)
Apply the subset construction algorithm to reduce the following NFA to a DFA. Indicate the NFA states associated with each state in your DFA.

5. (DFA Minimization, 12pts)
Consider the following DFA:
(a) (4 pts) Using the Hopcroft algorithm for DFA minimization, split the states into initial partitions.

(b) (8 pts) What criteria is used to determine when to further split partitions? Do the partitions from part (a) need to be split? List the new partitions from the next iteration of the Hopcroft algorithm, if any.

6. **(Context Free Grammars, 15 points)**
Consider the following grammar where $\Sigma = \{a,b\}$, $N = \{S,T\}$, and $S$ is the start symbol:

\[
\begin{align*}
S & \rightarrow S+T | T \\
T & \rightarrow a | b
\end{align*}
\]

(a) (2 pts) Give the language accepted by this grammar.

(b) (3 pts) Provide a leftmost derivation of the string $b+a+b$. 
(c) (2 pts) Draw a parse tree for the string \texttt{b+a+b}.

(d) (8 pts) What would happen if we tried to use a recursive descent parser (RDP) to parse this language? Use the algorithms described in class to transform the grammar so it can be parsed with an RDP.

7. (Ruby Features, 8 points)
What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

(a) \texttt{x = \{ \\
    x['foo'] = 4 \bigg \# OUTPUT = \\
    x['bar'] = 'hello'
    puts x['foo']
    puts x['foo'].class
    x['hello'] \}

8. (Ruby Programming, 30 points)
As mentioned in class, scripting languages are often used for text processing applications. In this problem, you will write a short Ruby program to analyze sensor data measuring the water quality of the Chesapeake Bay. The first line of a sensor data file will consist of a tab-delimited header describing the attributes in the rest of the file:

```
ID   DATE   TIME   PARAMETER   VALUE
```

The remaining lines of the file each represent a particular water quality measurement, containing the fields as outlined in the header. The entries are tab-delimited and will have the following format:

```
<integer><t:mm/dd/yyyy><t<hh:mm><t<string><t<float>
```

An example file might look like this:

<table>
<thead>
<tr>
<th>ID</th>
<th>DATE</th>
<th>TIME</th>
<th>PARAMETER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>177531</td>
<td>01/30/2009</td>
<td>09:12</td>
<td>PH</td>
<td>8.07</td>
</tr>
<tr>
<td>177531</td>
<td>01/30/2009</td>
<td>09:12</td>
<td>SALINITY</td>
<td>15.89</td>
</tr>
<tr>
<td>177532</td>
<td>01/30/2009</td>
<td>09:14</td>
<td>TN</td>
<td>0.4497</td>
</tr>
</tbody>
</table>

Implement a class `BaySensor` that includes the following methods:

(a) (18 pts) `parse(filename)` reads the contents of a file “filename” and stores the records in the `BaySensor` class. Return true if filename is a valid sensor file with the format above; otherwise, return false.
(b) (6 pts) `find_record(id)` takes a record id and returns the rest of the record information (i.e., date, time, parameter, value). If the id is not found, return nil.

(c) (6 pts) `avg(p)` takes a parameter name p and will output its average value over the entire sensor file. If the parameter does not occur in the file, return nil.

You can assume that in all properly formatted files, dates and times will be within valid ranges. In addition the `find_record` and `avg` methods will only be called for sensor data where `parse` returned true. The `parse` function can also store values in class instance variables that can be used by the `find_record` and `avg` methods.

Helpful functions:
- `File.new(filename, mode)` // opens filename in mode, returns File
- `file.eof?` // is File object file at end?
- `file.readline` // read & return single line from file
- `file.readlines` // read entire file, return array of lines