CMSC330 Fall 2011 Midterm #1

Name _____

Discussion Time (circle one): 9am 10am 11am 12pm 1pm 2pm

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

Instructions

- You have 75 minutes to take this midterm.
- This exam has a total of 100 points, so allocate 45 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- If you have a question, please raise your hand and wait for the instructor.
- Answer essay questions concisely using 2-3 sentences. Longer answers are not necessary and a penalty may be applied.
- 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.

	Problem	Score
1	Programming languages	/6
2	Ruby	/6
3	Regular expressions & finite automata	/10
4	RE to NFA	/10
5	NFA to DFA	/16
6	DFA minimization	/10
7	Ruby programming	/24
8	OCaml	/18
	Total	/100

1. (6 pts) Programming languages
   a. (3 pts) List a programming language feature that makes it easier to write programs, but make it more difficult to detect errors in the program. Explain why.

   b. (3 pts) What language feature in Ruby is similar to anonymous functions in OCaml / Java? Explain.

2. (6 pts) Ruby
   What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

   a. (2 pts)
```
a = "foo"
if a =~ /(o+)/ then
puts $1
end
```
 # Output =

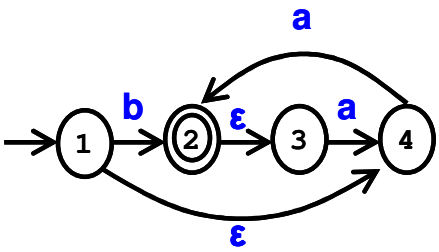
   b. (2 pts)
```
a = [ ]
a["foo"] = 1
puts a["foo"]
```
 # Output =

   c. (2 pts)
```
a = { }
a["foo"] = 1
puts a["foo"]
puts a["bar"]
```
 # Output =

3. (10 pts) Regular expressions and finite automata

   For this problem, provide regular expressions using only the concatenate, union, and closure operations. I.e., do not use Ruby regular expressions.

   a. (4 pts) Give a regular expression for the set of strings accepted by the following NFA:



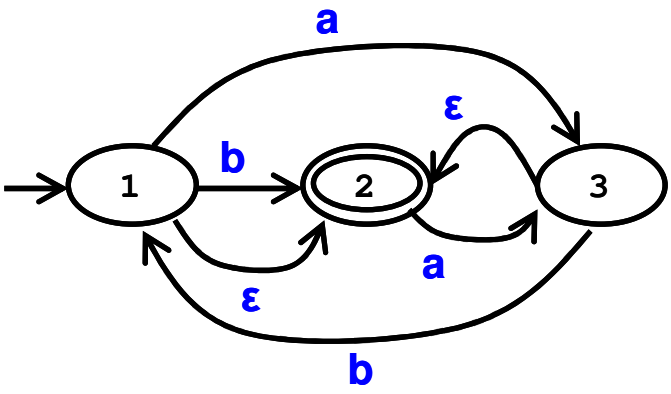
RE =

   b. (6 pts) Give a regular expression for OCaml expressions of type *int list*. Assume there are no empty lists, and all numbers are binary (i.e., digits consist of 0 or 1). For example, strings generated by your RE include: [1], [100110], [01;0], [0;1;1101], etc…

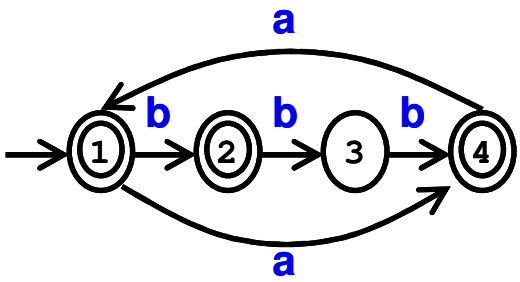
RE =

4. (10 pts) RE to NFA
   Create a NFA for the regular expression (a|bc)* using the method described in lecture.

5. (16 pts) NFA to DFA
Apply the subset construction algorithm discussed in class to convert the following NFA to a DFA. Show the NFA states associated with each state in your DFA.



6. (10 pts) DFA Minimization
Consider applying the Hopcroft DFA minimization algorithm discussed in class to the following DFA.



a. (2 pts) What are the initial partition(s) created by the Hopcroft algorithm?

b. (6 pts) Explain why one of the initial partitions needs to be split. What is the result after splitting the partition?

c. (2 pts) Is the DFA minimization algorithm finished at this point? Explain.

7. (24 pts) Ruby programming

Consider the following programming problem. Suppose you want to analyze a text file to determine how many different *unique* **and** *total* words immediately appear after each word in the file. For the purpose of this analysis you may assume that all words are lowercase or uppercase, ignore all whitespace and punctuation separating words, and assume the last word in the text file does not have any words appearing after it. Your program should accept the name of the text file as a command line argument. It should output a list of words in the text file in sorted order, with the number of different *unique* **and** *total* words following each word.

For instance, given the text file "foo" when executed as "ruby count.rb foo" the output should be as follows. The word "This" has 2 unique words (is, test) and 3 total words (is, is, test) following it. The output for "This" is thus "This 2 3". The word "test" has 2 unique words (This, should) and three total words (This, This, should) following it. The word "is" has one unique word (a) and 2 total words (2) following it. The word "work" has no words following it.

Input (content of foo)	Output
This is a test. This is a short test. This test should work.	This 2 3 a 2 2 is 1 2 short 1 1 should 1 1 test 2 3 work 0 0

Helpful functions:

```
f = File.new(filename, mode) // opens filename in mode, returns File f
f.eof?                       // is File object f at end?
ln = f.readline              // read single line from file f into String ln
a = f.readlines              // read all lines from file into array a
a = str.scan(…)              // finds patterns in String str, returns in array a
a = h.keys                   // returns keys in hash h as an array a
a.sort!                      // sorts elements of array a in place
a.size                       // number of elements in the array
a.each { … }                 // apply code block to each element in array
a.push / a.pop               // treat array as stack
ARGV                         // array containing command line arguments
```

You may NOT use methods such as include?, has_key?, uniq, join, split, chop, etc.

8. (18 pts) OCaml

a. (2 pts each) Give the type of the following OCaml expressions.

   i. [2 ; 5] **Type =**

   ii. let f x = x + 2 **Type =**

b. (2 pts each) Give the value of the following OCaml expressions. If an error exists, describe it.

   iii. 3::[2;1] **Value =**

   iv. let x = 2 in x + 5 **Value =**

c. (2 pts each) Write an OCaml expression with the following type.

   v. int list list **Code =**

   vi. int -> int **Code =**

d. (6 pts) Write an OCaml function *singleList* that returns true if its argument is a list with exactly one element and false otherwise. For instance, singleList [4] returns true, and singleList [] and singleList [3;2] both return false.