CMSC330 Fall 2010 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	Ruby features	/12
2	Regular expressions & finite automata	/12
3	RE to NFA to DFA	/20
4	DFA minimization	/10
5	OCaml Types	/18
6	Ruby programming	/28
	Total	/100

1. (12 pts) Ruby
   a. (6 pts) List 3 features of Ruby that are often found in scripting languages.

   b. (2 pts each) What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.
      i. puts "Found" if "College Park" =~ /([a-z]+)/ **Output =**
      puts $1

      ii. a = [ ] **Output =**
      a[2] = 1
      puts a

      iii. a = { 1 => 2 } **Output =**
      a.keys.each{ |x| puts "#{a[x]}" }

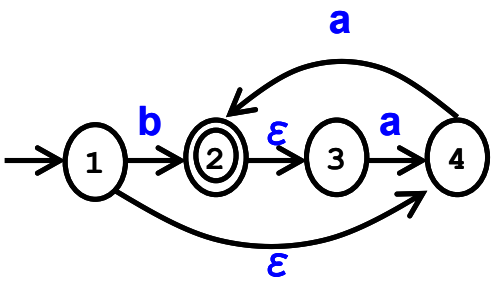
2. (12 pts) Regular expressions and finite automata
   a. (4 pts) Give a regular expression for all binary numbers (strings of 0s and 1s) beginning and ending with 1 (e.g., 1, 111, 10011).

   b. (4 pts) Give a DFA for all binary numbers (strings of 0s and 1s) beginning and ending with 1 (including 1).

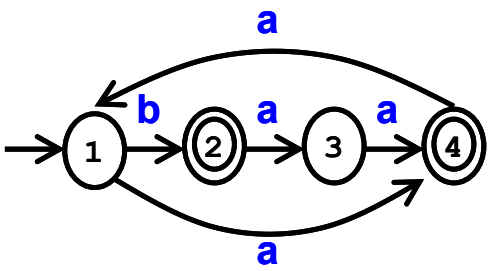
   c. (4 pts) Explain why adding a **not** operator to the standard **union**, **concatenation**, and **closure** operators does not make regular expressions more powerful.

3. (20 pts) RE to NFA to DFA
   a. (8 pts) Create a NFA for the regular expression ab|c*d using the method described in lecture. Remember precedence is in the order: closure > concatenation > union.

   b. (12 pts) 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.



4. (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. (5 pts) Which partition needs to be split first, if any? Explain.

c. (3 pts) Is the DFA minimization algorithm finished at this point? Explain.

5. (18 pts) OCaml
   a. (2 pts each) Give the type of the following OCaml expressions.
      i. [[1 ; 3] ; [2]] **Type =**
      ii. let x y = y + 1 **Type =**
   b. (3 pts each) Give the value of the following OCaml expressions. If an error exists, describe it.
      i. [5]::[6] **Value =**
      ii. let x = 1 in x + 3 **Value =**
   c. (3 pts) Write an OCaml expression with the following type
      i. int -> int list **Code =**
   d. (5 pts) Write an OCaml function non_empty that returns true if its argument is a non-empty list and false otherwise. For instance, non_empty [ ] returns false, and non_empty [1] returns true.

6. (28 pts) Ruby programming

Consider the following programming problem. Suppose you want to analyze a text file to determine how many different unique 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 words following each word.

For instance, given the text file "foo" when executed as "ruby count.rb foo" the output should be as follows. This has 2 unique words following it (is, test), and test has 2 unique words following it (This, should). All other words only have 1 unique word following them.

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

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