Midterm #1

CMSC 330: Organization of Programming Languages
Spring 2010

March 10, 2010

Name ____

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

Instructions

Do not start until told to do so!

- This exam has 8 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.
- 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.

You may avail yourself of the punt rule. If you write down *punt* for any part of a question with a specifically-assigned point value, you will earn 1/5 of the points for that question (rounded down).

	Question	Points	Score
1	Programming Languages	10	
2	Regular Expressions	8	
3	Finite Automata	10	
4	NFA to DFA	14	
5	DFA Minimization	10	
6	Ruby Features	18	
7	Ruby Programming	30	
	Total	100	

1. (Programming Languages, 10 points)

   (a) (5 points) When working with an interpreted language, after making a change to your program, you will be able to start running it more quickly than if you were using a compiled language. Why is this?

   (b) (5 points) In Ruby we say that "everything is an object." Explain what this means, using a Ruby code snippet to illustrate.

2. (Regular Expressions, 8 points)

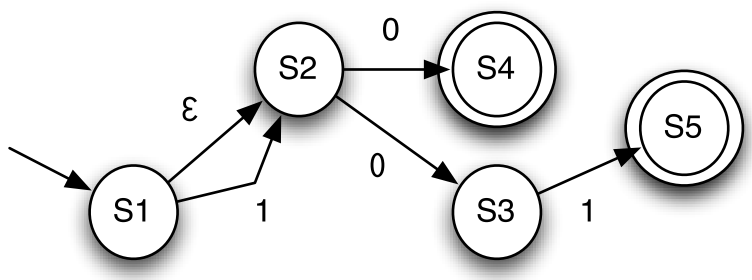
   (a) (4 points) Give a regular expression that matches all binary numbers (strings of 0s and 1s) containing three consecutive 0s.

   (b) (4 points) Convert the Ruby regular expression `/[01]2+0/` to an equivalent one that does not use `+` or character classes:

3. (Finite Automata, 10 points)

(a) (4 points) Give an DFA that accepts all binary numbers having exactly two digits where at least one digit is a 0.

(b) (6 points) For each of the following three strings, indicate whether it is accepted by the given NFA:

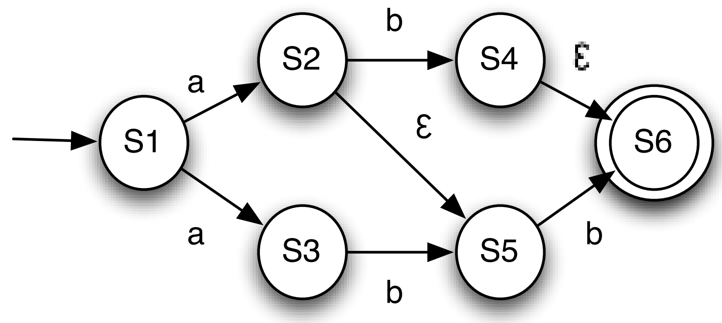


Write YES if it matches, NO if not

- 101
- 0
- 00

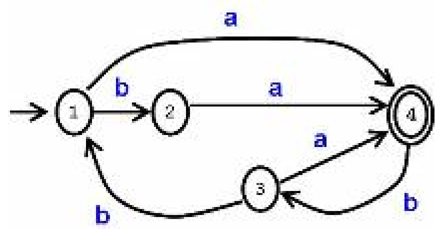
4. (NFA to DFA, 14 points)

   Apply the subset construction algorithm to convert the following NFA to a DFA. Show the NFA states associated with each state in your DFA.



5. (DFA minimization, 10 points)

   Consider the following automaton:



   (a) (4 points) Using the DFA minimization algorithm discussed in class, list the states in each initial partition.

   (b) (6 points) Do any partitions need to be split? If so, explain why.

6. (Ruby Features, 18 points)

   (a) (5 points) Write the output of the following Ruby code sequence. If the code fails with an exception, write the output to the point it fails, and then write FAIL.

```
x = { "hello" => 1, "bye" => 2}
puts x["hello"]
puts x[1]
```

   Output:

   (b) (5 points) Write the output of the following Ruby code sequence. If the code fails with an exception, write the output to the point it fails, and then write FAIL.

```
["hello","bye","jello"].each { |s|
  x = s =~ /ello/
  if x then puts "match"
  else puts "non-match" end
}
```

   Output:

   (c) (8 points) Convert the following Ruby class into an equivalent Java class. Be sure you use the correct access modifiers for your Java class to express equivalent access restrictions to the Ruby class.

```
class Point
  @@cnt = 0
  def initialize(x,y)
    @x = x; @y = y; @@cnt = @@cnt + 1;
  end
  def getX() @x end
  def getY() @y end
  def to_s() "(#{@x},#{@y})[#{@@cnt}]" end
end
```

7. (Ruby Programming, 30 points)

   (a) (16 points)

   Write a class `Set` that implements a set of elements. Your class should have the following interface:

   - `add(x)` adds `x` to the set; returns the set itself
   - `remove(x)` removes `x` from set if present; returns the set itself
   - `member?(x)` returns `true` if `x` is in the set, `false` otherwise
   - `elems` returns a *copy* of the contents of the set as an array

   Here is an IRB session with the class. (Some parts of the IRB responses have been replaced with ... to hide implementation details.)

```
>> s= Set.new
=> #<Set:0x5ee8cc ...>
>> s.add(1).add(2).add(3)
=> #<Set:0x5ee8cc ...>
>> s.remove(1).add(2)
=> #<Set:0x5ee8cc ...>
>> s.member?(2)
=> true
>> s.member?(1)
=> false
>> s.member?(4)
=> false
>> s.elems
=> [2, 3]
```

(b) (14 points)

Implement a method `tally` whose argument is an array of arrays, where each array contains a pair of strings: a student name, and a course name. The student name is first name, a space, and then a last name. The course name is four capital letters followed by three digits. You can assume the caller will always provide the input in this format.

The `tally` method should return a pair of arrays: the first contains all unique first names from the input, and the second contains all unique course names. You may use your `Set` class to implement this method, or use another data structure.

For example, given the input:

```
[["Mike Hicks","CMSC351"], ["Joe Smith","CMSC412"],
 ["Mike Stafford","CMSC351"], ["Alice Baker","CMSC330"]]
```

The `tally` method would return

```
[["Mike","Joe","Alice"],["CMSC351","CMSC412","CMSC330"]]
```

(Though the order of names and courses doesn't matter.)