

Name (PRINTED): _____		
University ID #: _____		
Circle your TA's name:	Guilherme	Nir
Circle your discussion time:	12:00	1:00

CMSC 330

Exam #1

Fall 2006

**Do not open this exam until you are told. Read these instructions:**

1. This is a closed book exam. **No notes or other aids are allowed.**
2. **You must turn in your exam immediately when time is called at the end.**
3. This exam contains 6 pages, including this one. **Make sure you have all the pages.** Each question's point value is next to its number. **Write your name on the top of all pages before starting the exam.**
4. In order to be eligible for as much partial credit as possible, show all of your work for each problem, and **clearly indicate** your answers. Credit **cannot** be given for illegible answers.
5. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.
6. If you have a question, raise your hand. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn't need to do this at all, so be careful when making assumptions.
7. If you need scratch paper during the exam, please raise your hand. Scratch paper must be turned in with your exam, with your name and ID number written on it. Scratch paper **will not** be graded.
8. Small syntax errors will be ignored in any code you have to write on this exam, as long as the concepts are correct.
9. The Campus Senate has adopted a policy asking students to include the following handwritten statement on each examination and assignment in every course: "*I pledge on my honor that I have not given or received any unauthorized assistance on this examination.*" Therefore, **just before turning in your exam**, you are requested to write this pledge **in full** and **sign it** below:

---



---

Good luck!

1	2	3	Total

1. [20 pts.] **Short Answer.**

- a. [5 pts.]
- Briefly**
- list one advantage of an interpreter versus a compiler, and one disadvantage.

**Answer:**

Advantages:

- Interpreters are far simpler than compilers
- Interpreters are often more portable than compilers
- Interpreters often include an interactive mode
- Interpreters often execute without time-consuming compilation activities

Disadvantages:

- Interpreters usually execute programs slower than compilers
- In order to run your program on an interpreter, you often (but not always) need to give them your code in source form.

- b. [5 pts.] List the four classes of variables in Ruby.

**Answer:** Local variables, instance variables (@), class variables (@@), and global variables (\$).

- c. [10 pts.] Suppose in Ruby we set
- `a = [1,2,3]`
- . Write Ruby code that copies
- `a`
- into variables
- `b`
- and
- `c`
- . The variable
- `b`
- should be a
- shallow copy*
- , and the variable
- `c`
- should be a
- deep copy*
- . Your solution should work for any array
- `a`
- , not just the particular example above.

**Answer:**

```
b = a
c = []
a.each { |x| c.push x }
# or...
c = Array.new(a)
```

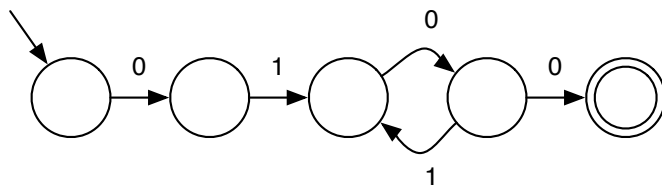
2. [50 pts.] **Regular Expressions and Finite Automata.** In your answers to these questions, please stick to the formal notation we showed you in class. If you want to use any other abbreviations, describe precisely what your abbreviations mean.

a. [10 pts.] Write a regular expression that accepts the following language

The set of strings of 0's and 1's that do not contain the sequence 11.

**Answer:**  $(10|0)^*(1|\epsilon)$

b. [15 pts.] Give a DFA that accepts the string  $01(01)^*00$



**Answer:**

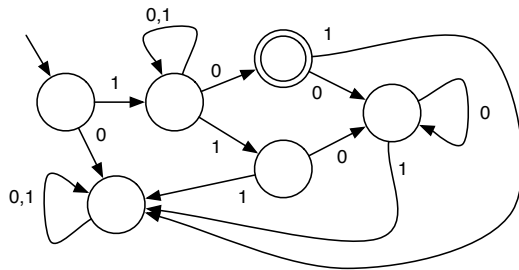
- c. [10 pts.] Divide the following 5 regular expressions, DFAs, and NFAs into subsets such that all the elements of each subset describe the same language. Write your answer in the boxes below by writing the letters for the elements of the sets (in any order) in the boxes, one set per box. You may or may not fill in all boxes.

a, d	b, c	e		
------	------	---	--	--

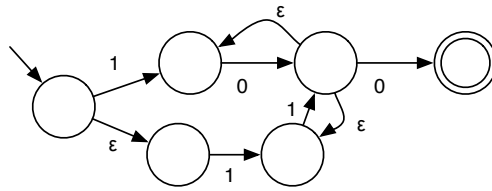
(a)  $1(0|1)^*0$

(b)  $(100^*|110^*)$

(c)  $1(1|0)0^*$



(d)



(e)

- d. [15 pts.] Show that if a DFA has  $n$  states and the DFA accepts a string of length  $k$  with  $k > n$ , then the language accepted by the DFA contains an infinite number of strings.

**Answer:** Since  $k > n$  and the string is accepted, there must be a cycle in the DFA such that the cycle has a path to a final state. Hence the DFA must accept an infinite number of strings.

3. [30 pts.] **Ruby.** Aunt Bertha needs some help. She has many text files on her computer, and she needs to identify which ones contain recipes and not things like, e.g., Apache web logs. Write a Ruby script that opens the file `recipe.txt` and prints `yes` if the file is a valid recipe, or `no` otherwise. You may use any Ruby standard library functions. If you forget the exact name or arguments of a library function, make a reasonable guess and explain your assumptions about the function. Minor mistakes in syntax will be ignored if you have the correct concept.

A recipe file consists of zero or more lines that only describe ingredients:

- Each ingredients line is of the form *num unit description*, each separated by one space, where
  - *num* is either a non-zero whole number such as 3 or 42, or a ratio of two non-zero whole numbers separated by a forward slash, such as 1/2 or 3/4.
  - *unit* is either `tsp`, `tbsp`, `cup`, `ounce`, or is empty (i.e., there is no unit). if there is no unit, then only one space should separate *num* and *description*. Notice that the units are all singular to keep things simple.
  - *description* is arbitrary text, made up of any number of any symbols, and cannot be empty.

**Answer:**

```
num = "[1-9]\\d*(/[1-9]\\d*)? "  
unit = "((tsp|tbsp|cup|ounce) )?"  
desc = ".+"  
ing = Regexp.new("^" + num + unit + desc + "$")  
  
valid = true  
File.open("recipe.txt", "r") { |f|  
  f.readlines.each { |line|  
    if not (line =~ ing)  
      valid = false  
    end  
  }  
}  
if valid then  
  puts "yes"  
else  
  puts "no"  
end
```

Name: \_\_\_\_\_

(You may continue or write your Ruby program here)