

# CMSC330 Fall 2009 Midterm #1

Name \_\_\_\_\_

Discussion Time (circle one): 10am 11am 12pm 1pm 2pm 3pm

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

## Instructions

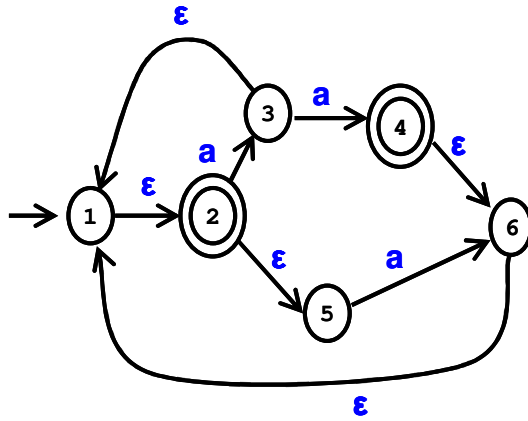
- You have 75 minutes to take this midterm.
- This exam has a total of 100 points, so allocate 45 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- If you have a question, please raise your hand and wait for the instructor.
- Answer essay questions concisely 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	/9
2	REs & finite automata	/15
3	NFA to DFA	/18
4	DFA minimization	/8
5	Context Free Grammars	/8
6	Ruby features	/6
7	Ruby programming	/36
	Total	/100

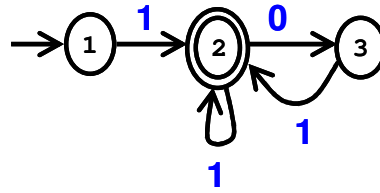
1. (9 pts) Programming languages
  - a. (3 pts) Explain the difference between interpretation and compilation.
  
  
  
  
  
  
  
  
  
  
  - b. (6 pts) Explain the difference between static and dynamic types. Write a small piece of Ruby code that relies on dynamic types.
  
  
  
  
  
  
  
  
  
  
2. (15 pts) Regular expressions and finite automata
  - a. (5 pts) Give a regular expression for all binary numbers (strings of 0s and 1s) with an odd number of 0s.
  
  
  
  
  
  
  
  
  
  
  - b. (4 pts) Give a NFA that accepts all binary numbers (strings of 0s and 1s) with an even number of 0s.
  
  
  
  
  
  
  
  
  
  
  - c. (6 pts) Create a NFA for the regular expression  $((abc)d)^*$  using any method.

3. (18 pts) NFA to DFA

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



4. (8 pts) DFA Minimization. Consider the following DFA.



a. (2 pts) What criterion is used to initially place DFA states in one of 2 partitions?

b. (6 pts) Assume states 1 & 3 are in one partition, and state 2 is in a different partition. Are states 1 & 3 distinct (i.e., should be placed in 2 different partitions)? Explain.

5. (8 pts) Context free grammars

a. (4 pts) Provide a *rightmost* derivation of the string “110” for the following grammar:  $S \rightarrow SS \mid 1 \mid 0$

b. (4 pts) Create a context free grammar for binary numbers (strings of 0s and 1s) with exactly one more 1 than 0s. Your grammar may be ambiguous.

6. (6 pts) Ruby features

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

a. `“330” =~ /(3+)/`      # **Output =**  
`puts $1`  
`puts $2`

b. `a = { }`      # **Output =**  
`a[“b”] = 3`  
`a.keys.each{ |x| puts x }`

7. (36 pts) Ruby programming

In this problem, you will write a short Ruby program that analyzes an assembly-like program stored in a text file. Each line in the file should have the form:

<label>: <opcode> <operands...>

I.e., label followed by instruction (every line must have a label). There is one space to the right of the :, and one to the right of the opcode. The following table lists the three kinds of instructions and their operand formats

Instructions	ldc <reg>, <int> sub <reg>, <reg> bz <reg>, <label>
<label>	Sequence of lower case letters
<opcode>	ldc, sub, bz
<reg>	r0, r1, ..., r9
<int>	Sequence of digits

Implement a class *AssemblyProgram* with the following methods:

- (22 pts). *compile*(filename) reads the contents of file "filename" and returns true if it matches the format above, and false otherwise.
- (7 pts) *find\_label*(x) takes a label x and returns the line number of the statement with label x. Line numbers start at 1. If label x does not exist for any instruction, then *find\_label* should return nil.
- (7 pts) *verify\_labels* checks that all branch target labels exist (these are labels that are operands of bz instructions). It returns true if all labels are verified to exist, and false otherwise.

For example, for the following program in file "prog"

```

start: ldc r0, 42
a: ldc r1, 37
b: sub r0, r1
c: sub r0, r2
branch: bz r0, b

```

**# p = AssemblyProgram.new**  
**# p.compile("prog") = true, since prog has correct syntax**  
**# p.find\_label("a") = 2**  
**# p.find\_label("start") = 1**  
**# p.find\_label("branch") = 5**  
**# p.verify\_labels = true, since branch target "b" exists**

You may assume that all instructions will have unique labels, and the functions *find\_label* and *verify\_labels* will be invoked only for programs for which *compile* returned true. The function *compile* may store information for use by *find\_label* and *verify\_labels*.

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