

Name (PRINTED):	_____		
University ID #:	_____		
Circle your TA's name:	David	Brandon	
Circle your discussion time:	10:00	11:00	12:00

CMSC 330

Exam #1

Fall 2008

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 7 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	4	Total

1. [10 pts.] **Short Answer.**

- a. [5 pts.] Consider the Ruby program `x=nil; y = x + "foo"; .` If we were to write a similar program in Java (substituting `null` for `nil`), the JVM would throw a `NullPointerException` at the second statement. What error would the Ruby interpreter issue for this program? (Roughly; don't worry about the exact wording.)

Answer: Since everything is an object, including `nil`, Ruby would report that there is no method `+` defined for `NilClass`. Specifically, the error is

```
NoMethodError: undefined method '+' for nil:NilClass
from (irb):1
```

- b. [5 pts.] Explain the difference between calling `s.chomp` and `s.chomp!`. (By default, both methods remove carriage returns or newlines from the end of a string.)

Answer: The first method, `s.chomp`, returns a new string without affecting `s`. The second method, `s.chomp!`, mutates the string `s`. In general, the convention is that methods that end in `!` mutate their receiver.

2. [30 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. For the following questions, $\Sigma = \{0, 1\}$.

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

The set of strings that have at least one 0 and at least one 1.

Answer: Three possible answers (out of many):

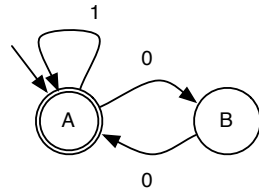
$$1^*0(0|1)^*10^* \mid 0^*1(0|1)^*01^*$$

$$(0|1)^*(01|10)(0|1)^*$$

$$0(0|1)^*1(0|1)^* \mid 1(0|1)^*0(0|1)^*$$

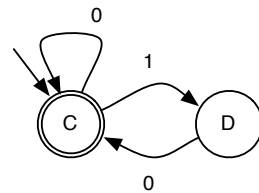
b. [20 pts.] Give DFAs that accept the following languages:

i. $(00 \mid 1)^*$



Answer:

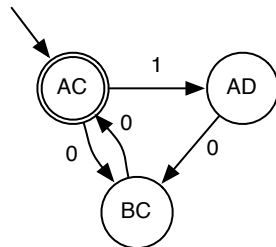
ii. $(10 \mid 0)^*$



Answer:

iii. The intersection of languages (i) and (ii). *Hint: Make a DFA whose states represent pairs of states in the DFAs (i) and (ii) above.*

Answer: The intersection is the language $(100|00)^*$ (you didn't have to give the regular expression, but it helps understand the DFA).



3. [30 pts.] **Context-Free Grammars** Consider the following grammar, where $\{E\}$ is the set of non-terminals, and $\{a, b, +, *\}$ is the set of terminals.

$$\begin{aligned} E &\rightarrow a \\ E &\rightarrow b \\ E &\rightarrow EE \\ E &\rightarrow E+E \\ E &\rightarrow E* \end{aligned}$$

- a. [8 pts.] Write “yes” to the right of the strings below that are accepted by the above grammar, and “no” to the right of those that are not:

ab+b	yes	ab++ba	no
aaabba	yes	a**	yes
a+*b	no	a*+b	yes
*ab	no	abb*aa	yes

- b. [8 pts.] Show that the grammar is ambiguous by giving two leftmost derivations for the same string.

Answer: There are many possible answers for this problem. For example,

$$E \Rightarrow EE \Rightarrow EEE \Rightarrow aEE \Rightarrow aaE \Rightarrow aaa$$

$$E \Rightarrow EE \Rightarrow aE \Rightarrow aEE \Rightarrow aaE \Rightarrow aaa$$

c. [14 pts.] Write a non-ambiguous grammar for the same language, resolving ambiguities as follows:

- Juxtaposition (the production $E \rightarrow EE$) and $+$ associate to the left.
- $*$ binds more tightly than juxtaposition, and juxtaposition binds more tightly than $+$.

For example, here are some strings and their parses.

String	Parse	String	Parse
aaa	(aa)a	a+a+a	(a+a)+a
ab*	a(b*)	a+b*	a+(b*)
a+bb	a+(bb)	a**	(a*)*

Answer:

$$E \rightarrow E+T \mid T$$

$$T \rightarrow TR \mid R$$

$$R \rightarrow R* \mid P$$

$$P \rightarrow a \mid b$$

4. [30 pts.] **Ruby.** Write a Ruby method `load_graph(f)` that takes a file name `f` as input, and returns an instance of the `Graph` class you wrote for project 2 that contains the edges specified in the file. A file describing a graph consists of zero or more lines of the following form:

- Each line is of the form `node -> node`, where the node name on the left is followed by a space, a dash, a greater-than sign, another space, and then another node name.
- A node name `node` is made up of one or more upper- or lowercase letters.

To make it a bit easier to use, your method should handle errors in the following way:

- If there is a line of text that does not match the above format, your method should print a message saying `Ignoring line n`, where `n` is the number of the line that is ignored. (The first line of the file is numbered 1.)
- If there is a line of text that specifies an edge already present in the graph, your method should print a message saying `Duplicate edge on line n`, where `n` is the number of the ignored line.

In both cases, your method should continue to process subsequent lines in the file after the error. You need not handle any other errors.

In your answer, 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. You may use the methods of `Graph` given in the project specification. In particular, you will likely want to call `add_edge(from, to)` and `has_edge?(from, to)`. Minor mistakes in syntax will be ignored if you have the correct concept.

Answer:

```
$r = /^[a-zA-Z]+ -> ([a-zA-Z]+)$/

def load_graph(f)
  g = Graph.new
  lines = open(f, "r") { |file| file.readlines }
  num = 0
  lines.each { |line|
    line.chomp!
    num = num + 1
    if (line =~ $r)
      if g.has_edge($1, $2)
        printf("Duplicate edge on line %d", num)
      end
      g.add_edge($1, $2)
    else
      printf("Ignoring line %d", num)
    end
  }
  return g
end
```

Several people chose to have `load_graph` be an instance method of `Graph`. This was not what we intended, but we did not take off points for it.

Name: _____

(You may continue or write your Ruby program here)