CS 340, Fall 2014 — Dec 11th/13th — Final Exam   Name: ____________________

Note: in all questions, the special symbol ϵ (epsilon) is used to indicate the empty string.

Question 1. [5 points] Consider the following regular expression;

(a|ab|bac)*

For each of the strings below, circle it if it is in the language generated by the regular expression, and cross it out if it is not in the language generated by the regular expression.

- ϵ
- **a**
- **b**
- **aba**
- **bab**
- **bac**
- **bacaa**
- **babac**
- **abbac**
- **abbbac**

Question 2. [5 points] Specify a regular expression that generates the language over the alphabet {**a**, **b**, **c**} of all strings not containing the substring **ac**.

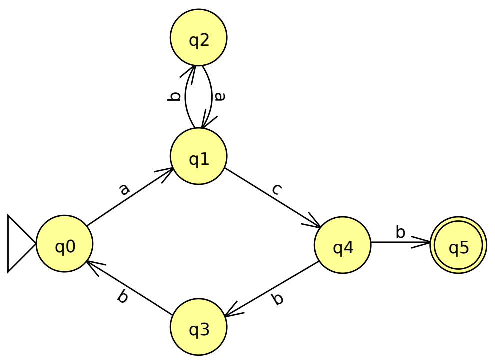
Examples of strings in the language:

- ϵ
- **aaab**
- **caaaa**
- **abc**
- **bbc**

Examples of strings not in the language:

- **ac**
- **aac**
- **cac**
- **bacb**
- **bbac**

Question 3. [5 points] Consider the following nondeterministic finite automaton (NFA):



For each of the strings below, circle it if it is in the language accepted by the NFA, and cross it out if it isn't.

- ϵ
- **a**
- **ac**
- **acb**
- **acbbac**
- **acbbacb**
- **abacb**
- **abacbb**
- **aabcb**
- **ababacb**

Question 4. [10 points] Specify a *deterministic* finite automaton (DFA) that generates the language over the alphabet {**a**, **b**, **c**} of all strings not containing the substring **ac**. (See Question 2 for lists of example strings in the language and not in the language.)

In your DFA, be sure to indicate a start state and one or more final states, and make sure each transition specifies a direction and a single input symbol consumed.

Question 5. [5 points] Consider the following context-free grammar (CFG), where **P** is the start symbol:

$$\mathbf{P} \rightarrow \mathbf{pP}$$
$$\mathbf{P} \rightarrow \mathbf{qP}$$
$$\mathbf{P} \rightarrow \mathbf{r}$$

For each of the strings on the right, circle the string if is in the language generated by the CFG, and cross it out if it isn't.

- **p**
- **q**
- **r**
- **ppr**
- **qqr**
- **prr**
- **pqrr**
- **pqqp**
- **pqqpr**
- **qq**

Question 6. [10 points] Specify a context-free grammar (CFG) which generates the language of all *AList*s. An *AList* has the following properties:

- It is a sequence of terminal symbols chosen from $\boxed{\textbf{( ) a ,}}$
- It begins with **(** and ends with **)**
- It contains (between the parentheses) a comma-separated list of 0 or more *AListMembers*

An *AListMember* is either the terminal symbol **a** or an *AList*.

Examples of strings in the language	Examples of strings not in the language:
()	ϵ
(a)	**a**
(a, a)	**a, (a)**
((a, (a, (a), a)), a)	**(a**
	(a, a))

Be sure to indicate which nonterminal symbol in your grammar is the start symbol.

Question 7. [10 points] Consider the following context-free grammar with start symbol $\boxed{\text{A}}$, nonterminal symbols $\boxed{\text{A E F V N}}$ and terminal symbols $\boxed{(\ )\ \lambda\ .\ \text{a b 1 2}}$

$$\begin{aligned}&\text{A} \rightarrow \text{E} \mid \text{E A}\\ &\text{E} \rightarrow \text{F} \mid \text{V} \mid \text{N}\\ &\text{F} \rightarrow (\ \lambda\ \text{V}\ .\ \text{E}\ )\\ &\text{V} \rightarrow \text{a} \mid \text{b}\\ &\text{N} \rightarrow 1 \mid 2\end{aligned}$$

Assume that you are writing a recursive descent parser for this grammar. Assume you have a lexer supporting `peek()`, `next()`, and `expect()` operations. (You can assume that the lexer operations are stateful, i.e., you don't have to assume that you are using a functional language.) Use pseudo-code to show how the following parse functions would be implemented. You don't have to build the parse tree, but do show all lexer operations and calls to parse functions. *Hint*: for `parseE`, think about how to use the lexer to decide whether tokens matching an occurrence of the F, V, or N nonterminal is about to begin.

(a) `parseE() {`

(b) `parseA() {`

Programming Questions

First things first: the following web page specifies which resources you may use during the exam, and links to the permitted resources:

```
http://ycpcs.github.io/cs340-fall2014/assign/final.html
```

Start by downloading the exam zipfile using the command

`wget` *zipfileURL*

where *zipfileURL* is the URL of the exam zipfile.

Import the zipfile as an Eclipse project. You should see a project called **cs340-final**. You will be editing the file `src/cs340_final/core.clj`.

There are four functions to implement: **double-it**, **double-them**, **count-all-nested**, and **vec->tower**.

Make sure you meet all of the requirements of each function: e.g., if the function is required to be tail recursive, make sure your function is tail recursive.

Some general hints:

- `(empty? s)` tests whether sequence `s` is empty
- `(first s)` gets the first element in sequence `s`
- `(second s)` gets the second element in sequence `s`
- `(last s)` gets the last element in sequence `s`
- `(rest s)` gets a sequence with all but the first element of sequence `s`
- `[]` is an empty vector
- `(conj v elt)` creates a vector which results from appending `elt` onto the vector `v`
- `(conj v elt1 elt2)` returns a vector which results from appending `elt1` and `elt2` onto the vector `v`
- `(subvec v i j)` returns a vector with of the elements of vector `v` from index `i` (inclusive) to index `j` (exclusive)

You can run the command `lein test` in a terminal window to run unit tests.

When you are done, submit your code using the blue up arrow icon (Submit project) in Eclipse, or run the command `make submit`. Type your Marmoset username and password when prompted.

Have fun!