CS 340, Fall 2013 — Oct 9th — Midterm 1 Name: _____

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

Question 1. [10 points] Consider the following regular expression:

ba(bac|bc)*a(c)*

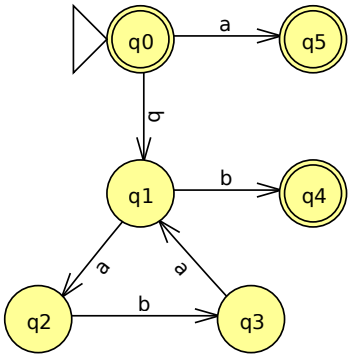
For the following strings, circle the strings that are in the language generated by this regular expression, and cross out the strings that are not in the language.

ϵ	**baac**
b	**babaca**
ba	**babcac**
baa	**babcacc**
bac	**babacbcaa**

Question 2. [10 points] Create a regular expression which generates the language of all strings over the alphabet {**a**, **b**} where each **b** is followed by exactly 2 or 3 occurrences of **a**. *I.e.*, there can never be fewer than 2 or more than 3 occurrences of **a** immediately following any **b**.

Examples of strings in the language:	Examples of strings not in the language:
ϵ	**b**
a	**ba**
aaaaaa	**abaab**
aaabaa	**abaaaa**
abaabaaabaaa	**bbaa**

Question 3. [10 points] Consider the following deterministic finite automaton (DFA):



For the following strings, circle the strings that are in the language accepted by this finite automaton, and cross out the strings that are not in the language.

ϵ	**baba**
a	**babab**
aa	**bababab**
b	**babaabab**
bb	**babaababb**

Question 4. [10 points] Create a deterministic finite automaton (DFA) that recognizes the language over the alphabet {**a**, **b**} of all strings not containing the substring **ab**.

Question 5. [10 points] Specify a context-free grammar (CFG) that generates all parenthesized, comma-separated lists of one or more list items, where each list item is an occurrence of the terminal symbol **a**. The terminal symbols are **a ( ) ,**

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

Be sure to indicate which nonterminal symbol is the start symbol.

Question 6. [20 points] Consider the context-free grammar (CFG) shown on the right, which generates strings of symbols chosen from the alphabet **a b 1 2 3 - ***

E is the start symbol.

E → T **-** E
E → T
T → F ***** T
T → F
F → **a** | **b** | **1** | **2** | **3**

(a) Show a derivation for the string

3 - 1 - 2 * a

String	Production	String	Production
E			

(b) Draw the parse tree corresponding to your derivation in (a).

(c) Briefly explain why this grammar doesn't follow the usual rules for evaluating infix expressions.

Question 7. [10 points] Consider the following productions in a context free grammar:

R → **a** S
R → **b** T

Assume that **a** and **b** are terminal symbols, and R, S, and T are nonterminal symbols. Show the pseudo-code for a recursive descent parse function for the nonterminal R (*i.e.*, a `parseR` function).

Important: Show how to use the lexical analyzer to choose between the two possible productions. Also, show how to raise an error if neither of the productions can be applied.

Programming Question

You may use the Ruby Doc website:

```
http://www.ruby-doc.org/core-1.9.3/
http://www.ruby-doc.org/core-1.9.3/Regexp.html
http://www.ruby-doc.org/core-1.9.3/String.html
```

Question 8. [20 points] Complete the program `sumint.rb` as follows. The program reads each line of text from a text file whose filename is specified on the command line. It should compute the sum of all of the integers in the file. An integer is defined as a string of 1 or more decimal digits (0-9), optionally preceded by a minus (-).

To run the program: `ruby sumint.rb` *filename*

For example, if the input file is

```
There are 42 reasons that Ruby is fun.
My favorite Yes album is 90125.
This file has 3 lines of text.
```

then the sum should be 90170. Some example text files are provided: `example.txt`, `example2.txt`, and `example3.txt`.

Approach: As each line is read, match it against a regular expression matching integers. You can use code like the following:

```
# assume that line is a string containing a line of text from the file
r = /a regular expression/
if m = r.match(line)
        # Get the text from the line that matched the regular expression
        s = m[0]

        # convert the matched part to an integer, add it to the sum
        ...
end
```

Hints:

- Use the `to_i` method to convert the matched string to an integer
- Add each integer value to the sum
- Do *not* anchor the regular expression to force it to match the beginning of a line; an integer could occur anywhere in a line

For extra credit, handle multiple integers per line.