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

**Question 1.** [10 points] Define a regular expression for the language containing all strings of symbols in the alphabet \{a, b, c\} where each occurrence of $b$ is immediately preceded and followed by at least one occurrence of $c$.

Examples of strings in the language: 

- $\epsilon$
- a
- ca
- cac
- acbcc
- cbcccacbc

Examples of strings not in the language:

- bc
- cbbc
- abc
- accb

**Question 2.** [10 points] Consider the following regular expression:

$$(bab|cab)(bac)^*(c^*)a$$

In the following list, circle each string that is in the language defined by the above regular expression:

- $\epsilon$
- babcab
- bab
- baba
- babca
- cabba
- cabbacca
- babbacca
- babacca
- cabca
Question 3. [10 points] Define a regular expression which generates all strings of symbols over the alphabet \{a, b, c\} of the form XYZ, where

- X is an a, followed by a string containing any number of (zero or more) b’s or c’s
- Y is any string beginning with bb and ending with ac, and
- Z is aa

Examples of strings in the language:
- abbacaa
- accbabcacaac
- abbcbbacaa

Examples of strings not in the language:
- \(\epsilon\)
- abacaa
- abbaacaa
- abbaacaa

Question 4. [10 points] Define a regular expression which generates all strings of symbols over the alphabet \{a, b, c\} which do not both begin with cb and end with a.

Examples of strings in the language:
- \(\epsilon\)
- abc
- cbaba
- cbaba

Examples of strings not in the language:
- cba
- cbcca
**Question 5.** [10 points] Consider the following finite automaton:

(a) Circle the strings below which are accepted by the finite automaton.

- abc
- cbbabb
- cbb
- cbbabc
- abcab

(b) Define a regular expression that generates the language recognized by the finite automaton.
Question 6. [10 points] Create a deterministic finite automaton that recognizes the language over the alphabet \{a, b, c\} of strings in which every occurrence of \textbf{a} is followed immediately by at least one \textbf{c}.

Examples of strings in the language: \begin{tabular}{ccc}
\(\epsilon\) & \(cbb\) & \(bacc\) \\
\(cbb\) & \(bacc\) & \(cac\) \\
\(bacc\) & \(cac\)
\end{tabular}

Examples of strings not in the language: \begin{tabular}{ccc}
\(cab\) & \(aac\) & \(cca\)
\end{tabular}

Be sure to indicate which state is the start state, which states are accepting (final) states, and for each transition, what input symbol is consumed.
Question 7. [10 points] Consider the following nondeterministic finite automaton (NFA):

Create a deterministic finite automaton (DFA) equivalent to this NFA. Number each state in the DFA. Include a table showing, for each reachable set of NFA states, the equivalent DFA state.
Question 8. [10 points] Consider the following context-free grammar (CFG), where uppercase letters are nonterminal symbols, and lowercase letters and digits are terminal symbols. S is the start symbol:

\[
\begin{align*}
S & \rightarrow c \ N \\
S & \rightarrow r \ N \ N \\
S & \rightarrow b \ L \ e \\
N & \rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9 \\
L & \rightarrow S \\
L & \rightarrow S \ L
\end{align*}
\]

(a) Show a derivation for the input string \textbf{b r 2 3 c 1 e} (Continue on the right-hand side if necessary.)

<table>
<thead>
<tr>
<th>String</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

(b) Draw a parse tree for the derivation in part (a).
**Question 9.** [10 points] Assume you are writing a recursive-descent parser for the context-free grammar described in the previous problem (Question 8). Show a pseudo-code sketch of the parse function for the S nonterminal symbol.

Be sure to show calls to lexical analyzer operations: `peek()` to find out the next token without consuming it, `next()` to consume the next token. Also, show calls to parse functions for nonterminal symbols: e.g., `parseN()` to expand an occurrence of the N nonterminal symbol.

You may assume that an `expect()` function is available to verify that the next token of input is a specific token. E.g., `expect('c')` would consume the next token and raise an error if it is not `c`.

You don’t need to show code to build a parse tree.
Question 10. [10 points] A simple language of postfix arithmetic expressions can be defined as follows:

- A number is one of the digits 0, 1, 2, ..., 9
- A variable is either \texttt{a} or \texttt{b}
- An operator is either + or *
- An expression is either
  1. A number
  2. A variable
  3. Two expressions followed by an operator

For example, the string

\[ 2 \ a \ + \ 3 \ * \]

Is a member of this language. Write a context-free grammar (CFG) that generates all strings in this language.