1. Write regular expressions for the following informally described languages:
   a. All strings of a’s and b’s with the subsequence abb.
   b. All strings of a’s and b’s with an even number of a’s and an odd number of b’s.

2. Consider the regular expression $\text{aac}^* | b(a|b)c^*$ defined on $\Sigma = \{a, b, c\}$.
   a. Construct the NFA for the regular expression. You can directly draw the NFA without going through the RE-to-NFA steps.
   b. Convert the NFA to DFA. Show the conversion steps.
   c. Minimize the DFA. Show the minimization steps.

3. Consider $\Sigma = \{a, b\}$. Answer the following DFA related questions. When constructing DFA, there is no need to show your construction steps, but you need to informally state how you get the DFAs.
   a. Construct a DFA that accepts $(a|b)^*$ except for abb.
   b. Construct a DFA that accepts $(a|b)^*$ except for $b^*a^*$.
   c. Based on the techniques you use in (a) and (b), can you come up with a DFA construction algorithm for the “except for” type of languages? (Just focus on the main idea.)

4. A token recognizer is designed to handle the following tokens, where $\Sigma = \{a, b, c, d\}$.
   
   $T_1 = aab$
   $T_2 = bcc^*$
   $T_3 = bcd^*$
   $T_4 = aabcd^*$

   a. Construct a minimized DFA for token recognition and the tokens are defined as follows.
   The DFA should specify the specific token names ($T_1, T_2, \ldots$) it accepts at the corresponding final states. You do not need to show the steps for the construction if you can draw the DFA directly. Note that the longest matching and first matching rules for ambiguity resolution should be used.
   b. Execute your DFA to process the following string to identify tokens. List every token string and its token name. Also, describe all the backtracking actions taken during the process.
      
      $aabacdbcbcacbcdaabbcd$ 

5. Consider the following grammar defined over $\Sigma = \{0, 1\}$.
   
   $S \rightarrow 0S11$
   $S \rightarrow S1$
   $S \rightarrow \varepsilon$

   a. Briefly describe the language generated by this grammar.
   b. Show that this grammar is ambiguous by giving a string that can be parsed in two different ways and showing the two corresponding parse trees.
   c. Rewrite the grammar to eliminate the ambiguity.

6. Let $L$ be a language defined over $\Sigma = \{0, 1\}$ and $L$ consists of all strings with the same, and even number of 0’s and 1’s.
   a. Give a context free grammar for $L$.
   b. Show a parse tree for the string 0110100.
   c. Give the leftmost derivation for (b).

7. Construct a regular grammar for the language $L$, where $L$ accepts $(a|b)^*$ except for $b^*a^*$ (start from your answer for 3).
8. Construct a type-0/1 grammar for the language \( w \text{c} \text{w} \), where \( w \) can be any string of a’s and b’s. Use some examples to illustrate how your grammar would work.

9. Eliminate left recursions for the following grammar.
   
   \[
   S \rightarrow A \text{ E} B \\
   A \rightarrow Ax | Ay | Ba | a \\
   E \rightarrow = | \neq \\
   B \rightarrow Ab | b
   \]

10. Consider the following grammar. Note that id, +, [, ], and “,” are terminals.
    
    \[
    E \rightarrow E + T | T \\
    T \rightarrow id | id[] | id[X] \\
    X \rightarrow E , E | E
    \]
    
    (a) Eliminate left recursion in the grammar.
    (b) Perform left factoring for the grammar.
    (c) Compute the First set for all symbols in the grammar.
    (d) Compute the Follow set for all non-terminals in the grammar.
    (e) Build an LL(1) parse table for the grammar.
    (f) Parse the string id + id[id+id, id[)]. Show the stack, the input, and the action taken.
    (g) Build the parse tree while you are parsing. Show your parse tree.

11. Consider the following grammar.
    
    \[
    S \rightarrow As \\
    A \rightarrow BCA \\
    A \rightarrow BCa \\
    B \rightarrow b \\
    C \rightarrow c
    \]
    
    (a) Show that the grammar is not LL(1).
    (b) Is the grammar LL(k)? If so, give the k value and show the parsing table.