1. (25pt) The identifiers in a programming language consist of one or more lower case letters, `a, b,..., z`, which must appear in non-decreasing alphabetical order. For example, correct identifiers include: `accent`, `begin`, `x`, `zzz`. Examples of incorrect identifiers are: `bad`, `id`.

   (a) (10pt) Write a regular expression for these identifiers.

   (b) (15pt) Draw a deterministic finite automaton that accepts precisely these identifiers.

2. (25pt) Consider the following grammar, G:

$$
\begin{array}{lrcl}
0. & P & \longrightarrow & S\texttt{\$} \\
1. & S & \longrightarrow & ABC \\
2. & A & \longrightarrow & \texttt{a}A \\
3. & A & \longrightarrow & \varepsilon \\
4. & B & \longrightarrow & \texttt{b}B \\
5. & B & \longrightarrow & \texttt{b}A \\
6. & C & \longrightarrow & C\texttt{c} \\
7. & C & \longrightarrow & \varepsilon
\end{array}
$$

(a) (10pt) Compute $\textsc{first}(X), \textsc{follow}(X)$, for nonterminals X, and $\textsc{predict}(p)$, for productions $p, 0 \leq p \leq 7$.

(b) (5pt) Explain why G is not LL(1). Include all conflicts G has in your explanation.

(c) (10pt) Modify G to become LL(1). Explain why the new, equivalent, grammar is LL(1). You don't need to rebuild the $\textsc{first}(), \textsc{follow}()$, and $\textsc{predict}()$ tables. Include sufficient explanation as to why the conflicts have been resolved.

3. (25pt) Consider the same grammar, G, from question 2 above, repeated here for convenience:

$$\begin{array}{rlcl}
0. & P & \longrightarrow & S\texttt{\$} \\
1. & S & \longrightarrow & ABC \\
2. & A & \longrightarrow & \texttt{a}A \\
3. & A & \longrightarrow & \varepsilon \\
4. & B & \longrightarrow & \texttt{b}B \\
5. & B & \longrightarrow & \texttt{b}A \\
6. & C & \longrightarrow & C\texttt{c} \\
7. & C & \longrightarrow & \varepsilon
\end{array}$$

(a) (15pt) Draw the LR parser in the form of the graph; the states contain the LR-items, the transitions are labelled by terminals. Reduce states are double circled. Include also (as jflap does and as shown in class) the trivial states, those containing a single LR-item with the dot at the end.

(b) (5pt) Is this grammar SLR(1)? Explain your answer.

(c) (5pt) In general, in the definition of an SLR(1) grammar, shift/reduce and reduce/reduce conflict are forbidden. What about shift/shift conflicts?

4. (25pt) Consider the following grammar, G, for arithmetic expressions in postfix notation:

$$\begin{array}{lcl} E & \longrightarrow & E\,E\,O \\ E & \longrightarrow & \texttt{const} \\ O & \longrightarrow & \texttt{+} \\ O & \longrightarrow & \texttt{-} \\ O & \longrightarrow & \texttt{*} \\ O & \longrightarrow & \texttt{/} \end{array}$$

(a) (10pt) Based on G, write an S-attributed grammar that computes the infix version of the postfix expression represented by the yield of the parse tree.

(b) (10pt) The same problem except that now you have to minimize the number of parentheses.

(c) (5pt) Draw the annotated parse tree for the string `1 2 + 3 4 5 - 6 7 - - * *` and the grammar you designed at (b). Show the attribute flow (arrows and values).

Alternative (c) (3pt) If you built a grammar for (a) but not for (b), then use the grammar you designed at (a). In this case, for a correct answer, you receive 3pt instead of 5pt. (If you solve the original (c), then you don't have to do this.)