Problem 1 [40 points]: Regular Expressions and Finite Automata

In this problem, we consider the translation of strings with integer values into a comma-separated-value (csv) format. The input string consists of a sequence of one or more letters or numerical digits (forming an identifier) followed by one or more blank space characters to which we have a set of three integers separated by the ‘-’ character and terminated by a ‘newline’ (‘\n’) character. The difficulty arises from the fact that the first integer can be a negative integer and thus begins with the ‘-’ character but the two subsequent integers are assumed to be positive. The output string has all ‘-’ character, except the first, replaced by an opening ‘(’, ‘:’ and a ‘)’ character respectively. Below you can find a couple of examples of the intended translation scheme where the ‘\n’ stands for the new-line character:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>F01 -1-0-1\n</td>
<td>F01, (-1:0), 1’\n</td>
</tr>
<tr>
<td>DX1 1-0-3\n</td>
<td>DX1, (1:0), 3’\n</td>
</tr>
</tbody>
</table>

String that do not comply with the structured described are not to be translated and the resulting translation generates an error condition. We can model this condition by having the translation program emit an error message as described in more detail in section d).

For this translation scheme, answer the following questions:

a) [05 points] Derive a regular expression RE over the ASCII character alphabet that captures valid strings and thus invalid strings.

b) [10 points] Convert the RE developed in a) into an NFA using the Thompson construction.

c) [10 points] Convert the NFA derived in section b) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.

d) [10 points] Minimize the DFA derived in section c) (or show it is already minimal) using the iterative refinement algorithm described in class.

e) [05 points] Using the C code skeleton described in class for the implementation of DFA with a table, implement a C code function that can recognize valid strings matching the RE derived in a). Whenever an error condition is detected, your implementation should print the ‘error’ string.
Problem 2 [30 points]: Predictive Top-Down Parsing

Consider the CFG grammar \( G = (N = \{ P, SL \}, T = \{ 'b', 'e', 's', ';' \}, P, E) \) where the set of productions \( P \) is given below.

\[
\begin{align*}
P & \rightarrow 'b' \text{ } SL \text{ } 'e' \\
SL & \rightarrow SL \text{ } ';' \text{ } 's' \text{ } | \text{ } 's'
\end{align*}
\]

Questions:

a) [10 points] Eliminate left-recursion in this grammar by deriving an equivalent grammar that is non-left recursive. If the resulting grammar needs to be left-factored, derive an equivalent grammar that is also left-factored and thus has the LL(1) property.

b) [10 points] Compute the FIRST and FOLLOW sets for each production’s RHS and the non-terminal symbol respectively. Use these to show that the grammar has in fact the LL(1) property.

c) [10 points] Derive the LL(1) parsing table as described in the lectures and show that in fact the grammar is parseable using the LL(1) parsing algorithm. Show the sequence of parsing actions for the input “b s ; s ; s e”.

Problem 3 [30 points]: Bottom-Up Parsing

Consider the CFG \( G = \{ NT = \{ E,T,F \}, T = \{ a,b,+,* \}, P, E \} \) with the set of productions as follows:

\[
\begin{align*}
(1) \ E & \rightarrow E + T \\
(2) \ E & \rightarrow T \\
(3) \ T & \rightarrow T \ F \\
(4) \ T & \rightarrow F \\
(5) \ F & \rightarrow F * \\
(6) \ F & \rightarrow a \\
(7) \ F & \rightarrow b
\end{align*}
\]

a) [05 points] Compute the FIRST and FOLLOW for all nonterminals in \( G \).

b) [10 points] Consider the augmented grammar \( G' = \{ NT, T, \{ (0) E' \rightarrow E S \} + P, E' \} \). Compute the set of LR(0) items for \( G' \).

c) [10 points] Compute the LR(0) parsing table for \( G' \). If there are shift-reduce conflicts use the SLR parse table construction algorithm.

d) [05 points] Show the movements of the parser for the input \( w = "a+ab*$" \).