1. Given the following grammar for expressions:

   E → E + T  \quad T → T / F
   E → E - T  \quad T → T * F
   E → T   \quad F → ( E )
   T → E

   write the generated string  \( a * (b-c) + (b-c) / a \) as a parse tree, as an abstract syntax tree, and as a DAG that is minimal.

2. Given the following attributed grammar

<table>
<thead>
<tr>
<th>PRODUCTION</th>
<th>SEMANTIC RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) L → E n</td>
<td>L.val = E.val</td>
</tr>
<tr>
<td>2) E → E_1 + T</td>
<td>E.val = E_1.val + T.val</td>
</tr>
<tr>
<td>3) E → T</td>
<td>E.val = T.val</td>
</tr>
<tr>
<td>4) T → T_1 * F</td>
<td>T.val = T_1.val * F.val</td>
</tr>
<tr>
<td>5) T → F</td>
<td>T.val = F.val</td>
</tr>
<tr>
<td>6) F → ( E )</td>
<td>F.val = E.val</td>
</tr>
<tr>
<td>7) F → digit</td>
<td>F.val = digit.lexval</td>
</tr>
</tbody>
</table>

   show the annotated parse tree for expression \((5+8*7) * 4n\).

3. Write down all the topological sorts of the following partial ordered graph:

   ![Graph Diagram]

4. This grammar generates binary numbers with a "decimal" point:

   \( S \rightarrow L \cdot L \mid L \quad L \rightarrow L \cdot B \mid B \quad B \rightarrow 0 \mid 1 \)

   Design an L-attributed SDD to compute B.val, the decimal-number value of an input string. For example, the translation of string 101.101 should be the decimal number 5.625. Hint: use an inherited attribute L.side that tells which side of the decimal point a bit is on.

5. Generate the three-address code sequences for the following instruction:

   a) \( a[i] = b*c - b*d \)
   b) \( a = b[i] + c[j] \)

6. A real array \( A[i, j, k] \) has index \( i \) ranging from 1 to 4, index \( j \) ranging from 0 to 4, and index \( k \) ranging from 5 to 10. Reals take 8 bytes each. Suppose array \( A \) is stored starting at byte 0. Find the location of:

   a) \( A[3, 4, 5] \)  \quad b) \( A[1, 2, 7] \)  \quad c) \( A[4, 3, 9] \).
7. Consider the following Post system rules:

\[
\begin{align*}
\rho \vdash e_1 : \text{bool} & \quad \rho \vdash e_2 : \text{bool} \\
\rho \vdash e_1 \text{ and } e_2 : \text{bool} & \quad \rho \vdash e_1 \text{ or } e_2 : \text{bool} \\
\rho \vdash \text{true} & \quad \rho \vdash \text{false} \\
\rho \vdash v : \text{t} & \\
\end{align*}
\]

Given \( \rho = \{ \langle a, \text{bool} \rangle, \langle b, \text{bool} \rangle, \langle c, \text{bool} \rangle \} \), show the proof of

\[ \rho \vdash a \text{ or } b \text{ and } c : \text{bool} \]

8. Define an L-attributed SDD on a top-down parsable grammar to generate the NFA associated with a regular expression, using Thompson’s algorithm sketched in the next figure. Assume that there is a token \texttt{char} representing any character, and that \texttt{char.lexval} is the character it represents. You may also assume the existence of a function \texttt{new()} that returns a new state, that is, a state never before returned by this function. Use any convenient notation to specify the transitions of the NFA.