1. [10 points]
The following NFA accepts binary numbers written \((0|1)^+b_2\) and trinary numbers written \((0|1|2)^+b_3\). (An edge labeled “a|b” is short-hand for two edges, one labeled \(a\) and the other labeled \(b\)). Construct the corresponding DFA.
2. [10 points] JLex, JFlex, and other lexical analysis generator tools in the Lex family resolve ambiguity among regular expression patterns by giving precedence to the pattern that appears first in the source file. For example, if a keyword pattern also matches the identifier pattern, we could write:

"if"    { return new Symbol(tokens.IF); }  
[a-z]+   { return new Symbol(tokens.IDENT); }  

The generated lexical analyzer will return the IF token when the first pattern is matched (even though the second pattern is also matched), but will return the IDENT token for all other strings matching the second pattern.

Given what you know about construction of lexical analyzers, describe briefly but clearly how JLex implements this precedence rule. I did not describe this in lecture, but you can figure it out; it might help to construct the DFA for the two patterns above. Remember that the overall complexity of matching a token in the input stream should be linear in the length of the token and independent of the number of regular expressions.
3. [10 points] Many languages in the C family, including Java, have declarations introduced by a type name that can be any identifier. For example:

        MyClass myObj = new MyClass();

This can introduce difficulties in parsing, as illustrated by the following grammar. The terminal symbols are \(\text{id} \), \(\text{;}\), \(\text{=}\), and the end-of-file symbol $; \lambda \) represents the empty string of symbols. Show that this grammar is not LL(1):

\[
\begin{align*}
\langle \text{block} \rangle & \rightarrow \langle \text{statements} \rangle \ \$$ \\
\langle \text{statements} \rangle & \rightarrow \langle \text{stmt} \rangle \langle \text{statements} \rangle \\
\langle \text{statements} \rangle & \rightarrow \lambda \\
\langle \text{stmt} \rangle & \rightarrow \langle \text{decl} \rangle \\
\langle \text{stmt} \rangle & \rightarrow \langle \text{assign} \rangle \\
\langle \text{decl} \rangle & \rightarrow \text{id} \ \text{id} \ \text{;} \\
\langle \text{assign} \rangle & \rightarrow \text{id} \ \text{=} \ \text{id} \ \text{;} 
\end{align*}
\]
4. [10 points] Here is nearly the same grammar, with right recursion replaced by left recursion for LR parsing. Is this grammar LALR(1)? Show why or why not.

\[
\begin{align*}
\langle \text{block} \rangle & \rightarrow \langle \text{statements} \rangle \$ \\
\langle \text{statements} \rangle & \rightarrow \langle \text{statements} \rangle \langle \text{stmt} \rangle \\
\langle \text{statements} \rangle & \rightarrow \lambda \\
\langle \text{stmt} \rangle & \rightarrow \langle \text{decl} \rangle \\
\langle \text{stmt} \rangle & \rightarrow \langle \text{assign} \rangle \\
\langle \text{decl} \rangle & \rightarrow \text{id} \text{id} \text{";"} \\
\langle \text{assign} \rangle & \rightarrow \text{id} \text{"=} \text{id} \text{";"}
\end{align*}
\]
5. [10 points] Here is nearly the same grammar again, except that all declarations must precede all assignments. Now is the grammar LALR(1)? Show why or why not.

\[
\langle \text{block} \rangle \quad \rightarrow \quad \langle \text{declarations} \rangle \langle \text{assignments} \rangle \$
\]

\[
\langle \text{declarations} \rangle \quad \rightarrow \quad \langle \text{declarations} \rangle \langle \text{decl} \rangle
\]

\[
\langle \text{declarations} \rangle \quad \rightarrow \quad \lambda
\]

\[
\langle \text{decl} \rangle \quad \rightarrow \quad \text{id} \; \text{id} \; \text{“;}”
\]

\[
\langle \text{assignments} \rangle \quad \rightarrow \quad \langle \text{assignments} \rangle \langle \text{assign} \rangle
\]

\[
\langle \text{assignments} \rangle \quad \rightarrow \quad \lambda
\]

\[
\langle \text{assign} \rangle \quad \rightarrow \quad \text{id} \; \text{“} = \text{”} \; \text{id} \; \text{“;}”
\]