1. (20 pts) LL(1) parsing
   a. (8 pts) Compute FIRST and FOLLOW for S, A for the above grammar, where ε is the empty string:

<table>
<thead>
<tr>
<th></th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>a, ε</td>
<td>$</td>
</tr>
<tr>
<td>A</td>
<td>a, ε</td>
<td>a, $</td>
</tr>
</tbody>
</table>

   b. (10 pts) Construct the LL(1) parse table for the grammar.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S → A</td>
<td>S → A</td>
</tr>
<tr>
<td>A</td>
<td>A → Aa</td>
<td>A → ε</td>
</tr>
</tbody>
</table>

   c. (2 pts) Is the grammar LL(1)? Explain.

   **No, since there are multiple entries for A for lookahead a.**

2. (6 pts) Table-driven LL(1) parsing

   Using the following LL(1) parse table, parse the input “baa”. You only need to show the stack and remaining input at each stage of the parse.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S → ba</td>
<td>S → AaC</td>
<td>S → C</td>
</tr>
<tr>
<td>A</td>
<td>A → ba</td>
<td>A → bBa</td>
<td>A → ε</td>
</tr>
<tr>
<td>B</td>
<td>B → ε</td>
<td>B → a</td>
<td>B → ε</td>
</tr>
<tr>
<td>C</td>
<td>C → ab</td>
<td>C → ba</td>
<td>C → ε</td>
</tr>
</tbody>
</table>

   Stack | Input
   --- | ---
   $ S$ | b a a $ |
   $ C$ a A | B a a $ |
   $ C$ a a B b | B a a $ |
   $ C$ a a B | A a $ |
   $ C$ a a | A a $ |
   $ C$ | $ |
   $ | $ |
3. (10 pts) Shift-reduce parsing

Given the ACTION/GOTO table above, show the parse if the current stack contents are 4 A 2 (i.e., current state is 2) and the remaining input is “bb”.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>c</td>
<td>$</td>
</tr>
<tr>
<td>0</td>
<td>Shift 2</td>
<td>Reduce S → Bb</td>
</tr>
<tr>
<td>1</td>
<td>Shift 4</td>
<td>Reduce S → c</td>
</tr>
<tr>
<td>2</td>
<td>Shift 3</td>
<td>Shift 4</td>
</tr>
<tr>
<td>3</td>
<td>Reduce A→ Ab</td>
<td>Reduce A → b</td>
</tr>
<tr>
<td>4</td>
<td>Reduce A → ε</td>
<td>Shift 3</td>
</tr>
</tbody>
</table>

4. (22 pts) LR(1) parsers
   a. (16 pts) Given the following grammar, derive its canonical sets of LR(1) items.

   \[
   S \rightarrow A \\
   A \rightarrow Aa \mid ε
   \]

   State 1
   - $S \rightarrow A$, $[$
   - $A \rightarrow Aa$, $\{a, \}$
   - $A \rightarrow \epsilon$, $\{\}$

   State 2
   - $S \rightarrow \epsilon$, $\[$
   - $A \rightarrow Aa$, $\{a, \}$
   - $A \rightarrow \epsilon$, $\{\}$

   State 3
   - $A \rightarrow Aa \epsilon$, $\{a, \}$

   Stack | Input | Action |
   --- | --- | --- |
   4 A 2 | b b $ | shift 3 |
   4 A 2 b 3 | b $ | reduce A→ Ab |
   4 A | b $ | goto 2 |
   4 A 2 | b $ | shift 3 |
   4 A 2 b 3 | $ | reduce A→ Ab |
   4 A | $ | goto 2 |
   4 A 2 | $ | accept |

   b. (2 pts) Is the grammar LR(1)? Explain.

   Yes, since there are no shift-reduce or reduce-reduce errors.

   c. (4 pts) How does this grammar example help explain a major difference between top-down and bottom-up parsers?

   Bottom up parsers can handle left recursive grammars (with productions such as $A \rightarrow Aa$), but top-down parsers cannot.
5. (16 pts) LR(1) parse table construction

Given the following sets of LR(1) items, construct the corresponding entries in the LR(1) parse table.

State 1
[S \rightarrow a \cdot A, $]  
[S \rightarrow a \cdot b, a]  
[A \rightarrow a, $]  
[A \rightarrow b, $]

State 2
[A \rightarrow b \cdot, $]  
[S \rightarrow ab \cdot, a]  

State 3
[A \rightarrow a \cdot, $]

State 4
[S \rightarrow aA \cdot, $]

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift 3</td>
<td>Shift 2</td>
</tr>
<tr>
<td>2</td>
<td>Reduce S \rightarrow ab</td>
<td>Reduce A \rightarrow b</td>
</tr>
<tr>
<td>3</td>
<td>Reduce A \rightarrow a</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Reduce S \rightarrow aA</td>
<td></td>
</tr>
</tbody>
</table>

6. (12 pts) LR(1) conflicts

Consider the following LR(1) items in a single state of a shift/reduce parser. List any conflicts that exist and describe for what lookaheads they occur.

Shift/reduce conflict:
[C \rightarrow b \cdot ad, c] and [E \rightarrow cab \cdot, a] for lookahead a

Reduce/reduce conflict:
[F \rightarrow aAab \cdot, c] and [G \rightarrow hab \cdot, c] for lookahead c

7.
(14 pts) LALR(1) parsers

a. (6 pts) Consider the following states in a LR(1) parser for a grammar G. Which states would be merged in a LALR(1) parser?

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
<th>State 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A → ab •, c]</td>
<td>[A → ab •, c]</td>
<td>[A → ab •, c]</td>
<td>[A → ab •, b]</td>
</tr>
<tr>
<td>[A → b •, a]</td>
<td>[A → b •, b]</td>
<td>[A → b •, c]</td>
<td>[A → b •, c]</td>
</tr>
<tr>
<td>[A → • b a, b]</td>
<td>[A → b • a, c]</td>
<td>[A → b • a, c]</td>
<td>[A → b • a, c]</td>
</tr>
</tbody>
</table>

States 2 and 4.

b. (4 pts) What new state(s) would result after the merge?

[A → ab •, b]  
[A → ab •, c]  
[A → b •, b]  
[A → b •, c]  
[A → b • a, c]  

or  

[A → ab •, {b,c}]  
[A → b •, {b,c}]  
[A → b • a, c]

c. (4 pts) Is the grammar G an LALR(1) grammar? Explain.

No, since there would be a reduce/reduce conflict on b and c.