1. Consider the following grammar:

\[ S \rightarrow S \ S \mid S \ S \ast \mid a \]

(a) Give a leftmost derivation for the string ‘aa+a\ast’.
(b) Give a parse-tree for the string.
(c) Is this grammar suitable for top-down parsing? If not, then make it suitable for top-down parsing.
2. Given the following grammar:

1. A → AB | c
2. B → AC | a
3. C → B | b

Use the algorithm presented in class to eliminate left recursion. List the final set of productions.
3. Consider grammar 4.28 and corresponding parsing table in Fig. 4.17 from the text. Show the moves made by a LL(1) predictive parser for the input string: \(( \text{id} \ast \text{id} )\).
4. Construct the sets of FIRST and FOLLOW for each non-terminal in the grammar below, using the algorithms we saw in class.

<table>
<thead>
<tr>
<th>Production</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → aAF</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>A → B</td>
<td>C</td>
<td>{}</td>
</tr>
<tr>
<td>B → D+</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>C → E*</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>D → x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>E → a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>F → A</td>
<td>ε</td>
<td>{}</td>
</tr>
</tbody>
</table>

5. Is the following grammar LL(1)? Prove your answer.

```
stmt → label unlabeled_stmt
label → ID : | ε
unlabeled_stmt → ID := expr ;
expr → ID | NUM
```
6. Use Figure 4.37 and the expression grammar 4.1 to parse the string \((\text{id} * \text{id}) + \text{id}\) using the LR parsing algorithm. Show the rightmost derivation in reverse.
7. Given the following grammar construct the LR(0) sets of items.

1. \( S \rightarrow XYa \)
2. \( X \rightarrow Yb \)
3. \( X \rightarrow b \)
4. \( Y \rightarrow c \)
5. \( Y \rightarrow \epsilon \)
8. Use the grammar and the set of items in the last problem to construct the SLR parsing table.
9. Construct the LR(0) set of items for the ambiguous grammar below:

1. \( E \rightarrow E \text{ sub } E \text{ sup } E \)
2. \( E \rightarrow E \text{ sub } E \)
3. \( E \rightarrow E \text{ sup } E \)
4. \( E \rightarrow \{ E \} \)
5. \( E \rightarrow c \)
10. Use the grammar and the set of items in the last problem to construct the SLR parsing table.

Resolve conflicts using the following rules: \texttt{sup} and \texttt{sub} have equal precedence and are right associative. Production number 1 has precedence over the other productions in case of a conflict.
11. Draw the syntax tree representation for the following expression. Use this syntax tree representation to derive the three-address representation of the given expression. Also provide the quadruple, triple and indirect triple representations of the expression.

\[(a \times c) + b \times (a \times c) + b - (a \times c)\]
12. Short answers:

(a) How is bottom-up parsing different from top-down parsing?
(b) Why is bottom-up parsing (with one symbol of lookahead) more powerful than top-
down parsing (with one lookahead symbol)?
(c) When would you want to use an ambiguous grammar for parsing?
(d) Explain the different error recovery strategies.
(e) What is the general strategy of panic-mode recovery? What are its main advantages?
(f) Why are grammars used for language specification?
(g) Given a grammar, what will you do to get it ready for top-down parsing?
(h) Give the advantages and disadvantages of using a three-address form of intermediate
representation over a zero-address representation.
(i) Why are the benefits of using intermediate code as the interface between the backend
and frontend portions of the compiler?
(j) What is the primary characteristic of SSA-based code? What is the need for $\phi$-functions
in SSA codes?
(k) What are synthesized and inherited attributes? Explain with example.
(l) What is an accumulator machine?