1. Consider the following grammar for a language of simple straight-line programs:

Non-Terminals = \{S, E, L\}
Terminals = \{id, num, (, ), +, :=, semicolon, comma\}
Keywords = \{print\}

Productions:

\[ S ::= S ; S | id ::= E | print(L) \]
\[ E ::= id | num | E + E | (S, E) \]
\[ L ::= E | L , E \]

This grammar is unsuitable for top-down parsing: it is ambiguous and has left-recursion. Transform the grammar to remove the ambiguity and the left-recursion, compute the First and Follow sets for non-terminals, and build an LL(1) table for the language.

2. Consider the following grammar, whose terminals are \{id, :\}:

\[ S ::= G \]
\[ G ::= P | P G \]
\[ P ::= id : R \]
\[ R ::= \epsilon | id R \]

Left-factor the grammar, and compute the First and Follow sets for the resulting grammar. Verify that the grammar is not LL(1) by indicating the conflicts that arise when trying to build the parse table. Show that the language is LL(2), by showing convincingly that the conflicts can be resolved by looking ahead one more token.

Consider the following list of grammars:

(a) \[ S ::= 0 S 1 | 0 1 \]
(b) \[ S ::= + S S | * S S | a \]
(c) \( S ::= S \ ( S ) \ S \mid \epsilon \)
(d) \( S ::= S + S \mid SS \mid ( S ) \mid S \ast \mid a \)
(e) \( S ::= ( L ) \mid a \)
\( L ::= L, S \mid S \)
(f) \( S ::= a S b S \mid b S a S \mid \epsilon \)

3. For grammars (a)–(f) and the respective strings “000111”, “+*aaa”, “((()))”, “((a,a),a,(a))”, “aabbab”, answer the following:

A. Give a leftmost derivation for the string.
B. Is the grammar ambiguous or unambiguous? Justify your answer.
C. Describe the language generated by this grammar.

4. For each of the grammars (a)–(f), answer the following:

A. Is the grammar suitable for top-down parsing?
B. For the cases you answered negatively to A, can you transform the grammar to an LL(1) grammar for the same language?

5. For each of the grammars (a)–(f), construct an LR(0) classifier, indicate the action conflicts if there exist any, and construct an SLR parsing table, if one exists.