Problem 1 – Ambiguity

You want to include an arithmetic switch statement in a programming language. The semantics of the new construct is as follows: The expression in the switch statement is evaluated to an integer value “x”, and the “x-th” statement in the statement list ⟨stmt_list⟩ is executed, if such a statement exists. If no such statement exists, and a default case is specified, the default case is executed. Here is a proposed partial grammar for the switch statement extension.

1. ⟨stmt⟩ ::= case ⟨arith_expr⟩ :: ⟨stmt_list⟩ |
2. case ⟨arith_expr⟩ :: ⟨stmt_list⟩ default ⟨stmt⟩
3. ⟨stmt_list⟩ ::= ⟨stmt⟩ ; ⟨stmt_list⟩ |
4. ⟨stmt⟩

1. Is the grammar ambiguous? Make a formal argument.

2. If the grammar is ambiguous, how can you change the language such that its grammar is unambiguous. Show the grammar.

Problem 2 – Parse trees, derivations, and left-recursion

Consider the following grammar with start symbol A:

1. A ::= ( B ) |
2. a |
3. B ::= B , A |
4. A |

1. Give parse trees for the sentences (a, a) and ((a, a), a).

2. Construct a leftmost and a rightmost derivation for the sentence ((a, a), a).

3. Eliminate the left recursion from the grammar and show the resulting grammar.
Problem 3 - LL(1) and Recursive Descent Parsing

Assume the following grammar of a simple, prefix expression language.

\[
\begin{align*}
\text{Program} &::= \text{Stmtlist} . \\
\text{Stmtlist} &::= \text{Stmt} \ \text{NextStmt} \\
\text{NextStmt} &::= ; \ \text{Stmtlist} \mid \epsilon \\
\text{Stmt} &::= \text{Assign} \mid \text{Print} \\
\text{Assign} &::= \text{ID} \ = \ \text{Expr} \\
\text{Print} &::= ! \ \text{ID} \\
\text{Expr} &::= + \ \text{Expr} \ \text{Expr} \mid \\
& \quad - \ \text{Expr} \ \text{Expr} \mid \\
& \quad \ast \ \text{Expr} \ \text{Expr} \mid \\
& \quad \text{ID} \mid \\
& \quad \text{ICONST} \\
\text{ID} &::= \text{a} \mid \text{b} \mid \text{c} \\
\text{ICONST} &::= \text{1} \mid \text{2} \mid \text{3}
\end{align*}
\]

The goal of this homework problem is to build an LL(1) parser for this simple language. Programs can be written in a single line (no new line characters), and there are no blanks. This means that all tokens are exactly one character long, and that you don’t have to worry about line breaks.

The homework problem consists of the following subproblems:

1. Compute the \textit{FIRST} and \textit{FOLLOW} sets for the grammar.

2. Compute the LL(1) parse table for the resulting grammar. Is the grammar LL(1) or not? Justify your answer.

3. If the resulting grammar is LL(1), show the behavior of the LL(1) skeleton parser as a sequence of states [stack content, remaining input, next action to be taken] on sentence c=3; !c. .