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 $\langle \text{stmt}_\text{list} \rangle$ 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. 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. 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.

```
Program ::= Stmtlist .
Stmtlist ::= Stmt NextStmt
NextStmt ::= ; Stmtlist | epsilon
Stmt ::= Assign | Print
Assign ::= ID = Expr
Print ::= ! ID
Expr ::= + Expr Expr |
      - Expr Expr |
      * Expr Expr |
      ID |
      ICONST
```

The goal of this homework problem is to build an ILOC compiler for this simple language. Here are a few more restrictions and hints:

1. all identifiers consist of single letter, lower or upper case letters, i.e., [a..z] and [A..Z].

2. all integer constants are single digits, i.e., [0..9].

3. 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.

4. use call to `next_register()` to get a fresh, virtual register

5. use calls to `offset(ID)` to get the local offset of variable ID from the base address 1024; register r0 should contain this base address

6. use calls to `value(ICONST)` to get the integer value of the constant

7. hint: recursive procedures return virtual register in which computation result will be stored at runtime, or “null”

The homework problem consists of the following subproblems:

1. Compute the `FIRST` and `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 a=5;!a.. .

4. If the resulting grammar is LL(1), write a recursive descent parser for it in pseudo code (or real code, if you want).