Problem 1

Ambiguous grammar:

```
L ::= R a | Q ba
R ::= aba | caba | R bc
Q ::= bbc | bc
```

Updated grammar:

```
A ::= bc | c
R' ::= bc R'

L ::= R a | Q ba
R ::= aba R' | caba R'
Q ::= bC
```

For the grammar to have the $\mathrm{LL}(1)$ property, for any nonterminal A with production rules $A \to \beta_1|\beta_2|\ldots|\beta_n$: $\mathrm{START}(A \to \beta_i) \cap \mathrm{START}(A \to \beta_j) = \emptyset, \forall\, 1 \le i, j \le n, i \ne j.$

For this grammar, we see that there is no nonterminal A that violates that rule. This is shown by the fact that there are no two production rules for the same nonterminal that begin with the same terminal/nonterminal. In the original grammar, nonterminal Q had this problem.

We also needed to eliminate the left-recursion in third production rule of nonterminal R in the original grammar. We did this by translating it to a left-recursion by introducing the nonterminal R' in the new grammar.

Problem 2

```
A ::= B a
B ::= dab | C b
C ::= c B | A c
```

No the grammar does not have the LL(1) property. There are no two productions rules for any nonterminal such that they begin with the same terminal/nonterminal. However, the grammar has a left-recursion `C ::= A c -> B a c-> C a b c` which can lead to an infinite recursion. The following grammar removes the left-recursion and therefore, has the LL(1) property:

```
A ::= B a
B ::= dab | b C'
C ::= c B | A c
```

```
C' ::= C | ϵ
```

Problem 3

Let C, U, D be the non-terminals for the following grammar:

```
C ::= UD
C ::= ϵ
U ::= ↑
U ::= ϵ
D ::= ↓D
D ::= ϵ
```

For the grammar to have the $\mathrm{LL}(1)$ property, for any nonterminal A with production rules $A \to \beta_1|\beta_2|\ldots|\beta_n$: $\mathrm{START}(A \to \beta_i) \cap \mathrm{START}(A \to \beta_j) = \emptyset, \forall\, 1 \leq i, j \leq n, i \neq j$.

This grammar satisfies that property because there is no nonterminal with two production rules that start with the same terminal/nonterminal.

Problem 4

Original Grammar:

```
S ::= S ; S
S ::= id | E
S ::= print( L )
E ::= id
E ::= num
E ::= E + E
E ::= ( S, E )
L ::= E
L ::= L , E
```

Updated Grammar:

```
S ::= id S' | E S'
S' ::= ; S | ϵ

E ::= id E' | num E' | ( S, E ) E'
E' ::= + E | ϵ

L ::= E L'
L' ::= , E | ϵ
```