**(a) Why is this grammar not LL(1)?**

An **LL(1) grammar** must not have:

* **Left recursion**
* **Ambiguity**
* **First/First or First/Follow conflicts**

**Left Recursion in ParamList:**  
  
ParamList = ParamList "," Param | Param .

This is left-recursive, which is not allowed in LL(1) grammars.  
  
  
**Ambiguity in Statement:**  
Statement = ident "=" Expr | ident | .

The issue is that all alternatives start with ident or empty, causing First/First and First/Follow conflicts:

If you see ident, you don't know whether it's:

* An assignment (ident = Expr)
* A standalone expression (ident)

Additionally, the empty alternative introduces a First/Follow conflict with the following, or }.  
  
First(ident "=" Expr) = {ident}

First(ident) = {ident}  
  
First(α₁) ∩ First(α₂) ≠ ∅ → not LL**(1)**

**(b) Rewriting the grammar to make it LL(1)**

**- Remove Left Recursion in ParamList**ParamList = ParamList “,” Param | Param .

Use a **right-recursive** structure with a helper nonterminal:

ParamList = Param { "," Param } .

### - Refactor Statement to remove ambiguity

Use **lookahead** to distinguish between alternatives:

Statement = ident StatementRest .

StatementRest = "=" Expr | ε .

This way, after seeing an ident, we look ahead:

* If it's =, parse assignment
* If it's something else (like , or }), it's a standalone ident

**LL(1)-Compatible Grammar**

FunctionDecl = "func" ident "(" [ ParamList ] ")" ":" Type Body .

ParamList = Param { "," Param } .

Param = ident ":" Type .

Type = "int" | "float" | "bool" .

Body = "{" [ StatementList ] "}" .

StatementList = Statement { “,” Statement } .

Statement = ident StatementRest .  
StatementRest = "=" Expr | ε . // ε represents matching the empty string

**No left recursion**

* **No ambiguous prefixes**
* **LL(1)-compliant structure**: each parsing decision can be made with a single token lookahead.