Transformed grammar into LL(1):

1) **Left Recursion:**
   ```
   arithExpr    -> arithExpr addOp term | term
   term         -> term multOp factor | factor
   ```
2) **Ambiguities & Non-Deterministic:**
   a) At *classDecl* line, both *varDecl* and *funcDecl* start with *type*
   b) At *funcHead* line, an *id* after *type* is non-deterministic
   c) At *funcBody* line, both *varDecl* and *statement* could start with *id*
   d) At *expr* line, a following *airthExpr* is non-deterministic because *relExpr* also start with *airthExpr*
   e) At *factor* line, both *variable* and *functionCall* start with {*idnest*}
   f) At *idnest*, a following *id* is non-deterministic
3) **Note:** all g0 ~ g5 file show step by step how the raw grammar is translated to a LL(1)

FIRST and FOLLOW sets:

Since the program uses table driven method, so all pop and scan is decided within the table entries, no additional first and follow sets needed.

Design (classes):

1) NonTerminalSet: store all non-terminal symbol, and its corresponding table index
2) TerminalSet: store all terminal symbol, and its corresponding table index
3) Parssing table: store the parsing table for further match checking usage
4) ProductionRule: store all production rule for generating derivation and updating stack
5) SyntacticAnalyzer:
   a. It uses LexicalAnalyzer to get token one by one, go through the table to find match. If there is a match, update stack and derivation, deal with error case otherwise.
   b. If an error occurs, either pop or scan based on the table corresponding entries.
   c. Error recovery, same as the lecture slides, after serial pop or scan, find a match, continue parse.
   d. The derivation and error are generated in separate file folders.

Use of tools:

1) AtoCC kfG Edit. Make sure the grammar translation is correct.
2) LL(1) Parser Generator. First, Follow, & Predict Sets. Table.
   http://hackingoff.com/compilers/ll-1-parser-generator