1. Left recursion and ambiguities:

   classDecl -> 'class' 'id' [':' 'id' {',' 'id'}] '{' {varDecl} {funcDecl} '}' ';' ***ambiguity***

   expr -> arithExpr | relExpr Left factoring ***ambiguity***
   relExpr -> arithExpr relOp arithExpr
   arithExpr -> arithExpr addOp term | term ***left recursion***
   term -> term multOp factor | factor ***left recursion***

   variable -> {idnest} 'id' {indice} ***ambiguity***
   functionCall -> {idnest} 'id' '(' aParams ')' ***ambiguity***
   idnest -> 'id' {indice} '.' | 'id' '(' aParams ')' '.' ***ambiguity***

   funcDecl -> type 'id' '(' fParams ')' ';' ***ambiguity***
   varDecl -> type 'id' {arraySize} ';'
   statement -> assignStat ';'
   assignStat -> variable assignOp expr
   variable -> {idnest} 'id' {indice}
   type -> 'integer' | 'float' | 'id'

2. Grammar after transformation:
   See “final_grammar.txt”

3. First and Follow sets
   See “First and Follow Sets.txt”

4. I implemented Recursive Descent Predictive Parsers. For each non-terminal, we have a corresponding function, and in each function, for each possible right-hand-side of the corresponding productions, we have a possible path to follow. Also, we have a tree data structure to store the result of the parser.

5. Used AtoCC’s kfGEditor for grammar analysis and grammar correction checking.