```
#ifndef SYNTAXANALYZER_H
#define SYNTAXANALYZER_H

#include <fstream>
#include "Lexer.h"
#include "SymbolTable.h"

class SyntaxError
{
public:
  // Constructor
  SyntaxError(std::string message, int lineNumber);

  ~SyntaxError();

  std::string getMessage() const;

private:
  std::string message;
  int lineNumber;
};

class SyntaxAnalyzer
{
public:
  // Constructor
  SyntaxAnalyzer(const std::vector<Lexer::Token> &tokens, std::ofstream &output, bool print = false);
  ~SyntaxAnalyzer();

  // Begins the analysis process with the given tokens
  void Analyze();

  std::string PrintAll();

private:
  enum ErrorType
  {
    TYPE_MISMATCH,
    DUPLICATE_SYMBOL,
        UNDECLARED_VARIABLE
  };

  void Rat18F();
  void OptFunctionDefinitions();
  void FunctionDefinitions();
  void Function();
  void OptParameterList();
  void ParameterList();
  void Parameter();
  void Qualifier();
  void Body();
  void OptDeclarationList();
  void DeclarationList();
  void Declaration();
  void IDs();
  void StatementList();
  void Statement();
  void Compound();
  void Assign();
  void If();
  void Return();
  void Print();
  void Scan();
  void While();
  void Condition();
  void Relop();
  void Expression();
  void Term();
  void Factor();
  void Primary();
  void Empty();
  void ExpressionPrime();
  void TermPrime();
  void Identifier();
  void Integer();
  void Real();

  void error(ErrorType errorType, int lineNumber, std::string expected = "");

  void getNextToken();
  void printCurrentToken();

  const std::vector<Lexer::Token> &tokens;
  std::vector<Lexer::Token>::const_iterator it;
  Lexer::Token currentToken;
  bool print;
  std::ofstream &output;
  SymbolTable symbolTable;
  std::string *savedOp;
  std::string *savedType;
  Lexer::Token *save;
  std::ostringstream err;
  int errCount;
  bool isDeclaration;
  bool assign;
};
```

```

#endif // SYNTAXANALYZER_H
```