# Report

**Name:** Adil Aman Mohammed
**Course:** Formal Language Theory
**Assignment Number:** 4
**CWID:** A20395630

**Description:** The provided code focuses on applying four key algorithms to a given CFG. These algorithms are designed to modify and optimize CFGs in various ways. The four algorithms are:

1. Epsilon Removal
2. Removal of Immediate Left Recursion
3. Removal of General Left Recursion
4. Removal of Unit Productions

**Key Functions and Their Descriptions:**

1. **readGrammarSymbols(filename)**:
   - This function reads grammar symbols from a specified file.
   - It initializes and populates sets for terminal and non-terminal symbols and identifies the start symbol.
   - This serves as the foundation for applying further CFG algorithms.
2. **removeEpsilon(cfg)**:
   - Purpose: To eliminate epsilon (empty string) productions from the CFG.
   - Method: The function identifies and removes productions that directly produce an epsilon, adjusting other productions accordingly to maintain the grammar's language.
3. **removeImmediateLeftRecursion(cfg, non_terminal)**:
   - This function targets immediate left recursion in CFG productions associated with a specific non-terminal.
   - Left recursion is a scenario where a production can led to an infinite recursive call, and removing it is crucial for certain types of parsing algorithms.
4. **removeLeftRecursion(cfg)**:
   - This is an extension of the previous function and is designed to handle general left recursion in the CFG.
   - It systematically processes each non-terminal and applies the removal of immediate left recursion, ensuring the CFG is free from any form of left recursion.
5. **removeUnitProductions(cfg)**:
   - Unit productions are those where a non-terminal directly produces another non-terminal.
   - This function identifies and eliminates such productions, replacing them with more direct production rules, thus simplifying the grammar.

**Error Handling and Validation:** The code includes mechanisms for handling file input errors and validates the structure of the CFG to ensure the correctness of the transformations applied.

**Conclusion:** This assignment is crucial in demonstrating the application of formal language theory concepts in practical scenarios. By implementing these algorithms, the program effectively optimizes CFGs, which is an essential step in the design of compilers and interpreters. The code showcases the student's ability to apply theoretical concepts to solve practical problems in computer science.