**PROJECT TITLE:** Mini compiler for a Custom Programming Language

**PROJECT OVERIVEW**

For the final exam, students must develop a mini compiler for a simple, custom programming language. This project will test their understanding of lexical analysis, syntax analysis, semantic analysis, intermediate code generation, and basic optimization techniques.

**PROJECT OBJECTIVES**

1. Implement a lexical analyzer (scanner) using regular expressions or finite automata.
2. Develop a syntax analyzer (parser) using context-free grammar rules and a parsing technique (LL(1), LR(1), etc.)
3. Perform semantic analysis to validate variable declaration, type checking, and scope management.
4. Generate intermediate representation (IR) such as three-address code (TAC) or abstract syntax trees (AST).
5. Optimize the intermediate code using simple optimization techniques (constant folding, dead code elimination, etc.)
6. Convert the intermediate representation into a low-level target code (assembly or bytecode for a virtual machine).
7. Provide a simple error-handling mechanism to detect and report syntax and semantic errors.

**PROJECT SPECIFICATIONS**

1. Lexical Analyzer
   1. Identify tokens such as keywords, identifies, literals, operators, and special symbols.
   2. Implement tokenization using regular expressions or a finite state machine.
   3. Output a token stream for the parser.
2. Syntax Analyzer (parser)
   1. Implement a parsing technique as LL(1) or LR(1).
   2. Use context-free grammar to validate program structure.
   3. Generate a parse tree or abstract syntax tree (AST).
3. Semantic Analyzer
   1. Perform type checking and variable scope validation
   2. Detect and report semantic errors such as undeclared variables or type mismatches
4. Intermediate Code Generation
   1. Generate an intermediate representation (IR) such as three-address code (TAC) or an AST.
   2. Handle arithmetic expressions, conditional statements, and loops.
5. Code Optimization
   1. Apply basic optimization techniques using constant folding
6. Code Generation
   1. Translate the IR into low-level assembly code for a hypothetical machine
7. Error Handling
   1. Implement meaningful error message for lexical, syntax, and semantic errors.
   2. Provide a debugging mechanism to help students identify issues.