Compiler Design

---

1. Remembering:
- Identify different types of translators like Assembler, Cross Assembler, Pre-Processor, Interpreter, Simulator, Loader.
- Recall the basic concepts of translators such as bootstrapping and cross compilation.

2. Understanding:
- Differentiate between various types of grammars and their recognizers.
- Explain the different phases of compilation and their significance in the translation process.

3. Applying:
- Design a lexical analyzer using Finite State Machines (FSM) for a given set of tokens and schemas.
- Implement syntax analysis techniques like Top-down and Bottom-up parsers for a given grammar.

4. Analyzing:
- Compare and contrast different parsing techniques such as LL(1), LR(1), and LALR(1) for efficiency and effectiveness in syntax analysis.
- Evaluate the representation of intermediate code generation using Quadruples, Triples, and Indirect Triples.

5. Evaluating:
- Critique the techniques and strategies used in code optimization like basic blocks, loop optimization, and DAG representations for improving program efficiency.
- Assess the challenges and solutions in code generation for producing optimal object programs.

6. Creating:
- Develop syntax-directed translation using attributes, semantic actions, and translation schemes to map source code to target code effectively.
- Design effective error handling mechanisms for detecting, reporting, recovering, and repairing syntax errors in the compilation process.

7. Applying:
- Implement symbol tables using various organization and management techniques for efficient storage and retrieval of symbols.
- Apply different techniques for runtime storage management such as static allocation, dynamic allocation, activation records, and recursive procedures in memory management.