**Basic C Compiler - Lexical to Code Generation**

### **Compilation Stages:**

### **1. Lexical Analysis**

* **Description**: The compiler's lexer scans the source code, breaking it down into tokens (keywords, operators, literals, and identifiers).
* **Outputs**: Tokens are generated to serve as the basis for subsequent parsing and analysis.
* **Error Handling**: Lexical errors are flagged (e.g., invalid tokens, malformed identifiers).

**2. Syntax Analysis (Parsing)**

* **Description**: The parser verifies the token sequence against C grammar rules to ensure syntactic correctness.
* **Outputs**: Constructs an Abstract Syntax Tree (AST) that represents the code's hierarchical structure.
* **Error Handling**: Reports syntax errors (e.g., missing semicolons, unmatched parentheses) with descriptive messages.

**3. Semantic Analysis**

* **Description**: This stage validates variable declarations, type compatibility, and function definitions.
* **Implementation**: A symbol table is created to manage variable scopes, function declarations, and types.
* **Error Handling**: Identifies semantic errors (e.g., undeclared variables, type mismatches).

**4. Intermediate Code Generation**

* **Description**: Transforms the AST into intermediate code, such as three-address code.
* **Purpose**: Serves as a bridge to machine code generation and allows for optimization.
* **Supported Operations**: Arithmetic expressions, assignments, and basic control flow.

**5. Basic Optimizations**

* **Description**: Improves intermediate code through optimizations like constant folding and dead code elimination.
* **Benefits**: Increases efficiency and readability of generated code.

**6. Code Generation**

* **Description**: Converts the optimized intermediate code into assembly or machine-level instructions.
* **Output**: Produces an executable format that enables end-to-end compilation from C code to machine code.