1. Lexical Analysis (Tokenization) The lexical analyzer scans the input source code and breaks it into tokens. This is handled by the `next()` function, which: - Reads characters one by one. - Identifies keywords, operators, and identifiers. - Converts numeric values into integer tokens. - Ignores unnecessary whitespace and comments. For instance, it distinguishes `if`, `while`, and `return` as special reserved words while treating numbers like `456` as integer tokens.

2. Parsing Process (Building a Syntax Tree) The parser organizes tokens into a structured representation following grammatical rules. - The `expr()` function manages arithmetic expressions like `a + b * c`. - The `stmt()` function interprets statements, including `if`, `while`, and `return`. - Functions are dynamically recognized and processed. It follows operator precedence, ensuring operations occur in the correct order, for example: `x = 10 - 2 * 3` evaluates `2 * 3` first before subtracting from `10`.

3. Virtual Machine (VM) Execution The C4 compiler utilizes a Virtual Machine (VM) to interpret and execute the compiled code. - It follows an instruction-based execution model, processing operations like `IMM`, `JMP`, `LEA`, and `ADD`. - It relies on a stack to manage variables and function calls. - System calls, such as `printf`, are handled via predefined VM instructions. For example, an instruction like `IMM 8` pushes `8` onto the stack, while `ADD` retrieves two values from the stack, adds them, and stores the result.

4. Memory Management in C4 organizes memory into three major sections: 1. Stack - Stores temporary values, local variables, and function call information. 2. Heap - Used for dynamic memory allocation via `malloc()` and freed with `free()`. 3. Global Data- Holds globally defined variables.