Explain Key Algorithms:

Lexical Analysis: The lexical analysis phase in C4 is handled by the next function, which reads the source code character by character and converts it into meaningful tokens. These tokens include keywords like if, while, return, operators such as plus, minus, multiplication, and division, numeric literals, and identifiers. The function skips whitespace and comments while also recognizing string and character literals. A key feature of this phase is its ability to distinguish between identifiers and reserved keywords by checking against entries in the symbol table. Additionally, escape sequences inside string literals are processed to ensure they are stored correctly in memory.

Parsing Process: C4 uses a recursive descent parser to process statements and expressions. The expr function constructs expressions using precedence climbing, ensuring operators are evaluated correctly based on their priority. Statements like if, while, and return are handled in the stmt function, which directs the flow of parsing based on the encountered tokens. Unlike traditional compilers that build an abstract syntax tree, C4 directly generates machine-like bytecode. This design simplifies execution but makes optimization more challenging. Variable declarations and function definitions are also processed at this stage, ensuring that identifiers are correctly classified as global, local, or system functions.

Virtual Machine Implementation: C4 includes a simple yet effective virtual machine that executes compiled instructions. The instruction set consists of low-level operations such as loading immediate values, jumping, pushing values onto the stack, and performing basic arithmetic operations. It also includes system calls for handling input, output, memory allocation, and file operations. The virtual machine operates in a loop, continuously fetching and executing instructions from memory. It uses a stack-based execution model, where function calls and local variables are managed within the stack. When a function is executed, space is allocated on the stack for its local variables, and upon returning, the stack is restored to its previous state.

Memory Management: Memory allocation in C4 is divided into multiple sections, including the symbol table, emitted code, global data storage, and the stack. The malloc function is used to allocate memory dynamically for these regions. The stack is primarily used for managing function calls and local variables, growing dynamically as functions are called and shrinking upon return. Global variables are stored in a separate data segment, while dynamically allocated memory, such as strings, is placed in the heap. The program ensures proper cleanup using free where applicable, but since C4 is a minimal compiler, it does not implement advanced garbage collection mechanisms.