Key Algorithms in C4 Compiler

1. Lexical Analysis

The lexical analysis process in C4 scans the source code character by character and converts it into tokens. This is handled by the next() function. It identifies different types of tokens such as keywords (e.g., if, while), identifiers, numeric literals, operators, and delimiters.

- It uses hash-based lookup for identifiers to improve efficiency.
- Numeric literals are parsed in decimal, hexadecimal, or octal formats.
- It ignores comments (// style) and preprocessor directives (# lines).
- Special characters and multi-character operators (e.g., <=, !=) are identified based on predefined precedence.

2. Parsing Process

The parser in C4 constructs an implicit abstract syntax tree (AST) using recursive descent parsing. The main function responsible for parsing expressions is expr(int lev), which follows precedence climbing to ensure correct operator binding.

- Function declarations and definitions are processed by checking identifiers followed by (.
- Control flow statements like if, while, and return are handled in stmt().
- Expressions are parsed recursively, handling precedence of arithmetic, logical, and assignment operations.
- Variable declarations and function definitions are stored in a simple symbol table.

3. Virtual Machine Implementation

C4 includes a stack-based virtual machine (VM) that executes compiled instructions. The VM executes an instruction cycle inside the while (1) loop in main().

- Instructions are represented as opcodes (e.g., LEA, IMM, JMP, ADD).
- It maintains a stack pointer (sp), base pointer (bp), and program counter (pc).
- Each instruction is fetched from memory and executed sequentially.
- Arithmetic and logical operations manipulate values on the stack.

- Function calls use JSR (Jump to Subroutine) and LEV (Return from Function) instructions.
- System calls such as file operations (open, read), memory allocation (malloc, free), and printing (printf) are handled within the VM.

4. Memory Management

C4 manages memory using global pointers to different memory sections:

- **Code segment (e)** stores compiled instructions.
- **Data segment (data)** stores global variables and string literals.
- **Stack (sp)** is used for local variables, function calls, and temporary expressions.
- **Heap memory (malloc)** is allocated dynamically when needed.

Memory is allocated dynamically at startup using malloc(). The stack grows downwards, while the heap grows upwards. The VM supports explicit deallocation of memory using free(). Garbage collection is not implemented, so manual memory management is required.

Conclusion

C4 is a minimal yet fully functional C interpreter with a compact lexical analyzer, efficient parsing strategy, a simple stack-based virtual machine, and a straightforward memory management model. Despite its simplicity, it supports function calls, control flow, and dynamic memory allocation.