**1. Lexical Analysis Process**

The lexical analysis process is responsible for converting the raw source code (a sequence of characters) into tokens that the compiler can understand. In C4, this process is handled by the next() function.

- **Tokenization**: The function scans the input character-by-character. Each character is processed, and based on its type (e.g., letter, digit, operator), it is classified into a specific token.
- **Keywords and Identifiers**: The function identifies keywords (such as int, char, return) by comparing the characters against predefined tokens. Identifiers (like variable names) are captured by checking for letters or underscores and then using a simple hashing technique to store and retrieve their values from the symbol table.
- **Operators and Punctuation**: Operators (e.g., +, -, *, =) and punctuation marks (e.g., ;, {, }) are also tokenized. If a character is identified as an operator or punctuation, it is immediately assigned a corresponding token.

The function processes tokens iteratively and assigns them to the tk variable, which stores the current token type. The next() function handles various cases like number literals, operators, and keywords.

**2. Parsing Process**

Parsing is the process of analyzing the tokens generated by the lexical analyzer and constructing a representation of the program's syntax. In C4, the parsing process is a combination of recursive descent parsing and manual token handling.

- **Recursive Descent Parsing**: Functions like expr() handle the parsing of expressions. The expr() function uses **precedence climbing** (a technique where the operator precedence guides the parsing process). It checks the type of each token and recursively calls itself to parse operands and operators with increasing precedence.
- **Abstract Syntax Tree (AST)**: C4 does not build a full AST, however, it builds an intermediate representation of the program in the form of emitted opcodes (assembly-like instructions). These opcodes are later executed by the virtual machine.
- **Syntax Analysis**: The stmt() function handles statements like if, while, and assignment. It uses token types to decide how to parse and execute different kinds of statements.

### 3. Virtual Machine Implementation

The virtual machine (VM) in C4 is responsible for executing the compiled code. It processes a sequence of opcodes that correspond to operations like arithmetic, memory access, and control flow.

- **Opcode Execution**: The VM interprets each instruction in the emitted code. Instructions are stored in the array e, and the pc (program counter) points to the current instruction being executed. The VM reads the opcode and performs the corresponding operation.
- **Memory Management**: The VM uses a stack-based architecture where operands and results are pushed and popped from the stack. It handles various instructions like IMM (immediate value), ADD (addition), SUB (subtraction), JMP (jump), and JSR (jump to subroutine).
- **Control Flow**: The VM handles conditional and unconditional jumps using BZ (branch if zero) and JMP (jump). These instructions help implement loops and conditional statements.

### 4. Memory Management Approach

Memory management in C4 involves the following key components:

- **Memory Allocation**: Memory for the symbol table, emitted code, data area, and stack is dynamically allocated using malloc(). The compiler allocates pools of memory for these areas and uses them throughout the compilation and execution processes.
    - The sym array holds the symbol table, which stores variable and function metadata.
    - The e array holds the emitted code (opcode instructions), and data stores dynamically allocated memory for variables.
    - The sp pointer manages the stack area, and bp is used as the base pointer to manage function frames.
- **Garbage Collection**: C4 does not implement advanced garbage collection. Instead, it relies on manual memory management. Memory is allocated when required (using malloc), and pointers are adjusted throughout the program to ensure that memory is used efficiently.
- **Deallocation**: C4 uses free() to release memory from the heap after it is no longer needed. However, in a self-hosting scenario, memory management primarily occurs during the lifetime of the compiler and VM execution.