Lexical Analysis, Parsing, Virtual Machine, and Memory Management in C4

1. Lexical Analysis (Tokenization)

The lexer (next()) in C4 is responsible for converting raw source code into tokens. It processes characters from the input buffer (p[]), identifying keywords, identifiers, numbers, and symbols. The lexer:

- Skips whitespace and comments.
- Recognizes identifiers and stores them in the symbol table (id[]).
- Converts numeric values into integer representations (ival).
- Produces tokens (tk) for the parser to process.

Example: Tokenizing if (x == 10)

Token: 'if' → Keyword
Token: '(' → Symbol
Token: 'x' → Identifier
Token: '==' → Operator
Token: '10' → Number
Token: ')' → Symbol

This sequence is then passed to the parser.

2. Parsing Process

The parser (expr(), stmt()) converts tokens into a structured representation. Instead of building an explicit Abstract Syntax Tree (AST), C4 follows recursive descent parsing:

- expr(level): Parses arithmetic and logical expressions (e.g., a + b * c).
- stmt(): Parses control structures (if, while, return).
- The parser generates bytecode instructions (e[]), which will be executed by the virtual machine.

Example: Parsing if (x == 10) return x;

1. Matches if, then calls expr() to parse x == 10.
2. Stores the parsed result as bytecode instructions.

3. Virtual Machine (VM) Implementation

C4 uses a stack-based virtual machine to execute compiled instructions. Key components:

- Registers: pc (program counter), sp (stack pointer), bp (base pointer).
- Execution loop (while(1)) fetches instructions from e[] and executes them.
- Supports arithmetic, memory operations, and control flow.

Example: Execution of x = 10 + 5;

```
PUSH 10    // Push 10 onto the stack
PUSH 5     // Push 5 onto the stack
ADD        // Pop two values, add them, push result
STORE x    // Store result in variable x
```

The virtual machine manages execution by manipulating the stack (sp) and memory (data[]).

4. Memory Management

C4 follows a simple memory model with global allocation:

- Symbol Table (sym[]) stores identifiers.
- Text Segment (e[]) holds compiled instructions.
- Data Segment (data[]) stores global variables.
- Stack (sp[]) is used for function calls and local variables.

Dynamic Memory Operations:

- malloc() is used for heap allocation.
- free() is available but limited in scope.

Example:

```
char *str = malloc(20);  // Allocate 20 bytes
strcpy(str, "Hello");    // Use allocated memory
free(str);               // Deallocate
```

However, C4 does not implement garbage collection, so memory must be managed manually.