## C4 Compiler: Key Algorithms and Concepts

This report explains the key algorithms and concepts used in the **C4 compiler**, a self-interpreting C compiler. The report covers the following topics:

1. **Lexical Analysis**: How input is tokenized.
2. **Parsing Process**: How syntax is analyzed and an AST is built.
3. **Virtual Machine**: How compiled instructions are executed.
4. **Memory Management**: How memory is allocated and freed.

### 1. Lexical Analysis – How Input is Tokenized

The **lexical analysis** process in C4 is implemented in the next() function. It reads the source code character by character and converts it into **tokens**, which are the basic building blocks of the compiler.

**Key Steps:**

1. **Character Reading**:
   - The next() function reads characters from the source code using the global pointer p.
   - It skips whitespace and comments (both single-line and multi-line).
2. **Token Generation**:
   - Identifiers and keywords are recognized using a simple hash function.
   - Numbers (integers and hex literals) are parsed and stored in ival.
   - Strings and character literals are handled and stored in the data section.
   - Operators and punctuation (e.g., +, -, ;) are directly mapped to tokens.
3. **Token Classification**:
   - Tokens are classified into categories such as Num, Id, Char, If, While, etc.
   - The tk variable holds the current token type.

**Why It Matters:**

Lexical analysis is the first step in the compilation process. It simplifies the source code into a stream of tokens, making it easier for the parser to analyze the syntax.

### 2. Parsing Process – How Syntax is Analyzed and an AST is Built

The **parsing process** in C4 is implemented in the expr() and stmt() functions. It converts tokens into an **implicit Abstract Syntax Tree (AST)**.

**Key Steps:**

1. **Expression Parsing**:
   - The expr() function handles expressions (e.g., arithmetic, logical, and relational operations).
   - It uses a **recursive descent parsing** approach to evaluate expressions based on operator precedence.
2. **Statement Parsing**:
   - The stmt() function handles statements (e.g., if, while, return).
   - It recursively calls expr() to evaluate conditions and expressions within statements.
3. **Implicit AST**:
   - Instead of explicitly building an AST, C4 directly emits **virtual machine instructions** during parsing.
   - This approach simplifies the compiler but makes it less modular.

**Why It Matters:**

Parsing ensures that the source code follows the syntax rules of the C language. The implicit AST allows the compiler to generate executable instructions directly

**3. Virtual Machine – How Compiled Instructions are Executed**

The **virtual machine (VM)** in C4 is implemented in the main() function. It executes the compiled instructions generated by the code generator.

**Key Steps:**

1. **Instruction Fetching**:
   - The VM fetches instructions from the e array using the program counter (pc).
2. **Instruction Execution**:
   - The VM supports a variety of instructions, such as:
     - IMM: Load immediate value.
     - JMP: Jump to a specific address.
     - PSH: Push a value onto the stack.
     - ADD, SUB, MUL, DIV: Arithmetic operations.
     - LEV: Return from a function.
3. **Stack Management**:

    - The VM uses a **stack** to manage function calls, local variables, and intermediate results.
    - The stack pointer (sp) and base pointer (bp) are used to track the stack state.

**Why It Matters:**

The VM executes the compiled instructions, simulating the behavior of a real CPU. It is the final step in the compilation process.

**4. Memory Management – How Memory is Allocated and Freed**

C4 uses a simple **memory management** approach, relying on global arrays for data and stack memory.

**Key Steps:**

1. **Global Memory**:
    - The data array is used to store constants, strings, and global variables.
    - Memory is allocated sequentially as the compiler processes the source code.
2. **Stack Memory**:
    - The sp (stack pointer) and bp (base pointer) manage the stack.
    - Function calls and local variables are stored on the stack.
3. **Heap Memory**:
    - C4 does not explicitly support heap memory allocation (e.g., malloc and free).
    - However, the VM provides instructions like MALC and FREE for basic memory management.

**Why It Matters:**

Memory management ensures that the compiler and VM can store and retrieve data efficiently. The stack-based approach simplifies memory handling but limits flexibility.

**Conclusion**

The C4 compiler is a compact and efficient implementation of a self-interpreting C compiler. Its key algorithms—lexical analysis, parsing, virtual machine execution, and memory management—work together to compile and execute C programs. While C4 is limited in features compared to modern compilers, its simplicity makes it an excellent tool for learning about compiler design and implementation.