جامعة خليفة
Khalifa University

COSC320 -Assignment 1

Deep Dive into C4 - Understanding a Self-Interpreting C Compiler

C4 Report

Joud Herzallah

100061269

Due Date: March, 03, 2025

The lexical analysis process: How does the code identify and tokenize input?

C4's lexer is mostly implemented in the next() method. It reads characters from the source code (via a global pointer p) and decides the next token based on patterns such as identifiers, numeric literals, string/character constants, operators, and punctuation.

Whitespace and comments (including the // and # directives) are skipped. When a new identifier is detected, the lexer generates a hash and examines the symbol table to determine if it has previously been saved. Numeric literals can be parsed as decimal, hexadecimal, or octal. Once the token has been discovered, tk is set to its type, like (Num, Id, If), and additional fields such as ival are filled with numbers or string addresses.

The parsing process: How does the code construct an abstract syntax tree (AST) or equivalent representation?

C4's expr() and stmt() functions employ a "precedence climbing" or recursive-descent style, in contrast to many other compilers that construct a formal ast. These routines directly output bytecode instructions into an array e after parsing statements and expressions, respectively. The parser uses next() to read tokens and then creates instructions (such as IMM, ADD, and JMP) that match the parsed constructs depending on the precedence of operators.

By keeping a symbol table (sym), local or global variables and function definitions are managed. The "expr()` method functions as a mini-ast builder internally and outputs code in postfix form right away, even though the parser in C4 doesn't create a separate ast data structure.

The virtual machine implementation: How does the code execute the compiled instructions?

The final portion of main() is where the VM executes. The code sets up registers (pc, sp, and bp) and starts a loop that retrieves and runs each opcode when the parser has filled the instruction array e. Arithmetic (ADD, MUL), jumps (JMP, BZ), function calls (JSR, ENT, LEV), loading/storing integers (LI, SI), and system calls (OPEN, READ, etc.) are among the supported instructions.

In the virtual machine, instructions manipulate the top of a contiguous stack in memory by pushing and popping values. While sp (stack pointer) and bp (base pointer) monitor local frames, pc (program counter) keeps track of which instruction is being performed at any one time.

The memory management approach: How does the code handle memory allocation and deallocation?

c4 manages memory sparingly. At launch, four big memory pools are set aside for the symbol table (sym), code array (e), data segment (data), and stack (sp). These pools are merely used to lay out the compiler's own data structures ex: token storage and symbol table entries. For dynamic allocations at runtime, c4 provides system calls like malloc(), free(), and memset(), which ultimately defer to the host operating system.

c4 is a small self-hosted compiler that relies on preallocated pools and the c standard library's malloc/free for dynamic use. It does not have a comprehensive garbage collector or sophisticated memory scheme.

جامعة خليفة
Khalifa University