C4 Rust Comparison

Differences in Implementation Between c4.c (C) and Rust-Based C4 Compiler

COSC - 320 Principles of Prog. Languages

Nasser Alzaabi 100061330

Zayed AlDhaldhaheri 100061222

Introduction

This project's primary objective is to transform the c4.c compiler into a Rust implementation. The original C implementation is a mini subset of a C compiler which was created to showcase the major constructs of the language and the execution on a virtual machine. In rewriting it, the goal is to achieve compatibility in the behavior with the original version while taking advantage of Rust's great safety features and modern development ecosystem.

Memory Safety and Design Impact

In Rust, the guiding principle of design is the memory safety guarantee without using a garbage collection system. This will affect our design in a few ways:

Variable scoping and Ownership: c4.c's variables and memory were implemented using raw pointer and manual memory allocation. In rust, we had the structure of Hashmap and Vec to serve as variable scope and stack. With Rust's ownership model, we could not introduce dangling pointers, double frees or uncontrolled dangling pointers which was a requirement.

Function calls and Stack frames: In C, function calls involved directly operating on the raw memory copies. In Rust, each call stack is modeled securely with a Vec and ownership of return addresses allows for transparent management of control call stacks. This made debugging far easier and much less complex.

Pattern Matching and Enum Safety: Rust simplifies enums with variants replacing the union-heavy int opcode style in c4.c. This not only makes instructions clearer, but also enhances safety. The match expression in Rust guarantees all instruction types were handled exhaustively at compile time.

Lifetimes and References: Typically, we avoided the need for complex lifetime annotations by utilizing owned data instead of references. This approach increased allocations but simplified the design by ensuring no use-after-free or dangling pointer errors.

Performance Comparison

Both implementations, the Rust and C4 compilers, seem to yield similar results for small scale programs. Here are a couple of points to note:

- **Execution Speed**: The performance of the virtual machine implementation in Rust is lower due to bounds checking on Vec and the shims for pointer arithmetic. Nevertheless, this is insignificant for simplistic C programs.
- **Compile Time:** The fourth version compiler implemented in Rust takes substantially longer to compile compared to the c4.c file with gcc. However, the resulting binary is more dependable and secure.
- **Debugging:** Compiler errors in Rust were highly supportive throughout the process. Rather than silently corrupt memory, as would be the case in c4.c, broken logic was flagged at compile time, enabling catches for condition checks that were added.

Problems and Solutions

1. Tokenization and Syntax Analysis: In Rust, the C-like syntax streaming had to be managed with a particular parsing match expression. The grammar also required precision in handling multi declarations and expressions such as *p = 10 to prevent token overconsumption.
2. Instruction Creation: During instruction emission, C has default config of integers and macros. While C instruction emission was more or less straightforward, with Rust there was a need to maintain instruction index for jumps and do label placement manually too.
3. Support For Functions: All the mapped functions implementing multiple issues required maintaining a name to entry point mapping and preserving the argument tracking. Storing instruction offsets alongside the call stack was made easier, but managing these explicitly with Rust's hashmap posed difficulties.

Conclusion

The effort to change c4.c into Rust focused on finding equilibrium dealing with low-level details alongside high-level pedantic safety. Having a type and memory structure like Rust's put heavy integrity on this project. Implementing direct point and memory alteration demanded a rethink, but the logic guarantees ease of use and learning value the original c4 had.