C4 Rust Compiler Comparison Report

1. Safety Features

Rust's ownership and borrowing system prevents issues like memory leaks, null pointer dereferencing, and use-after-free bugs, which are common in C. This made the compiler more secure and eliminated many runtime errors.

2. Design Differences

Rust's enums, pattern matching, and Option/Result types made our parser and lexer more expressive and readable. The bonus feature (position tracking) was trivial to integrate using structs and safe borrowing.

3. Performance

Initial testing showed performance similar to C4's original C version. Rust's speed combined with better memory management gives us a fast and reliable compiler. For more advanced benchmarking, we could use Criterion.

4. Challenges Faced

- Mapping C-style pointers and mutable memory to Rust's ownership system
- Designing AST and VM structures without garbage collection
- Implementing self-hosting test cases was skipped for MVP, but is planned

5. Bonus Feature: Error Reporting

Each token includes line and column numbers, making it easier to report syntax errors precisely. This is not present in the original C4 compiler.

6. Additional Improvements

The Rust version was extended to better support parsing binary operations (+, -, *, /) and simple variable assignments. This made the Rust compiler even more compatible with the C4 subset, going beyond just "return" statements to support basic expressions like return 1 + 2; and x = 5;.

7.Conclusion

Rust improved the overall structure, safety, and maintainability of the compiler. While the rewrite took effort due to borrow checker constraints, the final result is safer and ready for further features like floating point support or full C parsing.