Project: Rewriting the C4 Compiler in Rust

Due Date: May 02, 2025, 23:59
Team Size: 2 students per team
Current Date: March 02, 2025

Objective:

Building on your analysis of the C4 compiler from Assignment 1, your team will rewrite the C4 compiler in Rust. The Rust version must compile the same subset of C code as the original C4 compiler (e.g., the C4 source code itself), maintaining its self-hosting capability and core functionality. This project leverages Rust's safety, modern features, and performance to reimplement the compiler, ensuring equivalence to the original while improving design where possible.

Tasks:

1. **Rewrite the C4 Codebase in Rust:**
   - Translate the C4 compiler (lexer, parser, virtual machine, etc.) from C to Rust.
   - Ensure the Rust version compiles the same C code as the original C4, including its own source code (self-hosting).
   - Use Rust idioms and features (e.g., ownership, pattern matching, error handling with `Result`/`Option`) to enhance safety and clarity, while preserving functional equivalence.
   - Save the main Rust source code as `c4.rs` (or use a multi-file structure with `mod` if preferred).
2. **Collaborate via GitHub:**
   - Create a **public GitHub repository** named `c4_rust_<team_name>` (e.g., `c4_rust_team_alpha`).
   - Both team members must contribute code, with commits reflecting individual work (e.g., "Implemented lexer" or "Fixed VM instruction parsing").
   - Use branches and pull requests for major features or changes to facilitate collaboration and review.
   - Include a `README.md` with instructions to build, run, and test the compiler with sample C code (e.g., the original C4 source).
3. **Implement Unit Testing:**
   - Write unit tests to verify that the Rust compiler correctly processes the same C code as the original (e.g., tokenization, parsing, and execution of C4's supported C subset).
   - Recommended testing frameworks (optional):
     - **Rust's Built-in Test Framework:** Use `#[test]` attributes for lightweight, integrated tests.

    - **Cargo Test with `assert!` Macros:** Add assertions for quick validation of outputs.
    - **Criterion (Optional):** For performance benchmarking against the original C version.
  - Save tests in a separate module (e.g., `tests/` directory or inline with `#[cfg(test)]`).
  - Aim for at least 70% test coverage of critical functionality, including self-hosting.

4. **Document the Code:**
   - Add Rust-style documentation comments (`///`) to all public functions, structs, and modules.
   - Explain design decisions (e.g., how you ensured compatibility with C4's C subset or handled memory differently).
   - Generate documentation using `cargo doc` and include it in your repository.
5. **Analyze and Compare:**
   - Write a short report (1-2 pages) comparing the Rust implementation to the original C version. Address:
     - How Rust's safety features (e.g., memory safety, lifetimes) impacted the design while maintaining compatibility.
     - Performance differences (qualitative or quantitative, if measured) when compiling the same C code.
     - Challenges in replicating C4's behavior and your solutions.
   - Save the report as `c4_rust_comparison.pdf`.
6. **Bonus Task (Optional; +15%):**
   - Add a new feature to the Rust version not present in the original C4 (e.g., support for floating-point numbers or enhanced error reporting), while still supporting the original C subset.
   - Document the feature in the `README.md` and include tests for it.

Deliverables:

1. **GitHub Repository (`c4_rust_<team_name>`):**
   - Rust source code (`c4.rs` or multi-file structure) that compiles the same C code as the original C4.
   - Unit tests verifying compatibility with C4's C subset and self-hosting.
   - `README.md` with build/run instructions and example usage (e.g., compiling `c4.c`).
   - Generated documentation (via `cargo doc`).
   - Visible commit history showing contributions from both team members.
2. **`c4_rust_comparison.pdf`:** Report comparing the Rust and C implementations.
3. **(Optional) Bonus Feature:** Code and documentation for the bonus task.

Submission Instructions:

1. Push all deliverables to your GitHub repository by the due date.
2. Submit the zipped repository on Blackboard in a zip file named `c4_rust_submission_<team_name>.zip`.

Grading Criteria:

- **Functionality (30%):** The Rust compiler correctly compiles the same C code as the original C4, including self-hosting.
- **Collaboration (20%):** GitHub history shows balanced contributions, with effective use of branches/pull requests.
- **Code Quality (20%):** Code is idiomatic Rust, well-documented, and uses Rust features appropriately.
- **Testing (15%):** Tests confirm compatibility with C4's C subset and cover critical functionality.
- **Comparison Report (15%):** Report provides insight into the rewrite process and Rust's impact.
- **Bonus (Optional +15%):** New feature works, is tested, and is well-documented.

Recommendations:

- **Verify Compatibility Early:** Test with the original `c4.c` frequently to ensure the Rust version matches the C version's output.
- **Start Early:** Rust's ownership model may require rethinking some C4 logic—begin with small components like the lexer.
- **Divide Work Strategically:** One teammate could handle parsing/input, the other the VM/output, merging regularly.
- **Use GitHub Issues:** Track tasks and compatibility bugs for clarity.
- **Leverage Rust Tools:**
  - `cargo fmt` for consistent formatting.
  - `cargo clippy` for catching potential issues.
  - `cargo test` for continuous testing.
- **Test Incrementally:** Write tests for each component (e.g., lexer against C4's tokens) as you go.
- **Seek Help:** Use Rust's community (e.g., Rust Discord) or the instructor if stuck.

Helpful Tools:

- **Rust Analyzer:** IDE plugin for linting and autocompletion (VS Code, etc.).
- **Cargo:** Rust's build tool (pre-installed).
- **Valgrind (for Comparison):** Compare memory usage with the C version.
- **GitHub Actions:** Set up CI to build and test on each push, ensuring compatibility.

This project will solidify your course knowledge and Rust skills while emphasizing teamwork and version control. Good luck!