# Individual Addendum — Sam Ruby

Sam Ruby

December 10, 2025

## Individual Addendum — Sam Ruby

### 1. Personal Contribution

My primary contributions centered on implementing and integrating the core components of the Vulpes compiler. I was responsible for the lexer, the abstract syntax tree (AST) layer, and the code generation subsystem, as well as managing the repository structure and build workflow.

#### Specific tasks completed

- Implemented the **lexical analyzer**, including token definitions, scanning rules, and error handling paths.
- Designed and implemented the **AST node hierarchy**, ensuring consistent and extensible representations of language constructs.
- Developed the **code generation module**, mapping Vulpes types and expressions to canonical IR-like representations.
- Connected the major compilation stages into an end-to-end pipeline: source → tokens → AST → generated output.
- Maintained the project's **build systems**, including CMake and Makefile targets, ensuring a clean and portable build.
- Wrote and refined sample `.vlp` programs for testing and demonstration.

#### Role in the project

My role focused on compiler construction, emphasizing architecture, subsystem integration, correctness, and maintainability.

---

### 2. Skills Applied and Gained

#### Technical skills applied

- C++17 development and multi-module architecture design.

- Compiler pipeline fundamentals: **tokenization, AST design, and IR-style code emission**.
- Error-handling patterns for compiler front ends.
- Build system configuration and maintenance (CMake, Make).

#### Skills gained or strengthened

- Language design and documentation of language semantics.
- Increased proficiency in designing clear, maintainable APIs between compiler subsystems.
- Improved collaboration skills through code reviews and iterative refinement.
- Better intuition for managing complexity while preserving pedagogical clarity.

---

## 3. Challenges Faced

#### Grammar flexibility and optional elements

Supporting optional type annotations, optional parameter names, and implicit defaults required careful balancing between clarity and conciseness. Ensuring the AST and codegen layers handled these combinations predictably led to several iterations of redesign.

#### Keeping the language simple yet useful

Because Vulpes is a teaching language, we intentionally limited complexity. Avoiding feature creep while still producing a functional, extensible compiler was a repeated challenge—especially in codegen, where a more formal IR backend was tempting but beyond the project's scope.

---

## 4. Reflection

#### What went well

- The final compiler pipeline is **clean, comprehensible, and highly suitable for instructional use**.
- Collaboration was effective; our small team allowed rapid iteration and strong alignment on design decisions.
- The documentation and sample programs now serve as solid teaching aids, supporting both conceptual understanding and hands-on learning.

#### What I would do differently

- Introduce **automated tests** earlier to supplement manual testing.

- Formalize the IR layer from the beginning to better support future language growth.
- Establish feature boundaries earlier in development to reduce rework around grammar and semantics.

---

## 5. Teamwork Evaluation

I contributed consistently and communicated design decisions clearly, especially when they affected parsing, syntax rules, or codegen behavior. I maintained reliable code quality and helped ensure compatibility across subsystems.

Miles and I collaborated effectively, dividing responsibilities according to strengths—his work on the parser and documentation paired naturally with my work on lexing, AST, and codegen. The team dynamic was positive and productive throughout the project. Additional scheduled sync points early on might have improved velocity, but overall collaboration was strong and constructive.

---