Can't Spell "Curriculum" Without "C"

Rust Education Workshop 2022
Tiemoko Ballo, *tballo@alumni.cmu.edu*
Alex James, *ajames@alumni.cmu.edu*

Let's be frank: we don't yet know if Rust is the future of high-performance software. Rust offers a commercially-viable solution to the dire and enduring problem of memory safety without garbage collection latency [1]. Coupled with a growing ecosystem and a long-held "most-loved" status [2], it's a solution with undeniable appeal. But will Rust have *anywhere near* the societal relevance of the C language, even 10 years from now? Is it truly the most appropriate way to teach systems programming to burgeoning engineers with little prior?

The staples – operating systems, computer networks, and embedded development – can readily be taught in Rust. Thus it may be tempting to de-prioritize C within a university curriculum. Or remove coverage outright.

Side-lining C would be a grave mistake: a disservice to today's learners and, by extension, tomorrow's practitioners. From the perspective of a realist, not an apologist. C has formidable pedagogical and pragmatic value. Especially if taught alongside a more modern alternative.

Today's Learners: Type System Contrast

As learners, students should have the opportunity to think deeply about the impact of abstraction design on a problem domain. On one hand, C's simplicity, in count of concepts and keywords, makes it an ideal vehicle for teaching fundamental low-level abstractions. On the other, a relative lack of high-level abstractions (no traits or classes, generics, sum types, references, iterators, collections, async) forces a ruthlessly tactical approach to problem solving. Students must carefully consider mechanical minutia before implementing business logic. Pointer arithmetic/`null`-state [3], memory allocation, data layout/initialization, undefined behavior, etc. Even the most rudimentary task is fraught with subtle defect potential.

When a learner inevitably introduces a runtime error, they are asked to reason about the many complexities of program execution. Even if C's constructs no longer map directly to hardware [4], a student must develop a reasonably accurate mental model of "what the machine is doing" to fix a segmentation fault. Programming in C remains a literal crash course in computer architecture. Of a visceral nature Rust can't quite imitate.

Now the lasting lesson, the career-long imprint, isn't about the machinery of stack and heap memory. It's about the difficulty of reliable defect elimination in weakly-typed programs. A first-hand taste of C debugging prepares students to appreciate the intent and comprehend the benefit of Rust's borrow checker. Learning both languages elucidates pass-by-reference semantics, with their myriad performance and security implications, via juxtaposition.

That security piece – the connection between an easy-to-introduce bug and a possible-to-exploit vulnerability – is key. In 2021, 67% of zero-day exploits "detected and disclosed as used in-the-wild" relied on memory corruption [5]. Contemporary education must reflect this reality. Through the lens of C, students can understand spatial (e.g. buffer overflow) and temporal (e.g. use-after-free) bug classes. In sufficient depth to diagnose root cause – not develop exploits. And this hard-won knowledge is transferable: professional Rust developers make judicious use of `unsafe` [6], where C-like concepts and risks still apply. Rust's type system is progress, not panacea.

Tomorrow's Practitioners: CFFI Competence

As practitioners, students will go on to build production systems of ambitious scale and complexity. That often means contributing to multi-lingual code bases, where each business problem is solved by the best tool for the job.

Several mechanisms enable cross-language interoperability. In the cloud, modern microservices use standardized response-request formats – like REST and gRPC. But on the client, the C Foreign Function Interface (CFFI) is king. For better or worse, C-style data representation remains *the* de facto protocol for connecting languages [7].

Why? Whereas package managers let us compose source code libraries, all programs ultimately compose at the compiled shared library and process/OS boundaries. Because every mainstream OS is written in C, CFFI powers abstractions at these intersections – like Python's extension modules, the Java Native Interface (JNI), and most application-specific embedded scripting. No matter what level of the technology stack graduates go on to specialize in, they'll never truly escape C semantics. In fact, those who master them are well-equipped to integrate otherwise disparate technologies and deliver impactful solutions. So let's prepare them accordingly.

Closing

Framing Rust in the context of C makes for an accessible introduction to type systems and practical vocational training. Simultaneously. That's why we chose this approach in *High Assurance Rust* – a free online textbook about developing secure and robust systems software [8]. Now there are countless pedagogical challenges in seamlessly bridging a multi-decade (1972 to 2010) language paradigm gap – we need a range of books, courses, and tools as diverse as the learners they serve. Only one thing is certain: the ideal curriculum is largely undefined.

References

[1] L. Szekeres, M. Payer, T. Wei, and D. Song, "SoK: Eternal War in Memory," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pp. 48–62, IEEE Computer Society, 2013.

[2] "Stack Overflow Developer Survey 2022: Most loved, dreaded, and wanted." `https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted`, Jun 2022.

[3] T. Hoare, "Null References: The Billion Dollar Mistake." `https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/`.

[4] D. Chisnall, "C Is Not a Low-Level Language: Your Computer is Not a Fast PDP-11.," *Queue*, vol. 16, p. 18–30, apr 2018.

[5] M. Stone, "The More You Know, The More You Know You Don't Know." `https://googleprojectzero.blogspot.com/2022/04/the-more-you-know-more-you-know-you.html`, Apr 2022.

[6] V. Astrauskas, C. Matheja, F. Poli, P. Müller, and A. J. Summers, "How Do Programmers Use Unsafe Rust?," *Proc. ACM Program. Lang.*, vol. 4, nov 2020.

[7] A. Beingessner, "C isn't a programming language anymore." `https://gankra.github.io/blah/c-isnt-a-language/`, Mar 2022.

[8] T. Ballo, M. Ballo, and A. James, "High Assurance Rust: Developing Secure and Robust Software." `https://highassurance.rs`, 2022.