Mastering Rust: A Comprehensive Guide

- From Basics to Advanced Patterns and Best Practices -

Ji Yong-Hyeon

Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

March 16, 2024

Rust vs C

Memory Safety
One of Rust's core features is its guarantee of memory safety without a garbage collector. Rust achieves this through its ownership system, consisting of three main rules:

- **Ownership**: Each value in Rust has a single owner, the variable that binds to it.
- **Borrowing**: You can have references to a value, but these must adhere to borrowing rules. At any given time, you can have either one mutable reference or any number of immutable references to a particular piece of data.
- **Lifetimes**: The compiler uses lifetimes to ensure that references do not outlive the data they refer to. This system prevents common bugs found in C programs, such as dangling pointers, buffer overflows, and concurrent data races.

Concurrency
Rust's approach to concurrency is also built on the ownership and type system, enabling thread-safe programming without the fear of data races. This is a significant shift from C, where managing concurrent access to data can be error-prone and complex.

Error Handling
Rust encourages a more explicit approach to error handling compared to C's use of integers for indicating errors. Rust uses Result and Option types for functions that can fail or return nothing, respectively, forcing you to handle these cases explicitly and improving code reliability.

Package Management and Build System
Rust comes with Cargo, its package manager and build system, simplifying dependency management, compilation, and packaging. Think of it as combining make, gcc, and a package repository like npm or Maven. The Cargo.toml file (which you asked about) is part of this system, defining your project's dependencies, build configurations, and metadata.

Zero-Cost Abstractions
Rust provides high-level abstractions like iterators, closures, and pattern matching without sacrificing performance. These features enable expressive and efficient code but compile down to low-level code with performance equivalent to C.

Compatibility with C
Rust can interoperate with C by using foreign function interfaces (FFI). This means you can call C code from Rust and vice versa, allowing gradual migration of C codebases to Rust or leveraging existing C libraries.

Learning Curve
While Rust offers many benefits, it has a steeper learning curve than C, primarily due to its strict compiler checks, ownership model, and borrowing rules. However, these same features that make Rust challenging to learn also contribute to its safety and efficiency.

Summary As a C developer, you'll find familiar ground in Rust's performance and control but with added safety and modern features. The ownership model is the most novel aspect, eliminating entire classes of bugs prevalent in C programs. Transitioning to Rust will involve a shift in thinking, particularly around memory management and type safety, but it can lead to more reliable and robust system-level applications.

Contents

Chapter 1

Rust Tutorial

1.1 Variable, Constants and Shadowing

Bibliography

[1] "Introduction to Rust Part 1" YouTube, uploaded by Ryan Levick, 16 November 2020, https://www.youtube.com/watch?v=WnWGO-tLtLA