**Rust Language Report.**

Malachi Gage Sanderson.

Embry-Riddle Aeronautical University, CS332.

Fall 2021.

There are many incredibly popular and well-established Turing-complete languages available for programmers to work with currently. One of the most established of these languages being Microsoft’s C/C++. Despite being some of the most used due to their flexibility and applicability to both high- and low-level programs, C/C++ do have some major drawbacks which leaves room for alternative, newer, languages to come in and see if they can do it better. The Rust Corporation developed the Rust language to try and do this.

The Rust programming language can have its roots drawn back as far as 2006 but the first usable version of the language came out in 2010. The creation of the language is attributed to Graydon Hoare. It is a language that certainly seems to leverage the fact that it is much younger than its competition. It takes lessons and features from the other major and well-established languages and attempts to improve upon them. In addition to merging and building upon other languages, Rust leverages a more modern iterative development process where it has a “…six-week release cycle…”(Klabnik, 21.5) and “every two or three years…” a new edition is released. This allows for its creators to try and address problems as well as implement new changes and features in a much shorter timeframe. Rust is being used by many major companies. In addition to being utilized by Mozilla, it was also sponsored by them a few years ago. Additionally, due to some of the benefits, particularly when it comes to security/stability, Google is looking to utilizing it on their Linux kernel for Android and Microsoft is looking to use Rust to reduce memory-related bugs in Windows components.

Rust is a multi-paradigm language that is syntactically very similar to C++. It is a language that prioritizes high performance, competing with C/C++, while addressing the safety and stability issues that those languages face. With the many benefits it offers, Rust is ideal for many different users, including teams of developers, students, companies, open-source developers, as well as people who value speed and stability (Klabnik, Introduction).

For development teams, Rust is very useful for quickly identifying the elusive bugs in low-level code through the compiler which “plays a gatekeeper role by refusing to compile code with these elusive bugs, including concurrency bugs”. Rust also offers a feature called “Cargo, the included dependency manager and build tool” that makes managing dependencies smooth and consistent. Additionally, Rust offers ‘Rustfmt’ which “ensures a consistent coding style across developers”.

For students, Rust offers a very accessible way to learn about systems concepts and provides an incredibly easy to read book for free online. Additionally, Rust boasts that the compiler ensures stability through feature additions and refactoring which contrasts to “…the brittle legacy code in languages without these checks, which developers are often afraid to modify”. Additionally, massive companies like Mozilla use Rust for major parts of the Firefox web browser.

One of the most unique features of the Rust language is how it is sort of an amalgamation of basically every other major programming language and has major influences from many other paradigms (Klabnik, 17.1). One element that makes it stand out as a particularly useful language is its flexibility as a multi-paradigm language. “Rust is influenced by many programming paradigms, including [Object-Oriented]” (Klabnik, 17.1) imperative, and functional paradigms (Klabnik, 13). It simultaneously supports many different elements from other languages such as Generics and inheritance as seen in Java, excellent support for concurrency, and it also features pointers/smart pointers, as seen in C/C++. Proponents for the Rust language also praise it for its safety, performance, and especially safe concurrency and memory management. As these elements similar to other languages are mostly self-explanatory and not exclusive to Rust, I will not being going into further detail on them. But there are features that are very unique to Rust and key to making it stand out. One of these distinguishing features is the concept of ‘ownership’, which is a major feature of Rust that makes it unique and helps it solve many of the security, concurrency, and memory error issues that are prevalent in C/C++.

One of Rust’s most central and unique features is ‘Ownership’. It “enables rust to make memory safety guarantees without needing a garbage collector” (Klabnik, 4). “Some languages have garbage collection that constantly looks for no longer used memory as the program runs; in other languages, the programmer must explicitly allocate and free the memory” (Klabnik, 4) while Rust utilizes the concept of Ownership to take a different approach. Most importantly, none of the ownership features actually have any overhead for your program while its running. The details of ownership in Rust get complicated and confusing as you dive deeper into them as they are unique to this language, so I will not get very specific with it but it is useful to understand 3 of the main rules of ownership in Rust: 1. each value in Rust has a variable that’s called its owner, 2. there can only be one owner at a time, 3. when the owner goes out of scope the value will be dropped (Klabnik, 4.1). Generally speaking, the stack and heap have a greater effect on the language’s behavior in Rust than many other languages; and this has to do with the concept of ownership in Rust.

Another notably unique feature of the Rust language is the fact that “every reference in Rust has a lifetime” (Klabnik, 10.3) which distinguishes it from C/C++. This ‘lifetime’ is the scope for which a reference is valid. Usually lifetimes are “…implicit and inferred, just like most of the time, types are inferred”. Rust requires annotating of the relationships using generic lifetime parameters to “…ensure the actual references used at runtime will definitely be valid”. This plays into another unique feature of Rust, it doesn’t have null values. You are allowed to declare a variable and a type, yet you will get a compilation error if you try to use a variable before giving it a value. If you declare a variable without assigning it a value then in an inner-scope of a function you assign the value of a new variable to that initial variable, then once leaving that inner-scope you return to the outer-scope and try and print the initial variable, the code won’t compile because the value you set that initial variable equal to doesn’t have as long a lifetime and pretty much outlives the piece of memory that was allocated to the inner variable which is deallocated once the inner-scope is left.

One other unique feature Rust has is that it supports use of a “Foreign Function Interface (FFI)” (Klabnik, 19.1) allowing for rust to interact with/call functions from other languages. One of which languages Rust’s proponents say does a surprisingly good job of translating using its FFI is C++, which is notoriously difficult to work with across different languages. Yet it saves this feature for a subset of Rust features that it restricts to what it calls ‘Unsafe Rust’, a bunch of currently more unstable features in Rust. Especially notable when you account for the newness of the language.

Rust is relatively new and lacks some features and libraries of more established languages. Additionally, because errors are detected at compile time, to some users developing code is not as fast as in some other languages like Python. Another issue some of its detractors have with Rust is, with the language being constantly updated, there is the factor that pretty significant changes are made on a relatively frequent basis.

Now that we have gone through a bunch of information as to what Rust is and what makes it unique, lets finish up by looking at some of its syntax.

|  |  |  |
| --- | --- | --- |
| Declaration | Signed byte integer | i8 |
|  | Unsigned byte integer | u8 |
|  | 32-bit signed integer | i32 |
|  | 32-bit unsigned integer | u32 |
|  | Character | char |
|  | String | String |
| Assignment |  | let mut name«: type»« = initial\_value»; |
| Selection | IF | if condition {  expression } else if condition {  expression } <<else {  expression }>> |
|  | Select case (pattern-matching) | match variable {  pattern1 => expression,  pattern2 => expression,  pattern3 => expression } |
| Repetition | WHILE | while condition {   expression } |
|  | FOR | for i in first… =last {  expression } |

**Sources**

Klabnik, S., & Nichols, C. (2019). *The rust programming language* (Vol. 2). No Starch Press.

ISBN-13: 9781718500440.