Rust (programming language)

Rust is a systems programming language[9] sponsored by Mozilla Research,[10] which describes it as a "safe, concurrent, practical language,"[11] supporting functional and imperative-procedural paradigms. Rust is syntactically similar to C++, but its designers intend it to provide better memory safety while maintaining performance.

Rust is an open source programming language. Its designers have refined the language through the experiences of writing the Servo[12] web browser layout engine and the Rust compiler. A large portion of current commits to the project are from community members.[13]

Rust won first place for "most loved programming language" in the Stack Overflow Developer Survey in 2016 and 2017;[14][15] it is referenced in The Book of Mozilla as "oxidised metal".

Rust

Paradigm	Multi-paradigm: compiled, concurrent, functional, imperative, structured, generic
Designed by	Originally Graydon Hoare, then Rust project developers
Developer	Rust project developers
First appeared	2010
Stable release	1.22.1[1] / November 22, 2017
Typing discipline	static, strong, inferred, nominal, linear
Implementation language	Rust
OS	Linux, macOS, Windows, FreeBSD, Redox (operating system) Android, iOS (partial)[2]
License	Apache License 2.0 or MIT License[3]
Filename extensions	.rs, .rlib

Contents

Design and features

The goal of Rust is to be a language for highly concurrent and highly safe systems,[16] and "programming in the large," that is, creating and maintaining boundaries that preserve large-system integrity.[17] This has led to a feature set with an emphasis on safety, control of memory layout, and concurrency. Performance of idiomatic Rust is comparable to the performance of idiomatic C++.[18][19]

Syntax

The concrete syntax of Rust is similar to C and C++, with blocks of code delimited by curly brackets, and control flow keywords such as `if`, `else`, `while`, and `for`. Not all C or C++ keywords are implemented, however, while some Rust functionality (such as keyword `match` for pattern matching) will be less familiar to programmers coming from these languages. Despite the superficial resemblance to C

and C++, the syntax of Rust in a deeper sense is closer to that of the ML family of languages. Nearly every part of a function body is an expression[20], even control flow operators. For example, the ordinary `if` expression also takes the place of C's ternary conditional. A function does not need to end with a `return` expression; the last expression in the function is used as the return value.

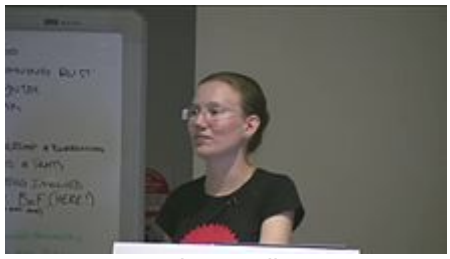
This program prints the string "Hello, world!" to standard output and exits.

```
fn main() {
    println!("Hello, world!");
}
```

Website	www.rust-lang.org
Influenced by	
Alef,[4] C#,[4] C++,[4] Cyclone,[4][5] Erlang,[4] Haskell,[4] Haxe,[4] Hermes,[4] Limbo,[4] Newsqueak,[4] NIL,[4] OCaml,[4] Ruby,[4] Scheme,[4] Standard ML,[4] Swift[4][6]	
Influenced	
Crystal, Elm,[7] Idris[8]	

Memory safety

The system is designed to be memory safe, and it does not permit null pointers, dangling pointers, or data races in safe code.[21][22][23][24] Data values can only be initialized through a fixed set of forms, all of which require their inputs to be already initialized.[25] To replicate the functionality in other languages of pointers being either valid or `NULL`, such as in linked list or binary tree data structures, the Rust core library provides an option type, which can be used to test if a pointer has `Some` value or `None`.[22] Rust also introduces additional syntax to manage lifetimes, and the compiler reasons about these through its *borrow checker*.



Play media

A presentation from linux.conf.au that explains the key concepts necessary for successful Rust programming.

Memory management

Rust does not use an automated garbage collection system like those used by Go, Java or .NET Framework. Instead, memory and other resources are managed through resource acquisition is initialization (RAII), with optional reference counting. Rust provides deterministic management of resources, with very low overhead. Rust also favors stack allocation of values and does not perform implicit boxing.

There is also a concept of references (using the & symbol), which do not involve run-time reference counting. The safety of using such pointers is verified at compile time by the borrow checker, preventing dangling pointers and other forms of undefined behavior.

Types and polymorphism

The type system supports a mechanism similar to type classes, called "traits", inspired directly by the Haskell language. This is a facility for ad-hoc polymorphism, achieved by adding constraints to type variable declarations. Other features from Haskell, such as higher-kinded polymorphism, are not yet supported.

Rust features type inference, for variables declared with the `let` keyword. Such variables do not require a value to be initially assigned to determine their type. A compile time error results if any branch of code fails to assign a value to the variable.[26] Variables assigned multiple times must be marked with the `mut` keyword.

Functions can be given generic parameters, which usually require the generic type to implement a certain trait or traits. Within such a function, the generic value can only be used through those traits. This means that a generic function can be type-checked as soon as it is defined. This is in contrast to C++ templates, which are fundamentally duck typed and cannot be checked until instantiated with concrete types.

However, the *implementation* of Rust generics is similar to the typical implementation of C++ templates: a separate copy of the code is generated for each instantiation. This is called monomorphization and contrasts with the type erasure scheme typically used in Java and Haskell. The benefit of monomorphization is optimized code for each specific use case; the drawback is increased compile time and size of the resulting binaries.

The object system within Rust is based around implementations, traits and structured types. Implementations fulfill a role similar to that of classes within other languages, and are defined with the `impl` keyword. Inheritance and polymorphism are provided by traits; they allow methods to be defined and mixed in to implementations. Structured types are used to define fields. Implementations and traits cannot define fields themselves, and only traits can provide inheritance. Among other benefits, this prevents the diamond problem of multiple inheritance, as in C++. In other words, Rust supports interface inheritance, but replaces implementation inheritance with composition, see composition over inheritance.

History

The language grew out of a personal project started in 2006 by Mozilla employee Graydon Hoare,[11] who stated that the project was possibly named after the rust family of fungi.[27] Mozilla began sponsoring the project in 2009[11] and announced it in 2010.[28][29] The same year, work shifted from the initial compiler (written in OCaml) to the self-hosting compiler written in Rust.[30] Known as `rustc`, it successfully compiled itself in 2011.[31] `rustc` uses LLVM as its back end.

The first numbered pre-alpha release of the Rust compiler occurred in January 2012.[32] Rust 1.0, the first stable release, was released on May 15, 2015.[33][34] Following 1.0, stable point releases are delivered every six weeks, while features are developed in nightly Rust and then tested with alpha and beta releases that last six weeks.[35]

In addition to conventional static typing, before version 0.4, Rust also supported typestates. The typestate system modeled assertions before and after program statements, through use of a special `check` statement. Discrepancies could be discovered at compile time, rather than when a program was running, as might be the case with assertions in C or C++ code. The typestate concept was not unique to Rust, as it was first introduced in the language NIL.[36] Typestates were removed because in practice they found little use, though the same functionality can still be achieved with branding patterns.[37]

The style of the object system changed considerably within versions 0.2, 0.3 and 0.4 of Rust. Version 0.2 introduced classes for the first time, with version 0.3 adding a number of features including destructors and polymorphism through the use of interfaces. In Rust 0.4, traits were added as a means to provide inheritance; interfaces were unified with traits and removed as a separate feature. Classes were also removed, replaced by a combination of implementations and structured types.

Starting in Rust 0.9 and ending in Rust 0.11, Rust had two built-in pointer types, `~` and `@`, simplifying the core memory model. It reimplemented those pointer types in the standard library as `Box` and (the now removed) `Gc`.

In January 2014, the editor-in-chief of *Dr Dobb's*, Andrew Binstock, commented on Rust's chances to become a competitor to C++, as well as to the other upcoming languages D, Go and Nim (then Nimrod): according to Binstock, while Rust was "widely viewed as a remarkably elegant language", adoption slowed because it repeatedly changed between versions.[38]

Rust was the third most loved programming language in the 2015 Stack Overflow annual survey,[39] and took first place in 2016 and 2017.[40][41]

Projects using Rust

The Rust compiler itself is written in Rust.[30]

Other projects developed in Rust include:

Web browser oriented:

- Firefox[42]
  - Servo – Mozilla's new parallel web browser engine[43] developed in collaboration with Samsung[44]
  - Quantum – a project, composed of several sub-projects, to improve the Gecko web browser engine of Firefox, developed by Mozilla[45]

Build tool oriented:

- Cargo – Rust's build automation system
- Habitat – a build and deployment tool from Chef Software[46]

Other projects:

- CITA - a fast and scalable permissioned blockchain for production[47]
- Magic Pocket – Dropbox's file system that powers their Diskotech petabyte storage machines[48]
- OpenDNS – used in two of its components[49][50][51]
- Redox – a microkernel operating system[52]
- Piston – a game engine[53]
- Amethyst – a data-oriented, data-driven game engine[54]
- OnePush – a notification delivery system developed by OneSignal[55]
- REmacs – a port of Emacs to Rust[56]
- MAIDsafe – a P2P Internet project currently being developed by a team in Troon, Scotland[57]
- Lucidscape Mesh – a distributed real-time simulation engine for virtual reality[58]
- Tock - an embedded operating system[59]
- Tor – an anonymity network[60]
- IOTA – a distributed ledger cryptocurrency[61]
- Pijul – a distributed version control system inspired by Darcs[62]
- Railcar – a container runtime by Oracle[63]
- Rustation - a PlayStation emulator[64]
- Parity - a Ethereum web browser and wallet management application[65]
- 'Spellbound' - an unnamed video game by Chucklefish[66]

See also

- Comparison of programming languages

References

1. "Announcing Rust 1.22 (and 1.22.1)" (https://blog.rust-lang.org/2017/11/22/Rust-1.22.html) *blog.rust-lang.org*. 2017-11-22. Retrieved 2017-11-24.
2. "Rust on iOS" (https://github.com/rust-lang/rust-wiki-backup/blob/master/Doc-building-for-ios.md) *GitHub*. 2015-01-09. Archived from the original (https://github.com/rust-lang/rust/wiki/Doc-building-for-ios) on 2015-01-09. Retrieved 2017-06-22.
3. "Copyright" (https://github.com/rust-lang/rust/blob/master/COPYRIGHT) Rust compiler source repository Retrieved 2015-11-09.
4. "The Rust Reference: Appendix: Influences" (https://doc.rust-lang.org/reference/influences.html) Retrieved July 14, 2017. "Rust is not a particularly original language, with design elements coming from a wide range of sources. Some of these are listed below (including elements that have since been removed): SML, OCaml [...] C++ [...] ML Kit, Cyclone [...] Haskell [...] Newsqueak, Alef, Limbo [...] Erlang [...] Swift [...] Scheme [...] C# [...] Ruby [...] NIL, Hermes"
5. "Note Research: Type System" (https://github.com/rust-lang/rust-wiki-backup/blob/master/Note-research.md#type-system). 2015-02-01. Retrieved 2015-03-25. "Papers that have had more or less influence on Rust, or which one might want to consult for inspiration or to understand Rust's background. [...] *Region based memory management in Cyclone* [...] *Safe memory management in Cyclone*"
6. "RFC for 'if let' expression" (https://github.com/rust-lang/rfcs/pull/160) Retrieved December 4, 2014.
7. "Command Optimizations?" (https://groups.google.com/forum/?fromgroups#!searchin/elm-discuss/rust/elm-discuss/lMX_9miTD2E/QBwdvL4JD9wJ) 2014-06-26. Retrieved 2014-12-10. "I just added the outline of a Result library that lets you use richer error messages. It's like Either except the names are more helpful. The names are inspired by Rust's Result library."

8. "Uniqueness Types" (https://web.archive.org/web/20141225103820/https://github.com/idris-lang/Idris-dev/wiki/Uniqueness-Types). 2014-08-22. Archived from the original (https://github.com/idris-lang/Idris-dev/wiki/Uniqueness-Types) on December 25, 2014 Retrieved 2014-10-27. "They are inspired by linear types, Uniqueness Types in the Clean programming language, and ownership types and borrowed pointers in the Rust programming language."
9. "Rust is a systems programming language" (https://www.rust-lang.org). *Rust-lang.org*. Retrieved 2017-07-17.
10. Noel (2010-07-08). "The Rust Language" (http://lambda-the-ultimate.org/node/4009) Lambda the Ultimate Retrieved 2010-10-30.
11. "FAQ - The Rust Project" (https://www.rust-lang.org/faq.html#project) *Rust-lang.org*. Retrieved 2 March 2016.
12. Peter Bright (2013-04-03). "Samsung teams up with Mozilla to build browser engine for multicore machines" (https://arstechnica.com/information-technology/2013/04/samsung-teams-up-with-mozilla-to-build-browser-engine-for-multicore-machines/). *Arstechnica.com*. Retrieved 2013-04-04.
13. "Contributors to rust-lang/rust · GitHub" (https://github.com/mozilla/rust/contributors) *Github.com*. Retrieved 2016-11-03.
14. "Stack Overflow Developer Survey 2016 Results" (https://stackoverflow.com/insights/survey/2016#technology-most-loved-dreaded-and-wanted) *Stack Overflow*. Retrieved 2017-03-22.
15. "Stack Overflow Developer Survey 2017" (https://stackoverflow.com/insights/survey/2017#most-loved-dreaded-and-wanted). *Stack Overflow*. Retrieved 2017-03-22.
16. Avram, Abel (2012-08-03). "Interview on Rust, a Systems Programming Language Developed by Mozilla" (http://www.infoq.com/news/2012/08/Interview-Rust) InfoQ. Retrieved 2013-08-17. "**GH:** A lot of obvious good ideas, known and loved in other languages, haven't made it into widely used systems languages... There were a lot of good competitors in the late 1970s and early 1980s in that space, and I wanted to revive some of their ideas and give them another go, on the theory that circumstances have changed: the internet is highly concurrent and highly security-conscious, so the design-tradeoffs that always favor C and C++ (for example) have been shifting."
17. Debian package description: rustc (https://packages.debian.org/sid/main/rustc)
18. Walton, Patrick (2010-12-05). "C++ Design Goals in the Context of Rust" (http://pcwalton.blogspot.com/2010/12/c-design-goals-in-context-of-rust.html) Retrieved 2011-01-21. "... It's impossible to be "as fast as C" in all cases while remaining safe... C++ allows all sorts of low-level tricks, mostly involving circumventing the type system, that offer practically unlimited avenues for optimization. In practice, though, C++ programmers restrict themselves to a few tools for the vast majority of the code they write, including stack-allocated variables owned by one function and passed by alias, uniquely owned objects (often used with `auto_ptr` or the C++0x `unique_ptr`), and reference counting via `shared_ptr` or COM. One of the goals of Rust's type system is to support these patterns exactly as C++ does, but to enforce their safe usage. In this way, the goal is to be competitive with the vast majority of idiomatic C++ in performance, while remaining memory-safe."..
19. "How Fast Is Rust?" (https://www.rust-lang.org/en-US/faq.html#how-fast-is-rust). *The Rust Programming Language FAQ*. Retrieved 3 August 2016.
20. "rust/src/grammar/parser-lalr.y" (https://github.com/rust-lang/rust/blob/5b13bff5203c1bdc6ac6dc87f69b5359a9503078/src/grammar/parser-lalr.y#L1309-L1573). 2017-05-23. Retrieved 2017-05-23.
21. Rosenblatt, Seth (2013-04-03). "Samsung joins Mozilla's quest for Rust" (http://reviews.cnet.com/8301-3514_7-57577639/samsung-joins-mozillas-quest-for-rust/) Retrieved 2013-04-05. "[Brendan Eich] noted that every year browsers fall victim to hacking in the annual Pwn2Own contest at the CanSecWest conference. "There's no free memory reads" in Rust, he said, but there are in C++. Those problems "lead to a lot of browser vulnerabilities" and would be solved by Rust, which is a self-compiling language".
22. Brown, Neil (2013-04-17). "A taste of Rust" (https://lwn.net/Articles/547145/) Retrieved 2013-04-25. "... Other more complex data structures could clearly be implemented to allow greater levels of sharing, while making sure the interface is composed only of owned and managed references, and thus is safe from unplanned concurrent access and from dangling pointer errors."
23. "`unsafe` - The Rust Programming Language" (https://doc.rust-lang.org/book/first-edition/unsafe.html)
24. "Data Races and Race Conditions" (https://doc.rust-lang.org/nomicon/races.html)
25. "The Rust Language FAQ" (https://web.archive.org/web/20150420104147/http://static.rust-lang.org/doc/master/complement-lang-faq.html) *static.rust-lang.org*. 2015. Archived from the original (http://static.rust-lang.org/doc/master/complement-lang-faq.html) on 2015-04-20. Retrieved 2017-04-24.
26. Walton, Patrick (2010-10-01). "Rust Features I: Type Inference" (http://pcwalton.blogspot.com/2010/10/rust-features-i-type-inference.html) Retrieved 2011-01-21.

27. "Internet archaeology: the definitive, end-all source for why Rust is named "Rust" : rust" (https://www.reddit.com/r/rust/comments/27jvdt/internet_archaeology_the_definitive_endall_source/). *Reddit.com*. 2014-06-07. Retrieved 2016-11-03.
28. "Future Tense" (http://www.slideshare.net/BrendanEich/future-tense-7782010). 2011-04-29. Retrieved 2012-02-06. "At Mozilla Summit 2010, we launched Rust, a new programming language motivated by safety and concurrency for parallel hardware, the "manycore" future which is upon us".
29. Hoare, Graydon (7 July 2010). *Project Servo* (http://venge.net/graydon/talks/intro-talk-2.pdf) (pdf). Mozilla Annual Summit 2010. Whistler, Canada. Retrieved 22 February 2017.
30. Hoare, Graydon (2010-10-02). "Rust Progress" (https://web.archive.org/web/20140815054745/http://blog.mozilla.org/graydon/2010/10/02/rust-progress/). Archived from the original (http://blog.mozilla.com/graydon/2010/10/02/rust-progress/) on 2014-08-15. Retrieved 2010-10-30.
31. Hoare, Graydon (2011-04-20). "[rust-dev] stage1/rustc builds" (https://mail.mozilla.org/pipermail/rust-dev/2011-April/000330.html). Retrieved 2011-04-20. "After that last change fixing the logging scope context bug, looks like stage1/rustc builds. Just shy of midnight :)"
32. catamorphism (2012-01-20). "Mozilla and the Rust community release Rust 0.1 (a strongly-typed systems programming language with a focus on memory safety and concurrency)" (https://www.reddit.com/r/programming/comments/opgxd/mozilla_and_the_rust_community_release_rust_01_a/). Retrieved 2012-02-06.
33. "Version History" (https://github.com/rust-lang/rust/blob/master/RELEASES.md). Retrieved 2017-01-01.
34. The Rust Core Team (May 15, 2015). "Announcing Rust 1.0" (http://blog.rust-lang.org/2015/05/15/Rust-1.0.html) Retrieved 2015-12-11.
35. "Scheduling the Trains" (https://blog.rust-lang.org/2014/12/12/1.0-Timeline.html). Retrieved 2017-01-01.
36. Strom, Robert E.; Yemini, Shaula (1986). "Typestate: A Programming Language Concept for Enhancing Software Reliability" (http://www.cs.cmu.edu/~aldrich/papers/classic/tse12-typestate.pdf) (PDF). IEEE Transactions on Software Engineering. ISSN 0098-5589 (https://www.worldcat.org/issn/0098-5589) Retrieved 2010-11-14.
37. Walton, Patrick (2012-12-26). "Typestate Is Dead, Long Live Typestate!" (https://pcwalton.github.com/blog/2012/12/26/typestate-is-dead/). *Pcwalton.github.com*. Retrieved 2016-11-03.
38. Binstock, Andrew. "The Rise And Fall of Languages in 2013" (http://www.drdobbs.com/jvm/the-rise-and-fall-of-languages-in-2013/240165192). *Dr Dobb's*.
39. "Stack Overflow Developer Survey 2015" (https://stackoverflow.com/research/developer-survey-2015). *Stackoverflow.com*. Retrieved 2016-11-03.
40. "Stack Overflow Developer Survey 2016 Results" (https://stackoverflow.com/insights/survey/2016#technology-most-loved-dreaded-and-wanted). *Stack Overflow*. Retrieved 2017-03-22.
41. "Stack Overflow Developer Survey 2017" (https://stackoverflow.com/insights/survey/2017#most-loved-dreaded-and-wanted). *Stack Overflow*. Retrieved 2017-03-22.
42. Herman, Dave (2016-07-12). "Shipping Rust in Firefox * Mozilla Hacks – the Web developer blog" (https://hacks.mozilla.org/2016/07/shipping-rust-in-firefox/). *Hacks.mozilla.org*. Retrieved 2016-11-03.
43. Serdar Yegulalp (3 April 2015). "Mozilla's Rust-based Servo browser engine inches forward" (http://www.infoworld.com/article/2905688/applications/mozillas-rust-based-servo-browser-engine-inches-forward.html). *InfoWorld*. Retrieved 2016-03-15.
44. Frederic Lardinois (3 April 2015). "Mozilla And Samsung Team Up To Develop Servo, Mozilla's Next-Gen Browser Engine For Multicore Processors" (https://techcrunch.com/2013/04/03/mozilla-and-samsung-collaborate-on-servo-mozillas-next-gen-browser-engine-for-tomorrows-multicore-processors/). *TechCrunch*.
45. Bryant, David. "A Quantum Leap for the Web" (https://medium.com/mozilla-tech/a-quantum-leap-for-the-web-a3b7174b3c12#.ldic6a78e). *Medium*. Retrieved 27 October 2016.
46. "Automate any app, anywhere with Habitat" (https://www.habitat.sh/). *Chef Software*. Retrieved 2017-08-01.
47. "CITA" (https://github.com/cryptape/cita/). *Github*. Retrieved 2017-09-09.
48. "The Epic Story of Dropbox's Exodus From the Amazon Cloud Empire" (https://www.wired.com/2016/03/epic-story-dropboxs-exodus-amazon-cloud-empire/). *Wired.com*. 2016-03-14. Retrieved 2016-11-03.
49. Balbaert, Ivo. *Rust Essentials* (https://books.google.com/books?id=TiuCQAAQBAJ&pg=PA6&lpg=PA6&dq=OpenDNS+Rust&source=bl&ots=UL5thAAi8w&sig=Wf-Z5xSRYU-IXyGiyIl2FVEQWEc&hl=en&sa=X&ved=0ahUKEwizzdSk59LLAhVpnoMKHWdbDrQQ6AEINzAF#v=onepage&q=OpenDNS%20Rust&f=false). Packt Publishing. p. 6. ISBN 1785285769. Retrieved 21 March 2016.

50. Frank, Denis. "Using HyperLogLog to Detect Malware Faster Than Ever" (https://labs.opendns.com/2013/12/05/hyperloglog-and-malware-detection/) *OpenDNS Security Labs* Retrieved 19 March 2016.
51. Denis, Frank. "ZeroMQ: Helping us Block Malicious Domains in Real Time" (https://labs.opendns.com/2013/10/04/zeromq-helping-us-block-malicious-domains/) *OpenDNS Security Labs* Retrieved 19 March 2016.
52. Yegulalp, Serdar. "Rust's Redox OS could show Linux a few new tricks" (http://www.infoworld.com/article/3046100/open-source-tools/rusts-redox-os-could-show-linux-a-few-new-tricks.html) *Infoworld*. Retrieved 21 March 2016.
53. "Piston A modular game engine written in Rust" (http://www.piston.rs/). *Piston.rs*. Retrieved 2017-08-01.
54. "Data-oriented game engine written in Rust" (https://www.amethyst.rs/). *Amethyst.rs*. Retrieved 2017-10-07.
55. "OneSignal now sends iOS push notifications 100x faster" (https://onesignal.com/blog/announcing-our-new-delivery-backend/). *OneSignal*. 2016-03-21. Retrieved 2017-01-06.
56. Larabel, Michael (2017-01-11). "Remacs: Re-Implementing Emacs In Rust" (https://www.phoronix.com/scan.php?page=news_item&px=Remacs-Rust-Emacs) *phoronix.com*. Retrieved 2017-01-19.
57. "MaidSafe - The New Decentralized Internet" (https://maidsafe.net/) *MaidSafe.net*. Retrieved 2017-08-01.
58. "Building the Metaverse" (http://lucidscape.com/) *Lucidscape.com*. Retrieved 2017-08-01.
59. "Tock Embedded Operating System" (https://www.tockos.org/). *Tock Embedded Operating System*. Retrieved 2017-11-13.
60. Hahn, Sebastian (2017-03-31). "[tor-dev] Tor in a safer language: Network team update from Amsterdam" (https://lists.torproject.org/pipermail/tor-dev/2017-March/012088.html) Retrieved 2017-04-01.
61. Sønstebø, David (2017-03-31). "[iota-development-roadmap] IOTA Development Roadmap" (https://blog.iota.org/iota-development-roadmap-74741f37ed01) Retrieved 2017-07-06.
62. "Pijul" (https://pijul.org/). *pijul.org*. Retrieved 8 July 2017.
63. "Building a Container Runtime in Rust" (https://blogs.oracle.com/developers/building-a-container-runtime-in-rust) 29 June 2017. Retrieved 8 July 2017. "Why Rust? (…) Rust sits at a perfect intersection of [C and Go]: it has memory safety and higher-level primitives, but doesn't sacrifice low level control over threading and therefore can handle namespaces properly."
64. "Rustation" (https://github.com/simias/rustation) Retrieved 8 July 2017.
65. "Parity" (https://www.reddit.com/r/rust/comments/78bowa/hey_this_is_kyren_from_chucklefish_we_make_and/) Retrieved 6 September 2017.
66. "Hey, this is kyren from Chucklefish, we make and publish cool video games. One of our two next projects is currently being written in rust, and I'd like to talk to you about it! • r/rust" (https://www.reddit.com/r/rust/comments/78bowa/hey_this_is_kyren_from_chucklefish_we_make_and/) *reddit*. Retrieved 2017-10-27.

External links

- Official website
- The Rust-dev Archives – electronic mailing list
- rust on GitHub – primary source code repository and bug tracker
- rust-rosetta on GitHub – implementations of common algorithms and solutions
- Rust by Example – web book
- Rust SubReddit – active dev community language development and support
- Rust compared to other programming languages – interactive comparison

---