Course Introduction

CS392: Rust, in Theory and in Practice

September 2, 2025 (Lecture 1)

Outline

Course Expectations

What is this course about?

Workshop: Install Rust

Minutiae

Instructor: Nathan Mull

Webpage: https://nmmull.github.io/CS392-F25/index.html

Midterm Date: October 23

Grade Breakdown

30%	Assignments
40%	Final Project
20%	Midterm (in class)
10%	Participation

Disclaimer

This is still a new course. It's going to be a bit disorganized. I appreciate your patience

I am *not* an expert of Rust, I'm an expert in **type theory**

Lectures

This course will be something like a flipped classroom/workshop hybrid. You'll be expected to read before lecture and do a small amount of pre-lecture participation

I will take attendance. You're allowed to miss 2-3 lectures

We'll spend the first part of the lecture reviewing the material you read about, and then we'll go into a workshop-style meeting during which you'll work on the homework assignments or final projects or other in-class tasks

I want this course to be very collaborative. I'll expect that you're working in groups, pair/group programming, and one-on-one discussions with me and the other students

Assignments

Assignments will consist of either programming exercises or larger programming tasks in Rust. We may have 1-2 written assignments

There are 6-7 total, I'll drop your lowest

Even if you pair/group program a problem, try to type your own solution and cite who you worked with

Final Project

The final project will be a self-defined, and will take up most of the second half of the course

We'll talk more about this as we get closer to the midterm

You should start thinking now about potential projects. I will provide many possible topics and projects if you're having a hard time coming up with something.

Other Stuff

We'll use Piazza for course communication

We'll use Gradescope for assignment submissions

Please read the course manual on the course webpage in its entirety

Questions?

If I missing anything, please ask on Piazza

Remember: This is a *small, experimental* course. You'll be helping me define the material. You'll get out of it what you put into it

Last Thing: What's your name?

Name:

Year:

Interest in CS:

Interest outside of CS:

Take a minute to think about it, then we'll go around the room.

Outline

Course Expectations

What is this course about?

Workshop: Install Rust

The Idea

First half: Learn enough Rust to be dangerous

- **Practically:** What do we need to know about Rust in order to use it?
- **Theoretically:** What mental model do we need to think about how Rust works (and how to we implement that mental model)?

Second half: Build something dangerous

- We'll also cover more advanced topics. You'll notice that the calendar says "TBD." You will help me figure out what we're going to cover.

Another Word of Warning

This is *not* an introductory programming course

Some things will move very fast (I'll assume you'll be able to write simple programs within the first week)

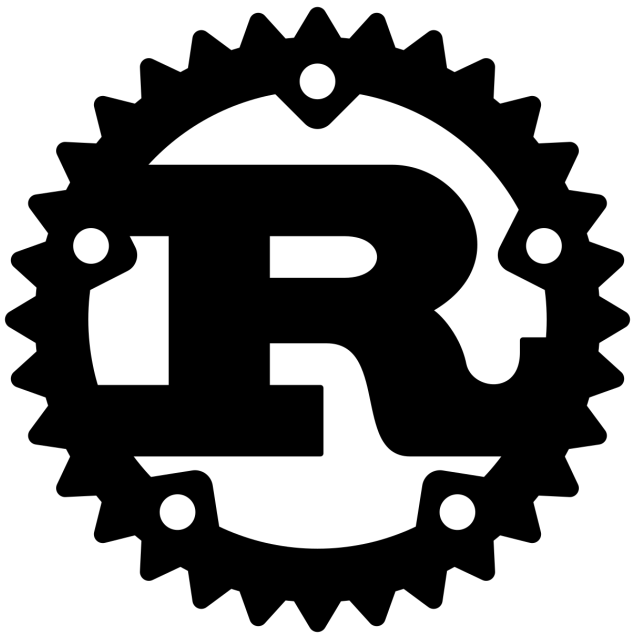
Some things will move very slow (We'll dwell a bit on things like memory management)

What's Rust?

Rust is a type-safe memory-safe PL

It's possible to write simple clean code that's guaranteed to be free of memory bugs

It's an **alternative to C or C++** which can be used in production settings for rapid development without fear of crashes or memory leaks



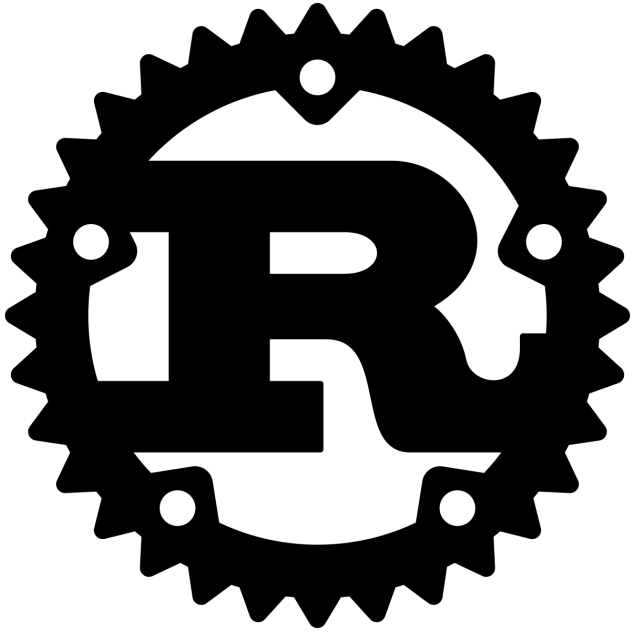
About Rust

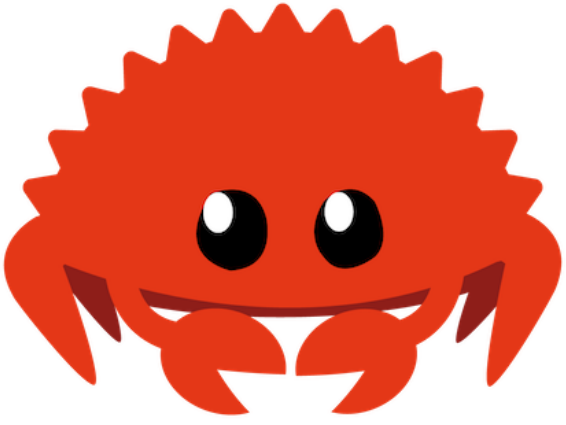
Developed by Graydon Hoare out of Mozilla in the 2000s (originally implemented in OCaml)

It became stable (in particular with its type system) in the late 2010s

The Rust Foundation was started in 2021 and is the basis for Rust information and adoption today

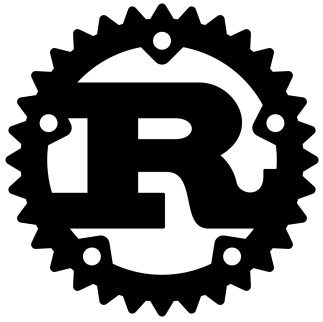
It's community members are called **Rustacians**, which is the basis for the unofficial mascot **Ferris the crab**





Why Rust?

1. *Rust is weird.* It uses a unique type system to achieve its memory safety, which programmers often have to wrestle with
2. *Rust is becoming popular.* Mozilla, Dropbox, Yelp, Amazon (along with lots of others) are all adopting Rust in large-scale projects



Example: Rust Weirdness

```
// THIS DOES NOT COMPILE
fn swap(x : &mut String, y : &mut String) {
  let z : String = *x;
  *x = *y;
  *y = z;
}
```

Rust has a notion of references, but it's *not possible* to write the swap-string-pointer function (in safe Rust)

A badly defined pointer-swap could cause a memory leak. Rust's type system disallows this *by fiat*

Example: Rust Sustainability

An interesting (slightly dated) article out of AWS [link]

Rust is performant, energy efficient and *a whole lot more interesting than many other options*

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Aside: How to Learn a PL

We tend to learn and teach PLs the "wrong" (fast) way, i.e., reading tutorials and doing examples

In this course we want to learn Rust the "right" (slow) way, i.e., formally describe what Rust is doing

We won't learn many cool, advanced features of Rust that are useful in practice

We will learn why Rust makes us tackle with the type system, and how it works

How *does* it work?

Linear/affine types, which are based on **Linear Logic** of Girard (1980s)

Rough idea:

- A function of type $A \to B$ take an A and gives us a B
- A function of type $A \multimap B$ **consumes** an A and gives a B

This ensures data is never unnecessarily duplicated or thrown away

And gives us high-fidelity control of our memory in a way that's recognized by the type system

Outline

Course Expectations

What is this course about?

Workshop: Install Rust

The Task

Take a look at the course webpage if you haven't already

Follow the in The Rust Programming Language (RPL) on installing **rustup**. If you're using windows I highly recommend using WSL. If you finish, then follow the tutorial in RPL called **Hello, Cargo!**

Note. This is how I'll take attendance, so please make sure to talk to me before the end of lecture