Lec 1: What's this "Rust" thing anyways?

Jack Duvall

"Knowledge check": https://forms.gle/Nru2UnL7W92p2zUP9

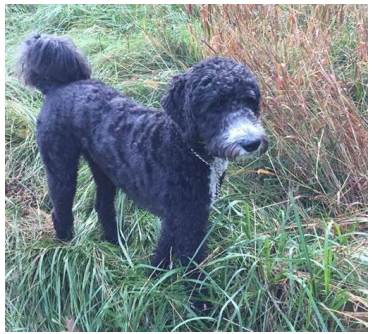
About Your Instructors

Jack Duvall

- SCS Junior
- Rust Experience: Personal projects, FB internship, and 15-451
- Links:
  - https://github.com/duvallj
  - https://duvallj.pw
- Dogs: 

Cooper Pierce

- SCS Junior
- Rust Experience: mostly around LSP & Compiler implementations
- Links:
  - https://github.com/kopecs
- Dog:



What This Course **Is**

- An overview of Rust's features
- A selection of beginner to advanced Rust topics
- Up-to-date with current Rust
- A supplement for The Rust Book

What This Course **Is Not**

- Comprehensive
- Professional
- Beginner-level (programming-wise)
- A replacement for The Rust Book

Why Are We Teaching This?

- We think Rust is nice

Syllabus Stuff

- StuCo policy: Only 2 unexcused absences
  - That means we must take attendance
  - Just let us know via email/Discord if you expect to miss a class
- Participation: 50%
  - Just show up to class!
- Homework: 10%
  - Mini Rust puzzles
- Midterm: 5%
  - Just a project idea
- Final: 35%
  - Completing a medium-sized Rust project

Let's Begin!

Rust At A High Level

- C/C++ replacement
  - Similar syntax, concepts, performance
- Goals:
  - Great performance, easily
  - Reliable and safe (especially memory safe!)
  - Have lots of good tooling
- Incorporates great ideas from many languages

Rust At Large Companies

- Mozilla, AWS, Microsoft, Google, Facebook
  - https://research.mozilla.org/rust/
  - https://aws.amazon.com/blogs/opensource/aws-sponsorship-of-the-rust-project/
  - https://cloudblogs.microsoft.com/opensource/2021/02/08/microsoft-joins-rust-foundation/
  - https://opensource.googleblog.com/2021/02/google-joins-rust-foundation.html
  - https://engineering.fb.com/2021/04/29/developer-tools/rust/

Rust at Slightly Smaller Companies

- Dropbox, Figma, npm, Discord
  - https://dropbox.tech/infrastructure/rewriting-the-heart-of-our-sync-engine
  - https://www.figma.com/blog/rust-in-production-at-figma/
  - https://www.rust-lang.org/static/pdfs/Rust-npm-Whitepaper.pdf
  - https://discord.com/blog/why-discord-is-switching-from-go-to-rust

Rust Is No Longer A Research Language

How is Mozilla involved with Rust?

Mozilla was the first investor for Rust and continues to sponsor the work of the open source project. Mozilla also utilizes Rust in many of its core initiatives including Servo and key parts of Firefox.

We're really excited to announce that AWS is sponsoring the Rust programming language! Rust is designed for writing and maintaining fast, reliable, and efficient code. It has seen considerable uptake since its first stable release four years ago, with companies like Google, Microsoft, and Mozilla all using Rust. Rust has also seen lots of growth in AWS, with services such as Lambda, EC2, and S3 all choosing to use Rust in performance-sensitive components. We've even open sourced the Firecracker microVM project!

Additionally, we are forming a Rust team within Microsoft to contribute engineering efforts to the Rust ecosystem. We expect to be working with the Rust community on the compiler, core tooling, documentation, and more.

Rust Is No Longer A Research Language: Pt. 2

Today, some examples of projects where Google is either already using Rust or contributing to the Rust ecosystem include:

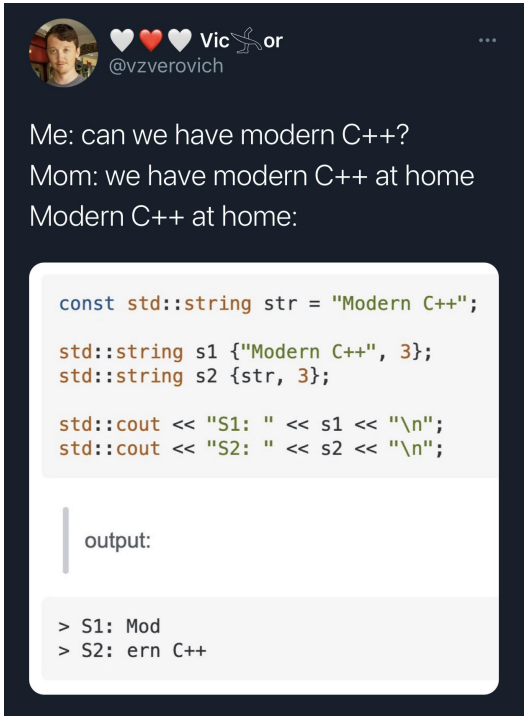
- Operating system modules in Android, including bluetooth and Keystore 2.0
- Low-level projects, such as the crosvm virtual machine monitor and drivers (alternative to QEMU) used in ChromeOS
- Contributing to open source projects that we use and use Rust, such as the Mercurial source code control system
- Firmware for FIDO security key support

Onward (2021 and beyond)

At the end of 2020, we re-upped our commitment by launching a Rust team in our Programming Languages organization, the same org responsible for Facebook's C++ standards work and toolchains.

Why is everybody so excited?

- C is a horrible language
  - zero memory safety
  - string manipulation is hell (strtok)
  - other footguns a-plenty
- C++ is a horrible language
  - inherits C's flaws
  - templates are hell
  - ABI compatibility => no room to undo mistakes
- Other attempts at replacements haven't seen widespread adoption from not enough tooling, features, or maintenance



Rust Goal: Great Performance, Easily

- C-like abstractions, close to hardware
- "Zero Cost Abstractions": paid at compile time, code is fast at runtime
- Backed by the LLVM compiler

Rust Goal: Reliable And Safe

- Ideally: "if it compiles, it works perfectly"
  - Not always true, but large classes of bugs excluded
- Lots of safe memory management
- Comprehensive standard library
- Testing is first-class feature

Rust Goal: Good Tooling

- Easy to install on any platform
- Great package manager
- Robust package ecosystem
- Linting, doc generation are first-class features

The Rust Compiler Is Your Friend

Catches your simple mistakes

- Dereferencing null pointers
  - It's impossible to have "pointers" that are "NULL" in safe Rust
- Forgetting to free/freeing memory twice
  - Free statements inserted when variable no longer accessible.
- Using null-terminated strings
  - All strings are length-checked :)

Simple Mistake 1: C Code

```
int f1(int *ptr) {
    return *ptr + 42;
}
```

Simple Mistake 1: C Code

```
int f1(int *ptr) {
    return *ptr + 42;
}
```

- Potential null pointer dereference!

Simple Mistake 1 Solution: Rust Code

```
fn f1(x: &i32) → i32 {
    *x + 42
}
```

- Rust references are guaranteed to be valid!
- Overflow in `+` checked when debugging, will 2's complement overflow during release builds

Simple Mistake 2: C Code

```
void s2(void) {
    int *ptr = malloc(sizeof *ptr);
    *ptr = 42;
    printf("%d\n", *ptr);
}
```

Simple Mistake 2: C Code

```
void s2(void) {
    int *ptr = malloc(sizeof *ptr);
    *ptr = 42;
    printf("%d\n", *ptr);
}
```

- Memory allocation could fail
- Memory not freed at end of function!

Simple Mistake 2 Solution: C++ code

```
void s2() {
    std::unique_ptr<int> ptr(new int(42));
    std::cout << *ptr << std::endl;
}
```

- Memory automatically freed now!
  - But what is that syntax?? ew disgusting

Simple Mistake 2 Solution: Rust Code

```
fn s2() {
    let ptr = Box::new(42);
    println!("{}", *ptr);
}
```

- So much less boilerplate!
- Memory still automatically freed

Simple Mistake 3: C Code

```
typedef struct { char *left; char *right; } pair;
pair split_mid(char *s, size_t len) {
    size_t mid = len / 2;
    pair p = { s, s + mid };
    return p;
}
```

Simple Mistake 3: C Code

```
typedef struct { char *left; char *right; } pair;
pair split_mid(char *s, size_t len) {
    size_t mid = len / 2;
    pair p = { s, s + mid };
    return p;
}
```

- p.left doesn't have a null terminator! Doesn't split in half like desired

Simple Mistake 3 Solution: C++ Code

```
std::tuple<std::string, std::string>
split_mid(std::string s) {
    size_t mid = s.length() / 2;
    std::string s1 = str.substr(0, mid);
    std::string s2 = str.substr(mid, len);
    return std::make_tuple(s1, s2);
}
```

Simple Mistake 3 Solution: Rust Code

```
fn split_mid(s: &str) → (&str, &str) {
    let mid = s.len() / 2;
    s.split_at(mid)
}
```

- Wow! Actual built-in string functions!
- Plus, no allocations, unlike the C++ code

Catches more complicated mistakes!

- Reference to stack variable living beyond scope of function: doesn't compile.
- Calling function that could fail without explicitly checking for success: doesn't compile.
- Multiple threads read/write to shared variable: must be behind mutex, or code doesn't compile.

Complex Mistake 1: C Code

```
int *c1(void) {
    int x = 42;
    return &x;
}
```

Complex Mistake 1: C Code

```
int *c1(void) {
    int x = 42;
    return &x;
}
```

- Returns pointer to local variable, will be invalid after function returns!

Complex Mistake 1: C++ code

```
std::unique_ptr<int> c1() {
    int x = 42;
    std::unique_ptr<int> ptr(&x);
    return ptr;
}
```

- No warnings when compiled with `-Wall -Wextra`??

Complex Mistake 1 Solution: Rust Code

```
fn c1() → &i32 {
    let x = 42;
    return &x;
}
```

```
error[E0515]: cannot return reference to local variable `x`
 --> src/lib.rs:3:12
  |
3 |     return &x;
  |            ^^ returns a reference to data owned by the current function
```

Complex Mistake 2: C Code

```
void c2(void) {
    FILE *f = fopen("output.txt", "w");
    fprintf(f, "Hello World!\n");
    fclose(f);
}
```

Complex Mistake 2: C Code

```
void c2(void) {
    FILE *f = fopen("output.txt", "w");
    fprintf(f, "Hello World!\n");
    fclose(f);
}
```

- `fopen`, `fprintf` could fail, not checked

Complex Mistake 2 Solution: Rust Code

```
use std::io::Write;

fn c2() → std::io::Result<()> {
    let mut f = std::fs::File::create("output.txt")?;
    writeln!(&mut f, "Hello World!")?;
    Ok(())
}
```

- `?`: if error is returned, return early (compiler warning if not present)
- File automatically closed at end of scope!

Complex Mistake 3: C++ Code

```
void c3(std::mutex m, std::shared_ptr<size_t> bal,
size_t amt) {
    m.lock();
    *bal += amt;
    m.unlock();
}
```

- Must remember to lock/unlock manually!

Complex Mistake 3 Solution: Rust Code

```
use std::sync::{Arc, Mutex};
fn c3(bal: Arc<Mutex<usize>>, amt: usize) {
    let mut bal = bal.lock().unwrap();
    *bal += amt;
}
```

- Mutex unlocked automatically at end of scope
- Mutex is part of type => *need* to lock in order to change data

Cargo is your friend

Friendly Things Cargo Does

- Formats/lints your code!
- Installs packages from specification!
- Builds your code in debug and release profiles!
- Makes sick af documentation!

Rustup is your friend

Friendly Things Rustup Does

- Makes it easy to install Rust! (https://rustup.rs)
- Makes it easy to select version per-project! (using rust-toolchain.toml)

Homework

Install Rust!

- Example time (follow along if you want)

Next Week: Basic Rust Concepts

- Homework is at https://github.com/Rust-Stuco/puzzles

Course Discord: https://discord.gg/pmuQphb6Xm