CIS 1905

Lecture 1: Hello!

Introductions and Rust Basics

About Us

Joy

- Junior in M&T
- Interested in systems, art, and devops

Favorite crate: tokio (love hate relationship)

Phillip

- PhD student
- Databases, programming languages

Favorite crate: egg

Thomas (S)

- Junior in DMD
- #1 Rust fan if it can compile
- Graphics programming pilled
- Instructing JS class (1962)

Favorite crate: tauri

Alexander

- Freshman in CIS/Physics
- Rust fnatic
- Doesn’t like python (slow)
- Can’t finish a project

Favorite crate: wgpu

Wait a minute, who are you?

You

- Name
- Year/Major

Pick One

- What's the last bug you had to fix?
- What's the most memorable bug you had to fix?
- Favorite Philly restaurant/food?

Logistics

Grade Breakdown

- 50% Homework
- 40% Final Project
- 10% Participation

Ed for questions

Office Hours

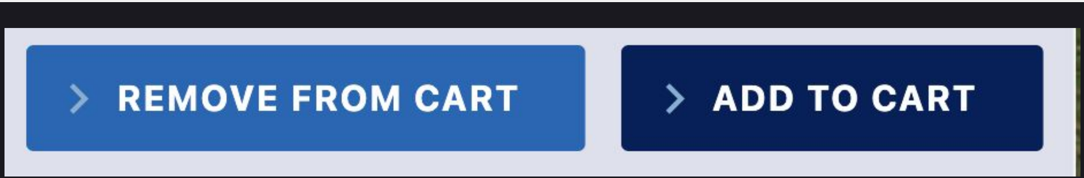
- Time 1
- Time 2

Aside: Rust Book

- Free online material to learn Rust!
- CIS 1905 <> The Book
  - Mirroring chapters (posted on websites), with more **practical applications** and **external crates** (what Rust calls libraries)
- Two modes of learning:
  - ✨**Read corresponding chapters, come to lecture with questions**
  - Come to lecture as pre-learning material for book
  - Recommend 1st to save time for you
- Schedule ramps up and order rearranged to Penn schedule

Aside: Rust Book

- Free online material to learn Rust!
- CIS 1905 <> The Book
  - Corresponding Chapter(s) on website
- Schedule ramps up and order rearranged to Penn schedule



Brb, lemme just drop this class and read the Book

~~Brb,~~ ~~op~~
~~this~~ ~~the~~
~~Boo~~

Homework

- Released on GitHub
- Submitted on Gradescope
- Due **Sunday Nights** at 11:59pm ET
- Homeworks can be submitted late, up to the start of the next class, for a 10% penalty per day late.
- Individual Assignments
    - Exception of HW3 Tetris Tournament and Final project where you can work in groups of 1-2

Us Rn

Rust Time

What is Rust

Rust is a language

1) run at **compile time**
2) enforces **memory safety** without automatic memory management.

C++, - - valgrind/gdb.

Is Rust beginner friendly?

Systems Programming

Functional

- Immutable by default
- Pattern Matching
- Strong Type System
    - Algebraic types (enums)
    - HOF
- No Null, Option/Result
- Monomorphization

Imperative

- Mutable State (mut keyword)
- Procedural Programming and Control Flow
    - Loops, conditionals
- Imperative Error Handling
    - panic!()
- Object-Oriented Features
    - Struct, Traits

Rust Ecosystem

Cargo: Rust’s build system and package manager.

- `cargo new, cargo build, cargo run, cargo test`

Crates: Rust binary / library

- Use in code
- Use locally: `cargo install`
  - github.com/rust-unofficial/awesome-rust

crates.io

Live Coding

Basic Syntax

Variable mutability: `let mut x = 5;`

Type annotations (optional, inferred)

- Primitives: integers (e.g. `usize`, `u32`, `i32`), floats, `bool`, `char`
- Tuples: `let tup: (i32, f64, u8) = (500, 6.4, 1);`
- Arrays: fixed length
  - `let a = [1, 2, 3, 4, 5];`
- Vectors: variable length (Python list, C++ std::vector, Java ArrayList)

```
let mut v1: Vec<i32> = Vec::new();
v1.push(1);
v1.push(2);
v1.push(3);
let v2 = vec![1, 2, 3];
```

Basic Syntax

Variable mutability with `mut` `mut` `mut` `mut` `mut` `mut` keyword

Type annotations (optional, inferred)

- Primitives: integers (e.g. `usize`, `u32`, `i32`), floats, `bool`, `char`
- Tuples: `let tup: (i32, f64, u8) = (500, 6.4, 1);`
- Arrays: fixed length
    - `let a = [1, 2, 3, 4, 5];`
- Vectors: variable length (Python list, C++ std::vector, Java ArrayList)

```
let v1: Vec<i32> = Vec::new();
v1.push(1);
v2.push(2);
v3.push(3);
let v2 = vec![1, 2, 3];
```

Basic Syntax

Variable mutability with **`mut`** keyword

Type annotations (optional, inferred)

- Primitives: integers (e.g. **`usize`**, **`u32`**, **`i32`**), floats, bool, char
- Tuples: **`let tup: (i32, f64, u8) = (500, 6.4, 1);`**
- Arrays: fixed length
  - **`let a = [1, 2, 3, 4, 5];`**
- Vectors: variable length (Python list, C++ std::vector, Java ArrayList)

```
let v1: Vec<i32> = Vec::new();
v1.push(1);
v2.push(2);
v3.push(3);
let v2 = vec![1, 2, 3];
```

Functions

Defined with `fn` keyword

Return keyword is optional if you omit `;`. See example:

```
fn five() -> i32 {
    5
}
```

Make sure you omit `;` at the end if you are using the simplified return!

Control Flow

`if` statement

`loop`

- Loop labels

`while`

Pattern Matching

In Rust, you can pattern match anything with a `match` statement:

```
fn is_palindrome(items: &[char]) -> bool {
    match items {
        [first, middle @ .., last] => first == last &&
is_palindrome(middle),
        [] | [_] => true,
    }
}
```

Closures

Rust Anonymous Functions that capture their environment in 3 ways:

- 1) borrowing immutably, 2) borrowing mutably, 3) taking ownership

```
let expensive_closure = |num: u32| -> u32 {
    println!("calculating slowly...");
    thread::sleep(Duration::from_secs(2));
    num
};
```

What is the environment being captured?

Custom Data Structures

Struct

Struct is a custom data type that lets you package together and name multiple related values that make up a meaningful group. If you’re familiar with an object-oriented language, a struct is like an object’s data attributes.

- Struct vs “Class”
- Impl

Enum

Enums allow you to define a type by enumerating its possible variants.

Classic Enum

```
enum TrafficLight {
    Red,
    Yellow,
    Green,
}
```

Enum with Data

```
enum IpAddr {
    V4(String),
    V6(String),
 }
```

Enum with Data

```
enum GameAction {
    Quit,
    Click { x: i32, y: i32 },
    Write(String),
    ChangeIconColor(i32, i32,
i32),
}
```

Pattern Matching Enums

```rust
fn value_in_cents(coin: Coin) -> u8 {
    match coin {
        Coin::Penny => 1,
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter(state) => {
            println!("State quarter from {:?}!", state);
            25
        }
    }
}
```

Rust Type System

Monomorphization: different types are created from polymorphic code

In C++ and Java, generic types are a meta-programming construct for the compiler: vector<int> and vector<char> in C++ are two different copies of the same boilerplate code for a vector type (known as a template) with two different types filled in.

In Rust, a generic type parameter creates what is known in functional languages as a “type class constraint”, and each different parameter filled in by an end user actually changes the type. In other words, Vec<isize> and Vec<char> are two different types, which are recognized as distinct by all parts of the type system.

Infinite Iterator

play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=b077bba91348l823d348d6a4ee64df46

HW1 TBReleased

Due 02/04

tinyurl.com/cis1905-logistics