ECE 459
Programming for Performance

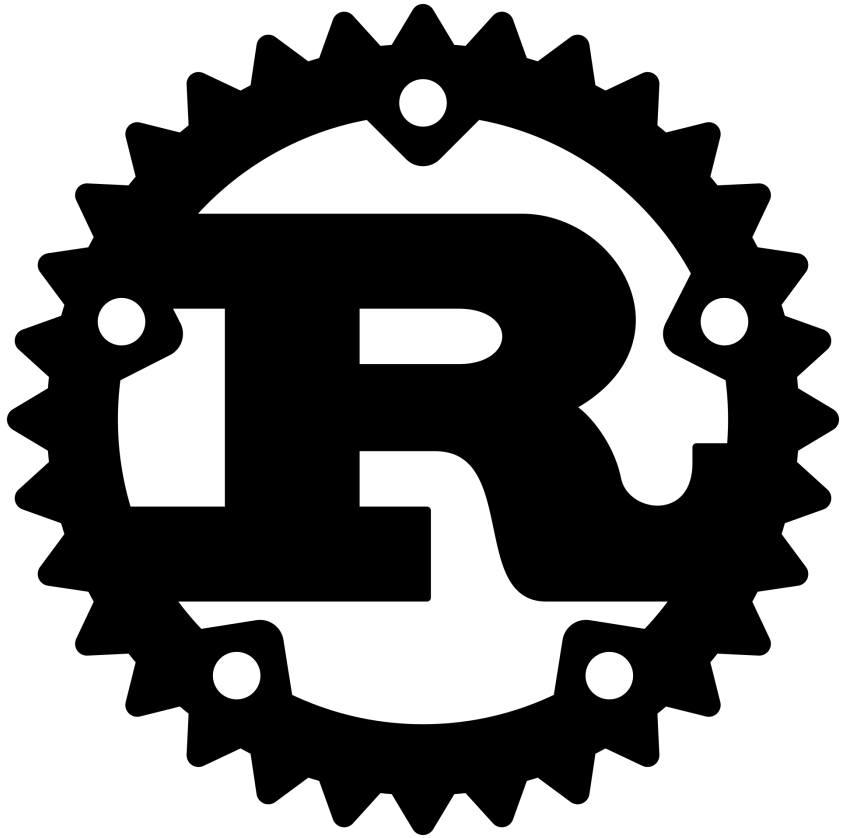
Lecture 2 Rust Basics

Winter 2023

Huanyi Chen
huanyi.chen@uwaterloo.ca

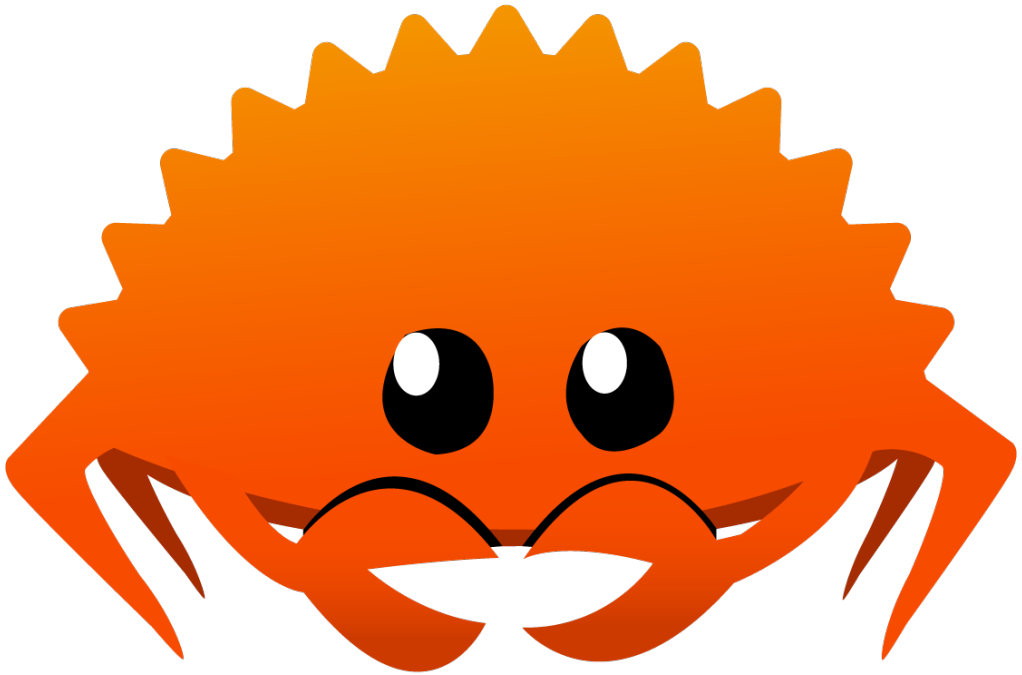
UNIVERSITY OF WATERLOO

Getting Started with Rust



https://en.wikipedia.org/wiki/Rust_(programming_language)

Getting Started with Rust



Rustacean

Crustacean

Learning Rust

- The official Rust book is a must-read
- https://doc.rust-lang.org/book/title-page.html

- There bunch of additional books on the official website
- https://www.rust-lang.org/learn

Rust in ECE459

- Focus on important features in the course
- Understand why & how they work towards the goal of programming for performance

Roadmap for today

- Some basic concepts of Rust
- In-class exercises to get you familiar with Rust

- Remaining time can be used for
  - Setting up your Rust environment or GitLab
  - Notes reading
  - Q&A
  - Self-learning

The Goal of Rust

- Avoid issues with

- Memory allocation
- Concurrency

Memory management

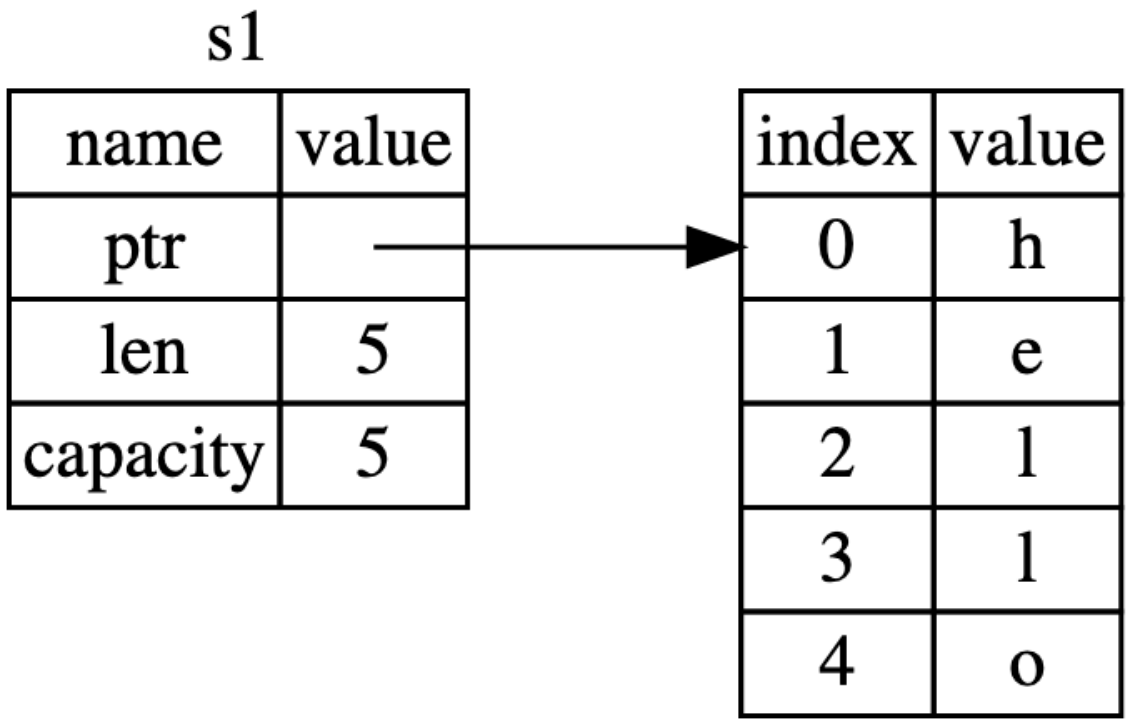
- In C, malloc / free
- In Java, Garbage collection
- In C++, Resource acquisition is initialization (RAII)

Memory management

- What about Rust?
  - Same as C++, RAII
  - But at compile time (!)

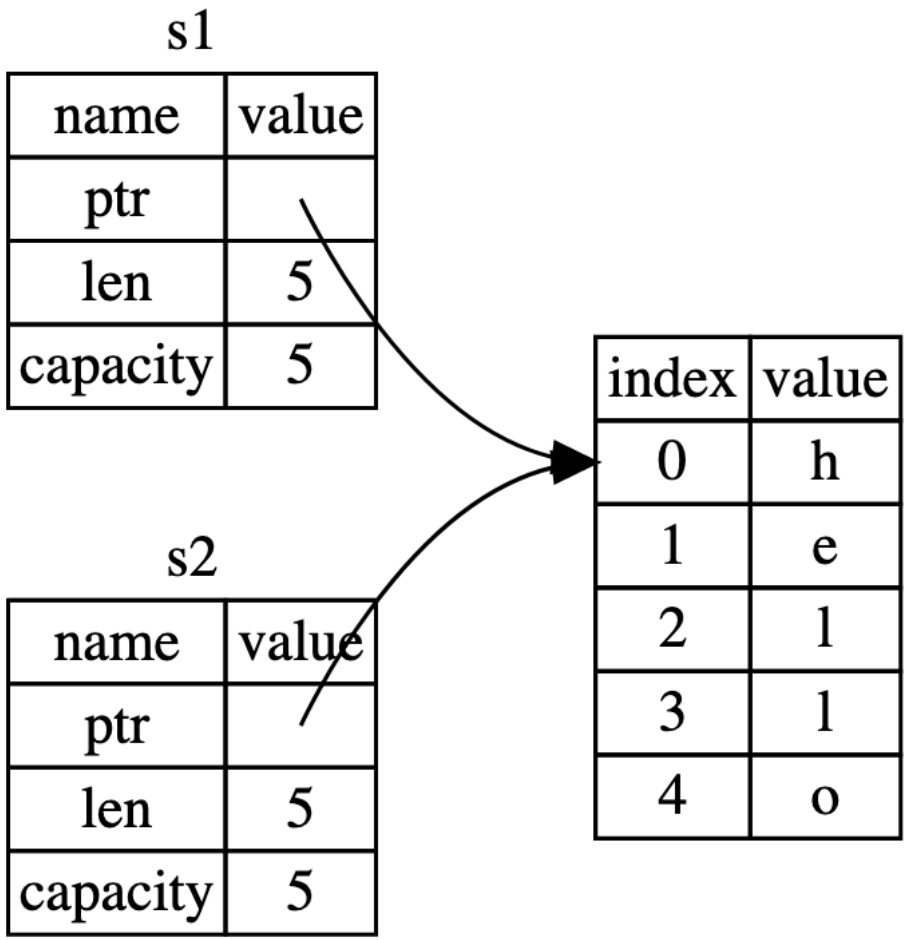
Stack vs. Heap

- String
- A stack part
- A heap part



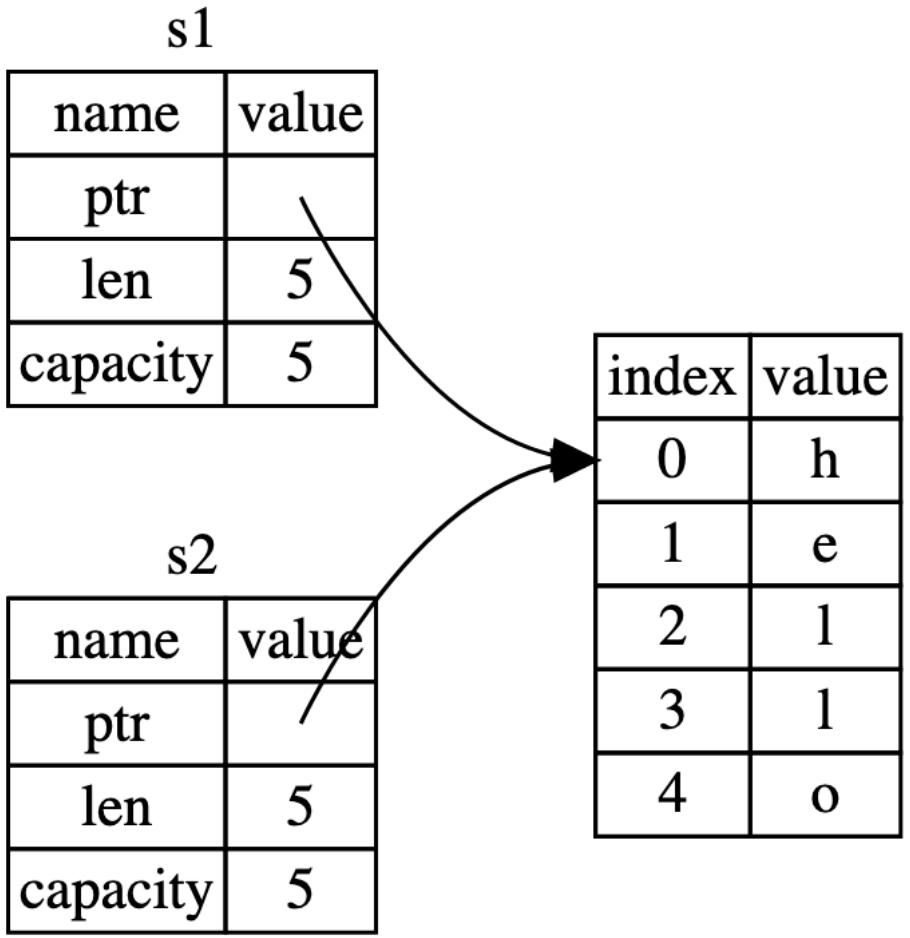
C/C++

- Can have multiple stack part



Potential issues

- Double free
- Use-after-free
- Content can be changed by both s1 and s2
  - single-thread: it's ok if you are careful
  - multi-thread: concurrency issues



Concurrency vs. Parallelism

- Use single-core CPU to do multi-threading

- Concurrent?
- Parallelism?

Ownership!

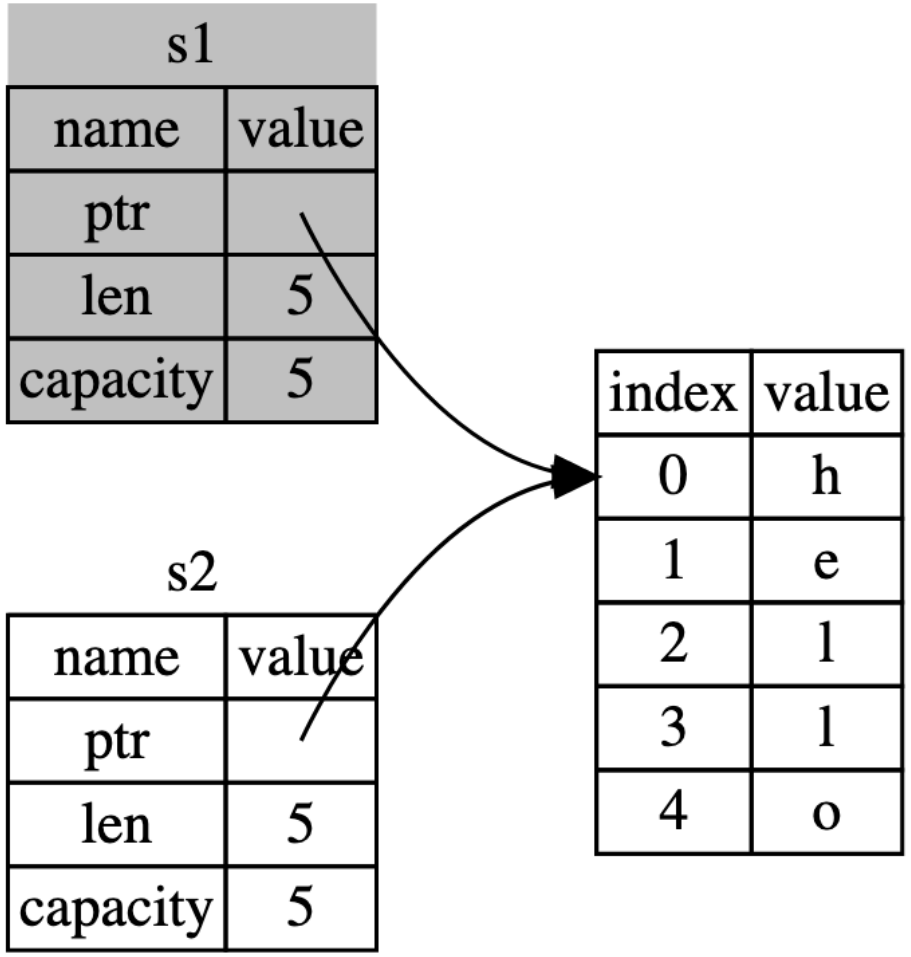
- In Rust, all magic is behind the concept: ownership

1. Every value has a variable that is its owner (stack)
2. Only one owner at a time
3. When the owner goes out of scope, the value (heap) is dropped

- Easy to understand the first and third rule
- But the second tells there exists the concept of **transferring ownership**

Ownership

- If s2 is created, s1 is no longer valid
- Compile-time error occurs if you want to access s1 after assigning it to s2



Move / Copy

- Transferring ownership follows the **Move** semantics
- When we have both stack part and heap part

- If only with stack part, we do **Copy**
  - Integers
  - Booleans
  - Floats
  - etc.

Copy

```
fn main() {
    let x = 5;
    let y = x;
    // x and y both are valid
}
```

Clone

- What if we want to make a deep copy of variables with both stack and heap?

- Then we need to explicitly do **Clone**
  - `clone()`

Other things

- Immutable / mutable
  - Variables are immutable by default
  - Use `mut` to explicitly create a mutable variable
- Semicolons
  - Separate expressions
- Shadowing
  - Reuse the original name but they are actually two variables

Live-coding

- Ownership
- Immutable / mutable
- Function
- Struct

In-class exercise

- Used to reinforce your understanding
- Not just about completion
- It becomes meaningless to complete without critical thinking

In-class exercise

q1

Create a string containing "hello", then create a function called `modify`, which will append a " world" to the string passed to it

q2

Create a `struct User` which contains two fields, `name: String`, and `lottomax: [u32; 7]`, and a `buy()` function.

The main() will create an instance of the User struct and call its buy() to buy a ticket and save it into itself. Next, it will then call a `check()` function which is implemented outside the struct and main() to check if the ticket wins.

If check() returns `true`, print a congratulation message, if not, print other message you name it.

In-class exercise

q3

- Follow the chapter 2 of the Rust book
- Build the guessing game

Remaining time

- Setup your GitLab environment if you haven't done so
- You can create a repo called "**ece459-practice**" and push your code there
- You can add me as a member if you want
- Q & A

Prepare for the next lecture

- Read the lecture 3 notes
- Read relevant chapters of the Rust book (https://doc.rust-lang.org/stable/book/)
- Bring your laptops and we will do some in-class exercises