
# Lecture 9

## Struct Methods in Rust

Slides: Neil Kaushikkar

# Goals For Today


- Answering Your Questions
- Review Vectors and Dereferencing
- Review Ownership & Borrowing
- Struct Methods
- Introduction to Traits
- Introduce MP2

# Reminders


- MP1 due TOMORROW 2/23 at 11:59 pm CT
- HW5 due TOMORROW 2/23 at 11:59 pm CT
- HW6 due 2/24 at 11:59 pm CT
- HW7 releasing tonight due 3/1 at 11:59 pm CT
- MP2 releasing tonight due 3/4 at 11:59pm CT

# Answering Your Questions!


- “I understand that this is a pretty independent class but I feel as if the lectures vs the homework are two completely different topics. I have never felt as if I understood how to do an assignment in this class without having to google each and every component. While this is useful to learn, I doubt it is the goal of the course.”
  - Reach out to us on Discord, on the forum or privately
  - Join office hours for clarification and help

# Answering Your Questions!


- “I feel like I'm not picking up rust quickly and using it isn't super intuitive to me so these quick hw's are taking me probably a lot longer than they should with all the testing and error fixing stuff I'm doing. ”

# Answering Your Questions!


- “Could you talk about different ways to navigate vectors?”
  - **.iter()** - get each element of the vector as a reference
  - **.iter_mut()** - get each element of the vector as a mutable reference
  - **0..v.len()** - get the index range and access elements by index

# Answering Your Questions!


- “How many programmers does it take to change a light bulb?”
  - None, that's a hardware problem!
- “There are 10 types of people, those that understand binary and those that don’t”

# Ownership & Borrowing Review

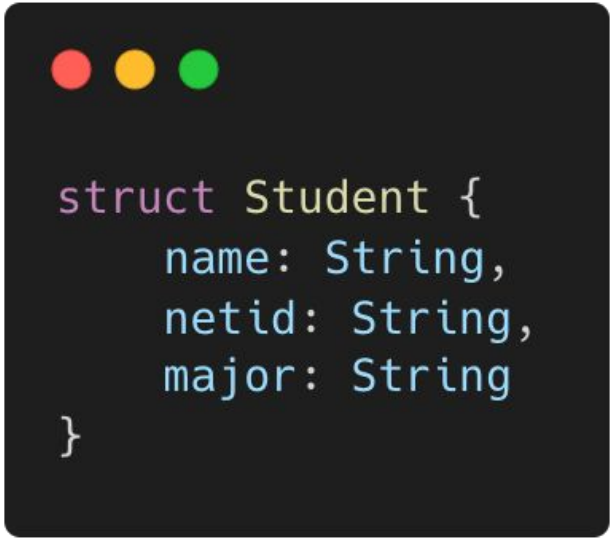
- Each value in Rust has a variable that's called its owner
- There can only be one owner at a time
- When the owner goes out of scope, the value will be dropped
- We can create references that borrow the data from the original owner while the owner retains the data

```rust
let v: Vec<u8> = vec![1, 2, 3, 4];

print_vector(&v);
duplicate(&mut v);

println!("{:?}", v);

// --- snip ---

fn print_vector(x: &Vec<u8>) { /* code here */ }
fn duplicate(x: &mut Vec<u8>) { /* code here */ }
```

Reference:

- https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html

# Structs Review



```
struct Student {
    name: String,
    netid: String,
    major: String
}
```

# Struct Methods


- Very similar to functions:
  - Use the **fn** keyword
  - Contain code that is run somewhere else
- However, they are defined in the context of a **struct** (or an **enum** or **trait**)
  - Use the **impl** keyword to implement methods for your custom type
  - The first parameter is always **self**, which represents the instance of the struct the method is being called on


# Struct Methods

# The self Keyword


- **&self** - IMMUTABLE borrow to the current instance
- **&mut self** - MUTABLE borrow to the current instance
- **self** - take ownership of the struct and discard after the method
    - This is rarely, if ever, used

- Use dot notation on self to access **struct** fields and call other **struct** methods within **struct** methods

# Associated Struct Functions


- Associated functions are defined on types but don't refer to instances of the type
- Rust does not have built-in constructors, but you can define your own
  - **String::new()**
  - **Vec::new()**
  - **Student::new()**


# Associated Struct Functions

# Deriving Struct Functionality


- **Debug** - make your custom type printable using **{:?}**
- **Clone** - allows you to easily make deep copies of your **struct** using **.clone()**
- **PartialEq** - allows you to use **==** on your **struct**
  - All fields of the struct must be equal for **==** to return **true**
  - All fields of the struct must implement **PartialEq** to enable this
- **Default** - create a **struct** with all fields set to their respective default values
  - All fields of the struct must implement **Default** to enable this

- Place **#[derive(...<traits here>...)]** on top of your custom type declaration


# Deriving Struct Traits

# Sneak Peak: Traits


- Traits define an interface for common behavior
  - **Debug**, **Clone**, **PartialEq**, **Default**, and many more…
- Traits are Rust's version of Object Oriented Programming
  - We'll go more into detail in the special topics section later in the course


# Introduce MP2


# That’s All Folks!