Smart Pointers

Ryan Eberhardt and Armin Namavari

April 23, 2020

The Plan for Today

- Review Box<T>
- Introduce Rc<T>
- Introduce RefCell<T>

Please ask Questions!

- Or else I will happily blast through the slides
- Feel free to unmute yourself
- Ryan: “At the end of the quarter, I’ll randomly select at least three people that participated 10 times, and I’ll make you a custom mug (see @paintedpeas) if you’re still around campus once I can access a ceramics studio again. Asking or answering a question in lecture (out loud, or in the chat) or on Slack all count as participation.”

Box<T>

- You’ve seen this already in the context of LinkedList
- Have a unique pointer to a chunk of heap memory
- What are some limitations of Box<T>?

Rc<T>

- What if I want to have multiple pointers to the same chunk of heap memory?
- Recall borrowing rules: can have multiple immutable references OR at most one mutable reference.
- Rc<T> lets you have multiple **immutable references** to a chunk of heap memory (i.e. we can’t modify this chunk of memory)
    - Why do we need this?
    - A: Rust’s borrow checking rules!
- Caution: you can get memory leaks if you create reference cycles! (if you need reference cycles, you need to throw other smart pointer types into the mix)

Example: Adding Multiple Views to Our List

- What if we want to be able to have our linked lists “intersect” one another so that they can share certain parts while the data structure is immutable? (this is a paradigm common in functional data structures)
- This can let us see into the “history” of our data structure!
- These are sometimes known as persistent data structures
- Playground example
  - Start
  - End

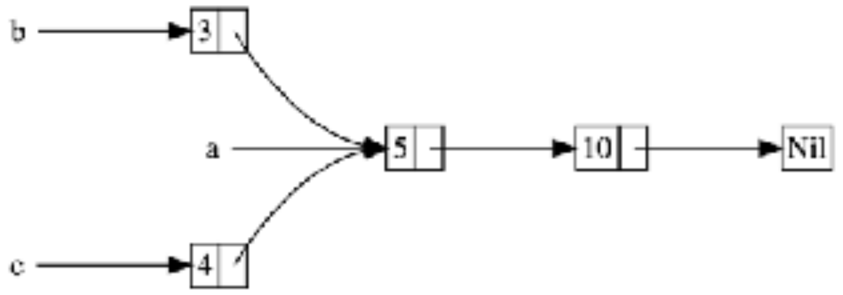

Image: https://doc.rust-lang.org/book/ch15-04-rc.html

RefCell<T>

- RefCell let’s you “lie” to the compiler by providing interior mutability
- That is, you can have shared references to the cell, but you can mutate what’s inside of it!
- Its new function **doesn’t** heap allocate, here are the things that do.
- This is still safe because it will enforce the reference rules at runtime (but this is now an additional cost)
- (try_)borrow/borrow_mut
- Common pattern: Rc<RefCell<T>>
  - You will often see this in fancier data structures that have multiple pointers pointing to the same piece of data, which might have to support mutability

Additional Reading

- The Rust book on Rc
- The Rust book on RefCell
- CS242 on Smart Pointers (this will show you how Box and Rc are implemented under the hood!)
- Quora thread about applications of persistent data structures (e.g. version control, optimizing React applications)
    - Concurrency