How to pass one Vec to multiple functions in Rust?

Ask Question

Asked 4 years, 9 months ago
Active 1 year, 11 months ago
Viewed 2k times

5

1

I wrote a `max` function which takes a `Vec` as a parameter. It works as I expected. Then I added a `min` function the same as the `max` function:

```
fn main() {
    let my_array = vec![61, 14, 71, 23, 42, 8, 13, 66];
    let max = max(my_array);
    let min = min(my_array);
    println!("Max value is {}.", max);
}

fn max(array: Vec<i32>) -> i32 {
    let mut max = array[0];
    for val in array {
        if max < val {
            max = val;
        }
    }
    max
}

fn min(array: Vec<i32>) -> i32 {
    let mut min = array[0];
    for val in array {
        if min > val {
            min = val;
        }
    }
    min
}
```

Rust reports an error if I put the same `my_array` parameter on the call to `min`:

```
error[E0382]: use of moved value: `my_array`
 --> src/main.rs:4:19
  |
2 |     let my_array = vec![61, 14, 71, 23, 42, 8, 13, 66];
  |         -------- move occurs because `my_array` has type `std::vec::Vec<i32>`, which does not implement the `Copy` trait
3 |     let max = max(my_array);
  |               -------- value moved here
4 |     let min = min(my_array);
  |               ^^^^^^^^ value used here after move
```

How can I write code that works?

vector rust ownership

Share
Improve this question
Follow

edited Jan 6 '20 at 15:22

Shepmaster
305k ● 59 ● 824 ● 1083

asked Mar 22 '17 at 14:06

tajihiro
1,873 ● 4 ● 27 ● 44

4

You are most advised to read about move semantics.
– E_net4 the flagger
Mar 22 '17 at 14:19

Add a comment

1 Answer

Active Oldest Votes

1

This is an issue that very beginners to Rust will experience. As a beginner, you should read *The Rust Programming Language*. A lot of effort has been put into this book, especially for newcomers to Rust. This will cover many things that you will run into.

Relevant sections:

- Data Types
- Understanding Ownership
- References and borrowing

The underlying problem is that you've *transferred ownership* of the vector when you call `max`. The value is then gone; `main` no longer has it.

The simplest thing to to is to clone the vector before passing to `max`. This allows `main` to keep ownership of `my_array` and then transfer ownership to `min` on the subsequent line:

```
let max = max(my_array.clone());
let min = min(my_array);
```

This is inefficient, as neither `max` nor `min` need to take ownership of the vector to do their work. Cloning the `Vec` also requires additional memory allocation. It's more idiomatic to pass in a *slice*, which is a type of reference to the data inside the `Vec`:

```
let max = max(&my_array);
let min = min(&my_array);


fn max(array: &[i32]) -> i32 {
    let mut max = array[0];
    for &val in array {
        if max < val {
            max = val;
        }
    }
    max
}
```

When iterating over a slice, you get back references to the items in the slice. With integers, we can dereference them (here using the `&` in `for &val in array`) and make a copy of the value.

See also:

- Why is it discouraged to accept a reference to a String (&String), Vec (&Vec), or Box (&Box) as a function argument?

Even better, there's no need to rewrite basic functions like this. You also **assume there's always at least one value**, which isn't true for an empty vector. The idiomatic solution is to use iterators:

```
fn main() {
    let my_array = vec![61, 14, 71, 23, 42, 8, 13, 66];
    let max = my_array.iter().max();
    let min = my_array.iter().min();
    println!("Max value is {:?}.", max);
    println!("Min value is {:?}.", min);
}
```

This uses `Iterator::min` and `Iterator::max`, which each return an `Option`, as an empty slice has no minimum or maximum value.

Technically, it's a little different from your original solution, as `min` and `max` are `Option<&i32>`; a reference to the original slice. You can get back to `Option<i32>` by using `Option::copied`:

```
fn main() {
    let my_array = vec![61, 14, 71, 23, 42, 8, 13, 66];
    let max = my_array.iter().max().copied();
    let min = my_array.iter().min().copied();
    println!("Max value is {:?}.", max);
    println!("Min value is {:?}.", min);
}
```

Bonus information: slices, `Vec`s, and arrays are all different types. It's not correct to refer to `my_array` as an array at all.

Share
Improve this answer
Follow
edited Jan 6 '20 at 15:26

answered Mar 22 '17 at 14:22

Shepmaster
305k ● 59 ● 824 ● 1083

Add a comment

Your Answer

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged vector rust ownership or ask your own question.

The Overflow Blog

- Sequencing your DNA with a USB dongle and open source code
- Don't push that button: Exploring the software that flies SpaceX rockets and...

Featured on Meta

- Providing a JavaScript API for userscripts
- Congratulations to the 59 sites that just left Beta

Linked

Why is it discouraged to accept a reference to a String (&String), Vec (&Vec), or Box (&Box) as a function argument?

Related

336
How do I print the type of a variable in Rust?

Cannot move out of borrowed content when borrowing a generic type

How to convert C variable-length array code to Rust?

How do I get a slice of a Vec<T> in Rust?

Convert Vec<String> into a slice of &str in Rust?

Store data that implements a trait in a vector

Pass vector of mutable trait objects to function

1
Rust: how to use await in function chain

Hot Network Questions

- Who (or what) created the atropal?
- Does saying "Keep it up" put me in an authoritative position?
- Why is my reasoning incorrect - probability?
- Would I be able to avoid the wash sale rule if I buy back the security on January 1st after selling it on December 31st?
- Text overflow in tabularx

more hot questions

Question feed