Collectives™ on Stack Overflow

Find centralized, trusted content and collaborate around the technologies you use most.

Learn more about Collectives

Stack Internal

Knowledge at work

Bring the best of human thought and AI automation together at your work.

Explore Stack Internal

How do I implement one of the std::ops:{Add, Sub, Mul, Div} operators without moving out the the arguments?

Ask Question

Asked 7 years, 3 months ago

Modified 7 years, 3 months ago

Viewed 3k times

2

I'm writing a ray-tracer and I want to be able to subtract my 3D vectors:

```
use std::ops::Sub;

#[derive(Clone, Debug)]
pub struct Vec3 {
   pub v: [f64; 3],
}

impl Sub for Vec3 {
   type Output = Vec3;

   fn sub(self, other: Vec3) -> Vec3 {
       Vec3 {
           v: [
               self.v[0] - other.v[0],
               self.v[1] - other.v[1],
               self.v[2] - other.v[2],
           ],
       }
   }
}
```

This seems to work. However, when I try to use it:

```
fn main() {
   let x = Vec3 { v: [0., 0., 0.] };
   let y = Vec3 { v: [0., 0., 0.] };
   let a = x - y;
   let b = x - y;
}
```

I get complaints from the compiler:

```
error[E0382]: use of moved value: `x`
 --> src/main.rs:26:13
  |
25 |     let a = x - y;
  |             - value moved here
26 |     let b = x - y;
  |             ^ value used here after move
  |
  = note: move occurs because `x` has type `Vec3`, which does not implement the `Copy` trait

error[E0382]: use of moved value: `y`
 --> src/main.rs:26:17
  |
25 |     let a = x - y;
  |                 - value moved here
26 |     let b = x - y;
  |                 ^ value used here after move
  |
  = note: move occurs because `y` has type `Vec3`, which does not implement the `Copy` trait
```

How can I write the subtraction operator so that the code above works?

Please don't tell me I should an existing 3D math module. I'm sure there's something better, but I'm after learning how to do it myself to learn the language.

How do I implement the Add trait for a reference to a struct? doesn't help as it requires specifying lifetimes for object which I'm not at yet.

- rust

Share

Improve this question

Follow

edited Aug 15, 2018 at 2:34

Shepmaster

439k116116 gold badges1.3k1.3k silver badges1.5k1.5k bronze badges

asked Aug 14, 2018 at 15:17

Jeffrey

11.1k11 gold badge2626 silver badges4646 bronze badges

4

1

Possible duplicate of How do I implement the Add trait for a reference to a struct?

mcarton – mcarton

2018-08-14 15:28:03 +00:00

Commented Aug 14, 2018 at 15:28

@trentcl thanks!! Consider making this an answer. I'm still processing the other options, but since I don't know yet what a lifetime is, I'm having difficulty comparing them. Your suggestion fixes my issue, but I'm not sure the copy is the correct trade-off.

Jeffrey – Jeffrey

2018-08-14 15:38:11 +00:00

Commented Aug 14, 2018 at 15:38

@Jeffrey The `Copy` marker permits to silently copy your struct when needed.

Boiethios – Boiethios

2018-08-14 15:44:08 +00:00

Commented Aug 14, 2018 at 15:44

I don't think the last paragraph is a legitimate constraint to put on a question. Would you ask your driving teacher "How do I parallel park? Oh, but I can't turn the wheel, because I'm not at *steering* yet." Fortunately, in this case lifetimes are not required (witness my answer), and the linked question does not address the question as posed anyway (you would have to use `&x - &y` instead of `x - y`).

trent – trent

2018-08-15 13:33:36 +00:00

Commented Aug 15, 2018 at 13:33

Add a comment |

1 Answer

Sorted by: Reset to default

Highest score (default) Trending (recent votes count more) Date modified (newest first) Date created (oldest first)

6

In the example, the compiler tells you why `x` has been invalidated by the move:

```
   = note: move occurs because `x` has type `Vec3`, which does not implement the `Copy` trait
```

In this case, you can simply add `#[derive(Copy)]` to give `Vec3` *copy semantics*:

```
#[derive(Clone, Copy, Debug)]
pub struct Vec3 {
   pub v: [f64; 3],
}
```

`Copy` is a marker trait that indicates to the compiler that values of a type do not become invalid when they are moved from. A type with this property is said to have copy semantics, while a type that does not implement `Copy` is said to have move semantics. Is it possible to make a type only movable and not copyable? and How does Rust provide move semantics? explain this concept in more detail.

However, you can only implement `Copy` for types that contain only other `Copy` types. If `Vec3` actually held a `Vec` inside it, the compiler would not let you implement `Copy` for it. Fortunately, references do implement `Copy`, so you can instead implement `Sub` for a *reference* to `Vec3`, using the approach described in How do I implement the Add trait for a reference to a struct?

Share

Improve this answer

Follow

answered Aug 14, 2018 at 16:07

trent

28.6k1010 gold badges6363 silver badges100100 bronze badges

Sign up to request clarification or add additional context in comments.

Comments

Add a comment

Your Answer

Thanks for contributing an answer to Stack Overflow!

- Please be sure to *answer the question*. Provide details and share your research!

But *avoid* …

- Asking for help, clarification, or responding to other answers.
- Making statements based on opinion; back them up with references or personal experience.

To learn more, see our tips on writing great answers.

Sign up or log in

Sign up using Google

Sign up using Email and Password

Submit

Post as a guest

Name

Email

Required, but never shown

Post as a guest

Name

Email

Required, but never shown

Post Your Answer Discard

By clicking “Post Your Answer”, you agree to our terms of service and acknowledge you have read our privacy policy.

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

- rust

See similar questions with these tags.

- The Overflow Blog
  - Introducing Stack Overflow AI Assist—a tool for the modern developer
  - Treating your agents like microservices
- Featured on Meta
  - Chat room owners can now establish room guidelines
  - AI Assist is now available on Stack Overflow
  - Policy: Generative AI (e.g., ChatGPT) is banned

Linked

80
How does Rust provide move semantics?
101
Is it possible to make a type only movable and not copyable?
98
How do I implement the Add trait for a reference to a struct?

Related

21
Can we create custom Rust operators?
15
How can I overload the += "plus equals" operator?
34
How do I require a generic type implement an operation like Add, Sub, Mul, or Div in a generic function?
14
Overloading the Add-operator without copying the operands
3
How can I implement an operator like Add for a reference type so that I can add more than two values at once?
0
How to implement arbitrary add operator in Rust?
0
How to overloading operator without using ops::Add trait?
2
Generic math in Rust without Copy (std::ops::Add, etc.)
3
Overloading of addition for arrays in Rust
4
Implementing Add, Sub, Mul, Div for all combinations of move and borrow

- Looking for a proper formal substitute for "quick fix" for formal letter
- I2C bus problems when using two distance sensors in Arduino
- What's the point of swapping polarity between mod slots?
- PSE Advent Calendar 2025 (Day 5): Cinnamon overdose
- Proportional odds logistic regression for ordered category outcome - how to convert odds ratios to probabilities in this case?
- Trying to understand UK tax brackets for salary above 150k
- How to use Bézier curves in `TikZ` to draw a quotation mark?
- Correct website for Thailand ETA application form

more hot questions

Question feed

Subscribe to RSS

Question feed

To subscribe to this RSS feed, copy and paste this URL into your RSS reader.

Stack Overflow

- Questions
- Help
- Chat

Business

- Stack Internal
- Stack Data Licensing
- Stack Ads

Company

- About
- Press
- Work Here
- Legal
- Privacy Policy
- Terms of Service
- Contact Us
- Cookie Settings
- Cookie Policy

Stack Exchange Network

- Technology
- Culture & recreation
- Life & arts
- Science
- Professional
- Business
- API
- Data
- Blog
- Facebook
- Twitter
- LinkedIn
- Instagram