immutable value is still being moved

Asked 1 year ago

Active 10 months ago

Viewed 109 times

4

I can't get this function to compile:

```
/// Return a String with all characters masked as '#' except the last 4.
fn maskify(cc: &str) -> String {
    let chars = cc.to_string().chars();
    chars
        .enumerate()
        .map(|(i, c)| {
            if i > chars.count() - 4 { '#' } else { c }
        })
        .collect()
}
```

The current errors are:

```
error[E0507]: cannot move out of `chars`, a captured variable in an `FnMut` closure
 --> src/lib.rs:7:21
  |
3 |     let chars = cc.to_string().chars();
  |         ----- captured outer variable
...
7 |             if i > &chars.count() - 4 { '#' } else { c }
  |                     ^^^^^ move occurs because `chars` has type `std::str::Chars<'_>`, which does not implement the `Copy` trait

error[E0716]: temporary value dropped while borrowed
 --> src/lib.rs:3:17
  |
3 |     let chars = cc.to_string().chars();
  |                 ^^^^^^^^^^^^^^         - temporary value is freed at the end of this statement
  |                 |
  |                 creates a temporary which is freed while still in use
4 |     chars
  |     ----- borrow later used here
  |
  = note: consider using a `let` binding to create a longer lived value

error[E0382]: use of moved value: `chars`
 --> src/lib.rs:6:14
  |
3 |     let chars = cc.to_string().chars();
  |         ----- move occurs because `chars` has type `std::str::Chars<'_>`, which does not implement the `Copy` trait
4 |     chars
  |     ----- value moved here
5 |         .enumerate()
6 |         .map(|(i, c)| {
  |              ^^^^^^^^ value used here after move
7 |             if i > &chars.count() - 4 { '#' } else { c }
  |                     ----- use occurs due to use in closure
```

I think the source of the error is that `chars` is an iterator, so it mutates, making it impossible to borrow in the closure, but even if I try to declare a local variable (such as `let count = chars.count()`), I still get borrow errors.

I've tried dereferencing it with `&`, but that didn't work either.

rust closures borrow-checker ownership borrowing

Share
Improve this question
Follow

edited Feb 19 at 12:26

pretzelhammer
11.7k ● 14 ● 39 ● 88

asked Dec 6 '20 at 22:44

mzedeler
3,753 ● 3 ● 22 ● 37

Add a comment

3 Answers

Active Oldest Votes

4

The crux of the issue here is that `Char::count()` consumes `self`. Even if you declare a local variable, you cannot use `chars` after you moved ownership to the `count` function:

```
fn maskify(cc: &str) {
  let chars = cc.to_string().chars();
   // ^^^^^ move occurs here
  let count = chars.count();
               // ^^^^^^ `chars` moved because `count` consumes self
  let _ = chars.enumerate();
      // ^^^^^ value used here after move - *this is not allowed*
}
```

You can fix this issue by creating a new iterator and consuming that to get the `count`:

```
fn maskify(cc: &str) -> String {
    let chars = cc.chars();
    let count = cc.chars().count();
             // ^^^ create and consume a new iterator over cc
    chars
        .enumerate()
        .map(|(i, c)| {
            if i < count - 4 { '#' } else { c }
        })
        .collect()
}

fn main() {
    assert_eq!(maskify("abcd1234"), "####1234");
}
```

Or you can get the length of the string with `.len()`:

```
fn maskify(cc: &str) -> String {
    let chars = cc.chars();
    chars
        .enumerate()
        .map(|(i, c)| {
            if i < cc.len() - 4 { '#' } else { c }
        })
        .collect()
}

fn main() {
    assert_eq!(maskify("abcd1234"), "####1234");
}
```

Note that `str.len()` can *only* handle ascii while `.chars().count()` can handle full utf8.

Share
Improve this answer
Follow
edited Dec 7 '20 at 14:59

answered Dec 6 '20 at 22:57
Ibraheem Ahmed
6,841 ● 1 ● 19 ● 30

2

I would use `cc.chars().count()`, not `cc.len()`, to better support Unicode. (That still doesn't handle everything but it's closer.)
– Lambda Fairy
Dec 6 '20 at 23:14

Great explanation - and yes - I also noticed that `len` isn't right here, but adding another `count()` makes this work.
– mzedeler
Dec 6 '20 at 23:40

@mzedeler Why isn't `len` right? The assertion passes.
– Ibraheem Ahmed
Dec 7 '20 at 3:05

@IbraheemAhmed `.len()` is the number of bytes (UTF8 code units), `chars` works in terms of codepoints. UTF8 has 1-4 bytes per codepoint, so the code will misbehave for anything outside of ascii (e.g. non-latin script, diacritics, emoji, ...). For instance if you maskify "école" using `cc.len()`, you will only get 3 unmasked characters, and things get worse as you get further away from ascii, "■■■■■■■■" will have *no characters unmasked* with a bytes-based count, because it has 8 codepoints but encodes as 24 bytes.
– Masklinn
Dec 7 '20 at 7:36

1

So your final line should really be that `str.len()` *only* handles ascii, it's completely broken otherwise (and one could argue that it's plain broken any time you're trying to manipulate characters). If using `str::len`, the function should probably fail on getting a non-ascii codepoint, whether panicing entirely or just resulting in an `Err`.
– Masklinn
Dec 7 '20 at 7:47

Show 4 more comments

3

Two slightly different approaches you can use to implement this function depending on whether or not `cc` is UTF8 or ASCII. The UTF8 implementation of course works for both cases as UTF8 is a superset of ASCII.

```
fn maskify_utf8(cc: &str) -> String {
    let last_four = cc.chars().count().saturating_sub(4);
    cc.chars()
        .enumerate()
        .map(|(i, c)| if i < last_four { '#' } else { c })
        .collect()
}

fn maskify_ascii(cc: &str) -> String {
    let mask_idx = cc.len().saturating_sub(4);
    format!("{0:#<1$}{2}", "#", mask_idx, &cc[mask_idx..])
}

fn main() {
    assert_eq!(maskify_utf8("□□□□1234"), "####1234");
    assert_eq!(maskify_utf8("abcd1234"), "####1234");
    assert_eq!(maskify_ascii("abcd1234"), "####1234");
}
```

playground

Share
Improve this answer
Follow
edited Dec 7 '20 at 0:47

answered Dec 6 '20 at 22:53

pretzelhammer
11.7k ● 14 ● 39 ● 88

Add a comment

0

Thanks to @ibraheem-ahmed, I wound up with this solution:

```
/// Return a String with all characters masked as '#' except the last 4.
fn maskify(cc: &str) -> String {
    let leading = cc.chars().count().saturating_sub(4);
    cc
        .chars()
        .enumerate()
        .map(|(i, c)| {
            if i >= leading { c } else { '#' }
        })
        .collect()
}
```

Share
Improve this answer
Follow
answered Dec 6 '20 at 23:49

mzedeler
3,753 ● 3 ● 22 ● 37

Add a comment

Your Answer

Post Your Answer

Not the answer you're looking for? Browse other questions tagged rust closures borrow-checker ownership borrowing 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

Related

Variable binding: moving a &mut or borrowing the referent?

Why can't I store a value and a reference to that value in the same struct?

How to capture mutable reference into move closure contained in iterator returned from a closure

Returning a reference from a HashMap or Vec causes a borrow to last beyond the scope it's in?

2
How does ownership of variables work between iterations?

0
Cannot borrow self twice in one function call

Error: use of moved value: `path`. How to correct this code?

Hot Network Questions

What's the largest REG_SZ value that Regedit can edit?

What caused this crash landing?

Is Elon Musk really exploiting a loophole to avoid taxes?

I've got a material setup that blends two shaders decently, but how would I apply it to a circular target?

What happened to this character in MCU Spider-Man's life?

more hot questions

Question feed

STACK OVERFLOW

Questions
Jobs
Developer Jobs Directory
Salary Calculator
Help
Mobile

PRODUCTS

Teams
Talent
Advertising
Enterprise

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