

# Practice 4414 Prelim

2026-09-17

## 0. Truth or Dare

Rust True/False:

| T | F                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | <p>Rust always uses dynamic polymorphism when a type implements a trait, even when the trait is used as a generic bound.</p> <p>The NewType idiom is appropriate when two values share the same underlying type but represent different concepts.</p> <p>A closure in Rust can only be used when it does not capture any variables from its surrounding environment.</p> <p>Rust uses a garbage collector to automatically manage memory, while C++ requires programmers to manually free memory.</p> <p>A Rust enum variant can contain associated data, allowing one enum type to represent multiple shapes of data.</p> |

## 1. Out for Coffee

Consider this piece of code

```
enum MacsCafe {
    Pastries(u32),
    Espresso(u32, u32),
    Tea(u16),
    Latte,
}

struct GoldiesCafe {
    latte: u8,
    tea: u16,
    pastries: u32,
    espresso: (u32, u32),
}

fn main() {
    println!("{}", std::mem::size_of::<MacsCafe>());
    println!("{}", std::mem::size_of::<GoldiesCafe>());
}
```

What is the size of MacsCafe?

A. 8 B. 12 C. 16 D. 17

What is the size of GoldiesCafe?

A. 8 B. 12 C. 16 D. 17

Is there a difference in size between these two? Explain briefly.

## 2. Once in a Lifetime

Fill in the two missing blanks below so that the code compiles successfully. Provide the correct lifetime parameters.

```
// Blank 1: Complete the struct definition
struct Substring_____ {
    text: &'a str,
}

// Blank 2: Complete the function signature
fn get_prefix<'a>(source: _____, end: usize) -> Substring<'a> {
    Substring { text: &source[..end] }
}

fn main() {
    let my_string = String::from("Systems Programming");
    let prefix = get_prefix(&my_string, 7);
    println!("Prefix is: {}", prefix.text);
}
```

Which of the following correctly fills in Blank 1?

A. ('a) B. <'a> C. {'a} D. ::<'a> E. <'static>

Which of the following correctly fills in Blank 2?

A. &str B. &mut str C. &'a str D. \*const str E. String<'a>

### 3. Seeking Closure

For each of the following `main` functions, write the exact output of the program. If the program does not compile, write “compile error” and briefly (one sentence) explain why.

```
fn main() {
    let mut count = 0;
    let mut inc = move || {
        count += 1;
        println!("inside: {count}");
    };
    inc();
    inc();
    println!("outside: {count}");
}
```

```
fn main() {
    let mut count = 0;
    let mut inc = || count += 1;
    println!("{count}");
    inc();
}
```

```
fn main() {
    let name = String::from("cs4414");
    let f = move || println!("{name}");
    f();
    f();
}
```

```
fn consume(s: String) {
    println!("{s}");
}

fn main() {
    let name = String::from("cs4414");
    let f = || consume(name);
    f();
    f();
}
```

## 4. Pain in the Asset

A game engine uses a generic trait `Renderer<D>` which represents how an in-game asset interacts with a specific hardware device `D`.

Consider the following Rust code:

```
pub trait Renderer<D> {
    fn render_to(&self, device: &mut D);
}

pub struct Asset {
    pub texture_id: u32,
}

pub trait GpuDevice {
    fn draw_quad(&mut self, texture_id: u32);
}

/* [ BLANK 1 ] */ Renderer<D> for Asset
where
    /* [ BLANK 2 ] */
{
    fn render_to(&self, device: &mut D) {
        device.draw_quad(self.texture_id);
    }
}
```

1. Which correctly fills in Blank 1

A. `impl` B. `impl<D>` C. `impl<Asset>` D. `impl<GpuDevice>`

2 Which correctly fills in Blank 2

A. `D: GpuDevice` B. `GpuDevice: D` C. `Asset: GpuDevice` D. `Self: GpuDevice`

## 5. Understanding the Idiom

Consider the four function signatures below:

```
// A: Retrieving a middle name from a web form whose field is "Name"
fn get_middle_name(user: &UserName) -> Option<String>;

// B: Load a configuration file. File read permissions should be reported.
fn load_config_path(config: &Path) -> Option<String>;

// C: Parsing an integer from a user-supplied string.
fn parse_integer(input: &str) -> Option<i32>;

// D: Looking up an element in a cache.
fn cache_lookup(cache: &Cache, key: &str) -> Result<Item, String>;
```

Assume that all types are declared and in scope, and that the program would compile. For each signature A, B, C, D, state whether the signature's choice of **Result**/**Option** and choices for **T/E** is idiomatic Rust. If you think the choice is idiomatic, write a short (1-2 sentence) justifying why. If you think the choice is not idiomatic, rewrite the function signature (you are free to define new Types for errors) and justify why.

## 6. Boxing Day

Consider the following code fragment:

```
fn main() {  
    let x: Box<String> = Box::new(String::from("CS4414"));  
  
    let r: &Box<String> = &x;  
  
    let b = r.len(); // L1  
    let a = *r;      // L2  
}
```

Which of the following is correct?

- A. L1 is invalid because `r` is a reference, but L2 is valid.
- B. L1 is valid because Rust does automatic dereferencing for method calls, but L2 is invalid because it tries to move the `Box<String>` out of a borrowed reference.
- C. L1 is invalid because `len()` belongs to `String`, while L2 is valid because dereferencing a reference gives ownership.
- D. Both L1 and L2 are valid.

## 7. Pause and Reflect

Suppose we represent 2D points as a tuple struct `Point2d` with two `f64` fields. How would you write a function `reflect` taking a `Point2d` and producing a `Point2d` that corresponded to reflection across the  $y$  axis (i.e. reflecting the first component).

## 8. Out of the Cell

Consider the code

```
use std::cell::RefCell;

fn main() {
    let x = RefCell::new(String::from("doc"));
    {
        let mut y = x.borrow_mut();
    }
    let u = x.borrow();
    let v = x.borrow();
    let w = x.borrow_mut();
    let z = x.borrow_mut();
}
```

Which line will cause the first compiler complaint? Explain briefly.

## 9. Be Positive

The following code adapts an iterator over a collection of `i32` to an iterator that produces `u32` outputs, keeping only the positive elements of the input iterator.

```
fn be_positive(v: /* A */) -> /* B */ {  
    v.filter(|x| x > 0).map(|x| /* C */)   
}
```

What goes in slots A, B, and C?