University of Toronto, Department of Electrical and Computer Engineering

ECE 1724-F1 — Performant Software Systems with Rust

Quiz 3 — Sample Questions

Wednesday, October 14, 2026

Note: Each question has exactly one correct answer. Unless a complete program is shown, assume the code is inside `fn main()`.

1. What does this code print?

```
struct Size { width: u32, height: u32 }

let first = Size { width: 4, height: 7 };
let mut second = Size { width: 9, ..first };
second.height = 2;
println!("{} {}", first.height, second.height);
```

Ⓐ `2 2`

Ⓑ It does not compile because `first.height` was moved.

Ⓒ `7 2`

Ⓓ `7 7`

2. Which line can replace the comment to print the struct using its derived debug format?

```
#[derive(Debug)]
struct Point { x: i32, y: i32 }

let point = Point { x: 2, y: 5 };
// Print point here.
```

Ⓐ `println!("{:d}", point);`

Ⓑ `println!("{}", point);`

Ⓒ `println!(point);`

Ⓓ `println!("{:?}", point);`

Quiz 3 — Sample Questions (continued)

3. What does this complete program print?

```
struct Panel { width: u32, height: u32 }

impl Panel {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

fn main() {
    let panel = Panel { width: 3, height: 5 };
    println!("{} {}", panel.area(), panel.width);
}
```

(A) `15 3`

(B) `15 15`

(C) It does not compile because `area()` moves `panel`.

(D) It does not compile because `panel` is not mutable.

4. What happens when this complete program is compiled and run?

```
struct Counter { value: u32 }

impl Counter {
    fn increment(&mut self) {
        self.value += 1;
    }
}

fn main() {
    let counter = Counter { value: 4 };
    counter.increment();
    println!("{}", counter.value);
}
```

(A) It prints `5`.

(B) It does not compile because `counter` is not a mutable binding.

(C) It prints `4` because the method changes a copy.

(D) It compiles but panics because mutable receivers cannot modify fields.

Quiz 3 — Sample Questions (continued)

5. Which expression can replace `/* expression */` to create a `Badge`?

```
struct Badge { number: u32 }

impl Badge {
    fn new(number: u32) -> Badge {
        Badge { number }
    }
}

let badge = /* expression */;
```

Ⓐ `Badge.new(8)`

Ⓑ `new::Badge(8)`

Ⓒ `Badge::new(8)`

Ⓓ `Badge(8)`

6. Given this definition, which expression constructs a valid `Event` value?

```
enum Event {
    Stop,
    Message(String),
    Move { x: i32, y: i32 },
}
```

Ⓐ `Event::Move { x: 1, y: 2 }`

Ⓑ `Event::Stop()`

Ⓒ `Event::Message { text: String::from("hi") }`

Ⓓ `Event::Move(1, 2)`

Quiz 3 — Sample Questions (continued)

7. What does this code print?

```
let bonus = Some(4);
let total = match bonus {
    Some(points) => points + 6,
    None => 6,
};
println!("{total}");
```

Ⓐ `4`

Ⓑ `6`

Ⓒ It does not compile because `Some` must be written as `Option::Some`.

Ⓓ `10`

8. Why does this complete program fail to compile?

```
enum Signal { Red, Yellow, Green }

fn score(signal: Signal) -> u32 {
    match signal {
        Signal::Red => 0,
        Signal::Green => 1,
    }
}

fn main() {
    println!("{}", score(Signal::Green));
}
```

Ⓐ An enum variant cannot be passed to a function.

Ⓑ The match does not handle `Signal::Yellow`.

Ⓒ Match arms cannot return integers.

Ⓓ Every enum must contain data in each variant.

Quiz 3 — Sample Questions (continued)

9. What does this code print?

```
let limit: Option<u32> = None;
if let Some(value) = limit {
    println!("limit {value}");
} else {
    println!("unlimited");
}
```

Ⓐ `limit 0`

Ⓑ It does not compile because `if let` requires a match arm for `None`.

Ⓒ `unlimited`

Ⓓ It prints nothing because `None` skips the entire conditional.

10. What does this complete program print?

```
fn next(value: Option<u32>) -> u32 {
    let Some(number) = value else {
        return 0;
    };
    number + 1
}

fn main() {
    println!("{} {}", next(None), next(Some(5)));
}
```

Ⓐ `0 6`

Ⓑ `1 6`

Ⓒ `0 5`

Ⓓ It does not compile because `number` is unavailable after `let else`.