University of Toronto, Department of Electrical and Computer Engineering

ECE 1724-F1 — Performant Software Systems with Rust

Quiz 1 — Sample Questions

Wednesday, September 23, 2026

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

1. Which command checks a Cargo project for compilation errors without producing an executable?

Ⓐ `cargo build`

Ⓑ `cargo check`

Ⓒ `cargo run`

Ⓓ `cargo clean`

2. What does this code print?

```
let x = 4;
let x = x + 1;

{
    let x = x * 2;
    println!("{x}");
}

println!("{x}");
```

Ⓐ `10`, then `5`, on separate lines.

Ⓑ `10`, then `10`, on separate lines.

Ⓒ `8`, then `4`, on separate lines.

Ⓓ It does not compile because `x` is not declared with `mut`.

Quiz 1 — Sample Questions (continued)

3. Why does this function fail to compile?

```
fn plus_one(x: i32) -> i32 {
    x + 1;
}
```

Ⓐ The parameter `x` must be declared with `mut`.

Ⓑ Every function returning an integer must use the `return` keyword.

Ⓒ The trailing semicolon makes the body evaluate to `()`, not `i32`.

Ⓓ Rust cannot infer the type of the integer literal `1`.

4. What happens when this code is compiled and run?

```
let condition = true;
let value = if condition { 5 } else { "six" };
println!("{value}");
```

Ⓐ It prints `5`; only the selected branch needs a valid type.

Ⓑ It prints `5`; Rust converts both branches to strings.

Ⓒ It compiles but panics if `condition` is false.

Ⓓ It does not compile because the branches have incompatible types.

5. What does this code print?

```
let mut counter = 0;
let result = loop {
    counter += 1;
    if counter == 3 {
        break counter * 2;
    }
};
println!("{counter} {result}");
```

Ⓐ `3 3`

Ⓑ `3 6`

Ⓒ `6 6`

Ⓓ It does not compile because a `loop` cannot produce a value.

Quiz 1 — Sample Questions (continued)

6. Which statement correctly explains this code?

```
let x: i32 = 5;
let y = x;
let s1 = String::from("hello");
let s2 = s1;
println!("{x} {y}");
println!("{s1} {s2}");
```

(A) Both assignments copy their values, so the program compiles.

(B) Both assignments move their values, so neither `x` nor `s1` can be used.

(C) `x` is copied, but `s1` is moved; the final line prevents compilation.

(D) The program compiles, but freeing the string twice causes a run-time error.

7. What does this code print?

```
let s1 = String::from("rust");
let mut s2 = s1.clone();
s2.push_str("ace");
println!("{s1} {s2}");
```

(A) `rust rustace`

(B) `rustace rustace`

(C) `rust rust`

(D) It does not compile because `clone()` invalidates `s1`.

Quiz 1 — Sample Questions (continued)

8. In this program, when is the heap allocation holding `"hello"` released?

```
fn consume(text: String) {
    println!("{text}");
} // point A

fn main() {
    let s = String::from("hello");
    consume(s);
    println!("done"); // point B
} // point C
```

(A) At point B, when `println!` next runs.

(B) At point C, when the original variable `s` goes out of scope.

(C) Later, when Rust's garbage collector next runs.

(D) At point A, when `text`, the current owner, goes out of scope.

9. Which replacement for the function body makes this complete program compile and print `hello 5`, without cloning the string?

```
fn measure(s: String) -> (String, usize) {
    (s, s.len())
}

fn main() {
    let s = String::from("hello");
    let (s, n) = measure(s);
    println!("{s} {n}");
}
```

(A) `let n = s.len(); (s, n);`

(B) `let n = s.len(); (s, n)`

(C) `(s.len(), s)`

(D) `let text = s; (text, s.len())`