University of Toronto, Department of Electrical and Computer Engineering

ECE 1724-F1 — Performant Software Systems with Rust

Quiz 1 — Answer Key to Sample Questions

Wednesday, September 23, 2026

1. Ⓑ **Checking a project.**

`cargo check` checks compilation without producing an executable. `build` produces a build, `run` builds and runs, and `clean` removes build artifacts. (Lecture 2; Book 1.3.)

2. Ⓐ **Scope and shadowing.**

The outer shadowed binding is `5`. The inner binding is `10` and ends with its block; the outer binding remains `5`. Shadowing introduces a new binding and needs no `mut`. (Lecture 3; Book 3.1.)

3. Ⓒ **Expressions and statements.**

The semicolon discards the value of `x + 1`. With no tail expression, the body yields unit, `()`, which conflicts with the declared `i32` return type. Removing the semicolon fixes it. (Lectures 2–3; Book 3.3.)

4. Ⓓ **Types of conditional branches.**

The integer and string branches must have compatible types even though the condition is `true`. Rust does not implicitly convert them to a shared string type. (Lecture 3; Book 3.5.)

5. Ⓑ **A value from a loop.**

The counter reaches `3`; `break counter * 2` ends the loop with the value `6`. The counter itself stays `3`. (Lecture 3; Book 3.5.)

6. Ⓒ **Copy versus move.**

`i32` implements `Copy`, so both integer bindings remain usable. Assigning the `String` moves ownership to `s2`, making the later use of `s1` a compile-time error. The whole program fails to compile and prints nothing. (Lecture 4; Book 4.1.)

7. Ⓐ **Cloning a string.**

`clone()` creates an independent copy of the string data. Appending to `s2` leaves `s1` unchanged. (Lecture 4; Book 4.1.)

8. Ⓓ **Ownership and function scope.**

The call moves ownership from `s` to `text`. The string is dropped when `text` goes out of scope at the end of `consume`, before `done` is printed. (Lecture 4; Book 4.1.)

9. Ⓑ **Returning ownership and a computed value.**

Compute the length before moving `s` into the returned tuple. A discards the tuple with a semicolon; C reverses the required tuple types; D uses `s` after moving it. The original body also uses `s` after moving it into the first tuple element. The caller's `let (s, n)` creates new bindings. (Lecture 4, cutoff slides; Book 4.1.)