CERTLABZ CERTIFICATION BOARD

Certified Rust Programmer (CRUP)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working Rust programmer would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1 — Domain 1: Rust Fundamentals and Syntax — CONFIG

Sarah Miller writes this snippet and the compiler rejects the second line.

```
let x = 5;
x = 6;
```

Listing 1.1 A rejected reassignment.

Which change lets the code compile?

A. Add a return type
B. Rename x to y
C. Use let mut x
D. Delete the let line

Correct answer: C

Bindings in Rust are immutable by default, so reassigning x fails until it is declared with let mut x. A rename or a return type does nothing about mutability, and deleting the let line removes the binding the second line needs.

Question 2 — Domain 2: Types, Structs and Enums — DATA

James Carter maps each standard type to what it represents, with one meaning missing.

Type	Represents
Option<T>	A value or nothing
Result<T, E>	(to choose)
bool	True or false
char	A single character

Table 2.1 Types and their meaning.

What does Result represent?

A Always a null value

B Ok or an error

C A list of items

D A fixed array

Correct answer: B
Result<T, E> models an operation that can succeed with Ok or fail with an error value in Err. Option covers a value or nothing, while a list or a fixed array are separate collection types that Result does not describe.

Question 3 — Domain 3: Control Flow and Functions — CONFIG

Olivia Davis transforms a vector using an iterator chain.

```
let nums = vec![1, 2, 3];
let doubled: Vec<i32> =
    nums.iter()
        .map(|x| x * 2)
        .collect();
```

Listing 3.1 An iterator chain.

What does collect do here?

A Builds a Vec

B Sorts the values

C Prints each item

D Sums the numbers

Correct answer: A
The map step yields a lazy iterator of doubled values, and collect consumes it to build the new Vec<i32> named doubled. It does not sort, print, or add the numbers, since map only applies the closure to each element.

Question 4 — Domain 4: Ownership, Borrowing and Lifetimes — CONFIG

Liam Anderson expects this to print the string, but it fails to build.

```
let s1 = String::from("hi");
let s2 = s1;
println!("{}", s1);
```

Listing 4.1 A use after move.

Why does the code fail?

A String is too long

B s2 is never used

C Missing a semicolon

D s1 was moved to s2

Correct answer: D
Assigning s1 to s2 moves ownership of the heap String, so s1 is no longer valid and the later use is rejected. Cloning with s1.clone or borrowing with a reference would let both names access the data safely.

Question 5

Domain 5: Traits, Generics and Collections

DATA

Ava Thompson matches each collection to its best use, with one entry missing.

Collection	Best for
Vec<T>	Ordered list of items
HashMap<K, V>	(to choose)
String	Growable text
array	Fixed-size storage

Table 5.1 Collections and their use.

What is HashMap best for?

A Fixed-size number arrays

B Key to value lookup

C Ordered sequence

D Single characters

Correct answer: B
HashMap<K, V> stores pairs and gives fast lookup of a value by its key. A Vec keeps an ordered sequence, an array holds a fixed number of items, and char covers single characters, none of which offer keyed lookup.

Question 6

Domain 6: Error Handling, Concurrency and Cargo Tooling

CONFIG

David Brown parses text into a number and returns a Result.

```
fn parse(s: &str)
    -> Result<i32, ParseIntError> {
    let n = s.parse::<i32>()?;
    Ok(n)
}
```

Listing 6.1 The question mark operator.

What does the ? operator do?

A Ignores any errors

B Loops until it works

C Returns error early

D Panics on success

Correct answer: C
The ? operator unwraps an Ok value or returns the Err from the function early, so the caller receives the failure. It never loops, ignores the error, or panics when the parse succeeds, which keeps error handling concise.

Question 7 Domain 4: Ownership, Borrowing and Lifetimes

A function needs to read a large String without taking ownership, and the caller must keep using it afterward. What should the parameter be?

A A shared reference

B An owned String

C A mutable copy

D A new global variable

Correct answer: A
A shared reference lets the function read the String while the caller keeps ownership and can use it again. Taking an owned String would move it away, a copy wastes memory, and a global variable changes the design instead of solving the borrow.

Question 8 Domain 3: Control Flow and Functions

You want to run a block of code once for each item in a vector, using the values only for reading. Which construct fits best?

A A while true loop

B A recursive call

C A match on length

D A for in loop

Correct answer: D
A for in loop over an iterator visits each item cleanly and reads the values without extra bookkeeping. A while true loop needs manual control, recursion adds overhead, and matching on length does not walk the elements.

Question 9 Domain 2: Types, Structs and Enums

A function returns Option<i32> and you must handle both the value and the empty case in one place. What is the idiomatic choice?

A. Always call unwrap first
B. Match on the Option
C. Ignore the None arm
D. Cast it to an int

Correct answer: B
Matching on the Option forces you to cover both Some and None, so the empty case is handled safely. Calling unwrap panics on None, ignoring the None arm drops a case, and casting an Option to an int is not valid.

Question 10 Domain 6: Error Handling, Concurrency and Cargo Tooling

Two threads must send computed values back to the main thread safely without sharing mutable state. What tool fits best?

A. A channel
B. A global mut static
C. A busy wait loop
D. A shared raw pointer

Correct answer: A
A channel lets threads send values to a receiver without sharing mutable memory, which the ownership model favors. A mutable static is unsafe, a busy wait wastes cycles, and a raw pointer gives up the safety guarantees Rust provides.

Published by the CertLabz Certification Board, a division of CertLabz LLC. Companion to objectives document CRUP-001. https://certlabz.com/certifications/certified-rust-programmer.html