UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

ECE 1724-F1 — Performant Software Systems with Rust
Sample Midterm Examination — Set 1
110 minutes / 100 marks

Coverage: Rust Book Chapters 1–6, 8, and 9. Chapter 7 is excluded.

This practice examination contains 40 multiple-choice questions. Select exactly one answer (A–D) for each question. There is no negative marking.

Suggested pacing: Allow 40 minutes for Questions 1–20 (2 marks each), approximately 67 minutes for Questions 21–40 (3 marks each), and 3 minutes to review your answers.

The harder questions assess your ability to read longer Rust programs using the same foundational concepts. All questions are independent.

The examination has 17 pages, including this cover.

First Name: _____ Last Name: _____

Student Number: _____

MARKS

Questions 1–20	Questions 21–40	Total
/40	/60	/100

Sample Midterm 1 — Questions (continued)

Code conventions. Unless a complete program is shown, treat code as the body of `main`; local type and function definitions are allowed. Compilation failures print nothing. Ignore compiler warnings; use Rust 2024.

For timed practice, work without a compiler and check solutions afterward.

Section A — Short questions (40 marks)

Question 1 [2 Marks]

Which Cargo command checks whether a project compiles without producing an executable?

(A) `cargo run`
(B) `cargo check`
(C) `cargo build`
(D) `cargo new`

Question 2 [2 Marks]

In a new binary project created by Cargo, which file normally contains the program entry point?

(A) `Cargo.toml`
(B) `src/lib.rs`
(C) `target/main.rs`
(D) `src/main.rs`

Question 3 [2 Marks]

What does this code print?

```
let input = " 42\n";
let count: u32 = input.trim().parse().expect("number");
println!("{}", count + 1);
```

(A) `43`
(B) `42`
(C) It does not compile.
(D) It panics.

Question 4 [2 Marks]

A program uses the following lines. The buffer is initially empty, and the user types 8 then presses Enter. On success, what does read_line do?

```
let mut input = String::new();
std::io::stdin().read_line(&mut input).expect("input");
```

(A) It replaces input with the integer 8.
(B) It removes the newline and stores only the digit.
(C) It appends the input text, including the newline, to input.
(D) It moves input so the caller cannot use it again.

Sample Midterm 1 — Questions (continued)

Question 5 [2 Marks]

What does this code print?

```
let count = 4;
let count = count + 2;
let count = count * 3;
println!("{}", count);
```

(A) `4`

(B) `12`

(C) It does not compile because count is immutable.

(D) `18`

Question 6 [2 Marks]

What does this complete program print?

```
fn charge(days: i32) -> i32 {
    days * 2 + 1
}
fn main() {
    println!("{}", charge(3));
}
```

(A) `6`

(B) `7`

(C) `3`

(D) It does not compile because return is missing.

Question 7 [2 Marks]

What does this code print?

```
let desks = [3, 5, 8];
let selected = (desks[1], desks[2]);
println!("{} {}", selected.0, selected.1);
```

(A) `5 8`

(B) `3 5`

(C) `8 5`

(D) It panics.

Question 8 [2 Marks]

What happens when this code is compiled?

```
let ticket = String::from("help");
let assigned = ticket;
println!("{}", ticket);
```

(A) It prints help.

(B) It prints an empty string.

(C) It fails: ticket was moved into assigned.

(D) It compiles, then panics.

Question 9 [2 Marks]

What does this code print?

```
let mut original = String::from("loan");
let saved = original.clone();
original.push('s');
println!("{} {}", original, saved);
```

(A) `loan loan`

(B) `loans loan`

(C) `loans loans`

(D) It does not compile.

Question 10 [2 Marks]

What does this code print?

```
let title = String::from("Rust guide");
let first = &title[..4];
println!("{}", first);
```

(A) `Rust` followed by one space.

(B) `guide`

(C) It does not compile.

(D) `Rust`

Question 11 [2 Marks]

What does this code print?

```
struct Loan {
    days: u32,
}
let mut loan = Loan { days: 4 };
loan.days += 2;
println!("{}", loan.days);
```

(A) `6`

(B) `4`

(C) It does not compile: fields are always immutable.

(D) It does not compile: days must be declared mut.

Question 12 [2 Marks]

What does this code print?

```
struct Desk {
    seats: u32,
}
impl Desk {
    fn available(&self) -> bool {
        self.seats > 0
    }
}
let desk = Desk { seats: 2 };
println!("{}", desk.available());
```

(A) `false`

(B) It does not compile: desk must be mutable.

(C) `true`

(D) It does not compile: methods must consume self.

Question 13 [2 Marks]

What does this code print?

```
let desk: Option<u32> = Some(7);
let number = match desk {
    Some(n) => n + 1,
    None => 0,
};
println!("{}", number);
```

(A) `7`

(B) `0`

(C) It does not compile.

(D) `8`

Question 14 [2 Marks]

What does this code print?

```
let note: Option<&str> = None;
let mut count = 2;
if let Some(_) = note {
    count += 5;
}
println!("{}", count);
```

(A) `2`

(B) `7`

(C) `0`

(D) It does not compile: None needs a String.

Question 15 [2 Marks]

What does this code print?

```
let mut desks = [2, 4, 6];
let previous = desks[1];
desks[1] = desks[0] + 5;
println!("{} {}", previous, desks[1]);
```

(A) `7 7`

(B) `4 7`

(C) `4 4`

(D) It does not compile because arrays cannot change.

Question 16 [2 Marks]

What does this code print?

```
let mut reserved = Some(3);
if let Some(count) = reserved {
    reserved = Some(count + 2);
}
match reserved {
    Some(n) => println!("{}", n),
    None => println!("none"),
}
```

(A) `3`

(B) `none`

(C) `5`

(D) It does not compile because Some cannot be reassigned.

Question 17 [2 Marks]

What does this code print?

```
let mut queue = vec![2, 4];
queue.push(6);
println!("{} {}", queue.len(), queue[1]);
```

(A) `3 4`

(B) `2 4`

(C) `3 6`

(D) It panics.

Question 18 [2 Marks]

What does this code print?

```
let mut label = String::from("Desk");
let suffix = String::from(" 4");
label.push_str(&suffix);
println!("{}|{}", label, suffix);
```

(A) `Desk| 4`

(B) It does not compile: suffix was moved.

(C) It prints Desk 4 and then panics.

(D) `Desk 4| 4`

Question 19 [2 Marks]

What does this code print?

```
use std::collections::HashMap;
let mut counts = HashMap::new();
counts.insert("open", 3);
counts.insert("open", 5);
println!("{}", *counts.get("open").unwrap());
```

(A) `3`

(B) `8`

(C) `5`

(D) It panics because a key is duplicated.

Question 20 [2 Marks]

What does this code print?

```
let parsed = "unknown".parse::<u32>();
match parsed {
    Ok(n) => println!("{}", n),
    Err(_) => println!("retry"),
}
```

(A) `0`

(B) `retry`

(C) It does not compile.

(D) It always panics when parse fails.

Section B — Longer code reading (60 marks)

Question 21 [3 Marks]

What does this code print?

```
let entries = [" 4 ", "bad", "7", " 2"];
let mut total = 0;
let mut rejected = 0;
for entry in entries {
    let number = match entry.trim().parse::<i32>() {
        Ok(n) => n,
        Err(_) => {
            rejected += 1;
            continue;
        }
    };
    if number > 3 {
        total += number;
    } else {
        total += 1;
    }
}
println!("{} {}", total, rejected);
```

(A) `13 1`

(B) `11 1`

(C) `12 1`

(D) `12 0`

Question 22 [3 Marks]

What does this code print?

```
let deliveries = [2, 5, 1, 4];
let mut score = 0;
let mut visited = 0;
for amount in deliveries {
    visited += 1;
    if amount == 1 {
        continue;
    }
    let amount = amount + 1;
    if amount % 2 == 0 {
        score += amount;
    } else {
        score += 2;
    }
    if score >= 8 {
        break;
    }
}
println!("{} {}", score, visited);
```

(A) `8 2`

(B) `10 4`

(C) `8 4`

(D) `7 2`

Question 23 [3 Marks]

What does this complete program print?

```
fn stamp(mut text: String) -> String {
    text.push('!');
    text
}
fn width(text: &String) -> usize {
    text.len()
}
fn main() {
    let request = String::from("book");
    let saved = request.clone();
    let updated = stamp(request);
    let size = width(&updated);
    let mut archive = saved;
    archive.push('?');
    println!("{} {} {}", updated, archive, size);
}
```

(A) `book! book! 5`

(B) `book! book? 4`

(C) It does not compile: request was moved.

(D) `book! book? 5`

Question 24 [3 Marks]

What happens when this complete program is compiled and run?

```
fn attach(text: &mut String) {
    text.push_str(" ready");
}
fn main() {
    let mut ticket = String::from("desk");
    let view = &ticket;
    let before = view.len();
    println!("{}", view);
    attach(&mut ticket);
    let after = ticket.len();
    let current = &ticket;
    println!("{}", current);
    println!("{} {}", before, after);
}
```

(A) It fails: view prevents every later mutable borrow.

(B) It prints three lines: `desk`, `desk ready`, and `4 10`.

(C) It prints desk twice, followed by 4 4.

(D) It compiles, then panics during attach.

Question 25 [3 Marks]

All text is ASCII. What does this complete program print?

```
fn first_word(text: &str) -> &str {
    let bytes = text.as_bytes();
    for (index, byte) in bytes.iter().enumerate() {
        if *byte == b' ' {
            return &text[..index];
        }
    }
    &text[..]
}
fn main() {
    let line = String::from("return blue book");
    let action = first_word(&line);
    let rest = &line[action.len() + 1..];
    let color = first_word(rest);
    println!("{} {} {}", action, color, rest.len());
}
```

(A) `return blue 9`

(B) `return blue 15`

(C) `return blue 10`

(D) It does not compile: two slices cannot coexist.

Question 26 [3 Marks]

What happens when this code is compiled and run?

```
struct Loan {
    title: String,
    days: u32,
    renewed: bool,
}
let first = Loan {
    title: String::from("Rust"),
    days: 7,
    renewed: false,
};
let second = Loan {
    days: first.days + 3,
    ..first
};
println!("{} {}", first.days, second.days);
println!("{}", first.title);
```

(A) It prints `7 10`, then `Rust`.

(B) It fails to compile because first.days was moved.

(C) It prints 7 10, then panics.

(D) It fails to compile because first.title was moved.

Question 27 [3 Marks]

What does this code print?

```
struct Desk {
    waiting: u32,
}
impl Desk {
    fn new(waiting: u32) -> Self {
        Self { waiting }
    }
    fn serve(&mut self, limit: u32) -> u32 {
        let served = if self.waiting < limit {
            self.waiting
        } else {
            limit
        };
        self.waiting -= served;
        served
    }
}
let mut desk = Desk::new(8);
let first = desk.serve(3);
let second = desk.serve(7);
println!("{} {} {}", first, second, desk.waiting);
```

(A) `3 7 8`

(B) `3 5 0`

(C) `3 7 0`

(D) `3 5 8`

Question 28 [3 Marks]

What does this complete program print?

```
enum Request {
    Borrow(i32),
    Return(i32),
    Closed,
}
fn change(request: Request) -> i32 {
    match request {
        Request::Borrow(n) => -n,
        Request::Return(n) => n,
        Request::Closed => 0,
    }
}
fn main() {
    let mut available = 10;
    available += change(Request::Borrow(3));
    available += change(Request::Return(2));
    available += change(Request::Closed);
    println!("{}", available);
}
```

(A) `5`

(B) `15`

(C) `9`

(D) It does not compile: variants have different data.

Question 29 [3 Marks]

What does this complete program print?

```
fn next_desk(current: Option<u32>) -> Option<u32> {
    match current {
        Some(n) => {
            if n < 5 {
                Some(n + 1)
            } else {
                None
            }
        }
        None => Some(1),
    }
}
fn main() {
    let mut desk = Some(4);
    desk = next_desk(desk);
    desk = next_desk(desk);
    desk = next_desk(desk);
    match desk {
        Some(n) => println!("{}", n),
        None => println!("none"),
    }
}
```

(A) `5`

(B) `6`

(C) `none`

(D) `1`

Question 30 [3 Marks]

What does this code print?

```
let arrivals = [Some(2), None, Some(5), Some(1)];
let mut total = 0;
let mut missing = 0;
for arrival in arrivals {
    if let Some(count) = arrival {
        if count > 2 {
            total += count - 1;
        } else {
            total += count;
        }
    } else {
        missing += 1;
    }
}
let bonus = if missing == 0 { 3 } else { 1 };
total += bonus;
println!("{} {}", total, missing);
```

(A) `9 1`

(B) `8 1`

(C) `10 0`

(D) `7 1`

Question 31 [3 Marks]

What does this complete program print?

```
struct Ticket {
    label: String,
    minutes: u32,
}
impl Ticket {
    fn estimate(&self) -> u32 {
        self.minutes + 2
    }
    fn finish(self) -> String {
        format!("{} done", self.label)
    }
}
fn main() {
    let ticket = Ticket { label: String::from("desk"), minutes: 5 };
    let estimate = ticket.estimate();
    let message = ticket.finish();
    println!("{} {}", estimate, message);
}
```

(A) `5 desk done`

(B) It fails: estimate consumes ticket.

(C) `7 desk done`

(D) It fails: ticket is still borrowed.

Question 32 [3 Marks]

What does this code print?

```
enum Change {
    Add(u32),
    Serve,
    Reset,
}
let changes = [
    Change::Add(4), Change::Serve, Change::Add(2),
    Change::Reset, Change::Serve, Change::Add(3),
];
let mut waiting = 0;
let mut served = 0;
for change in changes {
    match change {
        Change::Add(n) => waiting += n,
        Change::Serve => {
            if waiting > 0 {
                waiting -= 1;
                served += 1;
            }
        }
        Change::Reset => waiting = 0,
    }
}
println!("{} {}", waiting, served);
```

(A) `3 1`

(B) `3 2`

(C) `7 2`

(D) `3 0`

Question 33 [3 Marks]

What does this complete program print?

```
fn read_pages(text: &str) -> Result<u32, String> {
    match text.parse::<u32>() {
        Ok(n) => Ok(n),
        Err(_) => Err(String::from("invalid")),
    }
}
fn main() {
    let inputs = ["4", "bad", "6"];
    let mut accepted = Vec::new();
    let mut errors = 0;
    for input in inputs {
        match read_pages(input) {
            Ok(n) => accepted.push(n),
            Err(_) => errors += 1,
        }
    }
    accepted.push(errors);
    let mut total = 0;
    for pages in &accepted {
        total += *pages;
    }
    println!("{} {}", accepted.len(), total);
}
```

(A) `2 10`

(B) `3 11`

(C) `3 10`

(D) It panics on the input bad.

Question 34 [3 Marks]

What does this code print?

```
let mut waiting = vec![3, 6, 2];
for count in &mut waiting {
    if *count > 3 {
        *count -= 2;
    } else {
        *count += 1;
    }
}
waiting.push(5);
let mut total = 0;
for count in &waiting {
    total += *count;
}
let first = waiting[0];
let last = waiting[waiting.len() - 1];
println!("{} {} {}", first, last, total);
```

(A) `3 5 16`

(B) `4 5 11`

(C) It fails: mutable and immutable loops cannot share a vector.

(D) `4 5 16`

Question 35 [3 Marks]

What does this code print?

```
enum Ticket {
    Open(u32),
    Closed,
}
let tickets = vec![
    Ticket::Open(3),
    Ticket::Closed,
    Ticket::Open(6),
    Ticket::Open(2),
];
let mut work = 0;
let mut closed = 0;
for ticket in tickets {
    match ticket {
        Ticket::Open(minutes) => work += minutes + 1,
        Ticket::Closed => closed += 1,
    }
}
println!("{} {}", work, closed);
```

(A) `11 1`

(B) `15 1`

(C) `14 1`

(D) It fails: vector elements have different enum variants.

Question 36 [3 Marks]

What happens when this code is compiled and run?

```
let mut room = String::from("Lab");
let number = String::from("7");
let saved = format!("{}-{}", room, number);
room.push('!');
let label = room + &number;
let mut log = String::from("[");
log.push_str(&saved);
log.push(']');
let size = label.len();
if size > 4 {
    log.push('+');
} else {
    log.push('-');
}
println!("{} {} {}", label, number, log);
```

(A) It prints `Lab!7 7 [Lab-7]+`.

(B) It prints `Lab!7 7 [Lab!-7]+`.

(C) It fails: format! moved room and number.

(D) It fails: addition moved number.

Question 37 [3 Marks]

What does this code print?

```
use std::collections::HashMap;
let requests = ["book", "desk", "book", "book"];
let mut counts = HashMap::new();
counts.insert("desk", 2);
for request in requests {
    let count = counts.entry(request).or_insert(0);
    *count += 1;
}
{
    let count = counts.entry("book").or_insert(10);
    *count += 2;
}
let missing = counts.get("printer");
let extra = match missing {
    Some(n) => *n,
    None => 4,
};
println!("{} {} {}", *counts.get("book").unwrap(), *counts.get("desk").unwrap(), extra);
```

(A) `12 3 4`

(B) `5 1 4`

(C) `5 3 0`

(D) `5 3 4`

Question 38 [3 Marks]

What does this complete program print?

```
fn combined(a: &str, b: &str) -> Result<u32, String> {
    let first = match a.parse::<u32>() {
        Ok(n) => n,
        Err(_) => return Err(String::from("first")),
    };
    let second = read_second(b)?;
    Ok(first + second)
}
fn read_second(text: &str) -> Result<u32, String> {
    match text.parse::<u32>() {
        Ok(n) => Ok(n),
        Err(_) => Err(String::from("second")),
    }
}
fn main() {
    match combined("6", "bad") {
        Ok(n) => println!("total {}", n),
        Err(message) => println!("error {}", message),
    }
}
```

(A) `total 6`

(B) `error first`

(C) `error second`

(D) It panics at the question mark.

Question 39 [3 Marks]

What does this complete program print?

```
fn pages(text: &str) -> Result<u32, String> {
    match text.parse::<u32>() {
        Ok(0) => Err(String::from("empty")),
        Ok(n) => Ok(n),
        Err(_) => Err(String::from("invalid")),
    }
}
fn main() {
    let inputs = ["3", "0", "bad", "5"];
    let mut total = 0;
    let mut failures = 0;
    for input in inputs {
        match pages(input) {
            Ok(n) => total += n,
            Err(_) => {
                failures += 1;
                total += 1;
            }
        }
    }
    println!("{} {}", total, failures);
}
```

(A) `8 2`

(B) `10 2`

(C) `9 1`

(D) It panics at the input bad.

Question 40 [3 Marks]

Which description correctly gives the outcome of this complete program? Ignore the panic message itself.

```
fn read_count(text: &str) -> Result<u32, std::num::ParseIntError> {
    text.parse::<u32>()
}
fn main() {
    let first = read_count("4");
    match first {
        Ok(n) => println!("accepted {}", n),
        Err(_) => println!("retry"),
    }
    let second = read_count("oops");
    println!("checked");
    let count = second.expect("expected a count");
    println!("accepted {}", count);
    println!("done");
}
```

(A) It prints `accepted 4`, then `checked`, then panics.

(B) It prints `accepted 4`, `checked`, `accepted 0`, and `done`.

(C) It panics before printing checked.

(D) It does not compile because Result cannot be stored.