UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

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

Coverage: Rust Book Chapters 1–6, 8, and 9.

This 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), and approximately 70 minutes for Questions 21–40 (3 marks each).

All questions are independent. Unless a complete program is shown, code is inside `main`. Programs that fail compilation print nothing. Ignore compiler warnings; assume Rust 2024.

The examination has 28 pages, including this cover.

First Name: _____ Last Name: _____

Student Number: _____

MARKS

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

Section A: Short questions — 20 questions, 2 marks each (40 marks).

Question 1 [2 Marks]

Which command creates a new Cargo project named station, including its initial source file and Cargo.toml?

Ⓐ `cargo run station`
Ⓑ `rustc station`
Ⓒ `cargo new station`
Ⓓ `cargo check station`

Question 2 [2 Marks]

A program was built with cargo build. You then edit src/main.rs and directly run the existing executable in target/debug, without running Cargo or rustc again. Which code does that executable run?

Ⓐ The code from the previous successful build.
Ⓑ The newly edited source code.
Ⓒ A mixture of the old and new source code.
Ⓓ It refuses to run until Cargo.toml is edited.

Question 3 [2 Marks]

In the following code, why is &mut input passed to read_line?

```
let mut input = String::new();

std::io::stdin()
    .read_line(&mut input)
    .expect("input failed");
```

Ⓐ To convert the String into an integer.
Ⓑ To transfer ownership of input permanently to read_line.
Ⓒ To prevent read_line from changing input.
Ⓓ To let read_line append input through a mutable reference.

Question 4 [2 Marks]

What does this code print?

```
let guess = String::from("12");
let guess: u32 = guess.parse().expect("number required");
println!("{}", guess + 1);
```

Ⓐ `121`
Ⓑ `13`
Ⓒ `12`
Ⓓ It does not compile because shadowing cannot change a type.

Midterm Examination — Questions (continued)

Question 5 [2 Marks]

What does this code print?

```
let level = 3;

{
    let level = level + 4;
    println!("{}", level);
}

println!("{}", level);
```

(A) 7 then 7

(B) 7 then 3

(C) 3 then 7

(D) It does not compile because level is immutable.

Question 6 [2 Marks]

What does this code print?

```
let mut count = 0;

let value = loop {
    count += 1;
    if count == 3 {
        break count * 2;
    }
};

println!("{} {}", count, value);
```

(A) `3 3`

(B) `2 6`

(C) `6 3`

(D) `3 6`

Question 7 [2 Marks]

What does this code print?

```
let mut total = 0;

for number in 1..4 {
    total += number;
}

println!("{}", total);
```

(A) `6`

(B) `10`

(C) `3`

(D) `4`

Question 8 [2 Marks]

What does this complete program print?

```
fn send_back(mut text: String) -> String {
    text.push_str(" ready");
    text
}

fn main() {
    let message = String::from("train");
    let message = send_back(message);
    println!("{}", message);
}
```

(A) `train`

(B) It does not compile because a moved String cannot be returned.

(C) `train ready`

(D) It panics when send_back returns.

Question 9 [2 Marks]

What happens when this code is compiled and run?

```
let mut message = String::from("go");
let saved = &message;
message.push_str(" now");

println!("{}", saved);
```

(A) It does not compile because the immutable borrow is still needed when message is mutated.

(B) It prints `go`.

(C) It prints `go now`.

(D) It compiles, then panics.

Question 10 [2 Marks]

What does this code print?

```
let numbers = [10, 20, 30, 40];
let middle = &numbers[1..3];
println!("{} {}", middle.len(), middle[0]);
```

(A) `3 20`

(B) `2 10`

(C) `3 10`

(D) `2 20`

Question 11 [2 Marks]

What does this code print?

```
struct Position {
    row: u32,
    column: u32,
}

let start = Position { row: 2, column: 5 };
let next = Position { row: 4, ..start };

println!("{} {}", start.row, next.column);
```

(A) `4 5`

(B) It does not compile because start has been moved.

(C) `2 5`

(D) `2 4`

Question 12 [2 Marks]

What does this code print?

```
struct Counter {
    value: u32,
}

impl Counter {
    fn reset(&mut self) {
        self.value = 0;
    }
}

let mut counter = Counter { value: 8 };
counter.reset();

println!("{}", counter.value);
```

(A) `8`

(B) `0`

(C) It does not compile because methods cannot modify fields.

(D) It does not compile because reset consumes counter.

Question 13 [2 Marks]

Which statement about this enum definition is correct?

```
enum Delivery {
    Waiting,
    Assigned(String),
    Located { row: u32, column: u32 },
}
```

Ⓐ Every Delivery value stores all three variants at once.

Ⓑ Every variant must hold the same type of data.

Ⓒ Delivery::Waiting requires an empty String argument.

Ⓓ A Delivery value has one variant, and different variants may carry different data.

Question 14 [2 Marks]

What does this code print?

```
let original = 7;

let result = match original {
    1 => 10,
    other => other + 2,
};

println!("{} {}", original, result);
```

Ⓐ `7 9`

Ⓑ `9 9`

Ⓒ `7 10`

Ⓓ It does not compile because the match lacks an underscore arm.

Question 15 [2 Marks]

What does this code print?

```
let available: Option<u32> = Some(0);

if let Some(amount) = available {
    println!("count {}", amount);
} else {
    println!("missing");
}
```

Ⓐ `missing`

Ⓑ `count 1`

Ⓒ `count 0`

Ⓓ It does not compile because Some cannot contain zero.

Question 16 [2 Marks]

What does this code print?

```
let stops = vec![4, 8];

match stops.get(2) {
    Some(stop) => println!("{}", stop),
    None => println!("outside"),
}
```

Ⓐ `8`

Ⓑ `outside`

Ⓒ It panics because index 2 is out of bounds.

Ⓓ It does not compile because get requires a mutable vector.

Question 17 [2 Marks]

What does this code print?

```
let left = String::from("north");
let right = String::from("south");
let route = format!("{} to {}", left, right);

println!("{} | {}", left, route);
```

Ⓐ `north | north to south`

Ⓑ It does not compile because format! moves left.

Ⓒ `north to south | north to south`

Ⓓ `south | north to south`

Question 18 [2 Marks]

What does this code print? The character é occupies two bytes in UTF-8.

```
let word = String::from("é!");
let mut count = 0;

for _character in word.chars() {
    count += 1;
}

println!("{} {}", word.len(), count);
```

Ⓐ `2 2`

Ⓑ `3 3`

Ⓒ `2 3`

Ⓓ `3 2`

Question 19 [2 Marks]

What does this code print?

```
use std::collections::HashMap;
let mut seats = HashMap::new();
seats.insert("bus", 4);
seats.entry("tram").or_insert(6);
println!("{} {}", seats.len(), seats.get("tram").unwrap());
```

(A) `1 6`
(B) `2 0`
(C) `2 6`
(D) It panics because tram is not yet a key.

Question 20 [2 Marks]

What is printed to standard output before this code terminates? Ignore the panic diagnostic.

```
println!("begin");
let result: Result<u32, &str> = Err("closed");
let value = result.unwrap();
println!("value {}", value);
```

(A) `begin` then `value 0`
(B) `begin` only; the program then panics.
(C) Nothing; the program does not compile.
(D) `begin` then `value closed`

Section B: Code-reading questions — 20 questions, 3 marks each (60 marks).

Question 21 [3 Marks]

What does this complete program print?

```
fn main() {
    let readings = [3, 4, 0, 5, 6, 7, 0, 8];

    let mut current = 0;
    let mut longest = 0;
    let mut groups = 0;

    for reading in readings {
        if reading == 0 {
            current = 0;
        } else {
            if current == 0 {
                groups += 1;
            }

            current += 1;

            if current > longest {
                longest = current;
            }
        }
    }

    println!("{} {} {}", groups, longest, current);
}
```

(A) `2 3 1`

(B) `3 3 1`

(C) `3 6 6`

(D) `3 3 0`

Question 22 [3 Marks]

What does this complete program print?

```
fn main() {
    let mut number = 2403;
    let mut reversed = 0;
    let mut steps = 0;

    let last = loop {
        let digit = number % 10;
        number /= 10;
        steps += 1;

        if number == 0 {
            break digit;
        }

        reversed = reversed * 10 + digit;
    };

    println!("{} {} {}", reversed, last, steps);
}
```

(A) `3042 2 4`

(B) `34 2 3`

(C) `304 0 5`

(D) `304 2 4`

Question 23 [3 Marks]

What does this complete program print?

```
fn decorate(text: &mut String) -> usize {
    let before = text.len();
    text.push_str("!");

    before
}

fn main() {
    let mut label = String::from("box");
    let saved = label.clone();
    let first = decorate(&mut label);

    {
        let view = &label;
        println!("{} {}", view, first);
    }

    label.push_str(&saved);
    let second = decorate(&mut label);

    println!("{} {}", label, second);
}
```

(A) `box! 3` then `box!box! 7`

(B) `box! 4` then `box!box! 8`

(C) `box! 3` then `box!box!! 8`

(D) It fails to compile.

Question 24 [3 Marks]

What happens when this complete program is compiled and run?

```
fn select_name(mut name: String, shorten: bool) -> String {
    if shorten {
        name.clear();
        name.push_str("short");
    } else {
        name.push_str("-kept");
    }

    name
}

fn main() {
    let original = String::from("long-name");
    let backup = original.clone();
    let chosen = select_name(original, true);

    println!("{}", chosen);
    println!("{}", backup);
    println!("{}", original);
}
```

(A) It prints `short`, `long-name`, and `long-name` on successive lines.

(B) It prints `short` and `long-name`, then panics.

(C) It fails to compile because original was moved into select_name.

(D) It fails to compile because the function returns different String values.

Question 25 [3 Marks]

What does this complete program print? All characters are ASCII.

```
fn suffix(text: &str) -> &str {
    let bytes = text.as_bytes();
    let mut start = 0;

    for (index, &byte) in bytes.iter().enumerate() {
        if byte == b':' {
            start = index + 1;
        }
    }

    &text[start..]
}

fn main() {
    let mut path = String::from("a:bc:def");
    let end = suffix(&path);
    let size = end.len();

    println!("{}", end);

    path.push_str(":xy");
    println!("{} {}", suffix(&path), size);
}
```

Ⓐ `bc:def` then `xy 6`

Ⓑ `def` then `xy 2`

Ⓒ `def` then `xy 3`

Ⓓ It fails to compile because path is borrowed.

Question 26 [3 Marks]

What does this complete program print?

```
struct Wallet {
    coins: u32,
}

impl Wallet {
    fn send(&mut self, other: &mut Wallet, amount: u32) -> bool {
        if self.coins < amount {
            return false;
        }

        self.coins -= amount;
        other.coins += amount;
        true
    }
}

fn main() {
    let mut left = Wallet { coins: 9 };
    let mut right = Wallet { coins: 2 };
    let first = left.send(&mut right, 4);
    let second = left.send(&mut right, 6);
    let third = right.send(&mut left, 3);

    println!(
        "{} {} {} {} {}",
        left.coins, right.coins, first, second, third
    );
}
```

Ⓐ `8 3 true false true`

Ⓑ `2 9 true true true`

Ⓒ `5 6 true false false`

Ⓓ `8 3 true true true`

Question 27 [3 Marks]

What does this complete program print?

```
struct Parcel {
    address: String,
    weight: u32,
    express: bool,
}

fn main() {
    let first = Parcel {
        address: String::from("Oak"),
        weight: 4,
        express: false,
    };

    let mut second = Parcel {
        express: true,
        ..first
    };

    second.address.push_str(" St");
    second.weight += 2;

    println!("{} {}", first.weight, first.express);
    println!("{} {} {}", second.address, second.weight, second.express);
}
```

Ⓐ It fails to compile because no fields of first may be used.

Ⓑ `6 true` then `Oak St 6 true`

Ⓒ `4 false` then `Oak 6 true`

Ⓓ `4 false` then `Oak St 6 true`

Question 28 [3 Marks]

What does this complete program print?

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

impl Rectangle {
    fn widened(&self, extra: u32) -> Rectangle {
        Rectangle {
            width: self.width + extra,
            height: self.height,
        }
    }

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

fn main() {
    let original = Rectangle {
        width: 3,
        height: 4,
    };

    let mut expanded = original.widened(2);
    expanded.height += 1;
    println!("{} {}", original.area(), expanded.area());
}
```

Ⓐ `25 25`

Ⓑ `12 25`

Ⓒ `12 20`

Ⓓ It fails to compile because original was moved.

Question 29 [3 Marks]

What does this complete program print?

```
enum Action {
    Add(i32),
    Save,
    Undo,
}

fn main() {
    let actions = [
        Action::Add(4),
        Action::Save,
        Action::Add(7),
        Action::Undo,
        Action::Add(2),
        Action::Save,
    ];

    let mut value = 0;
    let mut saved = 0;
    let mut saves = 0;

    for action in actions {
        match action {
            Action::Add(amount) => value += amount,
            Action::Save => {
                saved = value;
                saves += 1;
            }
            Action::Undo => value = saved,
        }
    }

    println!("{} {} {}", value, saved, saves);
}
```

Ⓐ `13 13 2`

Ⓑ `6 4 2`

Ⓒ `2 2 1`

Ⓓ `6 6 2`

Question 30 [3 Marks]

What does this complete program print?

```
fn main() {
    let readings = [Some(3), Some(8), None, Some(5), Some(7)];
    let mut previous = None;
    let mut change = 0;
    let mut pairs = 0;

    for reading in readings {
        match reading {
            Some(value) => {
                if let Some(old) = previous {
                    change += value - old;
                    pairs += 1;
                }
                previous = Some(value);
            }
            None => previous = None,
        }
    }

    println!("{} {}", change, pairs);
}
```

Ⓐ `7 2`

Ⓑ `4 3`

Ⓒ `10 3`

Ⓓ `7 3`

Question 31 [3 Marks]

What does this complete program print?

```
fn discount(price: u32, coupon: Option<u32>) -> Option<u32> {
    match coupon {
        Some(amount) => {
            if amount > price {
                None
            } else {
                Some(price - amount)
            }
        }
        None => Some(price),
    }
}

fn main() {
    let mut total = 0;
    let mut rejected = 0;

    for coupon in [Some(3), Some(12), None, Some(10)] {
        match discount(10, coupon) {
            Some(price) => total += price,
            None => rejected += 1,
        }
    }

    println!("{} {}", total, rejected);
}
```

Ⓐ 17 2

Ⓑ 27 1

Ⓒ 17 1

Ⓓ 7 2

Question 32 [3 Marks]

What does this complete program print?

```
fn main() {
    let mut pending = vec![2, 4, 6];
    let mut total = 0;
    let mut taken = 0;

    loop {
        match pending.pop() {
            Some(value) => {
                total += value;
                taken += 1;
                if value == 6 {
                    pending.push(1);
                }
            }
            None => break,
        }
    }

    println!("{} {} {}", total, taken, pending.len());
}
```

(A) `12 3 0`

(B) `13 4 0`

(C) `13 4 1`

(D) `9 3 1`

Question 33 [3 Marks]

What does this complete program print?

```
fn main() {
    let mut values = vec![3, 1, 4, 2];
    let mut running = 0;

    for value in &mut values {
        running += *value;
        *value = running;
    }

    values.push(running);

    let mut count = 0;
    let mut total = 0;

    for value in &values {
        if *value >= 7 {
            count += 1;
            total += *value;
        }
    }

    println!("{} {} {}", values[1], count, total);
}
```

Ⓐ `1 0 0`

Ⓑ `4 2 18`

Ⓒ `4 3 30`

Ⓓ `4 3 28`

Question 34 [3 Marks]

What does this complete program print?

```
enum Item {
    Price(u32),
    Free,
    Limit(u32),
}

fn main() {
    let items = vec![
        Item::Price(6),
        Item::Free,
        Item::Limit(4),
        Item::Price(7),
        Item::Price(2),
    ];

    let mut limit = 10;
    let mut total = 0;
    let mut count = 0;

    for item in items {
        match item {
            Item::Price(price) => {
                if price <= limit {
                    total += price;
                    count += 1;
                }
            }
            Item::Free => count += 1,
            Item::Limit(value) => limit = value,
        }
    }

    println!("{} {}", total, count);
}
```

Ⓐ `8 3`

Ⓑ `15 4`

Ⓒ `8 2`

Ⓓ `2 2`

Question 35 [3 Marks]

What does this complete program print?

```
fn main() {
    let words = ["red", "", "blue", "green"];
    let mut line = String::new();
    let mut count = 0;

    for word in words {
        if word == "" {
            continue;
        }

        if count > 0 {
            line.push_str("/");
        }

        line.push_str(word);
        count += 1;
    }

    let saved = format!("[{}]", line);
    line.push_str("!");

    println!("{}", saved);
    println!("{} {}", line, count);
}
```

Ⓐ `[red//blue/green]` then `red//blue/green! 4`

Ⓑ `[red/blue/green!]` then `red/blue/green! 3`

Ⓒ `[red/blue/green]` then `red/blue/green! 3`

Ⓓ It fails to compile because format! moves line.

Question 36 [3 Marks]

What does this complete program print?

```
use std::collections::HashMap;

fn main() {
    let mut stock = HashMap::new();
    stock.insert(String::from("pen"), 5);
    stock.insert(String::from("pad"), 2);

    let orders = [("pen", 3), ("pad", 4), ("pen", 2), ("clip", 1)];
    let mut filled = 0;

    for (name, quantity) in orders {
        let available = stock.entry(String::from(name)).or_insert(0);

        if *available >= quantity {
            *available -= quantity;
            filled += 1;
        }
    }

    println!(
        "{} {} {}",
        stock.get("pen").unwrap(),
        stock.get("pad").unwrap(),
        filled
    );

    println!("{}", stock.len());
}
```

(A) `0 2 2` then `3`

(B) `0 -2 3` then `3`

(C) `0 2 2` then `2`

(D) `2 2 1` then `3`

Question 37 [3 Marks]

What does this complete program print?

```
use std::collections::HashMap;

fn main() {
    let mut counts = HashMap::new();

    for word in "up down up left down up".split_whitespace() {
        let count = counts.entry(word).or_insert(0);
        *count += 1;
    }

    counts.insert("up", 1);

    for word in ["left", "right", "up", "right"] {
        let count = counts.entry(word).or_insert(2);
        *count += 1;
    }

    let mut total = 0;

    for (_, value) in &counts {
        total += *value;
    }

    println!(
        "{} {} {}",
        counts.get("up").unwrap(),
        counts.get("right").unwrap(),
        total
    );
}
```

Ⓐ `4 2 10`

Ⓑ `2 2 8`

Ⓒ `4 4 12`

Ⓓ `2 4 10`

Question 38 [3 Marks]

What does this complete program print?

```
fn subtotal(texts: [&str; 3]) -> Result<i32, String> {
    let mut total = 0;

    for text in texts {
        let value = match text.parse::<i32>() {
            Ok(number) => Ok(number),
            Err(_) => Err(String::from("invalid")),
        }?;
        total += value;
        println!("added {}", value);
    }

    Ok(total)
}

fn main() {
    let mut saved = 10;

    match subtotal(["2", "bad", "5"]) {
        Ok(value) => saved += value,
        Err(_) => saved -= 1,
    }

    println!("saved {}", saved);
}
```

(A) `added 2` then `added 5` then `saved 17`

(B) `added 2` then `saved 9`

(C) `added 2` then `saved 11`

(D) It panics after printing `added 2`.

Question 39 [3 Marks]

What does this complete program print?

```
fn nonnegative(text: &str) -> Result<i32, String> {
    match text.parse::<i32>() {
        Ok(value) => {
            if value < 0 {
                Err(String::from("negative"))
            } else {
                Ok(value)
            }
        }
        Err(_) => Err(String::from("not a number")),
    }
}

fn main() {
    let mut total = 0;
    let mut rejected = 0;

    for text in ["4", "-2", "bad", "0", "3"] {
        match nonnegative(text) {
            Ok(value) => total += value,
            Err(_) => rejected += 1,
        }
    }

    println!("{} {}", total, rejected);
}
```

(A) `5 1`

(B) `7 3`

(C) `7 2`

(D) It panics when parsing bad.

Question 40 [3 Marks]

What happens when this complete program is compiled and run?

```
fn main() {
    let inputs = ["3", "bad", "4"];
    let mut total = 0;

    for input in inputs {
        let parsed = input.parse::<i32>();

        match &parsed {
            Ok(value) => println!("read {}", value),
            Err(_) => println!("retry needed"),
        }

        let value = parsed.expect("number required");
        total += value;
        println!("total {}", total);
    }

    println!("done {}", total);
}
```

(A) It prints `read 3`, `total 3`, `retry needed`, `total 3`, `read 4`, `total 7`, and `done 7`.

(B) It prints `read 3`, `total 3`, and `retry needed`, then panics.

(C) It prints `read 3` and `total 3`, then exits successfully.

(D) It fails to compile because matching &parsed moves parsed.