Week 10 Weekly Exercises

Objectives

- Show how the final exam works.
- Prac exam from 22T3, followed by real exam from 22T3
- None of these are marked, but obviously will be in the real exam
- The 23T1 exam will likely have 7 questions instead of 8

Activities To Be Completed

The following is a list of all the activities available to complete this week...

The following practice activities are optional and are not **marked, or required** to be completed for the week.

- **Exam Preamble**
- **Q1**
- **Q2**
- **Q3**
- **Q4**
- **Q5**
- **Q6**
- **Q7**
- **Q8**
- **22T3 Q1: Theory (10 marks)**
- **22T3 Q2: Practical (10 marks)**
- **22T3 Q3: Practical (10 marks)**
- **22T3 Q4: Theory (10 marks)**
- **22T3 Q5: Theory (10 marks)**
- **22T3 Q6: Practical (10 marks)**
- **22T3 Q7: Theory (10 marks)**
- **22T3 Q8: Theory (10 marks)**

Preparation

Before attempting the weekly exercises you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this week's exercises called `lab10`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab10
$ cd lab10
$ 6991 fetch lab 10
```

Or, if you're not working on CSE, you can download the provided code as a tar file.

(OPTIONAL) EXERCISE:

Exam Preamble

Exam Condition Summary

- This exam is "Open Book"
- Joint work is NOT permitted in this exam
- You are NOT permitted to communicate (email, phone, message, talk) with anyone during this exam, except for the COMP6991 staff via cs6991.exam@cse.unsw.edu.au
- The exam paper is confidential, sharing it during or after the exam is prohibited.
- You are NOT permitted to submit code that is not your own
- You may NOT ask for help from online sources.
- Even after you finish the exam, on the day of the exam, do NOT communicate your exam answers to anyone. Some students have extended time to complete the exam.
- Do NOT place your exam work in any location, including file sharing services such as Dropbox or GitHub, accessible to any other person.
- Your zpass should NOT be disclosed to any other person. If you have disclosed your zpass, you should change it immediately.
- **The use of AI assistants is strictly prohibited in this exam. This includes services such as Github Copilot and OpenAI ChatGPT.**
  As a remark, I have personally pre-emptively fed the entire exam through ChatGPT. It has a very distinct linguistic flavour to its responses, and often incredibly confidently gives an entirely nonsensical answer. Please don't mistakenly believe you won't be caught if you opt to violate these conditions.

Deliberate violation of these exam conditions will be referred to Student Integrity Unit as serious misconduct, which may result in penalties up to and including a mark of 0 in COMP6991 and exclusion from UNSW.

- You are **allowed** to use any resources from the course during the exam.
- You are **allowed** to use small amounts of code (< 10 lines) of general-purpose code (not specific to the exam) obtained from a site such as Stack Overflow or other publicly available resources. You should attribute the source of this code clearly in an accompanying comment.

Exam submissions will be checked, both automatically and manually, for any occurrences of plagiarism.

By starting this exam, as a student of The University of New South Wales, you do solemnly and sincerely declare that you have not seen any part of this specific examination paper for the above course prior to attempting this exam, nor have any details of the exam's contents been communicated to you. In addition, you will not disclose to any University student any information contained in the abovementioned exam for a period of 24 hrs after the exam. Violation of this agreement is considered Academic Misconduct and penalties may apply.

For more information, read the UNSW Student Code, or contact the Course Account.

- This exam comes with starter files.
- You will be able to commence the exam and fetch the files once the exam commences.
- You may complete the exam questions using any platform you wish (VLab, VSCode, etc). You should ensure that the platform works correctly.
- You may submit your answers, using the give command provided below each question.
- You can use give to submit as many times as you wish. Only the last submission will be marked.
- Do NOT leave it to the deadline to submit your answers. Submit each question when you finish working on it.
- Please make sure that you submit all your answers at the conclusion of the exam - running the autotests does not automatically submit your code.
- Autotests are available for all practical questions to assist in your testing. You can use the command: `6991 autotest`
- Passing autotests does not guarantee any marks. Remember to do your own testing!
- No marks are awarded for commenting - but you can leave comments for the marker to make your code more legible as needed

Language Restriction

- All programming questions must be answered entirely in Rust. You may not use any other programming languages.

- You are not permitted to use third-party crates other than the standard library (std).

Fit to Sit

By sitting or submitting an assessment on the scheduled assessment date, a student is declaring that they are fit to do so and cannot later apply for Special Consideration.

If, during an exam a student feels unwell to the point that they cannot continue with the exam, they should take the following steps:

1. Stop working on the exam and take note of the time
2. Contact us immediately, using cs6991.exam@cse.unsw.edu.au, and advise us that you are unwell
3. Immediately submit a Special Consideration application saying that you felt ill during the exam and were unable to continue
4. If you were able to advise us of the illness during the assessment (as above), attach screenshots of this conversation to the Special Consideration application

Technical Issues

If you experience a technical issue, you should take the following steps:

1. If your issue is with the connection to CSE, please follow the following steps:
    - **If you are using VLab:** Try exiting VLAB and reconnecting again - this may put you on a different server, which may improve your connection. If you are still experiencing problems, you can try changing how you connect to the CSE servers. Consider:
        - By using VSCode (with SSH-FS extension): https://www.cse.unsw.edu.au/~learn/homecomputing/vscode/
        - By using SSH: https://taggi.cse.unsw.edu.au/FAQ/Logging_In_With_SSH/
    - **If you are using VSCode remote-ssh:** Try disconnecting VSCode, and then changing the URL from `vscode.unsw.edu.au` to `vscode2.unsw.edu.au`.
    - **If you are using SSH:** Try disconnecting SSH and reconnecting again.
2. If things are still NOT working, take screenshots of as many of the following as possible:
    - error messages
    - screen not loading
    - timestamped speed tests
    - power outage maps
3. Contact should be made immediately to advise us of the issue at cs6991.exam@cse.unsw.edu.au
4. A Special Consideration application should be submitted immediately after the conclusion of the assessment, along with the appropriate screenshots.

(OPTIONAL) EXERCISE:
Q1

Question 1

Q1.1 (2 marks)

A C programmer who is starting to learn Rust has asked: "Aren't match statements just complicated if statements?". Give a specific example of a situation where you believe a match statement would significantly improve code quality, instead of a series of if/else statements.

Q1.2 (2 marks)

The following Rust code fails to compile, but equivalent code in other popular programming languages (e.g. C, Java, Python) compiles and/or works correctly. Explain what issue(s) prevent the Rust compiler from building this code, and the philosophy behind this language decision.

```
struct Coordinate {
    x: i32,
    y: i32,
};

let coord1 = Coordinate {x: 1, y: 2};
let coord2 = coord1;
let coord_sum = Coordinate { x: coord1.x + coord2.x, y: coord1.y + coord2.y };
```

Q1.3 (3 marks)

In other languages, the operation: `"first_string" + "second_string"` produces a new string, `"first_stringsecond_string"`. This particular operation **does not** work in Rust.

1. Why does Rust not implement this operation on the `&str` type?
2. Would it be possible for the Rust language developers to implement this? What rust feature would they use to implement it?
3. Do you think the Rust language developers should implement this operation? Give one reason to justify your answer.

Q1.4 (3 marks)

Rust beginners have posted some questions on a programming forum:

1. How can I turn an owned value into a shared borrow?
2. How can I turn a shared borrow into an exclusive borrow!
3. Why am I allowed to turn an exclusive borrow into a shared borrow?

Provide a short answer to each question. Importantly, note that some questions might ask for something that is not possible (in which case, you should say so and explain why).

(OPTIONAL) EXERCISE:

Q2

In this activity, you will be building a small text searching system. It should search a large string for sentences that contain a particular search term. Another function will then look through all the search results to determine how often each sentence was found.

You have been given starter code which does not yet compile. Your task is to fill in both `todo!()` statements, as well as to add lifetimes where required in order to build your code.

You are not permitted to change the return type of functions, the names of structs, or the types of structs. You may also not change the main function, and you should expect that the main function could be changed during testing. You will, however, have to add lifetimes to existing types in order to successfully compile your code.

This is an example of the expected behaviour:

```
$ 6991 cargo run test_data/test_data.txt
    Finished dev [unoptimized + debuginfo] target(s) in 0.36s
    Running `target/debug/prac_q2`
there
very
prove
the universe
  Ctrl-D
Found 1 results for 'there'.
Found 9 results for 'very'.
Found 1 results for 'prove'.
Found 11 results for 'the universe'.
'8 billion years ago, space expanded very quickly (thus the name "Big Bang")' occured 1 times.
'According to the theory the universe began as a very hot, small, and dense superforce (the mix of the four
fundamental forces), with no stars, atoms, form, or structure (called a "singularity")' occured 2 times.
'Amounts of very light elements, such as hydrogen, helium, and lithium seem to agree with the theory of the Big
Bang' occured 1 times.
'As a whole, the universe is growing and the temperature is falling as time passes' occured 1 times.
'Because most things become colder as they expand, scientists assume that the universe was very small and very
hot when it started' occured 2 times.
'By measuring the redshift, scientists proved that the universe is expanding, and they can work out how fast
the object is moving away from the Earth' occured 2 times.
'Cosmology is the study of how the universe began and its development' occured 1 times.
'Other observations that support the Big Bang theory are the amounts of chemical elements in the universe'
occured 1 times.
'The Big Bang is a scientific theory about how the universe started, and then made the stars and galaxies we
see today' occured 1 times.
'The Big Bang is the name that scientists use for the most common theory of the universe, from the very early
stages to the present day' occured 2 times.
'The more redshift there is, the faster the object is moving away' occured 1 times.
'The most commonly considered alternatives are called the Steady State theory and Plasma cosmology, according
to both of which the universe has no beginning or end' occured 1 times.
'The most important is the redshift of very far away galaxies' occured 1 times.
'These electromagnetic waves are everywhere in the universe' occured 1 times.
'This radiation is now very weak and cold, but is thought to have been very strong and very hot a long time
ago' occured 1 times.
'With very exact observation and measurements, scientists believe that the universe was a singularity
approximately 13' occured 2 times.
```

(OPTIONAL) EXERCISE:

Q3

In this question, your task is to complete two functions, and make them generic: `zip_tuple` and `unzip_tuple`. Right now, the `zip_tuple` function takes a `Vec<Coordinate>` and returns a tuple: `(Vec<i32>, Vec<i32>)`. The `unzip_tuple` function performs the inverse of this.

This code currently does not compile, because `q3_lib` (i.e. `lib.rs`) does not know what the type of `Coordinate` is. Rather than telling the functions what type `Coordinate` is, in this exercise we will make the functions generic, such that it works for both `q3_a` (i.e. main_1.rs) and `q3_b` (i.e. main_2.rs). This is to say, `tuple_unzip` should work for any `Vec<T>` such that `T` implements `Into` into a 2-tuple of any 2 types, and `tuple_zip` should work for any `Vec<(T, U)>` such that `(T, U)` implements `Into` into any type.

Once you have modified your function signatures for `tuple_unzip` and `tuple_zip`, you should find that the only *concrete* type appearing within the signature is `Vec`. In other words, the functions should work for **any type** which can be created from a 2-tuple and which can be converted into a 2-tuple.

(OPTIONAL) EXERCISE:

Q4

Q4.1 (2 marks)

Steve is writing some Rust code for a generic data structure, and creates a (simplified) overall design alike the following:

```
struct S {
    // some fields...
}

impl S {
    fn my_func<T>(value: T) {
        todo!()
    }
}
```

He soon finds that this design is not sufficient to model his data structure, and revises the design as such:

```
struct S<T> {
    // some fields...
}

impl<T> S<T> {
    fn my_func(value: T) {
        todo!()
    }
}
```

Give an example of a data-structure that Steve could be trying to implement, such that his first design would not be sufficient, and instead his second design would be required for a correct implementation. Furthermore, explain why this is the case.

Q4.2 (3 marks)

Emily is designing a function that has different possibilities for the value it may return. She is currently deciding what kind of type she should use to represent this property of her function.

She has narrowed down three possible options:

1. An enum
2. A trait object
3. A generic type (as `fn foo(...) -> impl Trait`)

For each of her possible options, explain one possible advantage and one possible disadvantage of that particular choice.

Q4.3 (5 marks)

Rust's macro system offers an extremely flexible method for code generation and transfiguring syntax, but this language feature comes with certain costs. Identify 3 downsides to the inclusion, design, or implementation of Rust's macro system.

(Note that your downsides may span any amount and combination of the categories above. e.g. you could write all 3 on just one category, or one on each, or anything in-between.)

(OPTIONAL) EXERCISE:

Q5

Q5.1 (3 marks)

In many other popular programming languages, mutexes provide `lock()` and `unlock()` methods which generally do not return any value (i.e. `void`).

What issues could this cause?

How does Rust differently implement the interface of a `Mutex`, and what potential problems does that help solve?

Q5.2 (2 marks)

In Rust, locking a `Mutex` returns a `Result`, instead of simply a `MutexGuard`. Explain what utility this provides, and why a programmer might find this important.

Q5.3 (3 marks)

While reviewing someone's code, you find the following type: `Box<dyn Fn() -> i32 + Send>`.

Explain what the `+ Send` means in the code above?

Explain one reason you might need to mark a type as `Send`, and what restrictions apply when writing a closure that must be `Send`.

Q5.4 (2 marks)

Your friend tells you they don't need the standard library's channels, since they've implemented their own alternative with the following code:

```
use std::collections::VecDeque;
use std::sync::Mutex;
use std::sync::Arc;
use std::thread;

#[derive(Clone, Debug)]
struct MyChannel<T> {
    internals: Arc<Mutex<VecDeque<T>>>
}

impl<T> MyChannel<T> {
    fn new() -> MyChannel<T> {
        MyChannel {
            internals: Arc::new(Mutex::new(VecDeque::new()))
        }
    }
    fn send(&mut self, value: T) {
        let mut internals = self.internals.lock().unwrap();
        internals.push_front(value);
    }

    fn try_recv(&mut self) -> Option<T> {
        let mut internals = self.internals.lock().unwrap();
        internals.pop_back()
    }
}

fn main() {
    let mut sender = MyChannel::<i32>::new();
    let mut receiver = sender.clone();
    sender.send(5);
    thread::spawn(move || {
        println!("{:?}", receiver.try_recv())
    }).join().unwrap();
}
```


Identify a use-case where this implementation would not be sufficient, but the standard library's channel would be.

Furthermore, explain why this is the case.

(OPTIONAL) EXERCISE:

Q6

The "Read Copy Update" pattern is a common way of working with data when many sources need to be able to access data, but also to update it. It allows a user to access a value whenever it's needed, achieving this by never guaranteeing that the data is always the latest copy. In other words, there will always be something, but it might be slightly old. In some cases, this trade-off is one that's worth making.

In this task, you will be implementing a small RCU data-structure. You should ensure that:

- Multiple threads are able to access a given piece of data.
- Threads can pass a closure to the type which updates the data.
- When created, the RCU type starts at generation 0. Every time it is updated, that counter is increased by one.

You have been given some starter code for the type `RcuType<T>`, including some suggested fields, and the required interface. Ensure you first understand the requirements of this task, and then implement the methods described in the starter code.

(OPTIONAL) EXERCISE:
Q7

Q7.1 (5 marks)

Gavin writes a blog post critical of Rust, especially with respect to `unsafe`. In his blog post, he claims that it's not possible to have any confidence in the overall safety of a Rust program since "even if you only write safe Rust, most standard functions you call will have unsafe code inside them".

1. State to what extent you agree with Gavin's claim.
2. Give at least three arguments that support your conclusion.

Q7.2 (5 marks)

Hannah writes a Rust program that intends to call some C code directly through FFI. Her C function has the following prototype:

```
int array_sum(int *array, int array_size)
```

and the following implementation:

```
int array_sum(int *array, int array_size) {
    int sum = 0;
    for (int i = 0; i < array_size; i++) { sum += array[i]; }
    return sum;
}
```

Note that you can assume that this C code is written entirely correctly, and the below `extern "C"` block is an accurate translation of the C interface.

Her Rust code is currently written as follows:

```rust
use std::ffi::c_int;

#[link(name = "c_array")]
extern "C" {
    fn array_sum(array: *mut c_int, array_size: c_int) -> c_int;
}

fn test_data() -> (*mut c_int, c_int) {
    let size = 10;
    let array = vec![6991; size].as_mut_ptr();
    (array, size as c_int)
}

fn main() {
    let sum = {
        let (array, size) = test_data();

        // Debug print:
        let message = format!("Calling C function with array of size: {size}");
        println!("{message}");

        unsafe { array_sum(array, size) }
    };

    println!("C says the sum was: {sum}");
}
```

She expects that if she runs her code, it should print that the C code summed to `69910`. To her surprise, she runs the program and finds the following:


```
$ 6991 cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/ffi`
Calling C function with array of size: 10
C says the sum was: -2039190222
```


Hannah correctly concludes that there must be a problem with her Rust code.

1. Identify the issue that is causing the program to misbehave.
2. Describe a practical solution Hannah could use to fix the bug.
3. Explain why Rust wasn't able to catch this issue at compile-time.

(OPTIONAL) EXERCISE:

Q8

The final question of the exam will be a more open-ended question which will ask you to perform some analysis or make an argument. Your argument will be judged alike an essay (are your claims substantiated by compelling arguments). Remember that you **will not** get any marks for blindly supporting Rust.

A friend of yours has just read this article, and thinks that it means they shouldn't learn Rust.

Read through the article, and **discuss** the following prompt:

Rust is not worth learning, as explained by this article.

The overall structure of your answer is **not** marked. For example, your answer may include small paragraphs of prose accompanied by dot-points, or could instead be posed as a verbal discussion with your friend. Regardless of the structure / formatting you choose, the **substance** of what you write is the most important factor, and is what will determine your overall mark for this question.

NOTE:

From UNSW's Glossary of Task Words:

Discuss	Investigate or examine by argument. Examine key points and possible interpretations, sift and debate, giving reasons for and against. Draw a conclusion.

(OPTIONAL) EXERCISE:

22T3 Q1: Theory (10 marks)

Q1.1 (3 marks)

Question:

1. Explain the difference between the `Option` and `Result` types. *(1 mark)*
2. Give an example of where each might be used. *(2 marks)*

Write your answer in `exam_q1/q1_1.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q1_1 q1_1.txt
```


Q1.2 (2 marks)

The following Rust code fails to compile, but equivalent code in other popular programming languages (e.g. C, Java, Python) compiles and/or works correctly.


```rust
let i: u32 = 32;
let j: i32 = -1;
println!("{}", i + j);
```


Question:

1. Explain what issue(s) prevent the Rust compiler from building this code *(1 mark)*.
2. Explain the philosophy behind this language decision *(1 mark)*.

Write your answer in `exam_q1/q1_2.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q1_2 q1_2.txt
```

Q1.3 (2 marks)

Your friend has asked you to teach them Rust. They think a great place to begin would be your teaching them to write their own implementation of a doubly-linked list (i.e. a list in which each node is stored on the heap, with references to the previous node and the next node).

Question:

Give two reasons to explain why this is not a problem well-suited for Rust. *(1 mark per reason)*

Write your answer in `exam_q1/q1_3.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q1_3 q1_3.txt
```

Q1.4 (3 marks)

A COMP6991 student has decided to build their own operating system in Rust. When implementing the ability to open and read from files, they design these functions based on how Linux deals with files.

```rust
/// This function takes a path to a file; and a mutable reference to a `usize`.
/// If the file at `file_path` can be opened, the function will write a unique
/// `file_id` to the given mutable reference, and return 0. If the file cannot
/// be opened, the function will return an error code.
fn open_file(file_path: &str, file_id: &mut usize) -> usize;

/// This`function takes a `file_id` the user has already obtained, as well as a
/// `buffer` to write bytes to, and a `max_read_size`. The function tries to
/// copy `max_read_size` bytes into `buffer`. It returns the actual number of
/// bytes read. If there is an error, it returns a negative error code.
fn read_file(file_id: i32, buffer: &[u8], max_read_size: usize) -> i32;
```

Question:

Identify three issues in this students plan (either with their design, or their ability to implement their plan in Rust), and three accompanying Rust features that could be used to fix them. *(1 mark per issue+fix)*

Write your answer in `exam_q1/q1_4.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q1_4 q1_4.txt
```

(OPTIONAL) EXERCISE:

22T3 Q2: Practical (10 marks)

In this activity, you will be finding the differences between two rolls of people. A COMP6991 tutor has collected two rolls of people, and wants to know who's unique to the first one; who's unique to the second one; and who is on both lists.

You will be given two string references, called `left` and `right`. On each line is the name of a person. For every person on the left roll, you should return either `DiffResult::LeftOnly` or `DiffResult::Both` containing a string reference to that line, depending on whether they're in the right list also. For every person on the right roll, you should return `DiffResult::RightOnly` containing a reference to that line if they are not in the left roll. You can assume that the people on any *single* roll are unique. That is, you won't see two of the same person on the left roll, nor two of the same person on the right roll.

Before returning, you should sort your list of differences alphabetically by their names. Note that since `DiffResult` derives `PartialOrd + Ord`, you should simply be able to call `.sort()` on your final `Vec`.

You have been given starter code which does not yet compile. Your task is to fill in the `todo!()` statement, as well as to modify the lifetimes where required in order to build your code.

You are **not permitted** to change the return type of functions, the names of structs, or the types of structs. You may also not change the main function, and you should expect that the main function will be changed during testing. Specifically, the main function could be changed to extract `DiffResult::RightOnly` or `DiffResult::Both`. You will, however, have to add or modify lifetimes to existing types in order to successfully compile your code.

This is an example of the expected behaviour:

```
$ 6991 cargo run test_data/test_data.txt
    Finished dev [unoptimized + debuginfo] target(s) in 0.36s
     Running `target/debug/prac_q2`
# Roll 1
Tom
Shrey
Zac
Hailey
Toby
Barry
Netanya

# Roll 2
Tom
Shrey
Hailey
Ctrl-D
Left Only: Barry
Left Only: Netanya
Left Only: Toby
Left Only: Zac
```

Write your answer in `exam_q2/src/lib.rs`.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 6991 autotest
```


When you are finished working on your answer, submit your work with `give`:

```
$ give cs6991 exam_q2 lib.rs
```

(OPTIONAL) EXERCISE: 22T3 Q3: Practical (10 marks)

In this task, you will build a data-structure, called a "DBMap". This type will wrap a Vec of tuples, of the form `(key, value)`. Currently, the type only works for tuples of the form `(i32, &'static str)`, however you will need to modify this so it works for any tuples where the key is `Eq`. Your task is to make this struct generic over all valid types for both its keys and values, then to implement a method on this data-structure.

The method you will implement is called `merge`. It should take ownership of two `DBMaps` with the same type of key, and then return a new DBMap with its values being tuples. To describe the algorithm merge uses, we will call one of the `DBMaps` `self`, and one of them `other`.

To create the new `DBMap`, you should iterate over each element in `self`. We will call these `key` and `value` You should then try to find the first element in `other` with an equal key. We will call that `other_value`. You should insert `(key, (value, Some(other_value)))` into the new `DBMap`. If you cannot find a matching value in `other`, you should insert `(key, (value, None))` into the new `DBMap`.

Your implementation should be generic, such that the key is any type which supports equality checking; and the value is any type.

Your implementation should compile with both the main functions provided, however you should assume that more main functions may be tested during marking.

You may assume that any **one individual `DBMap`** will be comprised of totally unique keys.

An example of a merge is shown below:

Stock Prices

Key	Value
Apples	3.5
Pears	4.5
Caviar	200

Stock Quantity

Key	Value
Pears	50
Apples	100
Peaches	200

Merge of Stock Prices and Quantity

Key	Value
Apples	(3.5, Some(100))
Pears	(4.5, Some(50))
Caviar	(200, None)

For example,

```
$ 6991 cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/exam_q3`
#1: Max Verstappen (Red Bull Racing)
#3: Daniel Riccardo (None)
#4: Lando Norris (McLaren)
#5: Sebastian Vettel (None)
#6: Nicholas Latifi (None)
#7: Kimi Räikkönen (None)
#9: Nikita Mazepin (None)
#11: Sergio Pérez (Red Bull Racing)
```

Write your answer in `exam_q3/src/lib.rs`.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 6991 autotest
```


When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q3 lib.rs
```

(OPTIONAL) EXERCISE:

22T3 Q4: Theory (10 marks)

Q4.1 (2 marks)

Therese is writing a library to help sell her car. The library defines a `Car` trait as follows:

```
trait Car {
    fn get_price(&self) -> u32;
}
```

Users of the library will create their own structs which represent specific cars by implementing the `Car` trait. As seen in the trait definition, all `Cars` have a price.

Therese wants to write a function to total the cost of all the cars in a slice. As she wants this to work for any models of `Car`, she considers two potential approaches:

```
fn get_total_price<C: Car>(cars: &[C]) -> u32;
```

and

```
fn get_total_price(cars: &[Box<dyn Car>]) -> u32;
```

Question:

1. Explain the difference between the two approaches. *(1 mark)*

2. Give one reason why Therese might choose each approach. *(1 mark)*

Write your answer in `exam_q4/q4_1.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q4_1 q4_1.txt
```

Q4.2 (2 marks)

Daniel is trying to design a function that counts the number of whitespace characters in some text. He starts with the first design...

```
fn number_of_whitespace_chars(string: String) -> usize {
    string.chars()
        .filter(char::is_whitespace)
        .count()
}
```

... and submits this for code review. Another software engineer on his team suggests the following change...

```
fn number_of_whitespace_chars(string: &str) -> usize {
    string.chars()
        .filter(char::is_whitespace)
        .count()
}
```

... which Daniel accepts and resubmits. Finally, a senior engineer suggests a further change:

```
fn number_of_whitespace_chars<T: AsRef<str>>(t: T) -> usize {
    t.as_ref()
        .chars()
        .filter(char::is_whitespace)
        .count()
}
```


Question:

1. Why is the first suggested change an improvement on Daniel's original design? *(1 mark)*
2. Why is the second suggested change an improvement on the second design? *(1 mark)*

Write your answer in `exam_q4/q4_2.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q4_2 q4_2.txt
```

Q4.3 (3 marks)

Olivia is working on a roll-call system to help take attendance at a class she teaches. She writes the following code:

```rust
trait IntoRollCall {
    fn into_roll_call(self) -> String;
}

struct UnswStudent {
    name: String,
    zid: u32,
}

impl IntoRollCall for UnswStudent {
    fn into_roll_call(self) -> String {
        let Self { name, zid } = self;
        format!("{name} z{zid}")
    }
}

fn call_roll(students: Vec<Box<dyn IntoRollCall>>) {
    for student in students.into_iter() {
        println!("{}", student.into_roll_call());
    }
}

fn main() {
    call_roll(vec![
        Box::new(UnswStudent { name: String::from("Alice"),   zid: 5000000 }),
        Box::new(UnswStudent { name: String::from("Bertie"),  zid: 5000001 }),
        Box::new(UnswStudent { name: String::from("Candice"), zid: 5000002 }),
    ]);
}
```


Although there is currently only one implementor of `IntoRollCall`, she plans to soon add more implementors for other schools.

However, when she attempts to compile the code, she gets a confusing error:


```
$ rustc my_program.rs
error[E0161]: cannot move a value of type `dyn IntoRollCall`
  --> my_program.rs:19:24
   |
19 |         println!("{}", student.into_roll_call());
   |                        ^^^^^^^^^^^^^^^^^^^^^^^^ the size of `dyn IntoRollCall` cannot be statically
determined

error: aborting due to previous error

For more information about this error, try `rustc --explain E0161`.
```


Question:

1. Explain the reasoning behind this error message, with specific reference to the requirements of trait objects. *(2 marks)*
2. Identify a simple solution to fix Olivia's program. *(1 mark)*

Write your answer in `exam_q4/q4_3.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q4_3 q4_3.txt
```

Q4.4 (3 marks)

Patel does not think Rust's generics system is that special. He claims he can write his own macro which does everything that Rust's generics system can do.

His macro, and an example of him using it, is shown below.

```
macro_rules! generic {
    ($($name:ident = $type:ty),+; fn $fn_name:ident($($arg_name:ident: $arg_type:ty),* ) -> $return_type:ty
$blk:block) => {
        $(type $name = $type);+;

        fn $fn_name($($arg_name: $arg_type),*) -> $return_type {
            $blk
        }
    }
}

generic!(I = i32, O = String; fn add(a: I, b: I) -> O {
    format!("{}", a + b)
});

fn main() {
    let a: i32 = 1;
    let b: i32 = 1;
    println!("{}", add(a, b));
}
```

Question:

Explain to Patel three features of Rust's generics system which his macro does not implement, with explicit reference to an example of how his system would have to implement it. *(1 mark per feature)*

Write your answer in `exam_q4/q4_4.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q4_4 q4_4.txt
```

(OPTIONAL) EXERCISE:

22T3 Q5: Theory (10 marks)

Q5.1 (3 marks)

The following code attempts to creates multiple threads which each increment an `i32` protected by a `Mutex` by one.

```
use std::thread;
use std::sync::Mutex;

fn main() {
    let mutex: Mutex<i32> = Mutex::new(0);

    thread::scope(|scope| {
        for _ in 0..3 {
            let mut i = mutex.lock().unwrap();
            scope.spawn(move || {
                *i += 1;
            });
        }
    });

    println!("{}", *mutex.lock().unwrap());
}
```

However, there is a mistake in this code. Fortunately, it is caught by the Rust compiler instead of causing a crash or undefined behaviour at runtime.

Question:

1. Explain how the Rust compiler knows (statically, at compile time) that `i` cannot be sent across threads. *(1 mark)*
2. Explain how you would change the code to compile, while maintaining the behaviour that each thread increments the `i32` by exactly one. *(1 mark)*
3. Explain why your change fixes the issue. *(1 mark)*

Write your answer in `exam_q5/q5_1.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q5_1 q5_1.txt
```

Q5.2 (2 marks)

Question:

1. Identify a type that should never be marked as `Sync`. *(1 mark)*
2. Explain why that type should never be marked as `Sync`. *(1 mark)*

Write your answer in `exam_q5/q5_2.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q5_2 q5_2.txt
```


Q5.3 (3 marks)

Zain claims on the course forum that Rust makes concurrency impossible to get wrong.

However, the next post shows a student struggling with their concurrent code:

```rust
use std::thread;
use std::sync::{Mutex, Arc};

fn main() {
    let mutex: Arc<Mutex<Vec<i32>>> = Arc::new(Mutex::new(Vec::new()));

    for _ in 0..3 {
        let mutex_clone = mutex.clone();
        thread::spawn(move || {
            {
                // Push 1 to the end of the vec...
                let mut vector = mutex_clone.lock().unwrap();
                vector.push(1);
            }

            {
                // ... then increment that element by 1.
                let mut vector = mutex_clone.lock().unwrap();
                let index = vector.len() - 1;
                vector[index] += 1;
            }
        });
    }

    // NOTE: this code makes it work better for some reason???
    for _ in 0..2000 {}

    println!("{:?}", *mutex.lock().unwrap());
}
```

Question:

1. Explain two concurrency issues with the student's code. *(1 mark per issue)*
2. Discuss to what extent you agree with Zain's claim. *(1 mark)*

Write your answer in `exam_q5/q5_3.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q5_3 q5_3.txt
```

Q5.4 (2 marks)

Question:

1. Analyse two Rust features which the Rayon crate is able to take advantage of in order to provide static (i.e. compile-time) guarantees as to the safety of a parallel computation. Ensure to explain both the Rust feature, and *how* it allows Rayon to provide such guarantees. *(1 mark per feature)*

Write your answer in `exam_q5/q5_4.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q5_4 q5_4.txt
```

(OPTIONAL) EXERCISE:

22T3 Q6: Practical (10 marks)

In this exercise, we will be making some mathematical operations occur in parallel. You have been given the code to calculate the factors of a number, and to find common factors between two numbers. This code is quite slow, as it only executes on a single thread.

Your task is to parallelise the code in your `main` function. You can use any concurrency tools within the standard library, but you **must not** use Rayon, or any other non-standard crates. You must not change any code outside the `main.rs` file.

You will receive full marks if you can ensure that (theoretically), all calls to `get_factors` could run simultaneously; and seperately that all calls to `get_common_factors` could run simultaneously.

You should **not** put a constant limit on the number of threads you spawn (i.e., do not use thread-pooling). We suggest that you spawn a new thread for each `get_factors` call, and a new thread for each `get_common_factors` call.

Write your answer in `exam_q6/src/main.rs`.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 6991 autotest
```

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q6 main.rs
```

(OPTIONAL) EXERCISE:

22T3 Q7: Theory (10 marks)

Q7.1 (5 marks)

Larry is trying to implement a difficult data structure in Rust, and is running into some issues with the borrow checker.

Steven, noticing Larry's struggle, offers some unsolicited advice: "Just use `unsafe`, it disables the borrow checker and the rest of the other safety checks."

Stephanie overhears the advice and retorts: "It's more complicated than that. Maybe you should rethink your ownership model before resorting to `unsafe`."

Question:

1. Which safety checks does `unsafe` code opt-out of? *(1 mark)*
2. How and why might `unsafe` be a useful tool for resolving borrow-checking issues when writing, for example, a complex data structure in Rust? *(2 marks)*
3. Explain why using `unsafe` is not always the optimal solution in a case like this, and what procedures might be important to attempt to validate its soundness? *(2 marks)*

Write your answer in `exam_q7/q7_1.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q7_1 q7_1.txt
```

Q7.2 (5 marks)

The following code attempts to write an (inefficient, but simple) implementation of a `Mutex` using `unsafe`.


```rust
use std::{cell::UnsafeCell, sync::Mutex, ops::{Deref, DerefMut}};

pub struct MyMutex<T> {
    data: UnsafeCell<T>,
    is_locked: Mutex<bool>,
}

impl<T> MyMutex<T> {
    pub fn new(data: T) -> Self {
        Self {
            data: UnsafeCell::new(data),
            is_locked: Mutex::new(false),
        }
    }

    pub fn lock<'lock>(&'lock self) -> MyGuard<'lock, T> {
        loop {
            let mut is_locked = self.is_locked.lock().unwrap();

            if !*is_locked {
                // we now hold the lock!
                *is_locked = true;

                return MyGuard { mutex: self };
            }
        }
    }
}

// Safety: Mutexes are designed to be used on multiple threads,
//         so we can send them to other threads
//         and share them with other threads.
unsafe impl<T> Send for MyMutex<T> {}
unsafe impl<T> Sync for MyMutex<T> {}

pub struct MyGuard<'lock, T> {
```

```
    mutex: &'lock MyMutex<T>,
}

impl<T> Deref for MyGuard<'_, T> {
    type Target = T;

    fn deref(&self) -> &Self::Target {
        // Safety: We hold the lock until we are dropped,
        //         so we have exclusive access to the data.
        //         The shared borrow of the data is tracked through
        //         the shared borrow of self (elided lifetime).
        unsafe { &*self.mutex.data.get() }
    }
}

impl<T> DerefMut for MyGuard<'_, T> {
    fn deref_mut(&mut self) -> &mut Self::Target {
        // Safety: We hold the lock until we are dropped,
        //         so we have exclusive access to the data.
        //         The exclusive borrow of the data is tracked through
        //         the exclusive borrow of self (elided lifetime).
        unsafe { &mut *self.mutex.data.get() }
    }
}

impl<T> Drop for MyGuard<'_, T> {
    fn drop(&mut self) {
        *self.mutex.is_locked.lock().unwrap() = false;
    }
}
```


It can be used like this, which produces the correct, expected output:


```
mod my_mutex;

fn main() {
    use std::thread;
    use my_mutex::MyMutex;

    const N_THREADS:    u64 = 20;
    const N_INCREMENTS: u64 = 1000;
    const EXPECTED:     u64 = N_THREADS * N_INCREMENTS;

    let my_mutex: MyMutex<u64> = MyMutex::new(0);

    thread::scope(|scope| {
        for _ in 0..N_THREADS {
            scope.spawn(|| {
                for _ in 0..N_INCREMENTS {
                    *my_mutex.lock() += 1;
                }
            });
        }
    });

    let final_value = *my_mutex.lock();
    println!("Final value: {final_value} (expected {EXPECTED})");
}
```

```
$ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/mutex_broken`
Final value: 20000 (expected 20000)
```

Furthermore, running with Miri (eventually) produces nothing out of the ordinary:

```
$ cargo +nightly miri run
Preparing a sysroot for Miri (target: x86_64-unknown-linux-gnu)... done
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `/home/zac/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu/bin/cargo-miri runner
target/miri/x86_64-unknown-linux-gnu/debug/mutex_broken`
Final value: 20000 (expected 20000)
```

Note that if you plan to run the code with Miri yourself, you might want to consider decreasing the constants' values - Miri is particularly slow to run.

However, there exists a subtle unsoundness in `MyMutex`. In certain conditions, it is possible for an end-user of `MyMutex` to cause **undefined behaviour** without writing any `unsafe`. In particular, it is possible to cause a data race using only safe code.

> **NOTE:**
>
> Note that `MyMutex` uses a `Mutex` internally to store the `lock_held` state, however a more appropriate type here would have been an `AtomicBool` instead. A `Mutex` was specifically chosen to limit the scope of unsoundness, and its choice in place of an `AtomicBool` is not relevant to the unsoundness you are tasked to find.

Question:

1. What is the soundness issue in `MyMutex`, and how could it be fixed? *(3 marks)*
2. Provide a short example `main` function using only safe Rust that can reproduce a data race by exploiting the located unsoundness when run with Miri. *(2 marks)*

Write your answer in `exam_q7/q7_2.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q7_2 q7_2.txt
```

(OPTIONAL) EXERCISE:

22T3 Q8: Theory (10 marks)

The final question of the exam will be a more open-ended question which will ask you to perform some analysis or make an argument. Your argument will be judged alike an essay (i.e. are your claims substantiated by compelling arguments). Remember that you **will not** get any marks for blindly supporting / opposing Rust.

A friend of yours has just read an article critical of Rust, written approximately 7 years ago. Following is a relevant excerpt of the article for you to read. The original article is a very long and challenging read, so please only read / consider this excerpt.

```
# Criticizing the Rust Language, and Why C/C++ Will Never Die

Eax Melanhovich, May 12 2015.

I believe Rust is overhyped, and that the death of C and C++ are
over-exaggerated. It is crystal clear for every sane programmer that C/C++ is
not going to die in the near future. No one is going to rewrite almost all of
the existing desktop applications, operating system kernels, browser engines,
tons of other C-libraries, and so on and so forth, into other languages. This is
a huge mass of fast, debugged, and time-proven code. Rewriting it is way, way
too expensive, risky, and, honestly, doesn't seem to make sense except in the
heads of the most frantic Rust fans. The demand for C/C++ programmers has always
been high and will remain so for a long time to come.

C/C++ is criticized for a variety of reasons. Briefly, the issue
with C++ is that it is very fast but it is not safe in the sense that it allows
array overruns, addressing freed memory, and so on. Back in the past, this
```

problem urged programmers to develop a variety of safe languages such as Java, C#, Python, and others. But they have proved to be too resource-demanding compared to C++. That's why programmers are struggling to create a language as fast as C++ but also safe. Rust is one of the candidates.

And what actually makes Rust safe, by the way? To put it simple, this is a language with a built-in code analyzer, and it's a pretty tough one: it can catch all the bugs typical of C++, dealing not only with memory management, but multithreading as well. Pass a reference to an assignable object through a pipe to another thread and then try to use this reference yourself - the program will just refuse to compile. And that's really cool.

But other languages haven't stood still during the last 30 years, and there are many compile-time and runtime analysers which will check your code for issues. In any serious project, you use a continuous integration system and run tons of tests when compiling builds. If you don't, then your troubles are much worse than the language's lack of safety because static typing doesn't guarantee correct execution logic! So, since you run tests anyway, why not use sanitizers as well? True, they don't find all the bugs. But it's still pretty good, and you don't need to deal with performance issues caused by checking the bounds of an array at runtime, or the pain that the Rust compiler forces you into. Even without sanitizers, you'll find lots of stuff just building the project with various compilers on different platforms with assert's checking your code's invariants in the "assert(obj->isValid)" fashion and with proper fuzzing.

Even apart from that, I'm also skeptical about the language's design as such. In particular with regards to the many types of pointers used in it. On the one hand, it's not bad to make programmers ponder if their variables are stored in the stack or heap and if they can or cannot be handled by several threads at a time. But on the other hand, imagine you are writing a program and discover at one moment that some variable should be stored in the heap instead of the stack. So you rewrite the code to use Box. Then you figure out that you actually need Rc or Arc. Again, you rewrite all that code. And then, once again, you rewrite it all to have an ordinary variable in the stack. Regular expressions won't help. Or you might just end up with a nightmare like "vec<Rc<RefCell<Box<Trait>>>>" - say hello to Java! It would be much more convenient to let the programmer simply declare a variable and explicitly specify Box or Rc where necessary. From this viewpoint, Rust's developers have screwed up the whole thing.

Like many of new languages, Rust is walking the path of simplification. I can generally understand why it doesn't have inheritance and exceptions, but the fact itself that someone is making decisions for me regarding things like that makes me feel somewhat displeased. C++ doesn't restrict programmers regarding what they can or cannot use.

I can't help but remind you one more time that the source of troubles is usually in humans, not technology. If your C++ code is not good enough or Java code is painfully slow, it's not because the technology is bad - it's because you haven't learned how to use it right. In that sense, you won't be satisfied with Rust either - but just for some other reasons. Isn't it easier to learn how to use more popular tools and start liking them?

So, to sum it up, personally I will be investing my time into studying C/C++ rather than Rust in the next 5 or so years. C++ is an industrial standard. Programmers have been using it to solve a huge variety of tasks for over 30 years now. As for Rust and stuff like that - they are just odd toys with a vague future. People have been predicting C++'s soon death since the 2000-s, but C/C++ haven't become less used and demanded for since then.

From https://pvs-studio.com/en/blog/posts/0324/.

Read through the excerpt, and **discuss** the following prompt:

Rust is too strict and too painful to be useful, as explained by this article.

The overall *structure* of your answer is **not** marked. For example, your answer may include small paragraphs of prose accompanied by dot-points, or could instead be posed as a verbal discussion with your friend. Regardless of the structure / formatting you choose, the **substance** of what you write is the most important factor, and is what will determine your overall mark for this question.

Only responses less than 1000 words will be marked for this question. There will be many good answers that are significantly shorter (see above), this limit is a cap to save our marking time, and your sanity.

NOTE:

From UNSW's Glossary of Task Words:

Discuss	Investigate or examine by argument. Examine key points and possible interpretations, sift and debate, giving reasons for and against. Draw a conclusion.

Write your answer in `exam_q8/q8.txt`.

When you are finished working on your answer, submit your work with **give**:

```
$ give cs6991 exam_q8 q8.txt
```

Submission

When you are finished each exercise make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **2023-05-15 21:00:00** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

Automarking will be run several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run you can view your results here.

Weekly exercise Marks

When all exercises of a week are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 6991 classrun -sturec
```
