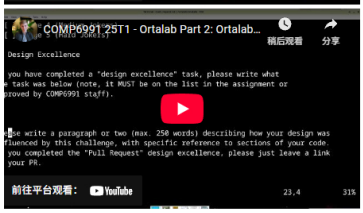
Assignment 1

EXERCISE:
Ortalab

Changelog

- 1.0.0: Released assignment in draft (2025-03-12)
- 1.0.1: Released assignment in full, added second video (2025-03-13)
- 1.1.0: Updated joker list to clarify which stage they were associated with (2025-03-16)
- 1.1.1: Removed stage-by-stage testing. Will update once fixed (2025-03-16)
- 1.1.2: Reintroduced stage-by-stage testing (2025-03-21)
- 1.1.3: Added extended Joker details in each stage-by-stage sections (2025-03-21)



The year is 2024, and a poker-themed rogue-lite game called *Balatro* has taken the world by storm. It is a huge hit, and has since sold over 5 million copies with an "overwhelmingly positive" 98% positive rating on Steam.

Balatro is all about building the best poker hand you can. The player starts with a standard 52 card deck, but can add and remove cards as they play. Playing simple poker hands (such as one pair, two pair, three of a kind, etc.) rewards the player with some points, but playing more complex hands (such as a flush, straight, full house, etc.) rewards the player with far more points!

The game also offers a set of special cards called "Jokers". These cards change the rules of the game, and can be used to create some very powerful poker hands! Each joker has a different effect, ranging from some simple score boosts, to some more complex effects that fundamentally change the way the game is played.

In this assignment, you will implement the scoring system for a simplified version of Balatro, called "Ortalab". The assignment is divided into several stages, each focusing on different aspects of the scoring system. The goals of this assignment are:

1. To get you to practice designing, structuring, and implementing a larger, more complex Rust program.
2. To understand and apply Rust's type system and traits effectively.
3. To have fun creating an interesting and engaging application.

We want to also be explicit about what the goals aren't:

1. To assess your ability to play poker or design a poker game. The focus is strictly on implementing the scoring system technically.
2. To assess your ability to write large amounts of unnecessary code. The code you write should be purposeful and demonstrate your understanding.

Importantly: the big challenge of this assignment is how to design your program. No individual feature is incredibly challenging, but it will be difficult to have everything work together and all of the functionality fit "neatly" into your program.

Much like the weekly lab exercises, there are autotests for this assignment! You should ensure that you pass them to avoid surprises during marking.

What you will submit

At the end of this assignment, you will submit:

- A Rust program that implements the Ortalab scoring system.
- A completed copy of `mark_request.txt` containing a list of features that you've completed; how to run your code; and good/bad things about your code.

Ortalab, the game

Ortalab operates within a simplified subset of Balatro, focusing specifically on calculating the score of a hand. While in Balatro, the game is played over many rounds, in Ortalab, we only consider a single round of play.

Example Round

Let's start with an example round of Ortalab. The round is described by a YAML file, which we will use to describe the cards played in the round.

```yaml
cards_played:
  - 10♥
  - 10♠
  - 10♦
  - A♣
  - J♥

cards_held_in_hand:
  - 2♥
  - 5♣
  - 9♦

jokers:
  - Joker
  - Mad Joker
  - Zany Joker
  - Raised Fist
```

Here we can see that the player has played 5 cards, and has 3 cards leftover in their hand. They have also have 4 jokers in their deck, which will augment the scoring of this round. The player's goal is to score as many points as possible by playing the cards in their hand, and by using the jokers to their advantage. In this case, the user has played a relatively powerful three of a kind, which will score them a base of Chips 30 and Mult 3.

We can feed this round into the reference solution, to see what the correct answer for this round is.

```
$ 6991 ortalab example_hand.yml
1380
```

The reference solution has calculated that the player should score 1380 points for this round. Although this is all that your

assignment solution is required to do, the reference solution can also explain exactly how it arrived at this final answer for your understanding. To see this, we can run the reference solution with the `--explain` flag.

```
$ 6991 ortalab example_hand.yml --explain
Three Of A Kind (30 x 3)
10♥ +10 Chips (40 x 3)
10♠ +10 Chips (50 x 3)
10♦ +10 Chips (60 x 3)
Raised Fist 2♥ +4 Mult (60 x 7)
Joker +4 Mult (60 x 11)
Zany Joker +12 Mult (60 x 23)
1380
```

Your assignment is not required to implement the `--explain` flag, but you may find implementing it yourself incredibly useful for debugging. Now, let's break down the explanation.

The first line shows the base score for the played poker hand. The player has played a three of a kind, which scores Chips 30 and Mult 3.

The next few lines show the score for each card played. For example, the player has played a 10♥, which scores an additional Chips 10, adding it to the 30 chips they already have.

Only the 10s are scored here, as they are the only cards that contribute to the three of a kind. Since the A♠ and J♥ are not part of the three of a kind, they are not scored. Furthermore, the cards held in hand are not scored by default.

The next few lines show scores for each joker played. Generally jokers activate from left to right (i.e. top to bottom), however there are additional ordering rules that, in this case, cause the Raised Fist joker to activate early.

The **Raised Fist** joker scores double the value of the lowest rank card held in hand as Mult. In this case, the player has a 2♥ held in hand, which scores a bonus Mult 4 for the **Raised Fist** joker.

Next, the **Joker** joker scores Mult 4 unconditionally.

Finally, the **Zany Joker** joker scores Mult 12 only if the player has played a three of a kind. Since they have, the **Zany Joker** scores in this round!

Sadly, although the player holds a **Mad Joker** joker, it only activates if the player has played a two pair hand. Since they have not, the **Mad Joker** does not score in this round.

Tallying up the scores, we get a total of Chips 60 and Mult 23. The final score is calculated by multiplying the chips by the mult, giving a final score of **1380**, which is shown on the final line.

Program Input

The program takes a single command line argument, which is the path to a YAML file describing a round of Ortalab.

A simple example was shown above, but a more complex example is given below.

```yaml
cards_played:
  - A♥ Bonus Foil
  - K♠ Mult Holographic
  - Q♦ Glass Polychrome
  - J♣ Wild
  - 10♥ Steel

cards_held_in_hand:
  - K♠ Steel Foil

jokers:
  - Joker Foil
  - Jolly Joker Holographic
  - Zany Joker Polychrome
```

In general, the YAML file will contain the following fields:

- `cards_played`: A list of cards played in the round.
- `cards_held_in_hand`: A list of cards held in the player's hand.
- `jokers`: A list of jokers in the player's deck.

The cards played and held in hand come from a standard 52 card deck, and they are described in detail further in the specification. There will always be a minimum of 1 card played, and a maximum of 5 cards played. For cards held in hand, there is a minimum of 0 cards, and a maximum of 5 cards.

The joker cards come from a fixed set of 34 jokers, which are described in detail further in the specification. There is a minimum of 0 jokers, and a maximum of 5 jokers.

Cards (including joker cards) may also optionally have editions and enhancements. These are described in detail further in the specification. Each card will have at most one edition and one enhancement.

Cards

Playing cards in Ortalab come from a standard 52 card deck. Each card has a rank and a suit.

The ranks are as follows:

- 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace

These ranks are listed in ascending order of value, where 2 is the lowest and Ace is the highest. Notably, Jack, Queen, King, and Ace are typically abbreviated as J, Q, K, and A respectively.

Jack, Queen, and King are all considered to be face cards. Note that Ace is **not** considered a face card.

The value of each rank is as follows:

Rank	2	3	4	5	6	7	8	9	10	J	Q	K	A
Value	2	3	4	5	6	7	8	9	10	10	10	10	11

Interestingly, the Ace is considered to be the highest rank, however it can also be used as the lowest rank in certain poker hands (e.g. a low Ace straight). The Ace is the only rank that has this property, and it always scores as 11.

The suits are as follows:

- **♠Spades**
- **♥Hearts**
- **♣Clubs**
- **♦Diamonds**

The suits have no particular order or value. Each suit also has a color. **♠Spades** and **♣Clubs** are both black, and **♥Hearts** and **♦Diamonds** are both red.

A standard 52 card deck contains 13 cards of each suit, with each rank appearing in each suit exactly once. This implies that each possible rank and suit combination appears exactly once in a standard deck.

However, in Ortalab, the same card can appear multiple times in a round. For example, the player may play three 10♠ in a single played hand. This is allowed, and there is no limit to the number of identical cards that can be appear in a single round. The same concept applies to jokers as well. This makes rounds such as the following entirely valid:

```yaml
cards_played: [10♠, 10♠, 10♠, 10♠, 10♠]

cards_held_in_hand: [10♠, 10♠, 10♠, 10♠, 10♠]

jokers:
  - Wrathful Joker
  - Wrathful Joker
  - Wrathful Joker
  - Wrathful Joker
  - Wrathful Joker
```

That round scores a whopping Chips 210 x Mult 91 for a final score of **19110**!

Poker Hands

Poker hands are a set of between 1 and 5 cards played in a single turn, which obtain Chips and Mult for scoring.

Higher tier poker hands score higher than lower tier poker hands. The lowest tier poker hand is the **High Card**, which is any poker hand that does not meet any of the other criteria. It scores Chips 5 and Mult 1. The next highest tier is the **Pair**, which is any poker hand that contains two cards with the same rank. It scores Chips 10 and Mult 2.

Eventually we reach the **Straight Flush**, which generally is the highest tier poker hand with a standard 52 card deck. It is obtained by playing five cards in consecutive order which are all from the same suit. It scores Chips 100 and Mult 8.

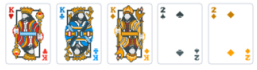
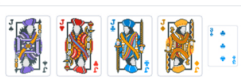
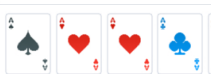
In Ortalab, we know that we can have duplicate cards in a played hand, so there are also three additional *"illegal"* poker hands: five of a kind, flush house, and flush five. These are each a higher tier than the **Straight Flush**, and are the greatest possible poker hands in Ortalab. Although they are colloquially called "illegal", they are valid poker hands in Ortalab, and can be played in a round.

Higher tier poker hands take precedence over lower tier poker hands invariably. For example, if your played hand is K K K K 2, and all of them are **♦Diamonds**, the scored poker hand will always be a **Four of a Kind** and never a **Flush**.

Generally, only the cards relevant to the poker hand are scored, and all others are unscored. For example, if an Ace is played high with 4 other cards, only the High card base amount and the Ace's values are used for the played hand's score. The other cards (up to 4) are just discarded fancily and have no effect. The main exception to this rule is with the **Splash** joker, which explicitly allows all played cards to score.

A detailed explanation of each poker hand, along with an example, is given below:

Base Scoring	**Poker**	**How to play the hand**	**Example**

	Hand		
Chips 5 x Mult 1	High Card	When no other poker hand is possible, the one highest card in your played hand. Aces are counted high for this poker hand.	
Chips 10 x Mult 2	Pair	Two cards with a matching rank. Suits may differ.	
Chips 20 x Mult 2	Two Pair	Two cards with a matching rank, and two cards with any other matching rank. Suits may differ.	
Chips 30 x Mult 3	Three of a Kind	Three cards with a matching rank. Suits may differ.	
Chips 30 x Mult 4	Straight	Five cards in consecutive order which are not all from the same suit. Aces can be counted high or low.	
Chips 35 x Mult 4	Flush	Five cards of any rank, all from a single suit.	
Chips 40 x Mult 4	Full House	Three cards with a matching rank, and two cards with any other matching rank, with cards from two or more suits.	
Chips 60 x Mult 7	Four of a Kind	Four cards with a matching rank. Suits may differ.	
Chips 100 x Mult 8	Straight Flush	Five cards in consecutive order, all from the same suit. Aces can be counted high or low.	
Chips 120 x Mult 12	Five of a Kind	Five cards with the same rank which are not all the same suit. (an "illegal" hand)	
Chips 140 x Mult 14	Flush House	Three cards with a matching rank, and two cards with any other matching rank, all from a single suit. (an "illegal" hand)	
Chips 160 x Mult 16	Flush Five	Five cards with the same rank and same suit. (an "illegal" hand)	

NOTE:

Typically an Ace is considered the highest card (above King), but for a straight (and a straight flush) it can *alternatively* be counted as the lowest card (below 2). However, a straight using an Ace must start or end with the Ace. For example, A♠ 2♦ 3♦ 4♣ 5♣ is a valid straight, 10♣ J♦ Q♦ K♣ A♣ is a valid straight, but Q♣ K♦ A♦ 2♦ 3♣ is **not** a valid straight. Another way to think about it is that straights cannot "wrap around" through an Ace.

The exact judgment criteria of a **Straight Flush** is "any played hand which is both a **Straight** and a **Flush**". With the **Four Fingers** joker, it's possible to make a **Straight Flush** with such cards as 9♣ 8♣ 7♥ 6♣ 3♣.

Two Pair and **Full House** require that the cards are of different ranks. Thus, a **Four of a Kind** is not considered as including a **Two Pair**. The same is true for **Five of a Kind** and **Flush House**.

Jokers

Jokers are a special type of card that can be used to score extra points. They are distinct from playing cards, which are described above.

Jokers each have a unique effect, which are described in detail in the stage specific sections. As you progress through the assignment, there will be a total of 34 jokers to implement. Some jokers have very simple effects, such as adding a fixed Mult to the score. Others have more complex effects, such as retriggering the scoring of played cards, copying the effects of other jokers, or changing the rules of the game! You can find a list of all 34 jokers in Ortalib below.

The same joker card can appear multiple times in a round, and each instance of the same joker card is distinct. For example, if the player plays two **Raised Fist** joker cards, these are considered as two distinct joker cards, and both will activate their abilities independently.

A brief explanation of each joker is given below:

#	Joker	Effect	Type	Act	Stage
1	 Joker	Mult +4	+m	Independent	3
2	 Jolly Joker	Mult +8 if cards played contains a Pair	+m	Independent	3
3	 Zany Joker	Mult +12 if cards played contains a Three of a Kind	+m	Independent	3
4	 Mad Joker	Mult +10 if cards played contains a Two Pair	+m	Independent	3
5	 Crazy Joker	Mult +12 if cards played contains a Straight	+m	Independent	3
6		Mult +10 if cards played contains a Flush	+m	Independent	3

	Droll Joker				
7	Sly Joker	Chips +50 if cards played contains a Pair	+c	Independent	3
8	Wily Joker	Chips +100 if cards played contains a Three of a Kind	+c	Independent	3
9	Clever Joker	Chips +80 if cards played contains a Two Pair	+c	Independent	3
10	Devious Joker	Chips +100 if cards played contains a Straight	+c	Independent	3
11	Crafty Joker	Chips +80 if cards played contains a Flush	+c	Independent	3
12	Abstract Joker	Mult +3 for each Joker card	+m	Independent	3
13	Raised Fist	Adds double the rank of the lowest card held in hand to Mult	+m	On Held	4
14	Blackboard	Mult ×3 if all cards held in hand are ♠Spades or ♣Clubs	×m	Independent	4
15	Baron	Each King held in hand gives Mult ×1.5	×m	On Held	4
16	Greedy Joker	Mult +3 for each ♦Diamonds card played	+m	On Scored	4
17	Lusty Joker	Mult +3 for each ♥Hearts card played	+m	On Scored	4
18	Wrathful Joker	Mult +3 for each ♠Spades card played	+m	On Scored	4
19	Gluttonous Joker	Mult +3 for each ♣Clubs card played	+m	On Scored	4
20	Fibonacci	Each played Ace, 2, 3, 5, or 8 gives Mult +8 when scored	+m	On Scored	4
21	Scary Face	Played face cards give Chips +30 when scored	+c	On Scored	4
22	Even Steven	Played even-ranked cards give Mult +4 when scored (10, 8, 6, 4, 2)	+m	On Scored	4
23	Odd Todd	Played odd-ranked cards give Chips +31 when scored (A, 9, 7, 5, 3)	+c	On Scored	4
24	Photograph	First scoring face card gives Mult ×2 when scored	×m	On Scored	4
25	Smiley Face	Played face cards give Mult +5 when scored	+m	On Scored	4
26		Mult ×3 if scoring cards contain a ♦Diamonds, ♣Clubs, ♥Hearts, and ♠Spades card	×m	Independent	4

	Flower Pot				
27	 Four Fingers	All Flushes and Straights can be made with 4 cards	Effect	N/A	5
28	 Shortcut	Allows Straights to be made with gaps of 1 rank (ex: 10 8 6 5 3)	Effect	N/A	5
29	 Mime	Retrigger all card held in hand abilities	Retrigger	On Held	5
30	 Pareidolia	All cards are considered face cards	Effect	N/A	5
31	 Splash	Every played card counts in scoring	Effect	N/A	5
32	 Sock and Buskin	Retrigger all played face cards	Retrigger	On Scored	5
33	 Smeared Joker	♥**Hearts** and ♦**Diamonds** count as the same suit, ♠**Spades** and ♣**Clubs** count as the same suit	Effect	N/A	5
34	 Blueprint	Copies the ability of Joker to the right (i.e. below)	Effect	Same as copied	5

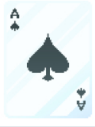
Card Modifiers

Modifiers are upgrades that can be applied to cards to change their scoring. There are two kinds of modifiers: **Card Enhancements** and **Card Editions**.

Each card may optionally have one enhancement, and may optionally have one edition. The enhancement and edition are chosen from a set of predefined options.

Card Enhancements

Enhancements can only be applied to playing cards (i.e. not jokers).

Appearance	Enhancement	Effect
	Bonus Card	Chips +30
	Mult Card	Mult +4
	Wild Card	Is considered to be every suit simultaneously
	Glass Card	Mult ×2
	Steel Card	Mult ×1.5 if this card is held in hand

Card Editions

Editions can be applied to any card, including both playing cards and jokers.

Appearance	Edition	Effect
	Foil	Chips +50
	Holographic	Mult +10

		
	Polychrome	Mult (×1.5)

Scoring

In Ortalab, there is a very specific sequence of steps that must be followed in order to score a round.

1. Determine the poker hand formed by the cards played. This is done by finding the best poker hand that can be made from the cards played. This may need to consider effect jokers such as **Four Fingers** and **Shortcut**, and any played cards that are wild.
   This provides the base score (in both Chips and Mult) for the round.
2. Score each card. Only cards that are part of the determined poker hand are scored, unless the **Splash** joker is active. For each scoring card, from left to right (i.e. top to bottom), its effects activate in the following order:
   1. Base Chips: Grant the base amount of Chips for the card.
   2. Apply an enhancement bonus, if present.
   3. Apply an edition bonus, if present.
   4. "On scored" Jokers activate.
      When multiple Jokers are triggered by the same card, they activate from left to right (i.e. top to bottom).
   5. Retriggers activate.
      For each "on scored" joker that caused a retrigger, the previous sequence (from base effect to "on scored" jokers) is repeated.
      Retriggers cannot trigger another retrigger, but multiple retriggers can stack additively.
3. Held in hand abilities: Cards held in hand are checked from left to right (i.e. top to bottom) if they can activate in-hand abilities. The sequence for each card is similar to scored cards.
   1. Enhancement (Steel card): If the card has a Steel enhancement, apply its multiplier to the score.
   2. "On held" Jokers activate.
      When multiple Jokers are triggered by the same card, they activate from left to right (i.e. top to bottom).
   3. Retriggers activate.
      For each "on held" joker that caused a retrigger, the previous sequence (from base effect to "on held" jokers) is repeated.
      Retriggers cannot trigger another retrigger, but multiple retriggers can stack additively.
4. Joker Editions and "independent" Jokers activate. Jokers are checked from left to right (i.e. top to bottom) to score any edition and activate any "independent" abilities in the following order:
   1. Edition (Foil or Holographic): If the card has a Foil or Holographic edition, apply its bonus to the score.
   2. "Independent" Jokers activate.
      These are jokers that will trigger their ability after all the playing cards have been scored. These do not get affected by retriggers.
   3. Edition (Polychrome): If the card has a Polychrome edition, apply its bonus to the score.

When this process is complete, a final score can be calculated by multiplying the Chips and Mult values.

Ortalib, the library

Ortalib is a Rust library that provides types, functions, and utilities to help you in implementing Ortalab. It is published to crates.io with the name `ortalib`, and you can access it at crates.io/crates/ortalib. Documentation is available at docs.rs/ortalib. In your starter code, you will find that it has already been added as a dependency to your project's `Cargo.toml` file.

The crate provides the following main types:

- `struct Round`
- `struct Card`
- `struct JokerCard`
- `enum PokerHand`
- `enum Joker`
- `enum Edition`
- `enum Enhancement`
- `enum Rank`
- `enum Suit`

It also provides the type aliases `Chips` and `Mult`, which simply alias to `f64`.

All of the above types, except `struct Round`, are `Copy`, meaning they opt-out of the ownership system and are copied on assignment. This should help to simplify implementing the assignment.

Please read through the entire documentation carefully, as it contains a lot of useful code that will help you in implementing your assignment.

Getting Started

Starter Code

We have provided a small amount of starter code for this assignment, which you can access by downloading this tar file.

You can also run the following command to fetch the starter code:

```
$ 6991 fetch assignment 01
```

Reference Solution

To help you check what the correct answer for a given test case is, you can use the command shown below to try our reference solution.

```
$ 6991 ortalab [round_file]
```

Where `[round_file]` is the path to a YAML file describing a round of Ortalab.

Stage 1: Basic Poker Hand Scoring (40% of 60% FC)

In this stage, you will implement the basic poker hand scoring system. You will calculate the score based on the poker hand formed by the cards played. You will also calculate the score for each card that forms part of the poker hand.

Required Features

- Implement scoring for all standard poker hands: High Card, Pair, Two Pair, Three of a Kind, Straight, Flush, Full House, Four of a Kind, Straight Flush.
- Implement scoring for "illegal" poker hands: Five of a Kind, Flush House, Flush Five.
- Implement scoring each card that forms part of the poker hand.

IMPLEMENTAION HINTS:

Writing a helper function to group cards by their rank or suit can help simplify your implementation for almost all of the required hand types. You may find the Itertools crate to be particularly useful.

Make use of Ortalib's `PokerHand` enum, which will help you implement the scoring for each poker hand type. In particular, the `pub fn hand_value(&self) -> (Chips, Mult)` function will give you the base Chips and Mult values for a given poker hand type.

When calculating the final score of the round, refer to the Scoring section above. In particular, you will only need to implement **step 1** and **step 2.1** of the scoring process.

You can find the base Chips for each card that forms part of the poker hand by using the `pub fn rank_value(&self) -> Chips` function on the `Rank` enum.

Example

For a played hand containing a Three of a Kind (e.g., 10♥ 10♦ 10♦ A♠ 2♥), the base score is Chips (30) and Mult (3).

The 10♥, 10♦, and 10♦ each score Chips (10), for a total of Chips (30). The A♠ and 2♥ do not score any Chips, even though they have been played, because they do not form part of the poker hand (Three of a Kind). The overall total is Chips (60) and Mult (3), producing a final score of 180.

```yaml
cards_played:
  - 10♥
  - 10♦
  - 10♦
  - A♠
  - 2♥
```

```
cards_held_in_hand: []

jokers: []
```

```
$ 6991 ortalab three_of_a_kind.yml
180
```

This is all that your program is required to print out. However, we can use the `--explain` flag to get a detailed explanation of the scoring process.

```
$ 6991 ortalab three_of_a_kind.yml --explain
Three Of A Kind (30 x 3)
10♥ +10 Chips (40 x 3)
10♠ +10 Chips (50 x 3)
10♦ +10 Chips (60 x 3)
180
```

Testing

You can use the following command to test your implementation:

```
$ 6991 autotest stage01
Found cargo project: ortalab
Located autotests for ortalab
./check_mark_request.sh # crate.tar
6991 cargo build --target-dir target # crate.tar
    Updating crates.io index
     Locking 1 package to latest compatible version
      Adding ortalib v1.0.0
   Compiling proc-macro2 v1.0.94
   Compiling unicode-ident v1.0.18
   Compiling libc v0.2.171
   ...
   Compiling ortalib v1.0.0
   Compiling ortalab v0.1.0 (/tmp/tmppoloednl/autotest)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 28.37s
Test stage01_flush_house_clubs_medium ('./target/debug/ortalab
"fixtures/staged/01/flush_house_clubs_medium.yml"') - passed
Test stage01_flush_diamonds ('./target/debug/ortalab "fixtures/staged/01/flush_diamonds.yml"') - passed
Test stage01_four_of_a_kind_high ('./target/debug/ortalab "fixtures/staged/01/four_of_a_kind_high.yml"') - passed
...
Test stage01_flush_clubs ('./target/debug/ortalab "fixtures/staged/01/flush_clubs.yml"') - passed
Test stage01_straight_flush_low_clubs ('./target/debug/ortalab
"fixtures/staged/01/straight_flush_low_clubs.yml"') - passed
Test stage01_flush_five_clubs_jacks ('./target/debug/ortalab "fixtures/staged/01/flush_five_clubs_jacks.yml"') -
passed
41 tests passed 0 tests failed
```

This will run a large number of tests, spanning all the different poker hands, and checking that the scoring is correct for each. These tests contain no joker cards, nor cards held in hand. They are only used to check that your implementation of the poker hand scoring is correct. Unfortunately, the autotests are in no particular order.

Stage 2: Card Modifiers (10% of 60% FC)

In this stage, you will enhance the scoring system by considering card enhancements and editions. You can find information on the specific effect of each card modifier in the Card Modifiers section above.

Required Features

- Complete all previous stages (Stage 1).
- Implement scoring adjustments based on playing card enhancements: Bonus, Mult, Wild, Glass, Steel.
- Implement scoring adjustments based on card editions: Foil, Holographic, Polychrome.

IMPLEMENTAION HINTS:

The most difficult part of this stage is likely to be the Wild Card enhancement. This may require you to modify your algorithm for determining the poker hand, as a Wild Card may be counted as any suit. This is relevant for determining if a played hand is a Flush, Straight Flush, Flush House, or Flush Five.

The remaining card modifiers are relatively straightforward scoring adjustments. This will expand your implementation of the Scoring section above.
In particular, you will need to implement **step 2.2**, **step 2.3**, and **step 3.1** of the scoring process.
Since there are no jokers yet in this stage, you do not *need* to implement **step 4.1** or **step 4.3** just yet, but you can choose to do so ahead of time if you wish.

Example

```
cards_played:
  - A♥ Bonus Foil
  - K♠ Mult Holographic
  - Q♦ Glass Polychrome
  - J♣ Wild
  - 10♥ Steel

cards_held_in_hand:
  - K♠ Steel Foil
  - 7♠ Steel Holographic
  - 3♠ Steel Polychrome

jokers: []
```

Notably, the `J♣` being `Wild` has no effect in this example, because we are not close to a flush of any kind. Similarly, the `10♥` being `Steel` has no effect, because the `Steel` effect only applies for cards held in hand. All the cards held in hand are `Steel` which will apply, but their editions (`Foil`, `Holographic`, `Polychrome`) do not count for cards held in hand.

```
$ 6991 ortalab modifiers.yml --explain
Straight (30 x 4)
A♥ +11 Chips (41 x 4)
A♥ Bonus +30 Chips (71 x 4)
A♥ Foil +50 Chips (121 x 4)
K♠ +10 Chips (131 x 4)
K♠ Mult +4 Mult (131 x 8)
K♠ Holographic +10 Mult (131 x 18)
Q♦ +10 Chips (141 x 18)
Q♦ Glass x2 Mult (141 x 36)
Q♦ Polychrome x1.5 Mult (141 x 54)
J♣ +10 Chips (151 x 54)
10♥ +10 Chips (161 x 54)
K♠ Steel x1.5 Mult (161 x 81)
7♠ Steel x1.5 Mult (161 x 121.5)
3♠ Steel x1.5 Mult (161 x 182.25)
29342
```

```
cards_played:
  - 2♥
  - 5♠ Wild
  - 5♥
  - 6♥
  - 6♠ Wild

cards_held_in_hand: []

jokers: []
```

In this example, we would typically find this played hand to be **Two Pair**, with the `5♠` and `5♥` forming the first pair, and the `6♥` and `6♠` forming the second pair. However, the `5♠` and `6♠` cards are `Wild`, so they can be counted as any suit. Here, when they are both treated as **♥Hearts**, the played hand forms a **Flush**, which is a higher tier poker hand than **Two Pair**.

```
$ 6991 ortalab wild.yml --explain
Flush (35 x 4)
2♥ +2 Chips (37 x 4)
5♠ +5 Chips (42 x 4)
5♥ +5 Chips (47 x 4)
6♥ +6 Chips (53 x 4)
6♠ +6 Chips (59 x 4)
236
```

Testing

You can use the following command to test your implementation:

```
$ 6991 autotest stage02
Found cargo project: ortalab
Located autotests for ortalab
./check_mark_request.sh # crate.tar
6991 cargo build --target-dir target # crate.tar
    Updating crates.io index
     Locking 1 package to latest compatible version
      Adding ortalib v1.0.0
```

```
   Compiling proc-macro2 v1.0.94
   Compiling unicode-ident v1.0.18
   Compiling libc v0.2.171
   ...
   Compiling ortalib v1.0.0
   Compiling ortalab v0.1.0 (/tmp/tmpl6o8y1is/autotest)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 29.07s
Test stage02_mult ('./target/debug/ortalab "fixtures/staged/02/mult.yml"') - passed
Test stage02_glass ('./target/debug/ortalab "fixtures/staged/02/glass.yml"') - passed
Test stage02_steel ('./target/debug/ortalab "fixtures/staged/02/steel.yml"') - passed
...
Test stage02_five_of_a_kind_kings ('./target/debug/ortalab "fixtures/staged/02/five_of_a_kind_kings.yml"') -
passed
Test stage02_wild_straight ('./target/debug/ortalab "fixtures/staged/02/wild_straight.yml"') - passed
Test stage02_five_of_a_kind_aces ('./target/debug/ortalab "fixtures/staged/02/five_of_a_kind_aces.yml"') - passed
14 tests passed 0 tests failed
```

This will run some tests, spanning all the different card modifers (both enhancements and editions) and checking that the scoring is correct for each. These tests contain no joker cards, since they only appear in stage 3. However, if you are planning to implement stage 3 onwards, you will need to make sure that editions work on Joker Cards also. There are some autotests containing Jokers spanning all possible editions in stage 3. Unfortunately, the autotests are in no particular order.

Stage 3: Easy Jokers (15% of 60% FC)

In this stage, you will implement the effects of easy jokers on the scoring system.

Required Features

- Complete all previous stages (Stage 1, 2).
- Implement the effects of easy jokers: Joker, Jolly Joker, Zany Joker, Mad Joker, Crazy Joker, Droll Joker, Sly Joker, Wily Joker, Clever Joker, Devious Joker, Crafty Joker, and Abstract Joker.

Joker Details

#	Joker	Effect	Act
1	 Joker	Mult +4 Simply adds Mult 4	Independent
2	 Jolly Joker	Mult +8 if cards played contains a Pair This applies even if the best poker hand is not a Pair. It also still applies even if the best poker hand doesn't contain both of the Pair cards. Three, Four, and Five of a Kind are all considered as containing a Pair.	Independent
3	 Zany Joker	Mult +12 if cards played contains a Three of a Kind This applies even if the best poker hand is not a Three of a Kind. It also still applies even if the best poker hand doesn't contain all of the Three of a Kind cards (e.g. with a Four Fingers Flush). Four and Five of a Kind are both considered as containing a Three of a Kind.	Independent
4	 Mad Joker	Mult +10 if cards played contains a Two Pair This applies even if the best poker hand is not a Two Pair. It also still applies even if the best poker hand doesn't contain all of the Two Pair cards (e.g. with a Four Fingers Flush). The two pairs must be of different ranks. For this reason, Four and Five of a Kind are **not** considered as containing Two Pair.	Independent
5	 Crazy Joker	Mult +12 if cards played contains a Straight This applies even if the best poker hand is not a Straight. It also still applies even if the best poker hand doesn't contain all of the Straight cards (e.g. with a Four Fingers Flush).	Independent
6	 Droll Joker	Mult +10 if cards played contains a Flush This applies even if the best poker hand is not a Flush.	Independent
7	 Sly Joker	Chips +50 if cards played contains a Pair This applies even if the best poker hand is not a Pair. It also still applies even if the best poker hand doesn't contain both of the Pair cards. Three, Four, and Five of a Kind are all considered as containing a Pair.	Independent
8	 Wily Joker	Chips +100 if cards played contains a Three of a Kind This applies even if the best poker hand is not a Three of a Kind. It also still applies even if the best poker hand doesn't contain all of the Three of a Kind cards (e.g. with a Four Fingers Flush). Four and Five of a Kind are both considered as containing a Three of a Kind.	Independent
9	 Clever Joker	Chips +80 if cards played contains a Two Pair This applies even if the best poker hand is not a Two Pair. It also still applies even if the best poker hand doesn't contain all of the Two Pair cards (e.g. with a Four Fingers Flush). The two pairs must be of different ranks. For this reason, Four and Five of a Kind are **not** considered as containing Two Pair.	Independent
10	 Devious Joker	Chips +100 if cards played contains a Straight This applies even if the best poker hand is not a Straight. It also still applies even if the best poker hand doesn't contain all of the Straight cards (e.g. with a Four Fingers Flush).	Independent
11	 Crafty Joker	Chips +80 if cards played contains a Flush This applies even if the best poker hand is not a Flush.	Independent
12	 Abstract Joker	Mult +3 for each Joker card Simply adds Mult 3 multiplied by the number of Joker Cards in the round (e.g. with 4 Joker Cards, Mult +12).	Independent

Testing

You can use the following command to test your implementation:

```
$ 6991 autotest stage03
Found cargo project: ortalab
Located autotests for ortalab
./check_mark_request.sh # crate.tar
6991 cargo build --target-dir target # crate.tar
    Updating crates.io index
     Locking 1 package to latest compatible version
      Adding ortalib v1.0.0
   Compiling proc-macro2 v1.0.94
   Compiling unicode-ident v1.0.18
   Compiling libc v0.2.171
```

```
   ...
   Compiling ortalib v1.0.0
   Compiling ortalab v0.1.0 (/tmp/tmpl68y1is/autotest)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 29.07s
Test stage03_jolly_joker_with_pair ('./target/debug/ortalab "fixtures/staged/03/jolly_joker_with_pair.yml"') -
passed
Test stage03_enhanced_cards ('./target/debug/ortalab "fixtures/staged/03/enhanced_cards.yml"') - passed
Test stage03_all_wild_cards ('./target/debug/ortalab "fixtures/staged/03/all_wild_cards.yml"') - passed
...
Test stage03_crazy_joker_without_straight ('./target/debug/ortalab
"fixtures/staged/03/crazy_joker_without_straight.yml"') - passed
Test stage03_zany_joker_with_three_of_a_kind ('./target/debug/ortalab
"fixtures/staged/03/zany_joker_with_three_of_a_kind.yml"') - passed
Test stage03_devious_joker_without_straight ('./target/debug/ortalab
"fixtures/staged/03/devious_joker_without_straight.yml"') - passed
34 tests passed 0 tests failed
```

This will run some tests, spanning all stage 3 Jokers and checking that the scoring is correct for each. Most tests will just focus on a single joker in isolation. Unfortunately, the autotests are in no particular order.

Stage 4: Medium Jokers (15% of 60% FC)

In this stage, you will implement the effects of medium jokers on the scoring system.

Required Features

- Complete all previous stages (Stage 1, 2, 3).
- Implement the effects of medium jokers: Raised Fist, Blackboard, Baron, Greedy Joker, Lusty Joker, Wrathful Joker, Gluttonous Joker, Fibonacci, Scary Face, Even Steven, Odd Todd, Photograph, Smiley Face, and Flower Pot.

Joker Details

#	Joker	Effect	Act
13	 Raised Fist	Adds double the rank value of the lowest rank card held in hand to +Mult Importantly, Raised Fist triggers while you are scoring the lowest rank card held in hand, not independently as you may assume. If multiple cards held in hand tie for the same rank, Raised Fist scores its bonus the right-most card. As usual, face cards count as 10 +Mult +20 and Aces count as 11 +Mult +22. With retriggers, Raised Fist can apply on the lowest ranked card multiple times.	On Held
14	 Blackboard	×Mult ×3 if all cards held in hand are ♠Spades or ♣Clubs This boost also activates if there are no cards held in hand. Any card with the Wild Card modifier will also count as ♠Spades / ♣Clubs.	Independent
15	 Baron	Each King held in hand gives ×Mult ×1.5 Baron scores on *each* King, not just the first King held in hand.	On Held
16	 Greedy Joker	+Mult +3 for each ♦Diamonds card played Any card with the Wild Card modifier will also count as ♦Diamonds.	On Scored
17	 Lusty Joker	+Mult +3 for each ♥Hearts card played Any card with the Wild Card modifier will also count as ♥Hearts.	On Scored
18	 Wrathful Joker	+Mult +3 for each ♠Spades card played Any card with the Wild Card modifier will also count as ♠Spades.	On Scored
19	 Gluttonous Joker	+Mult +3 for each ♣Clubs card played Any card with the Wild Card modifier will also count as ♣Clubs.	On Scored
20	 Fibonacci	Each played Ace, 2, 3, 5, or 8 gives +Mult +8 when scored	On Scored
21	 Scary Face	Played face cards give +Chips +30 when scored The face cards are King, Queen, and Jack. Notably, Ace is **not** considered a face card. You may find the `is_face` method from Ortalib helpful in implementing Scary Face.	On Scored
22	 Even Steven	Played even-ranked cards give +Mult +4 when scored (10, 8, 6, 4, 2)	On Scored
23	 Odd Todd	Played odd-ranked cards give +Chips +31 when scored (A, 9, 7, 5, 3)	On Scored
24	 Photograph	First scoring face card gives ×Mult ×2 when scored Photograph's bonus only applies to the first scoring face card. However, with retriggers, it can apply on this card multiple times.	On Scored
25	 Smiley Face	Played face cards give +Mult +5 when scored	On Scored
26	 Flower Pot	×Mult ×3 if scoring cards contain a ♦Diamonds, ♣Clubs, ♥Hearts, and ♠Spades card Flower Pot rewards ×Mult ×3 for playing at least one scoring card of every suit. Cards with multiple suits (such as Wild Cards or cards affected by Smeared Joker) will only count for a single suit per card, but they can replace missing suits required to activate Flower Pot's effect. For this reason, Flower Pot can only trigger when having at least 4 scoring cards.	Independent

Testing

You can use the following command to test your implementation:

```
$ 6991 autotest stage04
Found cargo project: ortalab
Located autotests for ortalab
./check_mark_request.sh # crate.tar
6991 cargo build --target-dir target # crate.tar
    Updating crates.io index
     Locking 1 package to latest compatible version
      Adding ortalib v1.0.0
   Compiling proc-macro2 v1.0.94
   Compiling unicode-ident v1.0.18
   Compiling libc v0.2.171
   ...
   Compiling ortalib v1.0.0
   Compiling ortalab v0.1.0 (/tmp/tmpl6o8yiis/autotest)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 29.07s
Test stage04_raised_fist_multiple_cards ('./target/debug/ortalab
"fixtures/staged/04/raised_fist_multiple_cards.yml"') - passed
Test stage04_fibonacci_all_fibonacci ('./target/debug/ortalab "fixtures/staged/04/fibonacci_all_fibonacci.yml"')
- passed
Test stage04_raised_fist_high_card ('./target/debug/ortalab "fixtures/staged/04/raised_fist_high_card.yml"') -
passed
...
Test stage04_scary_face_mixed_cards ('./target/debug/ortalab "fixtures/staged/04/scary_face_mixed_cards.yml"') -
passed
Test stage04_smiley_face_all_face ('./target/debug/ortalab "fixtures/staged/04/smiley_face_all_face.yml"') -
passed
Test stage04_lusty_joker_all_hearts ('./target/debug/ortalab "fixtures/staged/04/lusty_joker_all_hearts.yml"') -
passed
41 tests passed 0 tests failed
```

This will run some tests, spanning all stage 4 Jokers and checking that the scoring is correct for each. Most tests will just focus on a single joker in isolation. Unfortunately, the autotests are in no particular order.

Stage 5: Hard Jokers (20% of 60% FC)

In this final stage, you will implement the effects of hard jokers on the scoring system.

Required Features

- Complete all previous stages (Stage 1, 2, 3, 4).
- Implement the effects of hard jokers: Four Fingers, Shortcut, Mime, Pareidolia, Splash, and Sock and Buskin, Smeared Joker, Blueprint.

Joker Details

#	Joker	Effect	Act
27	 Four Fingers	All Flushes and Straights can be made with 4 cards Four Fingers is an effect-type joker that allows you to make Straights or Flushes using four cards instead of five. Five cards will still be prioritised if possible. Straight Flushes become more achievable with this joker, as a five-card hand that contains a 4-card Straight and a 4-card Flush will be considered to be a Straight Flush. For example, in a hand of 2♠ 3♠ 4♦ 5♠ 10♠, the 2 3 4 5 make a Straight of four cards and the 2 3 5 10 make a ♠Spades Flush of four cards, altogether being counted a Straight Flush.	N/A
28	 Shortcut	Allows Straights to be made with gaps of 1 rank (ex: 10 8 6 5 3) For example, the ranks 2 4 5 7 9 form a straight with Shortcut. As usual, Shortcut still does not allow "wrapping" Straights, so Aces remain either high or low and not both. Four Fingers creates an interesting combination with Shortcut, as this allows for a 4-length Straight with gaps, and Straight Flushes become even easier to make since the Straight and Flush don't need to be on the same 4 cards. (e.g. Q♠ J♠ 9♦ 7♠ 3♠ is a valid Straight Flush with both Jokers in effect)	N/A
29	 Mime	Retrigger all card held in hand abilities Mime retriggers all held in hand effects. This includes all steel cards held in hand, along with retriggering the effects of any "On Held" Jokers. As previously mentioned, retriggers cannot cause further retriggers. However, multiple retriggers can stack additively. For example, if three Mimes exist in a round, they will each cause their own retrigger, for a total of four triggers on each card held in hand.	On Held
30	 Pareidolia	All cards are considered face cards Pareidolia makes all playing cards count as face cards. This effect applies to other Jokers' effects (e.g. Scary Face). All cards otherwise retain their rank and suit.	N/A
31	 Splash	Every played card counts in scoring Splash makes all the played cards count when scoring, not just the ones that make up the best Poker Hand. This includes triggering editions and enhancements on each of the additional cards, and these will also trigger any Joker effects.	N/A
32	 Sock and Buskin	Retrigger all scoring face cards Sock and Buskin retriggers all scoring effects. This includes card base chips, enhancements, and editions, along with retriggering the effects of any "On Scored" Jokers. Sock and Buskin only retriggers scoring face cards. It does not retrigger face cards held in hand.	On Scored
33	 Smeared Joker	♥Hearts and ♦Diamonds count as the same suit, ♠Spades and ♣Clubs count as the same suit This effect applies to all Jokers which have an effect based on suit. You may find the `color` method from Ortalib helpful in implementing Smeared Joker.	N/A
34	 Blueprint	Copies the ability of Joker to the right (i.e. below) Not all Jokers are compatible with Blueprint, as Blueprint cannot copy any "passive modifier" (i.e. N/A) effects, but Blueprint itself is compatible. Blueprint does **not** copy the edition of the Joker to the right. If there is no Joker on Blueprint's right, it simply has no effect. This same logic also applies for a Blueprint copying another Blueprint with no effect.	Same as copied

Testing

You can use the following command to test your implementation:

```
$ 6991 autotest stage05
Found cargo project: ortalab
Located autotests for ortalab
./check_mark_request.sh # crate.tar
6991 cargo build --target-dir target # crate.tar
    Updating crates.io index
     Locking 1 package to latest compatible version
      Adding ortalib v1.0.0
   Compiling proc-macro2 v1.0.94
   Compiling unicode-ident v1.0.18
   Compiling libc v0.2.171
   ...
   Compiling ortalib v1.0.0
   Compiling ortalab v0.1.0 (/tmp/tmpl6o8yiis/autotest)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 29.07s
Test stage05_sock_buskin_no_face ('./target/debug/ortalab "fixtures/staged/05/sock_buskin_no_face.yml"') - passed
```

```
Test stage05_four_finger_flush ('./target/debug/ortalab "fixtures/staged/05/four_finger_flush.yml"') - passed
Test stage05_shortcut_four_fingers ('./target/debug/ortalab "fixtures/staged/05/shortcut_four_fingers.yml"') -
passed
...
Test stage05_shortcut_with_single_gap ('./target/debug/ortalab
"fixtures/staged/05/shortcut_with_single_gap.yml"') - passed
Test stage05_pareidolia_with_smiley_face_and_splash ('./target/debug/ortalab
"fixtures/staged/05/pareidolia_with_smiley_face_and_splash.yml"') - passed
Test stage05_complex_blueprint_chain ('./target/debug/ortalab "fixtures/staged/05/complex_blueprint_chain.yml"')
- passed
50 tests passed 0 tests failed
```

This will run some tests, spanning all stage 5 Jokers and checking that the scoring is correct for each. Most tests will just focus on a single joker in isolation. Unfortunately, the autotests are in no particular order.

Once you have completed stage 5, you may wish to run all autotests in one bulk command. You can use the following command to run all available tests (in stage order):

```
$ 6991 autotest
Found cargo project: ortalab
Located autotests for ortalab
./check_mark_request.sh # crate.tar
6991 cargo build --target-dir target # crate.tar
    Updating crates.io index
     Locking 1 package to latest compatible version
      Adding ortalib v1.0.0
   Compiling proc-macro2 v1.0.94
   Compiling unicode-ident v1.0.18
   Compiling libc v0.2.171
   ...
   Compiling ortalib v1.0.0
   Compiling ortalab v0.1.0 (/tmp/tmpl6o8yiis/autotest)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 29.07s
Test stage01_flush_house_clubs_medium ('./target/debug/ortalab
"fixtures/staged/01/flush_house_clubs_medium.yml"') - passed
Test stage01_flush_diamonds ('./target/debug/ortalab "fixtures/staged/01/flush_diamonds.yml"') - passed
Test stage01_four_of_a_kind_high ('./target/debug/ortalab "fixtures/staged/01/four_of_a_kind_high.yml"') - passed
...
Test stage05_shortcut_with_single_gap ('./target/debug/ortalab
"fixtures/staged/05/shortcut_with_single_gap.yml"') - passed
Test stage05_pareidolia_with_smiley_face_and_splash ('./target/debug/ortalab
"fixtures/staged/05/pareidolia_with_smiley_face_and_splash.yml"') - passed
Test stage05_complex_blueprint_chain ('./target/debug/ortalab "fixtures/staged/05/complex_blueprint_chain.yml"')
- passed
180 tests passed 0 tests failed
```

Design Excellence (5% of Assignment 1 Mark)

Design Excellence is designed for students who really want to get the most out of this assignment. It's a less structured part of the assignment where you can take on a challenge. When marking, we'll check that you've completed it reasonably but it won't be strictly tested -- the point is that you've challenged yourself and hopefully learned something interesting.

What to do for design excellence?

- Get 80% test coverage for your application. You could use a library like tarpaulin to calculate your test coverage. Consider approaching this assignment in a Test-Driven Development style if you're aiming for this design excellence. This is design excellence because it requires you to think about how to test your program, and to design a testable program.
- Implement a "fuzzer" for your assignment, that generates rounds and tests whether your answer is the same as the reference solutions.
- Implement a UI for Ortalib, either in the terminal or in the web. This will test your ability to integrate external components with the app, and it's a significant extra piece of work to design.
- Implement a "solver", which takes a given set of jokers/cards and determines what the largest score you could get is with those cards/jokers.
- Support users adding custom jokers as "plugins".
- Your own idea: we're very happy to have further ideas/suggestions for design excellence. You should check them with us on the forum however, before beginning.

Mark Request

When building modern software, you are expected to send in a pull request, explaining what you've done. In COMP6991, you must submit a "mark request". This is contained in `mark_request.txt`.

The mark request helps us mark you -- we will use it to help us find out what to mark. It is not meant to take more than 10 minutes to write. No marks are awarded for the mark request specifically; but if it is very inaccurate, we reserve the right to either request you resubmit it with a penalty; or mark using it as a basis (and therefore not mark things you've done).

FAQs

How do I access the contents of each test? You can access the contents of the tests with the command:

```
$ wget https://cgi.cse.unsw.edu.au/\~cs6991/current/api/activity/ortalab/autotest.tar
$ tar xf autotest.tar fixtures/
```

Make sure you do this regularly, in the unlikely event any tests are updated/removed.

Other Information

QandA / Clarifications

For any assignment-related questions, refer to the `Assignments` category on the forum. The following are a list of brief answers to common questions:

- **Can I use external crates?** Yes, you may use any crate from crates.io with at least 1000 downloads, excluding the last 30 days, and that does not impose license restrictions requiring you to share your own code.
- **How should I handle errors?** Use idiomatic Rust error handling. Avoid panics for expected errors and use Result types where appropriate.

Submission

See the instructions down the bottom of the page.

Marking Scheme

There are 4 things on which you will be marked:

- Mechanical Style
- Functional Correctness
- Idiomatic Design
- Design Excellence

You can see how the marks break down here:

	Mechanical Style	Functional Correctness	Idiomatic Design	Design Excellence
Ortalab Scoring System	5%	60%	30%	5%

And a detailed analysis is shown below:

1. Mechanical Style (5%):

We will look at your crates, and make sure they:

- Compile, with no warnings or errors.
- Raise no issues with `6991 cargo clippy`
- Are formatted with rustfmt
- Have any tests written for them pass

If they do all of the above, you get full. Otherwise, we will award partial marks. This is meant to be the "easy marks" of programming.

2. Functional Correctness (60%):

Your code should do what we describe. We will run the code through some obvious tests to make sure it behaves as we expect.

- Trying to start your program.
- Testing the behaviour we stated below.
- Trying out different poker hands and joker effects.

You *must* complete the checklist section of the `mark_request` faithfully. We will only test things which you say you have done. In other words, if you do not fill in the file, we will not be able to give you any marks. If you do fill in the `mark_request` and you clearly have not done the work, we may ask you to re-complete it at a penalty before we do marking.

3. Idiomatic Design (30%):

Your code should be well designed. This is where we will spend most of our time when marking. To help you, we have provided "design excellence" suggestions, which are ideas to make your design really excellent. You don't have to do them, but they would be good ways of getting a great design.

The following list of properties will be marked in your program:

- Code is not unnecessarily repeated.
- Code is abstracted appropriately.
- Types are used appropriately to express data in the program.
- The design does not impose unnecessary constraints on either the caller or callee through borrowing, lifetimes or ownership.
- Uses traits sensibly to add expressiveness.
- Data structures used are appropriate to store data.
- Functions perform error handling; cases that are expected do not panic.
- Code is sensibly organised, and split into appropriate modules.
- Documentation, where provided, is correct and readable.
- (optional) Uses external crates effectively to achieve the above goals.
- (optional) Where code is designed in a sub-optimal way, comments about how to improve it are made under "Design Limitations".

Your mark will be calculated based on the feedback you have received:

100% of available marks	Very little negative feedback is given on the above criteria.
85% of available marks	Some minor comments are made about some of the above criteria.
75% of available marks	Major comments are made about one or two criteria, with multiple small comments in different areas.
65% of available marks	Major comments are made about three or more criteria.
50% of available marks	Many areas have major comments made.
below 50% of available marks	Assignments in this category are likely written as "translations from C", and ignore many Rust features and design patterns.

4. Design Excellence (5%):

Design excellence is 5% of your mark, and intended only for those who truly want to shoot for full marks.

You can learn more about Design Excellence in the above section.

Note that the following penalties apply to your total mark for plagiarism:

0 for the assignment	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 for the assignment	Submitting any other persons work. This includes joint work.
0 FL for COMP6991	Paying another person to complete work. Submitting another persons work without their consent.

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ 6991 give-crate
```

The due date for this exercise is **Week 7 Wednesday 18:00:00**.

Note that this is an individual exercise; the work you submit with `give` must be entirely your own.