CS 246 Spring 2018 Project - Euchre

C. Kierstead, A. Moss, V. Sakhnini

Due Date 1: Monday, July 16, 11:59pm
Due Date 2: Wednesday, July 25, 11:59pm

Project Description

This project is intended to be doable by 2-3 people in 2 weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes 2 weeks' worth of work for 2-3 students is difficult to quantify. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, **MAKE SURE YOUR SUBMITTED PROGRAM COMPILES AND RUNS**. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does *something*.

1. Euchre Game Description

The course project is based on a multi-player version of the card game of euchre (pronounced "YOO-KER"). There are many different versions of the game worldwide, but we will only consider the North American version.

Below are a few web sites that have rules and strategies to the game:

- `http://userpages.bright.net/~double/euchre.htm`
- `https://www.pagat.com/euchre/euchre.html`
- `https://www.thespruce.com/euchre-rules-tricktaking-card-game-409350`

There is also a commercial site devoted to euchre at `http://www.thehouseofcards.com/euchre.html`. At this site, there are books, electronic games, sample software, and a list of online sites devoted to the game.

The game specification follows. If there is something in doubt in this specification, it will be addressed on Piazza. **You are thus required to read Piazza regularly!**

1.1 Objective

Euchre is a card game with two teams of two players using a 24-card deck, running from Aces down to the 9's (A, K, Q, J, T, 9) in 4 suits (H = Hearts, D = Diamonds, C= Clubs, S = Spades). The players sit around a common area, such as a tabletop, so that one team-pair (players 0 and 2) faces each other, and the other team-pair (players 1 and 3) faces each other, as in Figure 1.

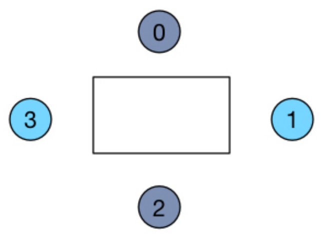

Figure 1

The objective of the game is to be the first team to score 10 (or more) points.

We ***HIGHLY*** recommend that you play this card game within your group ***BEFORE*** trying to implement the game.

1.2. Game Play

Before going into details of the game, there are a few terms that every euchre player ought to know.

1.2.1 Terminology

- A *round* is completed whenever each player has played a card.
- A *hand* consists of five rounds, *i.e.,* when all the cards that have been dealt have been played.
- The term *trump* is probably the most important concept in the game. The trump suit is set at the beginning of the hand by the trump-calling process, and the cards in that suit are the most powerful cards in the game. So, a 9 of the trump suit beats an ace of any other suit, although an ace of trump will defeat that same 9. Note, though, that the ace of trump is not the highest card for that hand (see the following definition of *bower*). This concept is similar to that used in the game of bridge.
- An *off-suit* is any suit but trump.
- A player is considered to be *void* in a particular suit if the player has no cards in that suit.
- The *kitty* is the stack of four cards left over in the deck after the deal is complete.
- The Jack of the trump suit is called the *right bower*. It is the highest card of all of the cards in the round. For example, if the trump suit is hearts, the Jack of hearts is the right bower.
- The Jack of the suit that is the same colour as the trump suit, but not the trump suit, is called the left bower. It is the second highest card of all of the cards in the round. If the trump suit is hearts, the Jack of diamonds is the left bower. After the bowers, the rest of the hearts cards in descending order (A, K, Q, T, 9) are highest. None of the other diamond cards count as trump, and follow the usual rules of play.
- In general, whenever a person wins a *round*, their team gets one *trick*. 1 point is given to the team who receives either 3 or 4 tricks in a hand, and 2 points if the team receives all 5 tricks in a hand. However, the opposing team can instead receive 2 points if a *euchre* is played (more on this later). *Hands* are played until one team (the winner) reaches 10 or more points.
- A player can *order up* the dealer during the calling trump process. This signifies that the dealer **must** pick up the card. This is clarified further in section 1.2.3.
- The player that is calling trump can choose to declare that they are *going alone*. (Note that if the player orders up their partner, they are implicitly choosing to go alone.) A player who does this believes that their hand is strong enough to take all 5 tricks without the assistance of their

partner. **Thus, their partner will not play any cards in the hand**. See section 1.2.5.1 for how this situation is scored.

- In certain circumstances, a player that has a single card in a particular suit can choose to *void* itself in that suit by playing that card when it cannot follow the suit of the led card. For example, if hearts are trump, the Ace of clubs is led and the next player plays the Queen of hearts, a player who has {9H, TH, AD, QS} in their hand may choose to play their Queen of spades so that if spades are led, they might have the opportunity to play one of their trump cards and possibly take the trick if everybody else has to follow suit.

1.2.2 Dealing

One player is chosen, normally at random, to deal the first hand. This player we will refer to as the *dealer*. After each hand is played, the deal passes clockwise around the table. So, as in Figure 2, if player 0 is chosen as the dealer for the first round, the deal passes to player 1 for the next round. From the viewpoint of player 0 sitting at the table, player 1 is sitting to its left.

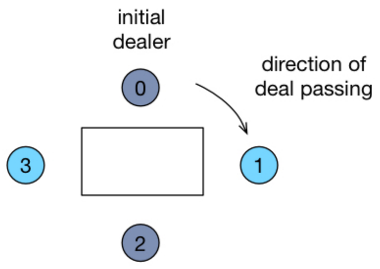

Figure 2

The dealer then deals 5 cards to each of the 4 players. At the end of the deal, 4 cards are left over, forming the kitty.

1.2.3 Calling Trump

The dealer turns over the **top** card of the kitty to reveal the suit proposed for trump. Starting with the first player to the dealer's left e.g. player 1, each player in turn has the option of either declining that suit as trump (called *passing*, and done by saying "pass"), or accepting that suit as trump by *ordering up* the dealer[1]. If all players pass, then the top card of the kitty is turned over to indicate this.

If the dealer is ordered up, they **must** pick up the top card of the kitty. It is up to them, however, as to which card from their hand is discarded, face-down, on the top of the kitty to bring their hand back down to 5 cards in number.

If all players pass, then starting with the player immediately to the dealer's left, each player in turn is given the chance to propose a trump suit. **A player may not propose the same suit as what was turned down!** The first player to declare a suit sets the trump suit for the round, and bidding stops. If nobody is willing to declare a suit trump, then the round is nullified i.e. all cards are returned to the deck and a new round is started.

[1] Note that the dealer can choose to make this suit trump by picking up the top card of the kitty and discarding one of the cards from their hand. There is no specific term for this, but it's not called "ordering up" in this situation.

1.2.4 Game play
The player to the left of the dealer *leads* by playing a card.[2] The card can be any card from the player's hand. Play then proceeds clockwise. The next 3 (2 if the person calling trump is going alone) players **must** play a card of the same suit, if able. If not, they may either *trump* (play a card of the trump suit) or *throw off* (play a card that is neither of the suit of the card played, nor of the trump suit). If one of the players does not follow suit, and that person is able to do so, then that player has *reneged* and an automatic 2 points is awarded to the other team. In real life, reneging is detected by a player noticing that someone played a card in a suit where they had previously not followed suit. In your game, the game logic can see the player's hands and tell if someone had a card in the suit to be followed or not.

1.2.4.1 Trumping
Trumping is accomplished by playing any card of the trump suit. Since even a nine of trump will beat any card of an off suit, this is sometimes a good idea.

1.2.4.2 Throwing off
Throwing off means playing a card that is neither trump nor follows suit. As mentioned above, if you can follow suit and you do not, then you have reneged. If you throw off, you personally cannot win the trick, so this is generally only a good idea when it looks like your partner will take the trick. It is also a good way to void yourself in a particular suit so that you can play trump on that suit if it is led later. For example, if a player has only a single spade card, and no clubs, when clubs are led, they may choose to play the spade if they can't take the trick so that they can later play a trump card if clubs are led again.

2.4.3 Winning rounds and finishing the hand
The person that played the highest trump card takes the trick. If no trump was played, then the trick is awarded to the person that played the highest card of the suit led. So, for example, if hearts are trump and clubs were led, and the cards played consisted of {AC, TC, 9C, 9S}, played in that order, then the Ace of clubs takes the trick. Remember that in off suits, aces are high. The player who took the trick leads the next round. The rest of the rounds are played in a similar fashion, with the winner of each round leading the next.

Once all 5 rounds have been played, each team totals their tricks. The team with the most tricks wins the hand.

1.2.5 Scoring
If the team that called trump takes either 3 or 4 tricks, they earn 1 point, plus 1 bonus point if they take all 5 tricks.

If the team that called trump fails to take at least 3 tricks, then this team is awarded 0 points and the other team is given 2 points. This is called a *euchre*.

1.2.5.1 Going alone
In the trump-calling process, if you think that your hand is really good, you may want to consider *going alone* if you are the one calling trump. This means that your partner does not play in the hand and your partner's cards are discarded face down. Thus, the hand is played with only 3

[2] The only exception to this rule is if the player directly clockwise of the dealer goes alone, then the player opposite the dealer leads; see section 1.2.5.1 for further details.

players. If you get all 5 tricks by yourself, then you score 4 points. If you get only 3 or 4 tricks, then you still only get 1 point. If you fail to obtain at least 3 tricks, the usual euchre scoring applies.[3]

2. Implementing Euchre

The program you are implementing has a text interface i.e. commands are entered from the keyboard, and output is sent to the screen.

Question: One possible additional enhancement is to add a graphical user interface. Explain briefly why, especially if you choose this enhancement, you would want to structure your project to follow the *MVC* design pattern.

Whenever input is required from the user, the program prints the prompt "`> `" (without the quotation marks). The list of valid commands is described in section 2.2.1.

2.1 Running your program

In order to be able to mark your program, your program needs to output the state of the game, such as the cards each player holds. This, however, doesn't make for a very interesting game, so this will be a feature that can be turned on by passing in a flag, `-d`, on the command line.

As well, the execution of the program must be repeatable for testing purposes, so your command-line argument must also be able to pass in an integer seed value, indicated by the `-s` flag, to the (pseudo) random number generator[4]. If no seed is specified on the command-line, the value of 0 is used for the seed. If both the seed and the debug flag are given, you may assume that the debug flag occurs first. You are not required to check that the seed is an integer.

For test purposes, you must also be able to initialize a game from a file. This lets us test very specific scenarios and responses. The name of the file will be provided on the command-line as the parameter following the `-f` flag. You may assume that if the `-f` flag is present, it will follow all previous flags. You may also assume that the specified file exists, is readable, and follows the format specified in section 3.

The format of the command line is as below, where the square brackets indicate the optionality of the argument, and are not physically present:

```
./euchre [[-d] [-s seed] [-f filename]]
```

If both a file and a seed are specified, the seed value is silently ignored since it is replaced by the value specified in the file. So, possible command invocations are:

```
./euchre
./euchre -d
./euchre -s 123
./euchre -d 123
./euchre -f saved-game.txt
./euchre -s 123 -f saved-game.txt
./euchre -d -s 123 -f saved-game.txt
```

When the game starts without being initialized from a file, the deck is initialized as described in section 2.3.1, and then shuffled as described in section 2.3.2. Player 0 is the initial dealer, and dealing

[3] There are scoring variations to this, but we'll stay with this simple version.
[4] See the description of how to shuffle the cards in section 2.3.2.

moves clockwise around the table, to player 1, then 2, etc. The cards will then be dealt simply, in that the first 5 cards will be given to player 1, the next 5 cards will be given to player 2, etc. The remaining 4 cards will be set aside as the *kitty*, and the top card of the kitty is turned over to face up. In debug mode, the contents of each hand and the kitty will be displayed as text to standard output; otherwise, just the top-card of the kitty and the human player's hand will be printed. See the sample executable, and section 2.4 for the output format.

As described in 2.2, initially there will be 1 human player and 3 computer players.

2.2 Implementing the players

Since teaching you how to communicate across a network is beyond the scope of the course, we will instead resort to having 1 player be the human playing the game, and the other 3 players be simple computer players. (If you are sufficiently knowledgeable to know of an easy way to do this, this would make a fine bonus.)

Question: There are several possible approaches to setting up the player classes so that the game can easily replace the human player with a computer player without having to change the game logic or how it interacts with the player class. Briefly explain the design approach you took.

Note: You will be often comparing cards to determine which is the largest, which will depend upon the trump suit, and the context, such as whether or not you must follow suit versus finding the largest card in a player's hand. You will also frequently want to count the number of cards in a suit, or the number of off-suit aces, or find the largest (or smallest) card in a particular suit. Take some care in designing your card class to make this easy. It would also be a good idea to create some helper methods to search the hand right at the beginning, and make sure that they work properly before you implement more complex sections of code.

2.2.1 Human player

The human player takes the role of player 0. Thus, player 2 is, by default, the human's partner while players 1 and 3 are the opposing team.

2.2.2 Computer player

The computer player (CP) that you code follows some very simple, fairly conservative strategies. This is the version that will be automatically tested, so even if you choose to provide other, more complex versions as part of your enhancements, this must always be the default version provided for testing.

The implementation, while not difficult, is somewhat complex. It is strongly suggested that you develop it incrementally, and start with something very simple, such playing/leading the first card in the hand, and declaring trump as the first suit in the player's hand.

Note: In all circumstances where the computer player must choose between suits, and its choice is not dictated by its hand and the specified strategies, **it will choose first hearts, then diamonds, then spades, then clubs** (described as H > D > S > C later in the document). For example, if it wants to void itself, and it has a choice of discarding either a 9S or a 9D, it will pick the 9D first due to the previously described preference (H > D > S > C).

2.2.2.1 Proposed trump strategies

Note that if the computer player is the dealer, it needs to include the top card of the kitty in its calculations for calling trump, though the card only becomes part of its hand if it picks it up. When a computer player must decide whether to accept the top-card of the kitty as trump:

- *If the CP is not the dealer, but its partner is the dealer*: it will order up its partner (hence going alone) only if it has:
    - the right bower, the left bower, and
        - either an ace in the trump suit, or
        - at least 1 ace in another suit.
    - the left bower and
        - either at least 2 more trump cards, or
        - at least 1 more trump card and at least one ace in another suit.
- *If the CP is not the dealer, and its partner is not the dealer*: it will order up the dealer and go alone if it has:
    - both bowers,
    - at least 1 ace in another suit, and
    - at least 1 more trump in addition to the bowers.
- *If the CP is not the dealer, and its partner is not the dealer*: it will order up the dealer if it has:
    - the right bower,
    - at least one other trump card, and
    - an ace in another suit.
- *If the CP is the dealer*: it will only pick up the card and go alone if it ends up with at least the right bower and one of the following situations:
    - at least 3 other trump, or
    - at least 2 other trump and at least 1 ace in another suit.
- *If the CP is the dealer*: it will only pick up the card if it ends up with:
    - at least the right bower and one of the following situations:
        - 2 additional trump cards, or
        - 1 additional trump card and an ace in another suit.
    - or a left bower and one of
        - at least 2 additional trump cards, or
        - at least 1 additional trump card and an ace in another suit.

In all other cases, the computer player passes on the suggested trump suit.

2.2.2.2 Declaring trump

If the computer player has the choice of declaring trump, you may assume that it will never try to declare the same suit as the top card of the kitty that was declined as trump. Note that the top card of the kitty is factored into the strength of the hand for the computer player **if it is the dealer**.

- *The CP will declare trump and go alone* if it has both the left and right bowers and any of the following situations:

    - at least 1 additional trump card[5], or
    - only 2 suits, including the trump suit, or
    - at least 1 outside ace.
- *The CP will declare trump but not go alone* if it has one of the following situations:
    - the right bower, and
        - either at least 2 other trump cards, or
        - 1 other trump card and an outside ace.
    - the left bower, and
        - either at least 3 other trump cards, or
        - 2 other trump cards and an outside ace.
    - 5 cards in the trump suit but no bowers i.e. AKQT9 in some suit.

In all other cases, it passes.

2.2.2.3 Discarding after picking up kitty

When deciding which card to discard, the computer player will find the lowest-ranked, non-trump card in its hand and discard that. If there are several cards with the same rank, it will choose to discard along the usual suit-preference rules (H>D>S>C).

If all of its cards are trump, it discards the lowest-ranked trump card in its hand.

2.2.2.4 Leading

Once a trump suit has been declared, the computer player chooses which card to lead based upon the following strategies:

- *If the CP called trump*, it leads the highest trump card in its hand. Once it is out of trump, it will lead any off-suit aces, then the highest card left, in order.
- *If the CP's partner called trump, and the CP has the right bower*, it leads the right bower to show its partner where the card is located.
- *If the CP's partner called trump, and the CP doesn't have the right bower*, it leads the lowest trump card in its hand in an attempt to flush out the opponents' trump cards, and let its partner take the trick. If it has no trump but at least one off-suit ace, it indicates this by leading an outside ace. If it doesn't have an outside ace, it leads the lowest card in the colour opposite to the trump suit, in the usual preference order. If it has no cards in the opposite colour, it discards the lowest-ranked card in the same colour.
- *If the opposing team called trump*:
    - if the CP has the right bower, it leads an outside ace if it has one in the hope that the ace will take one trick, and it can use the right bower to take another, thus hopefully taking at least 2 tricks.
    - If the CP doesn't have the right bower, it leads its highest card in an off-suit in an attempt to either take the trick or force the opposing team to use trump cards.
    - If the CP only has trump, it leads its highest trump card.

2.2.2.5 Following suit

When following suit, the computer player never reneges. It also has the following strategies:

[5] In terms of choosing which suit becomes trump in that case, it will always pick hearts over diamonds over spades over clubs (H>D>S>C). For example, if the top card of the kitty was 9S and it was turned down, and it has {JH, JD, AH, AD, 9C} in its hand, it could choose to make either hearts or diamonds trump, so it will pick hearts.

- *if the CP must follow suit, and 1) either it is going alone, or 2) its partner either doesn't have the trick or hasn't played yet*: if the computer player can take the trick, it will play the highest card in that suit from its hand. If the computer player must follow suit and can't take the trick, it will play the lowest card in that suit from its hand.
- *if the CP must follow suit, and its partner has the trick*: if the computer player can play a lower card than its partner's card, it will play the lowest card in that suit from its hand. If it doesn't have a lower card, it will play the highest card in its hand in that suit.
- *if the CP can't follow suit, and 1) either it is going alone, or 2) its partner either doesn't have the trick or hasn't played yet*:
    - *if the CP is not the last player and has trump*: play the highest trump card you own if it will take the trick so that you will either take the trick, or force out higher trump.
    - *if the CP is the last player and has trump*: play the highest trump card that will take the trick so that you take the trick but keep your high trump cards, if possible.
    - *if the CP has no trump*: void yourself in a suit if possible, or if that isn't possible, throw away the smallest off-suit card you can, subject to the usual suit-favouritism rule.
- *if the CP can't follow suit, and its partner has the trick*: void yourself in a suit if possible, or if that isn't possible, throw away the smallest off-suit card you can, subject to the usual suit-favouritism rule. If it only has trump cards, it plays the lowest trump card in its hand.

2.3 The deck

2.3.1 Initialization

The deck initially starts with the cards in the order:

```
9H TH JH QH KH AH 9D TD JD QD KD AD 9S TS JS QS KS AS 9C TC JC QC KC AC
```

2.3.2 Shuffling the deck

2.3.2.1 Algorithm

You are required to use the following algorithm and choice of pseudo-random number generator. Note that there must only be one random number generator, and it must be seeded exactly once. To help you, the code has also been provided in the file `shuffle.cc`.

```
#include <random>
int seed = 0;

void shuffle() {
    static mt19937 rng(seed);
    int n = CARD_COUNT;
    while ( n > 1 ) {
        int k = (int) (rng() % n);
        --n;
        Card *c = cards_[n];
        cards_[n] = cards_[k];
        cards_[k] = c;
    } // while
} // shuffle
```

Thus, for example, if the seed is 0, the deck after exactly 1 shuffle and no intervening calls to the random number generator, has the cards in the order:

```
AS QC KC KS JS 9D TS 9S JH JD AH 9H AD QS TC AC KD 9C TD QH QD TH KH JC
```

2.3.3 Dealing

We will use a non-standard but simple method for dealing the cards. The first 5 cards of the deck, `AS QC KC KS JS`, will be dealt to the player immediately to the dealer's left, clockwise around the table. The next 5 cards will be dealt to the second player to the dealer's left, etc. This is done until there are only 4 cards left. Those remaining cards form the kitty.

2.3.4 The Kitty

The kitty consists of the 4 remaining cards in the deck after dealing out the players' cards. To keep things consistent, we treat the first card in the sequence as the "top" card i.e. if the remaining cards consist of [`QD TH KH JC`], then `QD` is the top card. This makes diamonds the proposed trump suit.

2.3.5 Resetting

Because the game play has to be perfectly repeatable, you will need to find a way to remember the order of the cards after the deck is shuffled. Whenever the round is over, before the cards are shuffled the next time, you will have to return the deck's cards to their previous order.

For example, if the cards in the deck have the order:

```
AS QC KC KS JS 9D TS 9S JH JD AH 9H AD QS TC AC KD 9C TD QH QD TH KH JC
```

after being shuffled once with a seed of 0, they will need to be back in this order before being shuffled. If you shuffle the deck a second time, the subsequent order of cards will be:

```
9S KC 9D TS QD AH AC KD TD 9H KS QH QS AD JH KH 9C AS JC JD TH QC JS TC
```

2.4 The Display

2.4.2 Without the debug flag

At the start of each hand, the score for the two teams will be printed as follows:

```
Scores: Team 0-2 N, Team 1-3 N\n
```

Once the deck has been dealt out and the kitty has been formed, the game prints:

```
Top card of the kitty is: card\n
Proposed trump suit is: suit\n
Dealer is player X.\n
```

If the game state isn't being read from a file that sets the trump suit, the game prints:

```
Decision on trump starts with player Y.\n
```

As each player makes their decision to either pass or order up, the game prints either:

```
Player X: passes.\n
```

or one of:

```
Player X orders up player Y.\n
Player X orders up player Y and goes alone.\n
```

Note that the "pass" messages, including the turning down of the kitty's top card, are omitted if the trump suit is set in the input file used to initialize the game state. If the trump suit is set in the file, the appropriate "order up" or "declares" message is still printed.

If all players passed on the kitty, the game prints:

```
Kitty passed on, trump may not be suit.\n
```

For each player in turn, until a trump suit is chosen, the game prints:

```
Player X, choose a suit.\n
```

If a player chooses a trump suit, print one of:

```
Player X declares suit trump.\n
Player X declares suit trump and goes alone.\n
```

If all players pass, the game prints:

```
No trump declared, hand is nullified. Deal switches to player X.
```

The game will prompt the player for a response with one of the following messages:

```
Player X, choose card to discard.\n
Player X, choose card to lead.\n
Player X, choose card to play.\n
```

When a player leads, the game prints:

```
Player X leads card.\n
```

For all subsequent players, the game prints one of:

```
Player X plays card.\n
Player X skipped.\n
```

If a human player reneges, in addition to the error message described in section 2.5.1, the game prints the new score for the team that received the automatic point bonus, as in:

```
Score for Team X-Y is now N.\n
```

Once all eligible players have played in the round, the game prints:

```
Player X takes the trick.\n
Score for Team X-Y is now N.\n
```

If a team is euchred, the game prints:

```
Aw, team X-Y was euchred!\n
```

Once a team reaches 10 or more points, the game prints:

```
Team X-Y WINS!!!\n
```

2.4.2 With the debug flag

Providing the debug flag on the command-line causes additional information to be printed, so that you and the markers can verify what is occurring in your game logic. All of the previously discussed messages are still printed.

When the deck's seed is initialized, print the seed being used:

```
Seed: N\n
```

When the deck is shuffled at the start of the round, print the new order of the cards, where `AS` is the first card in the deck, and `JS` is the last card in the deck:

```
    Deck: AH QS KS KH JH 9D TH 9H JC JD AC 9C AD QH TS AS KD 9S TD QC QD TC KC
JS\n
```

When the kitty is formed, its contents will be printed as follows:

```
Kitty: card1 card2 card3 card4\n
```

Note that if the cards in the kitty are [`QD TH KH JC`], then *card*$_1$ is `QD`, *card*$_2$ is `TH`, etc. and the top card of the kitty is the `QD`.

When a player discards a card after picking up the top card of the kitty, print the following message:

```
Player X: discards card.\n
```

Whenever either the game state is read from file, or it is the start of a player's turn (one of choosing trump, discarding a card after being ordered up, leading or playing after the led card), the contents of their hand will be printed, along with the number of tricks they have taken so far in the round.

```
Player X: hand [card1 card2 card3 ... cardN] # tricks N\n
```

Note that the state of player 0's hand is always printed, so long as it is a human player.

2.5 The Command Interpreter

There are only a few commands that need to be supported, since the only input comes from the single human player, assuming they haven't rage quit yet. Note that whenever input is required from the human player, the game prints the prompt "`> `" so that the human knows to type a response. All output is sent to standard output, i.e. `std::cout`.

Command	**Explanation**
`q`	Terminates the program immediately.
`r`	Human player "rage quits", and is replaced by a computer player. The game prints "`Player 0 rage quit.\n`". The game play continues until the game is over i.e. one team wins with 10 or more points, and then the program terminates.
`p`	Used in the trump declaration process to indicate that the player is passing. Results in an unrecognized command error message if given in the wrong context, such as when the player needs to discard, lead, or play a card.
`o`	Used to order up the dealer in the trump declaration process while considering the top-card of the kitty as trump. Used even if the human player is the dealer and wants to pick up the top-card of the kitty.
`oa`	Same as before, but now the player is also going alone.
suit `n`	One of {`H`, `D`, `C`, `S`}. Used in the trump declaration process after the top-card of the kitty has been turned down. Specified suit is declared trump and the player is not going alone, unless it is an error as discussed in the following section. Results in an unrecognized command error message if given in the wrong context, such as when the player needs to discard a card. *Note that the parameter 'a' or 'n' is not read if the suit choice is invalid.*
suit `a`	Same as above, but the player is going alone.
card	Rank and suit, as in `9S`, for a card to be discarded/played/led, depending upon the context and prompt. Results in an unrecognized command error message if given in the wrong context, such as when the player needs to declare trump.

2.5.1 Dealing with errors

Being human, mistakes could happen, deliberately or otherwise. It is thus important that your implementation check the validity of all moves. Whenever input is required from the human, you need to produce the following error messages if the described error situation occurs.

Type of error	**Message produced**
Enters an invalid command. Human is re-prompted for a new command until a valid one is entered.	`ERROR: command 'x' is unrecognized. Please enter a valid command.\n`
Declares as trump the same suit as that of the turned-down top-card of the kitty. Human is re-prompted for a new, valid suit until one is entered.	`ERROR: invalid suit 'x' since previously turned down. Please enter a valid suit.\n`
Tries to discard, lead or play a card that isn't in the player's hand. Human is re-prompted for a new, valid card until one is entered.	`ERROR: card 'x' not in hand. Please enter a valid card.\n`
Enters the order-up command (normal or alone version) after the top-card of the kitty was turned down.	`ERROR: cannot order up if top card turned down. Please enter a valid command.\n`
Reneges when playing a card. Player is prompted for a new, valid card. Score for the other team increases by 2 points for every occurrence.	`ERROR: reneging with card 'x'. Please enter a valid card.\n`

3. Reading in a saved game

In order to keep things simple, you may assume that the format of the input file is always correct. (Our sample solution has some error-checking present to help detect errors in the input file content.) If trump hasn't been declared yet, assume that the processing of the top-card of the kitty needs to be done. Player 0 is always initially a human player.

While we have only shown a single space or a newline between values, the sample solution will skip intervening white space as necessary. You may assume that our test case input files will follow a similar format. See `providedFiles/in*.{txt,in,out}` for some sample game state files, input for player 0, and the resulting output.

Input file format	**Description**
`0 0\n`	Scores of team 0-2 and team 1-3 respectively.
`0 1\n`	Id of the current dealer, and id of which player is next. *This is done to simplify testing certain test case scenarios, and thus is not always the player immediately following the dealer.*
`?\n`	Trump suit: '`?`' if trump not declared, else one of '`H`', '`C`', '`D`', '`S`'.
`0 f\n`	If trump was declared, id of who called it and either `f` if not going alone, or `t` if going alone. *If trump was not declared, this line is omitted.*
`123\n`	Seed for pseudo-random number generator.

`AH QS KS KH JH 9D TH 9H JC JD AC 9C AD QH TS AS KD 9S TD QC QD TC KC JS\n`	Order of cards in the deck, from first to last.
`5 AH QS KS KH JH\n`	# of cards and cards currently in player 0's hand
`0\n`	# of cards and cards from tricks taken by player 0
`5 9D TH 9H JC JD\n`	# of cards and cards currently in player 1's hand
`0\n`	# of cards and cards from tricks taken by player 1
`5 AC 9C AD QH TS\n`	# of cards and cards currently in player 2's hand
`0\n`	# of cards and cards from tricks taken by player 2
`5 AS KD 9S TD QC\n`	# of cards and cards currently in player 3's hand
`0\n`	# of cards and cards from tricks taken by player 3

Grading

Your project will be graded as follows:

Correctness and Completeness 60%	Does it work? Does it implement all of the requirements?
Documentation 20%	Plan of attack; final design document.
Design 20%	UML; good use of separate compilation, good object-oriented practice; is it well-structured, or is it one giant function?

Even if your program doesn't work at all, you can still earn a lot of marks through good documentation and design, (in the latter case, there needs to be enough code present to make a reasonable assessment).

Question: Many games follow a common set of steps when being played. Explain how you could apply the *Template Method* design pattern to your euchre implementation i.e. what class or classes would be affected.

Question: The *Abstract Factory* design pattern, discussed in Chapter 4 of the "Head First Design Patterns" book, ensures that only related items within a family are created. For example, if you were creating a window displaying system, the look of the buttons on a Mac versus a Windows window are different, and if you were creating a window for a Mac, you would only want a Mac's "look and feel". Similarly, if you were marketing your software globally, you could use the Abstract Factory to ensure that all text and messages were in the appropriate language. If you wanted to provided different euchre variants, discuss how you would apply the Abstract Factory design pattern.

If Things Go Wrong

If you run into trouble and find yourself unable to complete the entire assignment, please do your best to submit something that works, even if it doesn't solve the entire assignment. For example:

- Is the deck correctly initialized and shuffled?
- Are the cards dealt out correctly and the kitty properly formed?
- Does the correct player start?

- Does the trump suit get passed or accepted correctly based upon the top card of the kitty and the player hands?
- If the proposed trump suit is declined, is the correct trump suit declared based upon the player hands?
- If the proposed trump suit is declined, do all computer players pass on declaring trump correctly based upon the player hands?
- Does a computer player lead the correct card based upon the trump suit?
- Do computer players play appropriately, based upon what is led and what is trump?
- Are tricks correctly taken and scored?

You will get a higher mark for fully implementing some of the requirements than for a program that attempts all of the requirements, but doesn't run.

Make sure that your program is at least able to load a pre-saved game and display the game.

A well-documented, well-designed program that has all of the deficiencies listed above, but still runs, can still potentially earn a passing grade.

Plan for the Worst

Even the best programmers have bad days, and the simplest pointer error can take hours to track down. So be sure to have a plan of attack in place that maximizes your chances of always at least having a working program to submit. Prioritize your goals to maximize what you can demonstrate at any given time.

One of the first things you should probably do is write a routine to read in and output a card and a container of cards, such as a hand or a deck. Then work on manipulating your container of cards. You should also develop the ability to read in pre-saved games early on. This will ensure that we are able to test features faster by loading our pre-saved games. You should also do the command interpreter early, so that you can interact with your program. You can then add commands one-by-one, starting with '`q`' and '`p`', and separately work on supporting the full command syntax. Work on your game steps and strategies one at a time. Don't try to build everything all at once! Start with a rough structure and fill in capabilities as you go.

Take the time to work on a test suite at the same time as you are writing your project. Although we are not asking you to submit a test suite, having one on hand will speed up the process of verifying your implementation.

You will be asked to submit a plan, with projected completion dates and divided responsibilities, as part of your documentation for Due Date 1.

If Things Go Well

If you complete the entire project, you can implement extra features for a few extra credits. These should be outlined in your design document, and markers will judge the value of your extra features.

Submission Instructions

See `Guidelines.pdf` for instructions about what should be included in your plan of attack and final design document.

Due Dates

Due Date 1 (July 16, 2018): Due on due date 1 is your plan of attack, `deadlines.txt`, and initial UML diagram for your implementation of euchre, `a5-uml-initial.pdf`.

Due Date 2 (July 25, 2018): Due on due date 2 is your actual implementation of euchre. All `*.h`, `*.cc`, `*.cpp` and any text files needed for your project to compile and run should be included in `euchre.zip`. The ZIP file must contain a suitable `[Mm]akefile` such that typing `make` will compile your code and produce an executable named `euchre`. In addition, upload `a5-uml-final.pdf` and `a5-design.txt` to the appropriate place on marmoset.