VE280 Project 3: Blackjack

Out June 29th 18:30, 2021; **Due** July 9th 23:59, 2021

Ⅰ. Background

The Cultural Festival turned out a success for `SOS Brigade`. After struggling for making the movie for so long, they finally went back to their normal routine. However, a group of five huge men broke into their club activity room and asked the members of `SOS Brigade` to play their designed game `Blackjack`. Actually, the computer storing `SOS Quantum Library` (see Lab1 ex.1) originally belonged to `Joseph Joestar` , but `Suzumiya Haruhi` robbed it. At that time `Joseph` was busy with dealing with `Dio Brando` and did not have time to deal with `Haruhi`. Now `Dio Brando` is dead and `Joseph` comes back to get his computer back. The `Stardust Crusaders` led by `Joseph Jostar` will compete `SOS Brigade` led by `Suzumiya Haruhi` through this game. `Joseph Jostar` will borrow 4 computers to `SOS Brigade`. If `Stardust Crusaders` wins, `Joseph` will get his computer back and get a member of `SOS Brigade` into `Stardust crusaders`. If `SOS Brigade` wins, those four computers will belong to `SOS Brigade` and `Stardust Crusaders` will be an affiliate group of `SOS Brigade`.

Ⅱ. Introduction

You need to firstly help `Joseph Joestar` to implement this card game. It is played with a standard deck of 52 playing cards. There are two participants, a **dealer** and a **player**. The **player** starts with a **bankroll**, and the game progresses in **hands** and **rounds**.

At the start of each **hand**, the **player** decides how much to **wager** on this **hand**. The **wager** is an integer greater or equal to a **minimum** allowable wager and less than or equal to the player's total bankroll.

After the wager, the dealer deals a total of four cards:

1. face-up to the player,
2. face-up to the dealer,
3. face-up to the player,
4. face-down to the dealer, **hole card**.

The player then examines his cards and calculate the total number of his card's worth. Different cards have different worth:

- 2-10 is worth its spot value
- face card (jack, queen, king) is worth 10
- Ace is worth 1 or 11, depending on which is more advantageous to the player.

If the total includes an ace counted as 11, the total is called **soft**, otherwise it is called **hard**.

The total only depends on the cards in the player/dealer's hand, but not on what he/she expose. Cheating should not be regarded as cheating as long as it does not get caught. They will not memorize the opposite's cards in his/her hand.

The game progresses first with the player, then the dealer. The player's goal is to build a hand that is as close to 21 as possible without going over---the latter is called a **bust**, and a player who busts loses the hand and his bankroll will decrease by his wager. For cards whether soft or hard, players/dealers only care about the current situation, i.e. they are **greedy**. They do not have the ability to foresee. As long as the player believes another card will help, the player **hits**---asks the dealer for another card. Each of these additional cards is dealt **face-up**. This process ends either when the player decides to **stand**---ask for no cards, or the player busts. Note that a player can stand with two cards; one need not hit at all in a hand.

Natural 21: The player is dealt an ace plus any 10 or face card (jack, queen, king) in his hand. His bankroll will increase by wager * 3 / 2.

If the player neither busts nor is dealt a natural 21, play then progresses to the dealer. The dealer **must** hit until he either reaches a total greater than or equal to 17 (hard or soft), or busts. If the dealer busts, the player wins. Otherwise, the two totals are compared:

- If the dealer's total is higher, the player's bankroll decreases by the amount of his wager.
- If the player's total is higher, his bankroll increases by the amount of his wager.
- If the totals are equal, the bankroll is unchanged; this is called a **push**.

A **hand** ends in four cases:

1. The player get paid.
2. The player's bankroll decreases.
3. They push.
4. The cards are used up, then they **shuffle** the cards and do this **hand** again.

A **round** consists of several hands. There is a maximum number of hands for each rounds. A round will end whenever one player is kicked out. A round ends in three cases:

1. `Joseph Joestar` will escape if he is the player, i.e. his bankroll is less than one half of the normal bankroll.
2. The player's bankroll cannot reach that **minimum** allowable wager. Then the player is kicked out and a new round start from Hand

   The dealer's remaining hand will be reset to the maximum number of hands.
3. Otherwise, if the allotted hands are all played and the player's bankroll can still reach that **minimum** allowable wager, then the dealer is kicked out and a new round start from Hand 1. The player's remaining hand will be reset to the maximum number of hands.

During the whole game process (before all the members of one team are kicked out), they will use the same set of cards, i.e. the cards will not be reset after the game starts.

When one **round** ends and the player wins, the bankroll will not be reset to the start bankroll. Instead, everyone will be provided the bankroll only when they enter the game. During the game process, i.e. the player has not been kicked out, the player can only get money from the dealer. When the player is kicked out of the game, he/she will **not** pass his/her remaining money to the next player.

Ⅲ. Programming Assignment

You will provide implementations of four separate abstractions for this project: a deck of cards, a blackjack hand, a blackjack player, and a game driver. All files referenced in this specification are located in the Projects/Project3 folder on Canvas.

1. The Deck ADT

Your first task is to implement the following ADT and put your implementation into a file named `deck.cpp`.

```
class DeckEmpty { // An exception type
};

const int DeckSize = 52;

class Deck {

// A standard deck of 52 playing cards---no jokers

    Card deck[DeckSize]; // The deck of cards
    int next; // The next card to deal

public:

    Deck();
    // EFFECTS: constructs a "newly opened" deck of cards. first the
    // spades from 2 to A, then the hearts, then the clubs, then the
    // diamonds. The first card dealt should be the 2 of Spades.

    void reset();
    // EFFECTS: resets the deck to the state of a "newly opened" deck
    // of cards.

    void shuffle(int n);
    // REQUIRES: n is between 0 and 52, inclusive.
    // MODIFIES: this
    // EFFECTS: cut the deck into two segments: the first n cards,
    // called the "left", and the rest called the "right". Note that
    // either right or left might be empty. Then, rearrange the deck
    // to be the first card of the right, then the first card of the
    // left, the 2nd of right, the 2nd of left, and so on. Once one
    // side is exhausted, fill in the remainder of the deck with the
    // cards remaining in the other side. Finally, make the first
    // card in this shuffled deck the next card to deal. For example,
    // shuffle(26) on a newly-reset() deck results in: 2-clubs,
    // 2-spades, 3-clubs, 3-spades ... A-diamonds, A-hearts.
    //
    // Note: if shuffle is called on a deck that has already had some
    // cards dealt, those cards should first be restored to the deck
    // in the order in which they were dealt, preserving the most
    // recent post-shuffled/post-reset state. After shuffling, the
    // next card to deal is the first one in the deck.

    Card deal();
    // MODIFIES: this
    // EFFECTS: returns the next card to be dealt. If no cards
    // remain, throws an instance of DeckEmpty.

    int cardsLeft();
    // EFFECTS: returns the number of cards in the deck that have not
    // been dealt since the last reset/shuffle.
};
```

The Deck ADT is specified in `deck.h`. The Deck ADT depends on the following Card type declared in `card.h` The file `card.cpp` defines SpotNames, SuitNames, SOS_Name, SC_Name for you, so that SuitNames[HEARTS] is the char string “Hearts”, and so on.:

```
enum Suit {
        SPADES, HEARTS, CLUBS, DIAMONDS
};

enum Team {
    SOSBrigade, StardustCrusaders
};

enum Spot {
    TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE
};

extern const string SuitNames[DIAMONDS + 1];
extern const string SpotNames[ACE+1];
extern const string SOS_Name[5]; // The full name of each member in SOS Brigade
extern const string SC_Name[5]; // The full name of each member in Stardust
Crusaders

struct Card {
    Spot spot;
    Suit suit;
};
```

2. The Hand Interface

Your second task is to implement Hand APT specified in `hand.h` and put your implementation into a file named `hand.cpp`.

```
struct HandValue {
    int  count;    // Value of hand
    bool soft;     // true if hand value is a soft count
};

class Hand {
    // OVERVIEW: A blackjack hand of zero or more cards

    // Note: this really is the only private state you need!
    HandValue curValue;

public:

    Hand();
    // EFFECTS: establishes an empty blackjack hand.

    void discardAll();
    // MODIFIES: this
    // EFFECTS: discards any cards presently held, restoring the state
    // of the hand to that of an empty blackjack hand.

    void addCard(Card c);
    // MODIFIES: this
    // EFFECTS: adds the card "c" to those presently held.

    HandValue handValue() const;
    // EFFECTS: returns the present value of the blackjack hand.  The
    // count field is the highest blackjack total possible without
    // going over 21.  The soft field should be true if and only if at
    // least one ACE is present, and its value is counted as 11 rather
    // than 1. If the hand is over 21, just return the handvalue as what it is.
};
```

3. The Player Interface

Your third task is to implement two different blackjack players, specified in `player.h`. You are to implement two different derived classes from this interface and put them in a file named `player.cpp`.

```
class Player {
    // A virtual base class, providing the player interface
protected:
    Team team; // The team of each member, either SOS Brigade or Stardust
Crusaders
    int ID; // The ID of each member.
    string name; // The full name of each member.

public:
    virtual int bet(unsigned int bankroll,
                    unsigned int minimum) = 0;
    // REQUIRES: bankroll >= minimum
    // EFFECTS: returns the player's bet, between minimum and bankroll
    // inclusive

    virtual bool draw(Card dealer,            // Dealer's "up card"
                    const Hand &player) = 0; // Player's current hand
    // EFFECTS: returns true if the player wishes to be dealt another
    // card, false otherwise.

    virtual void expose(Card c) = 0;
    // EFFECTS: allows the player to "see" the newly-exposed card.
    // For example, each card that is dealt "face up" is exposed, i.e. expose().
    // Likewise, if the dealer must show his "hole card", it is also exposed,
i.e.
    // expose().  Note: not all cards dealt are exposed, i.e. expose()---if the
    // player goes over 21 or is dealt a natural 21, the dealer need
    // not expose his hole card.

    virtual void shuffled() = 0;
    // EFFECTS: tells the player that the deck has been re-shuffled.\

    virtual string getName();
    // EFFECTS: get the full name of the player.

    virtual int getID();
    // EFFECTS: get the ID of the player.

    virtual Team getTeam();
    // EFFECTS: get the team of the player.
    virtual void setPlayer(Team tm, int id);
    // EFFECTS: set a member's name, ID and team
    // MODIFIES: name, ID, team

    virtual ~Player() { }
    // Note: this is here only to suppress a compiler warning.
    //       Destructors are not needed for this project.
};
```

The first derived class is the simple player, who plays a simplified version of basic strategy for blackjack. The simple player always places the minimum allowable wager, and decides to hit or stand based on the following rules and whether or not the player has a **hard count** or **soft count**:

The first set of rules apply if the player has a **hard count**, i.e., his best total counts an Ace (if any) for 1, not 11.

- If the player's hand totals 11 or less, he always hits.
- If the player's hand totals 12, he stands if the dealer shows 4, 5, or 6; otherwise he hits.
- If the player's hand totals between 13 and 16 inclusive, he stands if the dealer shows a 2 through a 6 inclusive; otherwise he hits.
- If the player's hand totals 17 or greater, he always stands.

The second set of rules applies if the player has a "**soft count**" , i.e., his best total includes **one** Ace worth 11. (Note that a hand would never count two Aces as 11 each--that's a bust of 22.)

- If the player's hand totals 17 or less, he always hits.
- If the player's hand totals 18, he stands if the dealer shows a 2, 7, or 8, otherwise he hits.
- If the player's hand totals 19 or greater, he always stands.

Note: the Simple player does nothing for **expose** and **shuffled** events.

The second derived class is the Counting player. This player counts cards in addition to playing the basic strategy. The intuition behind card counting is that when the deck has more face cards (worth 10) than low-numbered cards, the deck is favorable to the player. The converse is also true.

The Counting player keeps a running "count" of the cards he's seen from the deck. Each time he sees (via the expose() method) a 10, Jack, Queen, King, or Ace, he subtracts one from the count. Each time he sees a 2, 3, 4, 5, or 6, he adds one to the count. When he sees that the deck is shuffled(), the count is reset to zero. Whenever the count is +2 or greater **and** he has enough bankroll (**larger than or equal to** the double of the minimum), the Counting player bets double the minimum, **otherwise** (i.e., including the situation where count >= +2 but the bankroll is less than the double of the minimum) he bets the minimum. The Counting player should not re-implement methods of the Simple player unnecessarily.

You must also declare static global instances of each of the Players you implement in your player.cpp file. Finally, you should implement "access" functions like this that return pointers to each of these global instances in your player.cpp file. You should use this single function to get any player.

```
extern Player* get_Player(string& dealerSide, string& playerType, int& ID);
// EFFECTS: get a pointer to a player.
// "dealerSide" describes whether the dealer
is from SOS Brigade or Stardust Crusade. This depends on the last program
argument: [sos|sc]. sc means the dealer team is Stardust Crusaders, sos means the
dealer team is SOS Brigade.
// "playerType" describes whether Koizumi Itzuki and Mohammed Avdol are simple p
layer or count player. This depends on the penultimate program argument:
[simple|counting]. If this argument is "simple", then Itzuki and Avdol are simple
players. If this argument is "counting", then Itzuki and Avdol are
countingplayers.
// "ID" is the player's ID.
```

4. Characters

There are in total 10 people in this game who have different characteristics. They are from two sides. Normally, they have same bankroll and same hands. One team will be selected as player team and the other will be dealer team. This depends on program arguments. In "[]" are characteristics that these people have only when they are **players**. The ID of each character is given after the name of each character. i.e. The IDs of `Kyon` and `Kakyoin Noriaki` are both 4.

SOS Brigade

1. **Suzumiya Haruhi** (ID: 1) Leader of SOS Brigade. [Due to her preference to make things simple, she is a simple player. But she is very brave, she always bets double the minimum.]
2. **Nagato Yuki** (ID: 2) Human interface alien. [She is a normal counting player.]
3. **Asahina Mikuru** (ID: 3) Not good at games. [She is a counting player, but she will act contrary to a counting player, i.e. she will double the minimum only when the count is less than -2 and she has enough bankroll.]
4. **Kyon** (ID: 4) [He is a simple player.]
5. **Koizumi Itzuki** (ID: 5) [His player type (simple or count) depends on user input.]

Stardust Crusaders

1. **Joseph Joestar** (ID: 1) Leader of Stardust Crusaders. [He is a rich counting player, having 2 times of normal bankroll as his own bankroll, but he fears being poor. If his bankroll is less than one half of the **normal** bankroll, he will escape, shout "Ni Ge Run Da Yo" (Escape fast in Japanese) and lost the game.] If he successfully kicks out an enemy, he will cheer "Nice!".
2. **Kujo Jotaro** (ID: 2) [He is a count player.] However, every time he is about to bust, he will shout "Star Platinum, Za Warudo" to activate his stand power, stop the time, abandon his current cards and take the same number of cards again. He will stop time for multiple times until he got a set of cards that will not bust. He will not expose his card got when time is stopped. It may happen that the cards are used up. Please note that dealers can also bust.
3. **Jean Pierre Polnareff** (ID: 3) [He is a simple player.]
4. **Kakyoin Noriaki** (ID: 4) [He is a very cautious count player so that he will only bet double the minimum when count >= +4 and he has enough bankroll.] After he defeats an enemy, he will eat a cherry, producing a sound of "rerorerorero rerorerorero".
5. **Mohammed Avdol** (ID: 5) [His player (simple or count) type depends on user input.]

For SOS Brigade, they will come into the game according to the sequence: **Suzumiya Haruhi** -> **Nagato Yuki** -> **Asahina Mikuru** -> **Kyon** -> **Koizumi Itzuki**. For Stardust Crusaders, they will come into the game according to the sequence: **Joseph Joestar** -> **Kujo Jotaro** -> **Jean Pierre Polnareff** -> **Kakyoin Noriaki** -> **Mohammed Avdol** . The detailed kick-out rule is in `5. The`

`Driver Program`.

5. The Driver Program

Finally, you are to implement a driver program that can be used to simulate this version of blackjack given your implementation of the ADTs described above and put your implementation into a file named `blackjack.cpp`.

The driver program, when run, takes four arguments:

```
<bankroll> <hands> [simple|counting] [sc|sos]
```

The first argument is an integer denoting the player’s starting bankroll. The second argument is the maximum number of hands for each player to play in the simulation. **You can assume that these two integers input by the user are positive (≥1) and within an upper limit of 10000.** The third argument is one of the two strings “simple” or “counting”, denoting those player's type (simple or counting) which will depend on program arguments. The fourth argument decides which team is the dealer team.

For example, suppose that you program is called blackjack. It may be invoked by typing in a terminal:

```
./blackjack 100 3 simple sc
```

Then `Koizumi Itzuki` and `Mohammed Avdol` are simple players. The dealer team will be `Stardust Crusaders`. Each player will have at most 3 hands. When one person is kicked out, the other's remaining hand will be reset to 3.

5.1 Rounds

When one round start, the player and the dealer come into the game. The driver first shuffles the deck. To shuffle the deck, you choose **seven** cuts between 13 and 39 inclusive **at random**, shuffling the deck with each of these cuts. We have supplied a header, `rand.h`, and an implementation, `rand.cpp`, that define a function that provides these random cuts. Each time the deck is shuffled, first announce it:

```
cout << "Shuffling the deck" << endl;
```

And announce each of the **seven** cut points:

```
cout << "cut at " << cut << endl;
```

then be sure to tell the player via shuffle().

Note: you should always print the message corresponding to the initial shuffling before you do anything further.

We assume that the **minimum bet** is 5. Then, while the player's bankroll is larger than or equal to the minimum bet of 5 and there are hands left to be played, we enter the hand process in *5.2 Hands*.

You can now begin to read *5.2 Hands* and then come back here.

There are in total 3 cases where the process of *5.2 Hands* in a round should come to an end:

- the player has too little money to make a minimum wager
- the allotted hands have been played
- Player `Joseph Joestar` should escape (but he will firstly announce his bankroll before escape)

When the *5.2 Hands* process ends, the driver program should announce the outcome:

```
 cout << "Player: " << Player_Fullname << " has " << bankroll << " after " <<
thishand << " hands" << endl;
```

where the variable `thishand` is the current hand number. In the special case where the initial bankroll is less than the minimum, we have `thishand` = 0, since the player hasn't played any hand yet. Furthermore, in this special case, the initial shuffling of the deck should still be announced before you print the status of the player.

At this time, if `Joseph Joestar` is the player and he should escape, he will say:

```
Player: Joseph Joestar: "Ni Ge Run Da Yo"
```

, escape, lose this round and be automatically kicked out.

Then, it's time to release the outcome of the round. If the allotted hands have all been played and the player have enough money to at least make a minimum wager, the player will win this round and the dealer will be kicked out of the game. The next dealer will come and start a new round from Hand 1. The program will announce the outcome:

```
cout << "Dealer: " << Dealer_Fullname << " has been kicked out. The winner of
this round is "<< Player_Fullname <<"." << endl;
```

If the player has too little money to make a minimum wager, the dealer wins this round, and the player will be kicked out of the game. The next player will come and start a new round from Hand 1. The program will announce the outcome:

```
cout << "Player: " << Player_Fullname << " has been kicked out. The winner of
this round is "<< Dealer_Fullname <<"." << endl;
```

At this time, `Joseph Joestar` and `Kakyoin Noriaki` will get to know whether they wins. If they wins, they will cheer after the "kick-out" message like this:

```
Player: Kakyoin Noriaki: "reroererorero reroererorero"
Dealer: Joseph Joestar: "Nice!"
```

5.2 Hands

Firstly, the driver program will announce the hand:

```
cout << "Hand " << thishand << " bankroll " << bankroll << endl;
```

where the variable `thishand` is the hand number, **starting from 1**.

If there are fewer than 20 cards left, reshuffle the deck as described above.

The driver program will then ask the player for a wager and announce it:

```
cout << "Player: " << Player_Fullname << " bets " << wager << endl;
```

where `Player_Fullname` is current player's full name.

Then, the driver program will deal four cards: one face-up to the player, one face-up to the dealer, one face-up to the player, and one face-down to the dealer. Announce the face-up cards using `cout`. For example:

```
Player: Suzumiya Haruhi dealt Ace of Spades
Dealer: Joseph Joestar dealt Two of Hearts
```

Use the `SpotNames`, `SuitNames`, `SOS_Name`, `SC_Name` arrays for this, and be sure to expose() any face-up cards to the player.

If the player is dealt a natural 21, immediately pay the player 3/2 of his bet. In this case, announce as below, and this hand ends:

```
cout << "Player: " << Player_Fullname << " dealt natural 21" << endl;
```

If the player is not dealt a natural 21, have the player play his hand. Draw cards until the player either stands or busts. Announce and expose() each card dealt as above.

①If `Kujo Jotaro` is the player and he is about to bust when he get **the last card that will make him bust**, he will immediately shout

```
Player: Kujo Jotaro: "Star Platinum, Za Warudo"
```

activate his stand power, stop the time and take the same number of cards **without announcing (but still counting) that particular card**. He will not announce (but still count) his cards when picking cards when time is stopped. The time will begin to flow after he get the same number of cards. However, if he finds that he is about to bust again, **he will immediately stop the time again**...

②After `Kujo Jotaro` flows the time and **will not stop the time immediately again**, i.e. his cards will not make him bust, he should firstly announce (and also count) **his last card** that he get.

③In other words, `Kujo Jotaro` will only announce the last card that will make him not bust from ① to ②, but he will always keep on counting.

④Then he will check whether he should stand. If not, he will keep getting cards and exposing according to the rule. However, it may happen that he is again about to bust, then he should stop the time again **without announcing (but still counting) the last card that will make him bust**...

Then it's time to announce the player's total

```
 cout << "Player: " << Player_Fullname << "'s total is " << player_count <<
endl;
```

where the variable `player_count` is the total value of the player's hand.

Then, if the player busts, say so:

```
cout << "Player: " << Player_Fullname << " busts" << endl;
```

deducting the wager from the bankroll and moving on to the next hand.

If the player hasn't busted, announce and expose the dealer's hole card. For example:

```
Dealer: Joseph Joestar's hole card is Ace of Spades
```

(Note: the hole card is NOT exposed if either the player busts or is dealt a natural 21.)

If the player hasn't busted, play the dealer's hand. The dealer must hit until reaching seventeen or busting. Announce and expose each card as above.

①If `Kujo Jotaro` is the dealer and he is about to bust when he get **the last card that will make him bust**, he will immediately shout

```
Dealer: Kujo Jotaro: "Star Platinum, Za Warudo"
```

activate his stand power, stop the time and take the same number of cards **without exposing that particular card**. He will not expose his cards when picking cards when time is stopped. The time will begin to flow after he get the same number of cards. However, if he finds that he is about to bust again, **he will immediately stop the time again**...

②After `Kujo Jotaro` flows the time and **will not stop the time immediately again**, i.e. his cards will not make him bust, he should firstly expose **his last card** that he get.

③In other words, `Kujo Jotaro` will only expose the last card that will make him not bust from ① to ②.

④Then he will check whether he should stand. If not, he will keep getting cards and exposing according to the rule. However, it may happen that he is again about to bust, then he should stop the time again **without exposing the last card that will make him bust**...

Then it's time to announce the dealer's total

```
 cout << "Dealer: " << Dealer_Fullname << "'s total is " << dealer_count <<
endl;
```

where the variable dealer_count is the total value of the dealer's hand.

If the dealer busts, say so:

```
 cout << "Dealer: " << Dealer_Fullname << " busts" << endl;
```

crediting the wager from the bankroll and moving on to the next hand.

If neither the dealer nor the player bust, compare the totals and announce the outcome. Credit the bankroll, debit it, or leave it unchanged as appropriate.

```
cout << "Dealer: " << Dealer_Fullname << " wins this hand" << endl;
cout << "Player: " << Player_Fullname << " wins this hand" << endl;
cout << "Push" << endl;
```

If all of the three cases below are satisfied:

- the player's bankroll is larger than or equal to the minimum bet of 5,
- there are hands left to be played,
- `Joseph Joestar` will not escape if he is the player,

then continue to play the next hand (i.e., start again from the beginning of *5.2 Hands*).

If the cards are used up at any time during the hand, this **hand** will immediately be stopped, output the below message, shuffle the card and start this hand again.

```
cout << "Hand " << thishand << " card used up, this hand will start again" <<
endl;
```

5.3 Game Terminate

When all the members of one team are kicked out, game ends. The program's output depends on the winning side. For example,

```
Game over. The winner is Stardust Crusaders. SOS Brigade will return the computer
to Joseph Joestar and Suzumiya Haruhi will become a member of Stardust Crusaders
# When Stardust Crusaders wins
```

```
Game over. The winner is SOS Brigade. SOS Brigade got four computers and Stardust
Crusaders become an affiliate group of SOS Brigade # When SOS Brigade wins
```

Messages after # should not go to output. They are just comments.

IV. Implementation Requirements and Restrictions

- You may include `<iostream>`, `<iomanip>`, `<string>`, `<cstdlib>`, and `<cassert>`. No other system header files may be included, and you may not make any call to any function in any other library.
- Output should only be done where it is specified.
- You may not use the `goto` command.
- You may not have any global variables in the driver. You may use global state in the class implementations, but it must be `static` and (except for the two players) `const`.
- There is no user input. You may assume that functions are called consistent with their advertised specifications. This means you need not perform error checking. However, when testing your code in concert, you may use the assert() macro to program defensively.

V. Source Code Files and Compiling

There are five header files (`card.h`, `deck.h`, `hand.h`, `player.h`, and `rand.h`) and two C++ source files (`card.cpp` and `rand.cpp`) located in the `Project3_starter_file.zip` from our Canvas resources:

You should copy these files into your working directory. **DO NOT modify them!**

You need to write four other C++ source files: `deck.cpp`, `hand.cpp`, `player.cpp`, and `blackjack.cpp`. They are discussed above and summarized below:

- `deck.cpp`: your Deck ADT implementation
- `hand.cpp`: your Hand ADT implementation
- `player.cpp`: your player ADT implementations
- `blackjack.cpp`: your simulation driver

After you have written these files, you can type the following command in the terminal to compile the program:

```
g++ -Wall -o blackjack blackjack.cpp card.cpp deck.cpp hand.cpp player.cpp
rand.cpp
```

This will generate a program called blackjack in your working directory. In order to guarantee that the TAs compile your program successfully, you should name you source code files exactly like how they are specified above. For this project, the penalty for code that does not compile will be **severe**, regardless of the reason. You should submit four source code files `deck.cpp`, `hand.cpp`, `player.cpp`, and `blackjack.cpp` (in one compressed file) via JOJ. The due time is 23:59 on July 9th, 2021.

VI. Grading

Your program will be graded along three criteria:

1. Functional Correctness
2. Implementation Constraints
3. General Style

Functional Correctness is determined by running a variety of test cases against your program, checking against our reference solution. We will grade Implementation Constraints to see if you have met all of the implementation requirements and restrictions. General Style refers to the ease with which TAs can read and understand your program, and the cleanliness and elegance of your code. For example, significant code duplication will lead to General Style deductions.

VIII. Last

The game `SOS Brigade` played is actually `The Day of Sagittarius Ⅲ`, originally designed as a competition between `Computer Research Club` and `SOS Brigade` in `The Melancholy of Haruhi Suzumiya 2009`, `episode 27 The Day of Sagittarius`. You can watch the original game playing process in bilibili (need a VIP) or in 5dm (better subtitle, less danmaku).

I am currently working on `The Day of Sagittarius Ⅳ`, another game based on the same background as this project. Hopefully it will become a potential future project of VE280.

The game `Stardust Crusaders` played is in the chapter of `D'Arby the Player` which is also quite splendid. This is in `JoJo's Bizarre Adventure: Stardust Crusaders, episode 60 & 61`. You can watch it in bilibili (need a VIP)

Hope you will enjoy this project. Have fun!