CECS 282 – Advanced C++ Programming - CSULB
Program #6 – Mega War / Inheritance
Due: May 4, 2021

War is a simple game played with 2 players and one deck of cards. The deck is shuffles and then each player receives half the deck – so 26 cards per player. Then each player lays down one card. The player who owns the highest card wins the other player's card and the winner puts both cards into his hand. In the event that both cards are the same value, then there is a war. Both players lay down 3 cards, and the fourth card is used as the battle card. The player with the highest battle card wins all 10 cards that are in play. If the battle cards produce another tie, then anther war happens.

The winner is the person who eventually obtains all the cards and the loser will have no cards left to play.

Your task is to build a card game called Mega War!

Mega War is similar to War, but you can play with more than 1 deck and more than 2 players. The winner will be the player who accumulates all the cards.

When Mega War starts, the program will ask for the number of decks, and the number of players. You could easily choose 1 deck and 2 players to simulate regular War – but what fun would that be… When you demo the program, you will play with 3 decks and 5 players. The game will continue until there is a single winner (who has all the cards).

The game play will be controlled with a loop. Each iteration will cause all the players to have a group battle. The player with the highest card wins all the other cards and adds them to his hand. If 2 or more players have the same highest card, then all players with the highest value card will have a war. The winner of the war will take all the cards. If a player cannot complete the War (because they ran out of cards) then that player loses and the other player acquires the war card pile. If both (or all) players cannot continue a war, then they all lose and the war card pile gets send to Lost and Found. The next time there is a war, the Lost and Found cards go to the winner of the war.

The value of each card is as follows: Ace = 1, Two = 2 and so on, Ten = 10, Jack = 11, Queen = 12, and King = 13.

There are a several things each player should keep track of.

1. The number of cards in their hand
2. The ***Fierceness*** of their hand. You can imagine that a hand that has many high-value cards is more likely to win than a hand that has many low-value cards. The Fierceness of a hand is the average value of all cards. A hand that has a King, Queen, 9 and 3 has a Fierceness value of 9.25. A hand that has 3, 8, 5 and Jack has a Fierceness value of 6.75. Based solely on that value, we might expect the first player to win the game.
3. The number of battles the user played. The winner will probably play the most battles. However, that might not be the case if the winner did not participate in many wars.
4. The number of battles the user won.

The game play will be in a loop until a winner is determined. After each battle, the stats for each player will be displayed like this:

Battle 66 Stats:

Player 1: Fierceness = 8.34	Cards = 16	Battles = 45	Won = 12
Player 2: Fierceness = 0	Cards = 0	Battles = 37	Won = 15
Player 3: Fierceness = 5.29	Cards = 38	Battles = 56	Won = 20
Player 4: Fierceness = 9.77	Cards = 102	Battles = 54	Won = 19

In the above stats, you can see that Player 2 has lost. Player 4 looks to be a likely winner eventually. Also the total number of cards (add all the player cards together) never changes. The game will stop when a winner has all the cards.

Now let's think about the classes needed for this game. Obviously, a Card. And obviously a Deck. Beyond that we will need a Player, a MegaDeck (in case we user more than one deck), a War pile (when there is War), a Lost & Found pile.

But wait – notice that a Deck, a Player, a War pile and Lost & Found are very similar. They are all Card containers. Here is an opportunity for inheritance. Create a Base Class called a CardPile that has a dynamic storage container for Cards (vector) and also an add( ) and remove( ) function. Also include any other functions or data members that are common to ALL Card piles. Then make a derived class for each of the classes needed.

Other Requirements:

- Your Card class must use enumerated types (enum) for the rank and suit.
- The Deck (or card pile) must use vector of cards instead of array of cards as you did in Solitaire Prime
- You need to create a UML diagram for your classes. Show each class UML and also the relationship classes have to each other

What to submit to the Drop box:

Program source code.
Screenshot of final result for a game that has 3 decks and 5 players
UML diagrams