Program 2

Re-submit Assignment

Due Feb 2 by 11:59pm **Points** 60 **Submitting** a file upload **File Types** tar
Available Jan 20 at 12am - Feb 7 at 11:59pm 19 days

Crazy Eights

For this assignment, you will write a program that allows one player to play a game of Crazy Eights against the computer. Crazy Eights is a game that uses a standard deck of 52 cards. Specifically, each card has a rank and a suit. There are 13 ranks: the numbers 2 through 10, Jack (usually represented with the letter J), Queen (Q), King (K), and Ace (A). There are 4 suits: clubs, diamonds, hearts, and spades. In a 52-card deck, there is one card of each rank for each suit.

A game of Crazy Eights between two players proceeds as follows:

- The deck of cards is shuffled, randomizing the order of the cards.
- Each player is dealt 7 cards.
- The remaining cards are placed face-down (i.e. with their rank and suit hidden) in a stack on the table and becomes the "stock", with the top card turned over and displayed in a separate "pile".
- One of the players (player A) begins the game by playing a card from their hand that is the same suit or same rank as the top card on the pile, which becomes the new top of the pile.
- Player B then plays a card from their hand that is the same suit or rank as the top card, which becomes the new top of the pile.
- If a player does not have any cards of the required rank or suit, they must draw from the top of the deck (stock) and add cards to their hand until they draw a card that can be played on the pile, or until the deck runs out of cards.
- The game continues with the players alternating turns until one of the players gets rid of all the cards in their hand, or until all of the cards are drawn and no one can play a card. If neither player has zero cards at the end of the game, the player with the least amount of cards in their hand wins.
- Eights of any suit are considered "wild" and may be played on any turn. When an eight is played, the player must specify a suit for the next player to match.

Needed classes

To write your Crazy Eights game, you should implement the following classes, including the specified members and methods. You may also add more members and methods, as needed.

```
class Card {
  private:
    int rank;  // Should be in the range 0-12.
    int suit;  // Should be in the range 0-3.
```

```
  public:
  // must have constructors, destructor, accessor methods, and mutator methods

};
```

In the Card class above, rank and suit are represented with int values, but you must also have some way to map those values to representations players will be familiar with (e.g. a string representation of the suit or rank).

```
class Deck {
  private:
    Card cards[52];
    int n_cards;  // Number of cards remaining in the deck.
  public:
  // must have constructors, destructor, accessor methods, and mutator methods

};
```

The Deck class is the source of all of the cards. Cards will initially start in a Deck object and then be transferred to players' hands. An important method that should be implemented for the Deck class is one to remove a card and return it so it can be placed in a player's hand (deal a card from the deck to the player).

```
class Hand {
  private:
    Card* cards;
    int n_cards;  // Number of cards in the hand.
  public:
  // must have constructors, destructor, accessor methods, and mutator methods
};
```

The Hand class will hold the cards in one player's hand. The number of cards a player holds may change, so the size of the array of Card objects in a hand may also need to change. Cards may be added to a player's hand and removed from a player's hand, so the Hand class will need functions to do both of those things. Other useful methods might check for a given suit or rank.

```
class Player {
  private:
    Hand hand;
    string name;
  public:
  // must have constructors, destructor, accessor methods, and mutator methods

};
```

The Player class represents a single player. Each Player will have a Hand object representing its hand and an array keeping track of the books the player has laid down. Depending on how you implement things, the Player class may need methods to add and remove cards from their hand or to check their hand for cards with a specific suit or rank. Another useful method the Player class might have is one to figure out what suit they want to ask for if they play an eight. Note that the Player class must represent both the human

player and the computer player. You should write your class methods accordingly and add any extra data members needed to accomplish this.

```
class Game {
  private:
    Deck cards;
    Player players[2];
  public:
  // must have constructors, destructor, accessor methods, and mutator methods
}
```

The Game class represents the state of an entire game. It contains objects representing the deck of cards and both players. It would be useful to have methods in the Game class that check whether the game is over and that execute a player's turn.

Program features

Your program should do the following:

- Set up a deck of 52 cards as described above.
- Shuffle the deck (i.e. randomize the order of cards).
- Deal the cards to the two players.
- Play a game of Crazy Eights as described above, with the following gameplay features:
  - Print the current state of the game (e.g. the cards held by the human player and the card of on the top of the pile) after each turn (keeping the computer player's cards hidden).
  - On the human player's turn, prompt the user for a card to play. When they enter a card, either put the card on top of the pile or have them draw a card from the deck if they can't play. If the card is an 8, prompt the user to declare a suit for the next player.
- Once the game is over, you should announce the winner and ask the user if they want to play again.
- You should not have any memory leaks in your program.
- Note: You cannot use the vector class for handling dynamic memory

Programming Style/Comments

In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying to follow them, i.e. don’t align everything on the left or put everything on one line!

```
/*************************************************
** Program: crazyeights.cpp
** Author: Your Name
** Date: 02/01/2020
** Description:
** Input:
** Output:
*************************************************/
```

When you compile your code, it is acceptable to use C++11 functionality in your program. In order to support this, change your Makefile to include the proper flag.
For example, consider the following approach (note the inclusion of **-std=c++11**):

```
g++ -std=c++11 <other flags and parameters>
```

In order to submit your homework assignment, you must create a tarred archive that contains your .h, .cpp, and Makefile files. This tar file will be submitted to Canvas. In order to create the tar file, use the following command:

```
tar –cvf assign2.tar <list of all the .h and .cpp files> Makefile
```

Crazy 8s

Criteria	Ratings	Pts
Game Initialization Student correctly initializes the game with 52 cards and 2 players		6.0 pts
Deck Initialization Student correctly shuffles and deals cards from deck to players		6.0 pts
Turns Student correctly alternates turns between players		6.0 pts
Playing Cards Student correctly transfers cards from player hands to pile on their turn		6.0 pts
Drawing Cards Student correctly transfers cards from deck to player hands until they can play a card		6.0 pts
File Separation & Makefile Student correctly separates code into appropriate interface, implementation, and driver files and has a working Makefile for compilation, and submits via a single tarball.		6.0 pts
Comments & Style Student appropriately comments code and uses a consistent and appropriate style. At a minimum, header should contain author's name and a description of the program. Is code easy for the TA to read? Conditional blocks of code should always be indented. Every function contains it's own initial block comment (or multiple lines of comments prior to the function definition) that provides the reader with an explanation of the function's purpose. -1 pt for each function that is missing a header (up to 3 point penalty).		6.0 pts
Winning Student correctly determines when the game is over and who the winner is		6.0 pts
Required Classes Student correctly implements each of the required classes (Card, Deck, Hand, Player, Game)		5.0 pts
Memory Cleanup The Valgrind LEAK SUMMARY should report: "definitely lost: 0 bytes in 0 blocks". Each "new" operation should have a corresponding delete operation.		2.0 pts
Program Functionality Program does not crash during testing (e.g. segmentation fault or similar abrupt halts)		2.0 pts
Function Modularity No functions longer than approximately 20 lines of code.		3.0 pts
		Total Points: 60.0