Crazy 8

A JCM production

The aim of this project is to set up an entertaining game of crazy eight between two user-based players. The version we made here involves two players each dealt eight cards with a draw pile and a discard pile. The objective of this game, like the famous game Uno, is to get rid of all your cards.

Purpose Statement: The purpose of the project is to create an interactive and entertaining implementation of the card game Crazy Eights using a virtual platform to play the game seamlessly. This digital version of the game was created to offer players an engaging and satisfying way of coming up with play strategies without the use of a physical deck of cards.

**Setup:**

1. Use a standard 52-card deck.
2. Shuffle the deck and deal 8 cards to each player.
3. Place the remaining deck face-down in the center, forming the draw pile. Turn the top card face-up next to the draw pile to start the discard pile.

**Gameplay:**

1. Players take turns in a clockwise direction.
2. On your turn, you must play a card that matches the rank or suit of the top card on the discard pile. For example, if the top card is a 7 of hearts, you can play any 7 or any heart.
3. If you cannot play a card, you must draw from the draw pile and offer a turn to your opponent.
4. The "crazy" part of Crazy Eights comes from the special rules associated with the 8 cards. Eights are wild and can be played at any time. When you play an 8, you must declare the suit that the next player must play.
5. The game continues until a player has no cards left, signaling the end of the round.

Assumptions:

When Running

* During gameplay, when prompted to enter any key to start the game, the user does not enter more than one character to start the game (otherwise an infinite display error occurs)
* During gameplay, when prompted to enter any key to start the game, the user does not enter a period to start the game ‘ . ‘
* During gameplay, if the first player draws a card instead of playing and then player 2 plays the special card (2 of any suit), the special card 2 will not function as normal for the rest of the game (Tutu error highlighted in presentation)
* During gameplay, the numbers entered for each card they play must be one of the numbers clearly displayed in their available hand.

When Designing the Program

* Users will not be able to hide their hands from each other due to the nature c++ terminal display.

All Functions and Algorithms:

**1. Class-Based Design:**

* The use of the **Deck** class provides a structured and organized way to encapsulate behaviors related to a deck of cards. The class handles deck initialization, shuffling, card population, dealing, and card manipulation in terms of adding and removing cards. This allowed us to give parameters to the deck from both the player as well as the deck itself, this way we can handle both structures at the same time. This modular approach enhances readability and makes it easier to understand and maintain the code.

**2. Modular Functions:**

* The code employs various functions that focus on specific tasks, such as drawing cards, playing cards, and simulating a turn. Each of the functions were separated and used in tandem with one another to improve the overall flow of the function. This modularity adheres to the principle of code separation, making each function responsible for a specific aspect of the game logic.
* The use of modular functions was purposefully chosen for our program mainly because of the initial design and its flaws that we encountered. Originally, all the functions were under the class as behaviors, although it seemed it would work perfectly as the class was meant for cards and the functions provided gameplay, the functions created more separation within the programming. The realization that the class was meant for creating a unique data type rather than a series of actions, guided the program to functionalize all the actions needed in the card game. Prior to the realization, the terminal was constantly shifting in its output, while also struggling to maintain basic inputs from the users.

3. Shuffling behaviour

* Our initial view to shuffling behavior was an iterator function that went on from 1 to 52 using the random function in the <ctime> library. We then changed our approach by applying a randomizer that allowed us to shuffle the deck of cards through the indices of the array deck. Although the initial randomizer or iterator for the shuffle behavior was more effective in eliminating unique placeholder values for each value of the card, we went ahead with the second option because it was the approachable method to use for a CS1 student using C for the first time, it was also more intuitive this way.

The initial alternative to shuffle is here below with the iterator class:  
// //////////// DO NOT TOUCH, SHUFFLES DECK ONLY

    // srand(time(NULL));

    // int array1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52};

    // random\_shuffle(array1, array1 + 52); // Shuffle array1

    // cout << "After random shuffle, array1: ";

    // ostream\_iterator<int> output(cout, " ");

    // copy(array1, array1 + 52, output);

    // /////////////////////////////////////////////

4. Switching players

* We initially wanted to switch between players by creating a variable named curr\_count would have been able to increment itself as the game continued or as gameplay persisted. Instead of allowing her count to increment by one each time and growing to some indefinite number we decided to let the Kirkland switch between one and two this way we could make sure to establish which player was currently in play as well as write code around Kirk count that would allow us to determine whether a special card was permitted to be playable or not at the current time in the game.

5. What the code does not do

* Our program has succeeded in the sense that it can run through a whole game of Crazy 8 and returns a winner of the game. It does a really good job of alternating between players, and it keeps track of all the different amounts of cards in each deck. Due to the way the class behaviors are crafted we do a really good job of avoiding duplicating or altering cards. I would say that deck/hand manipulation here is probably the strongest portion of our code. What our code doesn’t do well is work around the special cards. The draw 2 concept caused a lot of bugs that were a little above our knowledge of fixing in the time we had left. The 8 wild cards didn’t end up letting the user choose what suit they want, and while the skip works just fine all three of these are also a little hard to recognize when exactly they occur in the terminal. Just the way our program is formatted makes it easy to miss when the previous user played a special card and you can get lost in the lines of text that it outputs and easily miss it.