How to run the code:
Two methods to running the code:

1. In the terminal, type on the command line:
   ```
   git clone https://github.com/nfell/blackjack.git
   ```
   Open the blackjack folder by typing:
   ```
   cd blackjack
   ```
   Run the executable by typing:
   ```
   ./Kleinerperkins
   ```

2. Visit the url submitted and hit "clone or download." Then hit "download zip," and open the downloaded blackjack folder and open the executable file named Kleinerperkins

Rules:

The coded game is blackjack, and it follows the common rules of the game. Some important rules to note are:

1. When the cards are first dealt, the dealer's first card is dealt face down, and the dealer's second card is dealt face up. Both the player's cards are dealt face up.
2. If the dealer's face up card is either a face card, 10, or an ace, the dealer checks their face down card for a blackjack (value of 21). If a player is dealt a blackjack, they automatically win unless the dealer has a blackjack as well.
3. Aces can count as a value of either 11 or 1.

Design choices:

I chose to program blackjack because I wanted to be able to program the game while keeping as much as I could within the recommend time to spend. I had originally thought about building it to be a multiplayer game as well but felt that coding that would push me too far over the recommended time spent.

I used classes to create cards and the deck. The deck class holds a vector of pointers to card objects to represent the card deck. This class also holds two more vectors of pointers to cards that represent the dealer's hand and the player's hand, as well as functions to shuffle the deck vector, deal out cards to the player and dealer vectors, and functions that check hand's values. I used dynamically allocated space for each card object in the array, so that each vector's size was variable when cards are dealt from the deck to a player's hand. I chose to use vectors as the containers so that I could easily pop off the top card in the deck and delete that card from the deck once it was added to a player's hand.

I created a dealer algorithm which decides, based on the dealer's total card value, whether to hit or stand as well as whether an ace should be counted as 1 or 11 points. Based on how many casinos play the game, I wrote the algorithm so that the dealer will hit if they have less than 17 points, or if they have 17 points with an ace counting as an 11. This means if the dealer hits and goes over 21, the algorithm will try counting the ace as 1 instead of 11 in order to stay under 21 points and not bust.

Choice of tooling:

I chose to use C++ because it is the language I am most familiar with and the language that I have learned the most advanced syntax/data structures in. I used object-oriented programming so that I could make each card and object and the deck itself an object, which made it easier to shuffle the deck and deal cards. I used only the C++ library.

I also thought about using MATLAB instead to write the code because I wanted to incorporate graphical user interfaces to make the game more user friendly; however, I have less experience with MATLAB and felt I could make a better program in C++ than in MATLAB in the time given.