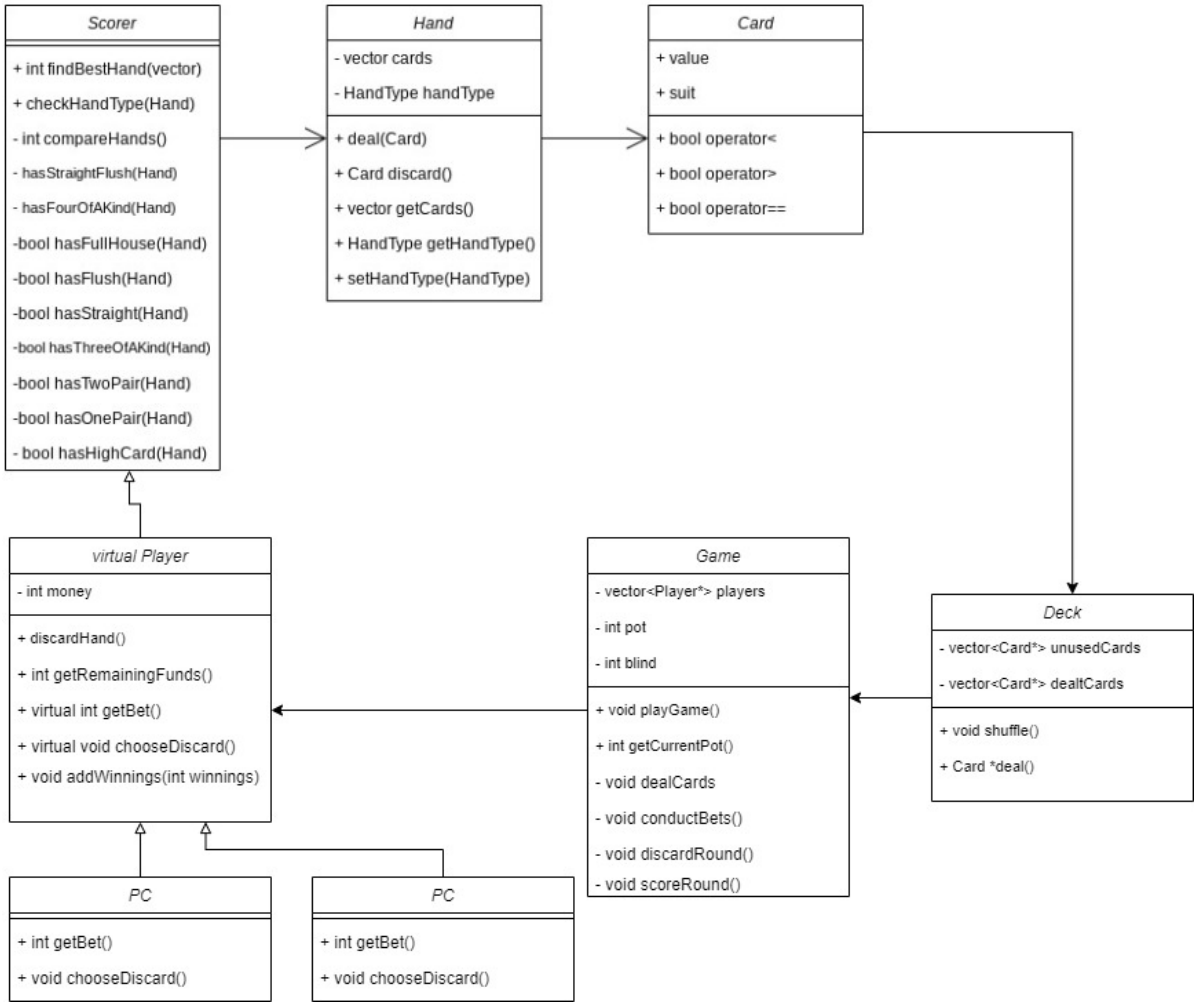
Nathan Roe
OOP with C++
EN.605.604 Section 81
5/1/2022

Module 12 Assignment: Poker Game

For this assignment, I will be generating a Poker Game following the rules for 5-Card Draw Poker.

The order for this game:

1. Deal Cards
2. Accept bets
    a. First bettor will have to meet blind
    b. Second bettor will have to meet big blind
    c. After this, all players will have to meet the last raise, or fold
3. Allow players to discard
4. Accept new bets
5. Score Cards
6. Distribute winnings
7. Clean up, start again if more than 1 player has funds



```cpp
/// Run a 5-Card Draw Poker Game
#include <iostream>
#include "deck.h"
#include "game.h"

/// Run main game loop, with compile-time parameters
int main()
{
    // Set number of Human/Computer Players
    const int NUM_COMPUTERS = 2;
    const int NUM_PLAYERS = 1;
    // Set starting money amount
    const int BUY_IN = 100;
    // Set minimum bet values
    const int BLIND = 3;

    // Run Poker Game
    Game *game = new Game(NUM_PLAYERS, NUM_COMPUTERS, BUY_IN, BLIND);
    game->playGame();
} // End function main

```

```cpp
/// @file
/// @author Nathan Roe
#pragma once
#include <stdexcept>
#include <string>
#include <vector>

class Player;
class Deck;
class Scorer;

using namespace std;

/// Game object for playing a game of 5-Card Draw Poker
///
/// Main game loop is played using playGame(), and there
/// can be a total of 7 players, either PCs or NPCs
class Game
{
public:
    /// Game constructor
    /// @param numPlayers - number of Human players (PCs)
    /// @param numComputers - number of Computer players (NPCs)
    /// @param buyIn - starting money amount for each player
    /// @param blind - Min starting bet for each round
    Game(int numPlayers, int numComputers, int buyIn, int blind);

    /// Error thrown when too many or too few players required
    class InvalidGameException : public std::runtime_error
    {
    public:
        InvalidGameException(int playerCount)
            : std::runtime_error("NumPlayers must be between 2 and 7, "
                                 + to_string(playerCount)
                                 + " were given") {}
    };

    /// Start main game loop (Bet, Draw, Bet, Score)
    ///
    /// Plays a game of Poker on loop until only 1
    /// player remains with money
    void playGame();

    /// Getter for current amount of money in the pot
    /// Returns the amount of money in the pot
    int getCurrentPot() { return this->pot; };

private:
    const int MIN_PLAYERS = 2;
```

```
    const int MAX_PLAYERS = 7;

    enum class BetType
    {
        NONE,
        FOLD,
        CALL,
        RAISE,
        ALL_IN
    };

    vector<Player*> players;
    vector<Player*> remainingPlayers;
    vector<BetType> betTypes;
    vector<int> amountIn;

    Deck *deck;
    Scorer *scorer;

    int pot = 0;
    int blind = 0;
    int blindIdx = 0;

    /// Deal 5 cards to each player
    void dealCards();

    /// Run a round of bets
    ///
    /// Runs betting until all players have called, folded,
    /// or are all in
    /// @param requireBlinds - indicates whether minimum
    ///    bet values should be used in the round (True
    ///    during the first round
    void conductBets(bool requireBlinds);

    /// Remove players from the round who have folded
    void dropPlayers();

    /// Handle round for redraws
    void discardRound();

    /// Score players who are still in, and distribute winnings
    void scoreRound();
};


```

```cpp
#include "game.h"
#include "deck.h"
#include "player.h"
#include "pc.h"
#include "npc.h"
#include "Scorer.h"
#include <iostream>

/// Game object for playing a game of 5-Card Draw Poker
Game::Game(int numHumans, int numComputers, int buyIn, int blind)
{
    int numPlayers = numHumans + numComputers;
    if (numPlayers < MIN_PLAYERS ||
        numPlayers > MAX_PLAYERS)
    {
        throw InvalidGameException(numPlayers);
    }

    // Add PCs
    for (int num = 0; num < numHumans; ++num)
    {
        Player *player = new PC(this, buyIn);
        this->players.push_back(player);
    }
    // Add NPCs
    for (int num = 0; num < numComputers; ++num)
    {
        Player *player = new NPC(this, buyIn);
        this->players.push_back(player);
    }

    this->blind = blind;
    this->scorer = new Scorer();
    this->deck = new Deck();
} // End Game constructor

/// Start main game loop (Bet, Draw, Bet, Score)
void Game::playGame()
{
    while (players.size() > 1)
    {
        // Setup Current Players
        vector<Player*>::iterator iter = this->players.begin();
        while (iter != this->players.end())
        {
            this->remainingPlayers.push_back(*iter++);
        }
        vector<BetType> startBet(this->remainingPlayers.size(),
          BetType::NONE);
```

```cpp
        this->betTypes = startBet;
        vector<int> startCash(this->remainingPlayers.size(), 0);
        this->amountIn = startCash;

        // Deal out initial cards
        dealCards();

        // Accept first round of bets
        conductBets(true);
        dropPlayers();

        // Let players redraw
        discardRound();

        // Betting round, no blinds
        vector<BetType> newBets(this->remainingPlayers.size(), BetType::NONE);
        this->betTypes = newBets;
        conductBets(false);
        dropPlayers();

        // Score round
        scoreRound();

        // Discard hands at end of round
        for (Player *player : this->players)
        {
            player->discardHand();
        }
        this->remainingPlayers.clear();

        // Remove Players who lost
        iter = this->players.begin();
        while (iter != this->players.end())
        {
            Player *p = *iter;
            if (p->isBroke())
            {
                cout << "Player " << p << " is out of funds." << endl;
                iter = this->players.erase(iter);
            }
            else
            {
                ++iter;
            }
        }

        // Increment Blind position player
        ++this->blindIdx;
```

```
        this->blindIdx %= this->players.size();
    }
} // End function playGame

// Remove players from the round who have folded
void Game::dropPlayers()
{
    vector<Player*>::iterator iter = this->remainingPlayers.begin();
    vector<BetType>::iterator betIter = this->betTypes.begin();
    vector<int>::iterator cashIter = this->amountIn.begin();
    while (iter != this->remainingPlayers.end())
    {
        BetType b = *betIter;
        // Remove Players who folded
        if (b == BetType::FOLD)
        {
            iter = this->remainingPlayers.erase(iter);
            betIter = this->betTypes.erase(betIter);
            cashIter = this->amountIn.erase(cashIter);
        }
        else
        {
            ++iter;
            ++betIter;
            ++cashIter;
        }
    }
} // End function dropPlayers

// Deal 5 cards to each player
void Game::dealCards()
{
    this->deck->shuffle();
    const int NUM_CARDS = 5;
    for (int n = 0; n < NUM_CARDS; n++)
    {
        for (Player *player : this->players)
        {
            player->deal(this->deck->deal());
        }
    }
} // End function dealCards

// Run a round of bets
void Game::conductBets(bool requireBlinds)
{
    // Exit if too few players
    if (this->remainingPlayers.size() < 2)
    {
```

```
        return;
    }

    int playerIdx = this->blindIdx;
    playerIdx %= this->remainingPlayers.size();
    int bet = 0;
    int baseBet = this->amountIn[playerIdx];
    int amountToCall = 0;
    Player *p;

    if (requireBlinds)
    {
        // Handle Small Blind
        p = this->remainingPlayers[playerIdx];
        cout << "Player " << p << " is the Small Blind" << endl;
        bet = p->getBet(0, this->blind);
        cout << "Player " << p << " bet $" << bet << endl << endl;
        pot += bet;
        this->amountIn[playerIdx] += bet;
        // Handle scenario when player can't afford small blind
        if (bet < this->blind)
        {
            this->betTypes[playerIdx] = BetType::ALL_IN;
        }
        else
        {
            this->betTypes[playerIdx] = BetType::RAISE;
        }
        ++playerIdx;
        playerIdx %= this->remainingPlayers.size();
        baseBet = bet;

        // Handle Big Blind
        p = this->remainingPlayers[playerIdx];
        cout << "Player " << p << " is the Big Blind" << endl;
        amountToCall = baseBet - this->amountIn[playerIdx];
        bet = p->getBet(amountToCall, 2 * this->blind);
        cout << "Player " << p << " bet $" << bet << endl << endl;
        pot += bet;
        if (bet >= amountToCall)
        {
            baseBet += (bet - amountToCall);
        }
        this->amountIn[playerIdx] += bet;
        // Handle scenario when player can't afford big blind
        if (bet < 2 * this->blind)
        {
            this->betTypes[playerIdx] = BetType::ALL_IN;
        }
```

```cpp
        else
        {
            this->betTypes[playerIdx] = BetType::RAISE;
        }
        ++playerIdx;
        playerIdx %= this->remainingPlayers.size();
    }

    // Main betting loop
    while (true)
    {
        // Check whether player needs to bet
        if (!(this->betTypes[playerIdx] == BetType::FOLD ||
              this->betTypes[playerIdx] == BetType::ALL_IN))
        {
            p = this->remainingPlayers[playerIdx];
            // Get required bet amount to avoid folding
            amountToCall = baseBet - this->amountIn[playerIdx];
            bet = p->getBet(amountToCall);
            pot += bet;
            // See if bet is a raise or a call
            if (bet >= amountToCall)
            {
                baseBet += (bet - amountToCall);
            }
            this->amountIn[playerIdx] += bet;
            // Handle Calls
            if (bet == amountToCall)
            {
                this->betTypes[playerIdx] = BetType::CALL;
                cout << "Player " << p << " Calls" << endl << endl;
            }
            // Handle All In
            else if (p->getRemainingFunds() == 0)
            {
                this->betTypes[playerIdx] = BetType::ALL_IN;
                cout << "Player " << p << " is All In" << endl << endl;
            }
            // Handle Folds
            else if (bet == 0)
            {
                this->betTypes[playerIdx] = BetType::FOLD;
                cout << "Player " << p << " Folds" << endl << endl;
            }
            // Other bets are raises
            else
            {
                this->betTypes[playerIdx] = BetType::RAISE;
                cout << "Player " << p << " Raises $" << bet -
```

```
                    amountToCall << endl << endl;
            }
        }

        // Check to see if bets are over
        int numRaises = 0;
        for (BetType betType : this->betTypes)
        {
            if (betType == BetType::RAISE || betType == BetType::NONE)
            {
                ++numRaises;
            }
        }
        if (numRaises == 0)
        {
            break;
        }

        ++playerIdx;
        playerIdx %= this->remainingPlayers.size();
    }
} // End function conductBets

// Handle round for redraws
void Game::discardRound()
{
    int discardingPlayers = this->remainingPlayers.size();
    // Find cards availible to redraw
    int remainingCards = this->deck->cardsRemaining();
    int playerIdx = 0;
    int discard = 0;
    Player *p;
    // Let each player discard
    for (int idx = 0; idx < discardingPlayers; ++idx)
    {
        playerIdx = (this->blindIdx + idx) % discardingPlayers;
        p = this->remainingPlayers.at(playerIdx);
        discard = p->chooseDiscard(remainingCards);
        cout << "Player " << p << " discarded " << discard << " cards" << endl << endl;
        // Deal new card for each discard
        for (int n = 0; n < discard; n++)
        {
            p->deal(this->deck->deal());
        }
    }
} // End function discardRound

// Score players who are still in, and distribute winnings
```

```cpp
void Game::scoreRound()
{
    int winIdx = -1;
    vector<Hand*> temp(this->remainingPlayers.begin(), this->remainingPlayers.end());
    // Print player hands
    for (Player *p : this->remainingPlayers)
    {
        cout << "Player " << p << "'s hand: " << p->printCards() << endl;
    }

    // Find number of winning hands, and bring winners to front
    if (this->remainingPlayers.size() > 1)
    {
        winIdx = this->scorer->findBestHand(&temp);
    }

    // If only one player is left, give pot to player
    if (this->remainingPlayers.size() == 1)
    {
        int idx = 0;
        Player* p = static_cast<Player*>(temp[idx]);
        p->addWinnings(this->pot);
        cout << "Player " << p << " Wins $" << this->pot << endl << endl;
        this->pot = 0;
    }
    // If multiple players left, distribute winnings amongst winners
    else if (winIdx >= 0)
    {
        int winnings = this->pot / (winIdx + 1);
        for (int idx = 0; idx <= winIdx; ++idx)
        {
            Player* p = static_cast<Player*>(temp[idx]);
            p->addWinnings(winnings);
            this->pot -= winnings;
            cout << "Player " << p << " Wins $" << winnings << endl << endl;
        }
    }
} // End function scoreRound
```

```
/// @file
/// @author Nathan Roe
#pragma once
#include <vector>
#include <exception>
#include "card.h"

using namespace std;

// Deck object for responsible for tracking all cards
//
// Allows for shuffling and dealing of cards. Dealing
// from an empty deck throws and error.
class Deck
{
public:
    /// Deck Constructor
    Deck();

    /// Error thrown when dealing from empty deck
    class EmptyDeck : public std::exception
    {
    public:
        virtual const char* what() const throw()
        {
            return "Cannot deal, deck is empty";
        }
    };

    /// Shuffle deck; returns previously dealt cards to deck
    void shuffle();

    /// Deal a card
    ///
    /// Returns a pointer to the dealt card
    Card *deal();

    /// Getter for cards left in deck
    /// Return the number of cards remaining in the deck
    int cardsRemaining() { return this->unusedCards.size(); };

private:
    vector<Card> unusedCards;
    vector<Card> dealtCards;
};

```

```cpp
#include "deck.h"
#include "card.h"
#include <array>
#include <list>
#include <algorithm>
#include <stdlib.h>

using namespace std;

// Deck object for responsible for tracking all cards
Deck::Deck()
{
    const int RAND_SEED = 7;
    srand(RAND_SEED);

    array<Card::Value, 13> values = { Card::Value::Two, Card::Value::Three, Card::Value::Four,
                                      Card::Value::Five, Card::Value::Six, Card::Value::Seven,
                                      Card::Value::Eight, Card::Value::Nine, Card::Value::Ten,
                                      Card::Value::Jack, Card::Value::Queen, Card::Value::King,
                                      Card::Value::Ace };
    array<Card::Suit, 4> suits = { Card::Suit::Clubs, Card::Suit::Diamonds,
                                   Card::Suit::Hearts, Card::Suit::Spades };

    // Create one card of each value/suit combination
    for (Card::Suit suit : suits)
    {
        for (Card::Value value : values)
        {
            Card *card = new Card(value, suit);
            this->unusedCards.push_back(*card);
        }
    }
} // End Deck constructor

// Shuffle deck; returns previously dealt cards to deck
void Deck::shuffle()
{
    // Return dealt cards to deck
    while (!this->dealtCards.empty())
    {
        this->unusedCards.push_back(this->dealtCards.back());
        this->dealtCards.pop_back();
    }
    sort(this->unusedCards.begin(), this->unusedCards.end());
```

```cpp

    // Shuffle using Fisher-Yates (https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle)
    for (unsigned int idx = 0; idx < this->unusedCards.size(); ++idx)
    {
        int swapIdx = rand() % this->unusedCards.size();
        swap(this->unusedCards[idx], this->unusedCards[swapIdx]);
    }
} // End function shuffle

// Deal a card
Card *Deck::deal()
{
    if (!this->unusedCards.empty())
    {
        // Shift card from unused to dealt
        this->dealtCards.push_back(this->unusedCards.back());
        this->unusedCards.pop_back();
        // Return pointer to dealt card
        return &this->dealtCards.back();
    }
    // Throw error if deck is empty
    else
    {
        throw EmptyDeck();
    }
} // End function deal

```

```
/// @file
/// @author Nathan Roe
#pragma once
#include "hand.h"

class Game;

/// Abstract Player class representing Poker player
///
/// Handles money, betting and discards for 5-Card
/// draw poker. Card information inherited from Hand
class Player : public Hand
{
public:
    /// Player constructor
    /// @param *game - Pointer to poker Game object
    /// @param buIn - Starting money value
    Player(Game *game, int buyIn);

    /// Player destructor
    virtual ~Player();

    /// Remove all cards from hand
    void discardHand();

    /// Check whether player has funds
    /// Returns True if player has no money, false otherwise
    bool isBroke() { return this->money <= 0; };

    /// Getter for amount of money remaining
    /// Returns an int of remaining money
    int getRemainingFunds() { return this->money; };

    /// Get a bet for a betting round in Poker
    /// @param prevRaise - Last raise value, min for calling
    /// @param minBet - Minimum allowed bet, used for blinds
    /// Returns an int of the Player's bet
    virtual int getBet(int prevRaise, int minBet = 0) = 0;

    /// Adds Winnings to the Player's pool of funds
    /// @param winnings - amount of money won in round
    void addWinnings(int winnings) { this->money += winnings; };

    /// Choose which cards to discard for draw phase of 5-Card draw
    /// @param maxDiscard - Maximum cards that are availible to draw
    /// Returns the number of cards discarded
    virtual int chooseDiscard(int maxDiscard) = 0;

protected:
```

```
    Game *game;
    int money = 0;
};


```

```cpp
#include "player.h"
#include "card.h"

/// Abstract Player class representing Poker player
Player::Player(Game *game, int buyIn)
{
    this->game = game;
    this->money = buyIn;
} // End Player constructor

/// Default Player destructor
Player::~Player()
{
    return;
} // End Player destructor

/// Discard all cards in hand
void Player::discardHand()
{
    while (!this->cards.empty())
    {
        const Card card = this->cards.front();
        discard(&card);
    }
} // End function discardHand

```

```cpp
/// @file
/// @author Nathan Roe
#pragma once
#include "player.h"

class Game;

/// Player class representing Human Poker player
class PC : public Player
{
public:
    /// Human Player constructor
    /// @param *game - Pointer to poker Game object
    /// @param buIn - Starting money value
    PC(Game *game, int buyIn) : Player(game, buyIn) {}

    /// Get a bet for a betting round in Poker
    /// @param prevRaise - Last raise value, min for calling
    /// @param minBet - Minimum allowed bet, used for blinds
    /// Returns an int of the Player's bet
    int getBet(int prevRaise, int minBet = 0);

    /// Choose which cards to discard for draw phase of 5-Card draw
    /// @param maxDiscard - Maximum cards that are availible to draw
    /// Returns the number of cards discarded
    int chooseDiscard(int maxDiscard);
};

```

```cpp
#include "pc.h"
#include <iostream>

using namespace std;

// Get a bet for a betting round in Poker
int PC::getBet(int prevRaise, int minBet)
{
    int bet = 0;
    cout << "Player " << this << "'s Turn" << endl;
    cout << "Your Cards: " << this->printCards() << endl;
    while (true)
    {
        if (minBet > 0)
        {
            cout << "Minimum Bet is $" << minBet << endl;
        }
        cout << "The previous raise was $" << prevRaise << endl;
        cout << "You have $" << this->money << endl;
        cout << "Enter your bet, or $0 to fold: ";
        cin >> bet;

        // Verify bet is a valid bet when a required bet is needed
        if (bet >= minBet && bet >= prevRaise && bet <= this->money)
        {
            break;
        }
        // Accept All In bets
        else if (bet == this->money)
        {
            break;
        }
        // Accept fold/calls when no min bet is needed
        else if (bet == 0 && minBet == 0)
        {
            break;
        }
        // Catch error for betting unavailible money
        else if (bet > this->money)
        {
            cout << "Please bet an amount of $" << this->money << " or less" << endl;
        }
        // Catch error for betting under required value
        else if (bet < minBet)
        {
            cout << "Please bet at leat $" << minBet <<", or go All In" << endl;
        }
```

```cpp
        // Catch error for not Calling or Folding
        else
        {
            cout << "Bet at least $" << prevRaise << " to Call/Raise" << endl;
        }
    }

    this->money -= bet;
    return bet;
} // End function getBet

// Choose which cards to discard for draw phase of 5-Card draw
int PC::chooseDiscard(int maxDiscard)
{
    // Find max possible discards
    if (maxDiscard > this->cards.size())
    {
        maxDiscard = this->cards.size();
    }
    cout << "Player " << this << "'s Turn to Discard" << endl;
    cout << "You may discard up to " << maxDiscard << " cards." << endl;

    int cardSelection = 0;
    int numDiscarded = 0;
    // Discard up to max cards, or until player exits loop
    while (numDiscarded < maxDiscard)
    {
        cout << this->printCards() << endl;
        cout << "Choose a card to discard (1-" << this->cards.size() << "), or 0 to Stop: ";
        cin >> cardSelection;
        // Exit Loop condition
        if (cardSelection == 0)
        {
            break;
        }
        // Player selects valid card for discard
        if (cardSelection > 0 && cardSelection <= this->cards.size())
        {
            // Discard indexed from 1-N, not by 0
            this->cards.erase(this->cards.begin() + cardSelection - 1);
            ++numDiscarded;
        }
    }
    return numDiscarded;
} // End function chooseDiscard

```

```cpp
/// @file
/// @author Nathan Roe
#pragma once
#include "player.h"

class Game;

/// Player class representing Computer Poker player
class NPC : public Player
{
public:
    /// Computer Player constructor
    /// @param *game - Pointer to poker Game object
    /// @param buIn - Starting money value
    NPC(Game *game, int buyIn) : Player(game, buyIn) {};

    /// Get a bet for a betting round in Poker
    /// @param prevRaise - Last raise value, min for calling
    /// @param minBet - Minimum allowed bet, used for blinds
    /// Returns an int of the Player's bet
    int getBet(int prevRaise, int minBet = 0);

    /// Choose which cards to discard for draw phase of 5-Card draw
    /// @param maxDiscard - Maximum cards that are availible to draw
    /// Returns the number of cards discarded
    int chooseDiscard(int maxDiscard);
};

```

```cpp
#include "npc.h"
#include <algorithm>

// Get a bet for a betting round in Poker
int NPC::getBet(int prevRaise, int minBet)
{
    // If required bet, bet minimum amount
    if (minBet > 0)
    {
        // Bet required value
        if (this->money > minBet)
        {
            this->money -= minBet;
            return minBet;
        }
        // Bet remaining money if less than required
        else
        {
            int prevFunds = this->money;
            this->money = 0;
            return prevFunds;
        }
    }

    // Call if funds are availible
    if (prevRaise < this->money)
    {
        this->money -= prevRaise;
        return prevRaise;
    }
    // Otherwise, fold
    else
    {
        return 0;
    }
} // End function getBet

// Choose which cards to discard for draw phase of 5-Card draw
int NPC::chooseDiscard(int maxDiscard)
{
    if (maxDiscard > this->cards.size())
    {
        maxDiscard = this->cards.size();
    }
    int numDiscard = rand() % maxDiscard;
    // Sort cards High to Low
    sort(this->cards.begin(), this->cards.end());
    // Randomly select some number of cards to discard
    for (int count = 0; count < numDiscard; ++count)
```

```
    {
        // Discard lowest n cards
        this->cards.erase(this->cards.end() - 1);
    }
    return numDiscard;
} // End function chooseDiscard

```

```
Player 000001476AC3F8E0 is the Small Blind
Player 000001476AC3F8E0's Turn
Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of Diamonds
Minimum Bet is $3
The previous raise was $0
You have $100
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 bet $3

Player 000001476AC39E40 is the Big Blind
Player 000001476AC39E40 bet $6

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of Diamonds
The previous raise was $3
You have $97
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 Raises $3

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of Diamonds
The previous raise was $0
You have $91
Enter your bet, or $0 to fold: 5
Player 000001476AC3F8E0 Raises $5

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of Diamonds
The previous raise was $0
You have $86
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
```

```
8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of Diamonds
Choose a card to discard (1-5), or 0 to Stop: 5
8 of Hearts, 7 of Spades, J of Clubs, K of Hearts
Choose a card to discard (1-4), or 0 to Stop: 1
7 of Spades, J of Clubs, K of Hearts
Choose a card to discard (1-3), or 0 to Stop: 1
J of Clubs, K of Hearts
Choose a card to discard (1-2), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 3 cards

Player 000001476AC39E40 discarded 0 cards

Player 000001476AC37E50 discarded 4 cards

Player 000001476AC3F8E0's Turn
Your Cards: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of Diamonds
The previous raise was $0
You have $86
Enter your bet, or $0 to fold: 10
Player 000001476AC3F8E0 Raises $10

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of Diamonds
The previous raise was $0
You have $76
Enter your bet, or $0 to fold: 5
Player 000001476AC3F8E0 Raises $5

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of Diamonds
The previous raise was $0
You have $71
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of Diamonds
Player 000001476AC39E40's hand: 4 of Clubs, 5 of Spades, 7 of Diamonds, Q
```

```
of Spades, Q of Hearts
Player 000001476AC37E50's hand: 2 of Spades, J of Hearts, 9 of Spades, 7 of Clubs, 2 of Diamonds
Player 000001476AC3F8E0 Wins $87

Player 000001476AC39E40 is the Small Blind
Player 000001476AC39E40 bet $3

Player 000001476AC37E50 is the Big Blind
Player 000001476AC37E50 bet $6

Player 000001476AC3F8E0's Turn
Your Cards: J of Diamonds, 8 of Hearts, J of Clubs, 9 of Hearts, 10 of Clubs
The previous raise was $6
You have $158
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC39E40 discarded 4 cards

Player 000001476AC37E50 discarded 3 cards

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
J of Diamonds, 8 of Hearts, J of Clubs, 9 of Hearts, 10 of Clubs
Choose a card to discard (1-5), or 0 to Stop: 2
J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs
Choose a card to discard (1-4), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 1 cards

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs, 6 of Diamonds
The previous raise was $0
You have $152
Enter your bet, or $0 to fold: 5
Player 000001476AC3F8E0 Raises $5

Player 000001476AC39E40 Calls

```

```
Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs, 6 of Diamonds
The previous raise was $0
You have $147
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs, 6 of Diamonds
Player 000001476AC39E40's hand: 4 of Hearts, A of Spades, K of Diamonds, 4 of Diamonds, 3 of Clubs
Player 000001476AC37E50's hand: 7 of Hearts, 8 of Diamonds, K of Clubs, A of Clubs, J of Spades
Player 000001476AC3F8E0 Wins $33

Player 000001476AC37E50 is the Small Blind
Player 000001476AC37E50 bet $3

Player 000001476AC3F8E0 is the Big Blind
Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of Spades
Minimum Bet is $6
The previous raise was $3
You have $180
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 bet $6

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of Spades
The previous raise was $0
You have $174
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 Raises $3

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of
```

```
Spades
The previous raise was $0
You have $171
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC37E50 discarded 1 cards

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of Spades
Choose a card to discard (1-5), or 0 to Stop: 2
Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades
Choose a card to discard (1-4), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 1 cards

Player 000001476AC39E40 discarded 1 cards

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades
The previous raise was $0
You have $171
Enter your bet, or $0 to fold: 10
Player 000001476AC3F8E0 Raises $10

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades
The previous raise was $0
You have $161
Enter your bet, or $0 to fold: 10
Player 000001476AC3F8E0 Raises $10

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades
The previous raise was $0
You have $151
```

```
Enter your bet, or $0 to fold: 10
Player 000001476AC3F8E0 Raises $10

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades
The previous raise was $0
You have $141
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades
Player 000001476AC39E40's hand: 2 of Clubs, 4 of Diamonds, 6 of Spades, J of Clubs, A of Hearts
Player 000001476AC37E50's hand: 2 of Hearts, 4 of Spades, 7 of Hearts, J of Hearts, K of Hearts
Player 000001476AC3F8E0 Wins $117

Player 000001476AC3F8E0 is the Small Blind
Player 000001476AC3F8E0's Turn
Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
Minimum Bet is $3
The previous raise was $0
You have $258
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 bet $3

Player 000001476AC39E40 is the Big Blind
Player 000001476AC39E40 bet $6

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
The previous raise was $3
You have $255
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 Calls

Player 000001476AC3F8E0's Turn to Discard
```

```
You may discard up to 5 cards.
9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

Player 000001476AC39E40 discarded 3 cards

Player 000001476AC37E50 discarded 4 cards

Player 000001476AC3F8E0's Turn
Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
The previous raise was $0
You have $252
Enter your bet, or $0 to fold: 8
Player 000001476AC3F8E0 Raises $8

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
The previous raise was $0
You have $244
Enter your bet, or $0 to fold: 8
Player 000001476AC3F8E0 Raises $8

Player 000001476AC39E40 Folds

Player 000001476AC37E50 Folds

Player 000001476AC3F8E0's Turn
Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
The previous raise was $0
You have $236
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
Player 000001476AC3F8E0 Wins $0

Player 000001476AC39E40 is the Small Blind
Player 000001476AC39E40 bet $3

Player 000001476AC37E50 is the Big Blind
```

```
Player 000001476AC37E50 bet $6

Player 000001476AC3F8E0's Turn
Your Cards: 2 of Diamonds, J of Clubs, K of Hearts, 5 of Hearts, 7 of Diamonds
The previous raise was $6
You have $286
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC39E40 discarded 4 cards

Player 000001476AC37E50 discarded 2 cards

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
2 of Diamonds, J of Clubs, K of Hearts, 5 of Hearts, 7 of Diamonds
Choose a card to discard (1-5), or 0 to Stop: 1
J of Clubs, K of Hearts, 5 of Hearts, 7 of Diamonds
Choose a card to discard (1-4), or 0 to Stop: 4
J of Clubs, K of Hearts, 5 of Hearts
Choose a card to discard (1-3), or 0 to Stop: 3
J of Clubs, K of Hearts
Choose a card to discard (1-2), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 3 cards

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: J of Clubs, K of Hearts, K of Clubs, 4 of Diamonds, Q of Diamonds
The previous raise was $0
You have $280
Enter your bet, or $0 to fold: 5
Player 000001476AC3F8E0 Raises $5

Player 000001476AC39E40 Folds

Player 000001476AC37E50 Folds

Player 000001476AC3F8E0's Turn
Your Cards: J of Clubs, K of Hearts, K of Clubs, 4 of Diamonds, Q of Diamonds
```

```
The previous raise was $0
You have $275
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: J of Clubs, K of Hearts, K of Clubs, 4 of Diamonds, Q of Diamonds
Player 000001476AC3F8E0 Wins $23

Player 000001476AC37E50 is the Small Blind
Player 000001476AC37E50 bet $1

Player 000001476AC3F8E0 is the Big Blind
Player 000001476AC3F8E0's Turn
Your Cards: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
Minimum Bet is $6
The previous raise was $1
You have $298
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 bet $6

Player 000001476AC39E40 Folds

Player 000001476AC3F8E0's Turn
Your Cards: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
The previous raise was $0
You have $292
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

Player 000001476AC37E50 discarded 2 cards

Player 000001476AC3F8E0's Turn
Your Cards: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
The previous raise was $0
You have $292
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC37E50 is All In
```

```

Player 000001476AC3F8E0's hand: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
Player 000001476AC37E50's hand: 5 of Diamonds, 5 of Clubs, 10 of Hearts, A of Hearts, Q of Diamonds
Player 000001476AC37E50 Wins $7

Player 000001476AC3F8E0 is the Small Blind
Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 2 of Hearts, 6 of Diamonds
Minimum Bet is $3
The previous raise was $0
You have $292
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 bet $3

Player 000001476AC39E40 is the Big Blind
Player 000001476AC39E40 bet $1

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 2 of Hearts, 6 of Diamonds
The previous raise was $0
You have $289
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
8 of Clubs, 10 of Hearts, A of Clubs, 2 of Hearts, 6 of Diamonds
Choose a card to discard (1-5), or 0 to Stop: 4
8 of Clubs, 10 of Hearts, A of Clubs, 6 of Diamonds
Choose a card to discard (1-4), or 0 to Stop: 4
8 of Clubs, 10 of Hearts, A of Clubs
Choose a card to discard (1-3), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 2 cards

Player 000001476AC39E40 discarded 4 cards

Player 000001476AC37E50 discarded 4 cards

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 9 of Spades, 3 of Hearts
The previous raise was $0
You have $289
Enter your bet, or $0 to fold: 2
```

```
Player 000001476AC3F8E0 Raises $2

Player 000001476AC39E40 is All In

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 9 of Spades, 3 of Hearts
The previous raise was $0
You have $287
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: 8 of Clubs, 10 of Hearts, A of Clubs, 9 of Spades, 3 of Hearts
Player 000001476AC39E40's hand: 2 of Spades, 7 of Hearts, 7 of Diamonds, 9 of Hearts, Q of Diamonds
Player 000001476AC37E50's hand: 5 of Clubs, 6 of Hearts, 10 of Diamonds, Q of Spades, 8 of Spades
Player 000001476AC39E40 Wins $11

Player 000001476AC39E40 is the Small Blind
Player 000001476AC39E40 bet $3

Player 000001476AC37E50 is the Big Blind
Player 000001476AC37E50 bet $2

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
The previous raise was $3
You have $287
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 Calls

Player 000001476AC39E40 discarded 3 cards

Player 000001476AC37E50 discarded 4 cards

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

Player 000001476AC39E40 Calls

```

```
Player 000001476AC37E50 is All In

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
The previous raise was $0
You have $284
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
Player 000001476AC39E40's hand: 7 of Spades, 9 of Clubs, 8 of Spades, 4 of Diamonds, 4 of Clubs
Player 000001476AC37E50's hand: 3 of Clubs, K of Diamonds, 10 of Hearts, K of Spades, J of Spades
Player 000001476AC37E50 Wins $8

Player 000001476AC37E50 is the Small Blind
Player 000001476AC37E50 bet $3

Player 000001476AC3F8E0 is the Big Blind
Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
Minimum Bet is $6
The previous raise was $3
You have $284
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 bet $6

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $278
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC39E40 Calls

Player 000001476AC37E50 Calls

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $277
```

```
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC39E40 Folds

Player 000001476AC37E50 Folds

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $276
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $275
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $274
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $273
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC3F8E0's Turn
Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
The previous raise was $0
You have $272
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

Player 000001476AC3F8E0's hand: 8 of Clubs, A of Clubs, 6 of Clubs, A of
```

```
Diamonds, 7 of Spades
Player 000001476AC3F8E0 Wins $26

Player 000001476AC3F8E0 is the Small Blind
Player 000001476AC3F8E0's Turn
Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
Minimum Bet is $3
The previous raise was $0
You have $298
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 bet $3

Player 000001476AC39E40 is the Big Blind
Player 000001476AC39E40 bet $1

Player 000001476AC37E50 Folds

Player 000001476AC3F8E0's Turn
Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
The previous raise was $0
You have $295
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC3F8E0's Turn
Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
The previous raise was $0
You have $294
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

Player 000001476AC39E40 discarded 4 cards

Player 000001476AC3F8E0's Turn
Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
The previous raise was $0
You have $294
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls
```

```

Player 000001476AC39E40 is All In

Player 000001476AC3F8E0's hand: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
Player 000001476AC39E40's hand: 4 of Clubs, 8 of Hearts, 8 of Clubs, 7 of Spades, 5 of Clubs
Player 000001476AC39E40 Wins $5

Player 000001476AC39E40 is the Small Blind
Player 000001476AC39E40 bet $3

Player 000001476AC37E50 is the Big Blind
Player 000001476AC37E50 bet $1

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, 2 of Clubs, K of Clubs, 4 of Diamonds, A of Spades
The previous raise was $3
You have $294
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 Calls

Player 000001476AC39E40 discarded 4 cards

Player 000001476AC37E50 discarded 0 cards

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
Q of Spades, 2 of Clubs, K of Clubs, 4 of Diamonds, A of Spades
Choose a card to discard (1-5), or 0 to Stop: 2
Q of Spades, K of Clubs, 4 of Diamonds, A of Spades
Choose a card to discard (1-4), or 0 to Stop: 3
Q of Spades, K of Clubs, A of Spades
Choose a card to discard (1-3), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 2 cards

Player 000001476AC39E40 Calls

Player 000001476AC37E50 is All In

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
The previous raise was $0
You have $291
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1
```

```

Player 000001476AC39E40 Calls

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
The previous raise was $0
You have $290
Enter your bet, or $0 to fold: 1
Player 000001476AC3F8E0 Raises $1

Player 000001476AC39E40 Folds

Player 000001476AC3F8E0's Turn
Your Cards: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
The previous raise was $0
You have $289
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's hand: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
Player 000001476AC37E50's hand: 2 of Diamonds, 4 of Clubs, 9 of Hearts, K of Hearts, A of Clubs
Player 000001476AC3F8E0 Wins $10

Player 000001476AC37E50 is out of funds.
Player 000001476AC3F8E0 is the Small Blind
Player 000001476AC3F8E0's Turn
Your Cards: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
Minimum Bet is $3
The previous raise was $0
You have $299
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 bet $3

Player 000001476AC39E40 is the Big Blind
Player 000001476AC39E40 bet $1

Player 000001476AC3F8E0's Turn
Your Cards: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
The previous raise was $0
You have $296
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
```

```
5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

Player 000001476AC39E40 discarded 1 cards

Player 000001476AC3F8E0's Turn
Your Cards: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
The previous raise was $0
You have $296
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 is All In

Player 000001476AC3F8E0's hand: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
Player 000001476AC39E40's hand: 3 of Spades, 7 of Diamonds, 10 of Spades, K of Hearts, 10 of Clubs
Player 000001476AC39E40 Wins $4

Player 000001476AC39E40 is the Small Blind
Player 000001476AC39E40 bet $3

Player 000001476AC3F8E0 is the Big Blind
Player 000001476AC3F8E0's Turn
Your Cards: 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
Minimum Bet is $6
The previous raise was $3
You have $296
Enter your bet, or $0 to fold: 6
Player 000001476AC3F8E0 bet $6

Player 000001476AC39E40 Folds

Player 000001476AC3F8E0's Turn
Your Cards: 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
The previous raise was $0
You have $290
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
Choose a card to discard (1-5), or 0 to Stop: 0
Player 000001476AC3F8E0 discarded 0 cards

```

```
Player 000001476AC3F8E0's hand: 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
Player 000001476AC3F8E0 Wins $9

Player 000001476AC3F8E0 is the Small Blind
Player 000001476AC3F8E0's Turn
Your Cards: 4 of Clubs, 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
Minimum Bet is $3
The previous raise was $0
You have $299
Enter your bet, or $0 to fold: 3
Player 000001476AC3F8E0 bet $3

Player 000001476AC39E40 is the Big Blind
Player 000001476AC39E40 bet $1

Player 000001476AC3F8E0's Turn
Your Cards: 4 of Clubs, 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
The previous raise was $0
You have $296
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC3F8E0's Turn to Discard
You may discard up to 5 cards.
4 of Clubs, 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
Choose a card to discard (1-5), or 0 to Stop: 1
6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
Choose a card to discard (1-4), or 0 to Stop: 1
7 of Hearts, 2 of Clubs, 10 of Diamonds
Choose a card to discard (1-3), or 0 to Stop: 1
2 of Clubs, 10 of Diamonds
Choose a card to discard (1-2), or 0 to Stop: 1
10 of Diamonds
Choose a card to discard (1-1), or 0 to Stop: 1
Player 000001476AC3F8E0 discarded 5 cards

Player 000001476AC39E40 discarded 4 cards

Player 000001476AC3F8E0's Turn
Your Cards: 9 of Spades, Q of Clubs, 3 of Clubs, 9 of Diamonds, J of Clubs
The previous raise was $0
You have $296
Enter your bet, or $0 to fold: 0
Player 000001476AC3F8E0 Calls

Player 000001476AC39E40 is All In
```

```

Player 000001476AC3F8E0's hand: 9 of Spades, Q of Clubs, 3 of Clubs, 9 of Diamonds, J of Clubs
Player 000001476AC39E40's hand: 6 of Hearts, 4 of Diamonds, 2 of Hearts, 10 of Hearts, 8 of Clubs
Player 000001476AC3F8E0 Wins $4

Player 000001476AC39E40 is out of funds.
```

Below are changes to the previous modules code, changing Hands to use pointers, as well as the results showing that the scorer output remains unchanged.

```
#pragma once
#include <vector>

class Card;
class Hand;

using namespace std;

// @file
// @author Nathan Roe
// Class to assess and find winner for set of Poker Hands.
//
// Given a set of hands, will evaluate the Poker Hand Type
// and determine the winner or winners.
class Scorer
{
public:
    // Evaluates a set of hands to determine winner(s)
    //
    // Accepts a vector of Hands and moves winner or
    //    winners to the front of the vector. The return
    //    value indicates the last index of a winner.
    //    One winner returns 0, two-way tie returns 1, etc.
    //    Returns -1 for error
    // @param *hands - pointer to vector of Hands to evaluate and rank
    // @return the index of the final winner in the partially
    //     sorted vector
    int findBestHand(vector<Hand*> *hands);

    // Determines and sets the HandType for a given poker hand
    //
    // @param *hand - Hand for which to set HandType
    // @return vector of Cards sorted for comparison based on HandType
    vector<Card> checkHandType(Hand *hand);

private:
    const int HAND_SIZE = 5;

    // Compare two Hands to determine which is better
    //
    // @param hand1 - First Hand for comparing
    // @param hand2 - Second Hand for comparing
    // @return 1 for hand1, 2 for hand2, 0 for tie,
    //    or -1 for error
    int compareHands(Hand *hand1, Hand *hand2);

    // Check hand for presence of Straight Flush
    //
    // If type is present, sort cards for comparison to
```

```
    // similar sets of cards
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasStraightFlush(vector<Card> *cards);

    // Check hand for presence of Four of a Kind
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasFourOfAKind(vector<Card> *cards);

    // Check hand for presence of Full House
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasFullHouse(vector<Card> *cards);

    // Check hand for presence of Flush
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasFlush(vector<Card> *cards);

    // Check hand for presence of Straight
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasStraight(vector<Card> *cards);

    // Check hand for presence of Three of a Kind
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasThreeOfAKind(vector<Card> *cards);

    // Check hand for presence of Two Pair
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasTwoPair(vector<Card> *cards);

    // Check hand for presence of One Pair
    //
```

```
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasOnePair(vector<Card> *cards);

    // Check hand for presence of High Card
    //
    // If type is present, sort cards for comparison
    // @param *cards - pointer to vector of cards to check
    // @return true if hand type is present, false otherwise
    bool hasHighCard(vector<Card> *cards);
};

```

```cpp
#include "scorer.h"
#include "hand.h"
#include "card.h"
#include <vector>
#include <algorithm>

using namespace std;

// Evaluates a set of hands to determine winner(s)
int Scorer::findBestHand(vector<Hand*> *hands)
{
    // Return error if not enough hands to score
    if (hands->size() < 2)
    {
        return -1;
    }

    int numTied = 0;
    // Begin with first hand as current best
    Hand *bestHand = hands->at(0);
    // Compare remaining hands to the current best hand
    for (int idx = 1; idx < hands->size(); ++idx)
    {
        int result = compareHands(bestHand, hands->at(idx));
        // If hands tie, move new hand to front section of vector
        if (result == 0)
        {
            ++numTied;
            Hand *hand = hands->at(idx);
            hands->erase(hands->begin() + idx);
            hands->insert(hands->begin() + numTied, hand);
        }
        // If current best hand is better, leave vector unchanged
        else if (result == 1)
        {
            continue;
        }
        // If new hand is better, move to front and reset number of ties
        else if (result == 2)
        {
            numTied = 0;
            Hand *hand = hands->at(idx);
            hands->erase(hands->begin() + idx);
            hands->insert(hands->begin(), hand);
        }
        // Return error for unexpected value
        else
        {
            return -1;
```

```
        }
    }
    return numTied;
} // End function findBestHand

// Compare two Hands to determine which is better
int Scorer::compareHands(Hand *hand1, Hand *hand2)
{
    // If hands are not sets of 5 cards, return error
    if (!(hand1->size() == HAND_SIZE) ||
        !(hand2->size() == HAND_SIZE))
    {
        return -1;
    }

    // Check type of each hand, and prep cards for comparison
    vector<Card> cards1 = checkHandType(hand1);
    vector<Card> cards2 = checkHandType(hand2);

    // Verify that each hand is a valid Poker Hand Type
    if (hand1->getHandType() == Hand::HandType::None ||
        hand2->getHandType() == Hand::HandType::None)
    {
        return -1;
    }

    int result = -1;

    // Check to see if one Hand is a higher rank than the other
    if (hand1->getHandType() > hand2->getHandType())
    {
        result = 1;
    }
    else if (hand1->getHandType() < hand2->getHandType())
    {
        result = 2;
    }
    // If hands are of same rank, compare cards to find best hand
    else
    {
        // Since hands are pre-sorted, iterate through cards until
        // a higher card is found
        for (int idx = 0; idx < HAND_SIZE; ++idx)
        {
            if (cards1.at(idx) > cards2.at(idx))
            {
                result = 1;
                break;
            }
```

```
            else if (cards1.at(idx) < cards2.at(idx))
            {
                result = 2;
                break;
            }
            // If all cards are of same value, return tie
            else
            {
                result = 0;
            }
        }
    }
    return result;
} // End function compareHands

// Determines and sets the HandType for a given poker hand
vector<Card> Scorer::checkHandType(Hand *hand)
{
    vector<Card> cards = hand->getCards();
    // Verify that all cards in the hand are valid
    for (Card card : cards)
    {
        if (!card.isValid())
        {
            return cards;
        }
    }

    // Test hand against Poker Hand Types from Best to Worst
    if (hasStraightFlush(&cards))
    {
        hand->setHandType(Hand::HandType::StraightFlush);
    }
    else if (hasFourOfAKind(&cards))
    {
        hand->setHandType(Hand::HandType::FourOfAKind);
    }
    else if (hasFullHouse(&cards))
    {
        hand->setHandType(Hand::HandType::FullHouse);
    }
    else if (hasFlush(&cards))
    {
        hand->setHandType(Hand::HandType::Flush);
    }
    else if (hasStraight(&cards))
    {
        hand->setHandType(Hand::HandType::Straight);
    }
```

```
    else if (hasThreeOfAKind(&cards))
    {
        hand->setHandType(Hand::HandType::ThreeOfAKind);
    }
    else if (hasTwoPair(&cards))
    {
        hand->setHandType(Hand::HandType::TwoPair);
    }
    else if (hasOnePair(&cards))
    {
        hand->setHandType(Hand::HandType::OnePair);
    }
    else if (hasHighCard(&cards))
    {
        hand->setHandType(Hand::HandType::HighCard);
    }
    // If no valid hand type is found, assign None
    else
    {
        hand->setHandType(Hand::HandType::None);
    }

    // Return vector of cards sorted for comparison based on HandType
    return cards;
} // End function checkHandType

// Check hand for presence of Straight Flush
bool Scorer::hasStraightFlush(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    // Sort cards from high to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    // Special case for Ace-Low
    if (cards->at(0).value == Card::Value::Ace &&
        cards->at(1).value == Card::Value::Five)
    {
        // Set starting card value/suit to the 5
        int startVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(1).value);
        Card::Suit startSuit = cards->at(0).suit;
        // Iterate through remaining cards, and exit if not in
        // descending order and matching suit
```

```cpp
        for (int idx = 2; idx < cards->size(); ++idx)
        {
            int cardVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(idx).value);
            Card::Suit cardSuit = cards->at(idx).suit;
            if (!(cards->at(idx).isValid()) ||
                !(cardSuit == startSuit) ||
                !(cardVal == startVal-- - 1))
            {
                return false;
            }
        }
        // Move ace to end
        Card ace = cards->front();
        cards->erase(cards->begin());
        cards->push_back(ace);
    }
    // Check Cases with no Aces
    else
    {
        // Set starting card value/suit
        int startVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(0).value);
        Card::Suit startSuit = cards->at(0).suit;
        // Iterate through remaining cards, and exit if not in
        // descending order and matching suit
        for (int idx = 1; idx < cards->size(); ++idx)
        {
            int cardVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(idx).value);
            Card::Suit cardSuit = cards->at(idx).suit;
            if (!(cards->at(idx).isValid()) ||
                !(cardSuit == startSuit) ||
                !(cardVal == startVal-- - 1))
            {
                return false;
            }
        }
    }
    return true;
} // End function hasStraightFlush

// Check hand for presence of Four of a Kind
bool Scorer::hasFourOfAKind(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
```

```
    {
        return false;
    }

    // Sort cards from hight to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    int FOAK = 4;
    int foundFOAK = false;
    Card::Value startVal = Card::Value::Invalid;
    // Test cards in sets (1-4) and (2-5) to see if all values match
    for (int idx = 0; idx <= cards->size() - FOAK; ++idx)
    {
        foundFOAK = true;
        startVal = cards->at(idx).value;
        // Iterate over 3 following cards and check for match
        for (int innerIdx = idx + 1; innerIdx < idx + FOAK; ++innerIdx)
        {
            Card::Value cardVal = cards->at(innerIdx).value;
            // If different value is found, no FOAK
            if (!(cards->at(innerIdx).isValid()) ||
                !(startVal == cardVal))
            {
                foundFOAK = false;
                break;
            }
        }
        // If first four cards are FOAK, break loop
        if (foundFOAK)
        {
            break;
        }
    }

    // If cards have FOAK, sort matching cards to the front
    if (foundFOAK)
    {
        vector<Card> unusedCards;
        for (int idx = 0; idx < cards->size(); ++idx)
        {
            if (!(cards->at(idx).value == startVal))
            {
                unusedCards.push_back(cards->at(idx));
                cards->erase(cards->begin() + idx);
            }
        }
        // Sort non-FOAK cards low to high
        sort(unusedCards.begin(), unusedCards.end());
```

```
        // Put unused cards at the end of card set from
        // hight to low
        for (int idx = 0; idx < unusedCards.size(); ++idx)
        {
            cards->push_back(unusedCards.back());
            unusedCards.pop_back();
        }
    }

    return foundFOAK;
} // End function hasFourOfAKind

// Check hand for presence of Full House
bool Scorer::hasFullHouse(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    bool foundFF = false;
    Card::Value pair = Card::Value::Invalid;
    Card::Value triple = Card::Value::Invalid;
    // Check wether first two cards match
    if (cards->at(0).value == cards->at(1).value)
    {
        // If first two cards match, check whether
        // first three cards make a set of 3
        if (cards->at(1).value == cards->at(2).value)
        {
            triple = cards->at(0).value;
        }
        // Otherwise, set pair value as value of first card
        else
        {
            pair = cards->at(0).value;
        }

        if (!(pair == Card::Value::Invalid))
        {
            // If first two cards are a pair, check remaining
            // cards to see if they make a set of 3
            if (cards->at(2).value == cards->at(3).value &&
                cards->at(3).value == cards->at(4).value)
            {
```

```
                foundFF = true;
                triple = cards->at(2).value;
                // Move set of 3 to the front
                cards->push_back(cards->front());
                cards->erase(cards->begin());
                cards->push_back(cards->front());
                cards->erase(cards->begin());
            }
        }
        else if (!(triple == Card::Value::Invalid))
        {
            // If first three cards make a set of 3, check
            // remaining cards to see if they are a pair
            if (cards->at(3).value == cards->at(4).value)
            {
                foundFF = true;
                pair = cards->at(3).value;
            }
        }
    }
    return foundFF;
} // End function hasFullHouse

// Check hand for presence of Flush
bool Scorer::hasFlush(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    // Sort cards from high to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    Card::Suit startSuit = cards->at(0).suit;
    // Iterate through cards to check whether suits match
    for (int idx = 1; idx < cards->size(); ++idx)
    {
        Card::Suit cardSuit = cards->at(idx).suit;
        // If different suit is found, return false
        if (!(cards->at(idx).isValid()) ||
            !(cardSuit == startSuit))
        {
            return false;
        }
    }

```

```cpp
    return true;
} // End function hasFlush

// Check hand for presence of Straight
bool Scorer::hasStraight(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    // Sort cards high to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    // Special case for Ace-Low
    if (cards->at(0).value == Card::Value::Ace &&
        cards->at(1).value == Card::Value::Five)
    {
        // Set highest card as the 5
        int startVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(1).value);
        for (int idx = 2; idx < cards->size(); ++idx)
        {
            int cardVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(idx).value);
            // Iterate through remaining cards to see if they are consecutive
            if (!(cards->at(idx).isValid()) ||
                !(cardVal == startVal-- - 1))
            {
                return false;
            }
        }
        // Move Ace to the back
        Card ace = cards->front();
        cards->erase(cards->begin());
        cards->push_back(ace);
    }
    // Check Cases with no Aces
    else
    {
        // Set starting card value
        int startVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(0).value);
        for (int idx = 1; idx < cards->size(); ++idx)
        {
```

```cpp
            int cardVal = static_cast<typename std::underlying_type<Card::Value>::type>(cards->at(idx).value);
            // Iterate through remaining cards to see if they are consecutive
            if (!(cards->at(idx).isValid()) ||
                !(cardVal == startVal-- - 1))
            {
                return false;
            }
        }
    }
    return true;
} // End function hasStraight

// Check hand for presence of Three of a Kind
bool Scorer::hasThreeOfAKind(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    // Sort cards high to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    int TOAK = 3;
    int foundTOAK = false;
    Card::Value startVal = Card::Value::Invalid;
    // Check (1-3), (2-4), (3-5) for three of a kind
    for (int idx = 0; idx <= cards->size() - TOAK; ++idx)
    {
        foundTOAK = true;
        startVal = cards->at(idx).value;
        // Compare three consecutive cards for matching values
        for (int innerIdx = idx + 1; innerIdx < idx + TOAK; ++innerIdx)
        {
            Card::Value cardVal = cards->at(innerIdx).value;
            // If different value is found, move to next set of 3
            if (!(cards->at(innerIdx).isValid()) ||
                !(startVal == cardVal))
            {
                foundTOAK = false;
                break;
            }
        }
        // End search if three of a kind is found
```

```
        if (foundTOAK)
        {
            break;
        }
    }

    // If three of a kind is found, sort cards for comparison
    if (foundTOAK)
    {
        vector<Card> unusedCards;
        int cardsMoved = 0;
        // Remove cards that are not part of TOAK
        for (int idx = 0; idx < HAND_SIZE; ++idx)
        {
            int cardIdx = idx - cardsMoved;
            if (!(cards->at(cardIdx).value == startVal))
            {
                unusedCards.push_back(cards->at(cardIdx));
                cards->erase(cards->begin() + cardIdx);
                ++cardsMoved;
            }
        }
        // Place non-TOAK cards from high-low and end of cards
        sort(unusedCards.begin(), unusedCards.end());
        size_t startSize = unusedCards.size();
        for (int idx = 0; idx < startSize; ++idx)
        {
            cards->push_back(unusedCards.back());
            unusedCards.pop_back();
        }
    }

    return foundTOAK;
} // End function hasThreeOfAKind

// Check hand for presence of Two Pair
bool Scorer::hasTwoPair(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    // Sort cards from high to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    bool foundTP = false;
```

```
    vector<Card> pair1;
    vector<Card> pair2;

    // Search cards for a pair
    for (int idx = 1; idx < cards->size(); ++idx)
    {
        int prevIdx = idx - 1;
        Card *card = &cards->at(idx);
        Card *prevCard = &cards->at(prevIdx);
        // If pair is found, store cards
        if (card->value == prevCard->value)
        {
            foundTP = true;
            pair1.push_back(*prevCard);
            pair1.push_back(*card);
            cards->erase(cards->begin() + prevIdx);
            cards->erase(cards->begin() + prevIdx);
            break;
        }
    }

    // If pair is found, search remaining cards for second pair
    if (foundTP)
    {
        foundTP = false;
        for (int idx = 1; idx < cards->size(); ++idx)
        {
            int prevIdx = idx - 1;
            Card *card = &cards->at(idx);
            Card *prevCard = &cards->at(prevIdx);
            // If pair is found, store cards
            if (card->value == prevCard->value)
            {
                foundTP = true;
                pair2.push_back(*prevCard);
                pair2.push_back(*card);
                cards->erase(cards->begin() + prevIdx);
                cards->erase(cards->begin() + prevIdx);
            }
        }
    }

    // If two-pair found, sort cards highPair-lowPair-spareCard
    if (foundTP)
    {
        // Place pairs in set highest to lowest
        if (pair1.front().value > pair2.front().value)
        {
            cards->push_back(pair1.front());
```

```
            cards->push_back(pair1.back());
            cards->push_back(pair2.front());
            cards->push_back(pair2.back());
        }
        else
        {
            cards->push_back(pair2.front());
            cards->push_back(pair2.back());
            cards->push_back(pair1.front());
            cards->push_back(pair1.back());
        }
        // Move spare card to the end
        cards->push_back(cards->front());
        cards->erase(cards->begin());
    }
    // Place unused pair back into card set
    else if (pair1.size() > 0)
    {
        cards->push_back(pair1.front());
        cards->push_back(pair1.back());
    }

    return foundTP;
} // End function foundTwoPair

// Check hand for presence of One Pair
bool Scorer::hasOnePair(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    bool foundPair = false;
    vector<Card> pair;

    // Search cards for pair of equal value
    for (int idx = 1; idx < cards->size(); ++idx)
    {
        int prevIdx = idx - 1;
        Card *card = &cards->at(idx);
        Card *prevCard = &cards->at(prevIdx);
        // If pair is found, store pair
        if (card->value == prevCard->value)
        {
```

```
            foundPair = true;
            pair.push_back(*prevCard);
            pair.push_back(*card);
            cards->erase(cards->begin() + prevIdx);
            cards->erase(cards->begin() + prevIdx);
            break;
        }
    }

    // If pair is found, place pair at front of cards
    if (foundPair)
    {
        cards->insert(cards->begin(), pair.front());
        cards->insert(cards->begin(), pair.back());
    }

    return foundPair;
} // End function hasOnePair

// Check hand for presence of High Card
bool Scorer::hasHighCard(vector<Card> *cards)
{
    // Return false if vector is not 5 cards
    if (!(cards->size() == HAND_SIZE))
    {
        return false;
    }

    // Sort cards from high to low
    sort(cards->begin(), cards->end());
    reverse(cards->begin(), cards->end());

    return true;
} // End function hasHighCard

```

```cpp
#include "card.h"
#include "hand.h"
#include "scorer.h"
#include <iostream>
#include <vector>
#include <libconfig.h++>

using namespace std;
using namespace libconfig;
// @file
// @author Nathan Roe
// Compares Poker Hands provided in file to find winners
//
// Given a set of hands, will evaluate the Poker Hand Type
// and determine the winner or winners.

// Creates Card object using integer value and string suit
Card makeCard(int value, string suit)
{
    Card::Value cardValue = Card::Value::Invalid;
    Card::Suit cardSuit = Card::Suit::Invalid;
    // Set card value
    try
    {
        cardValue = static_cast<Card::Value>(value);
    }
    catch (const exception &e)
    {
        cout << e.what() << endl;
        cout << "Invalid Card Value: " << value << endl;
        cardValue = Card::Value::Invalid;
    }

    // Set card suit
    if (suit.size() > 0)
    {
        char cardChar = suit[0];
        switch (cardChar)
        {
        case 'C':
            cardSuit = Card::Suit::Clubs;
            break;
        case 'D':
            cardSuit = Card::Suit::Diamonds;
            break;
        case 'H':
            cardSuit = Card::Suit::Hearts;
            break;
        case 'S':
```

```
            cardSuit = Card::Suit::Spades;
            break;
        default:
            cardSuit = Card::Suit::Invalid;
            break;
        }
    }

    return Card(cardValue, cardSuit);
} // End function makeCard

// Creates Card object using string value and suit
Card makeCard(string value, string suit)
{
    Card::Value cardValue = Card::Value::Invalid;
    Card::Suit cardSuit = Card::Suit::Invalid;

    // Set value for Ten through Ace
    if (value == "Ten" || value == "T")
    {
        cardValue = Card::Value::Ten;
    }
    else if (value == "Jack" || value == "J")
    {
        cardValue = Card::Value::Jack;
    }
    else if (value == "Queen" || value == "Q")
    {
        cardValue = Card::Value::Queen;
    }
    else if (value == "King" || value == "K")
    {
        cardValue = Card::Value::King;
    }
    else if (value == "Ace" || value == "A")
    {
        cardValue = Card::Value::Ace;
    }
    // If card is not 10-Ace, try to set based on numerical value
    else
    {
        try
        {
            cardValue = static_cast<Card::Value>(stoi(value));
        }
        catch (const invalid_argument &e)
        {
            cout << e.what() << ": Invalid Card Value: " << value << endl;
            cardValue = Card::Value::Invalid;
```

```
        }
        catch (const exception &e)
        {
            cout << e.what() << ": Invalid Card Value: " << value << endl;
            cardValue = Card::Value::Invalid;
        }
    }

    // Set suit value
    if (suit.size() > 0)
    {
        char cardChar = suit[0];
        switch (cardChar)
        {
        case 'C':
            cardSuit = Card::Suit::Clubs;
            break;
        case 'D':
            cardSuit = Card::Suit::Diamonds;
            break;
        case 'H':
            cardSuit = Card::Suit::Hearts;
            break;
        case 'S':
            cardSuit = Card::Suit::Spades;
            break;
        default:
            cardSuit = Card::Suit::Invalid;
            break;
        }
    }

    return Card(cardValue, cardSuit);
} // End function makeCard

// Format and print test results for input files
void printResults(int lastWinIdx, vector<Hand*> *players)
{
    // Print out error message if error value returned
    if (lastWinIdx == -1)
    {
        cout << "ERROR IN SCENARIO" << endl;
        return;
    }

    // Print out winning hands
    cout << "(" << players->at(0)->printCards() << ")";
    for (int idx = 1; idx <= lastWinIdx; ++idx)
    {
```

```
        cout << ", "
             << "(" << players->at(idx)->printCards() << ")";
    }
    // Format string based on whether ties are present
    if (lastWinIdx > 0)
    {
        cout << " Tie for the Win.";
    }
    else
    {
        cout << " Wins.";
    }

    // Print out losing hands
    bool multiLoser = false;
    for (int idx = lastWinIdx + 1; idx < players->size(); ++idx)
    {
        cout << " "
             << "(" << players->at(idx)->printCards() << ")";
        if (idx < players->size() - 1)
        {
            multiLoser = true;
            cout << " and";
        }
    }
    // Format string based on presence of multiple losing hands
    if (multiLoser)
    {
        cout << " Lose.";
    }
    else if (lastWinIdx < players->size() - 1)
    {
        cout << " Loses.";
    }
    cout << endl;
} // End function printResults

// Main function; runs Poker Hand Scorer for each input file provided
int main(int argc, char **argv)
{
    Scorer *scorer = new Scorer();

    // Iterate through input args, and read as paths
    for (int idx = 1; idx < argc; ++idx)
    {
        Config *cfg = new Config();
        char *path = argv[idx];

        // Attempt to parse Config files
```

```
        try
        {
            cout << path << endl;
            cfg->readFile(path);
        }
        // Catch file path errors
        catch (const FileIOException &fioex)
        {
            cerr << fioex.what() << " while reading file " << path << endl;
            return (EXIT_FAILURE);
        }
        // Catch config file format errors
        catch (const ParseException &pex)
        {
            cerr << "Parse error at " << pex.getFile() << ":" << pex.getLine()
                << " - " << pex.getError() << endl;
            return (EXIT_FAILURE);
        }

        vector<Hand*> *players = new vector<Hand*>;

        const Setting &root = cfg->getRoot();
        // Find "hands" data set in config file
        try
        {
            const Setting &hands = root["hands"];
            int numHands = hands.getLength();

            // Create a Hand object for each hand in Config
            for (int handIdx = 0; handIdx < numHands; ++handIdx)
            {

                Hand *hand = new Hand();

                // Read Cards list in config
                const Setting &cards = hands[handIdx];
                int numCards = cards.getLength();
                for (int cardIdx = 0; cardIdx < numCards; ++cardIdx)
                {
                    const Setting &card = cards[cardIdx];

                    string suit;
                    bool suitFound = false;
                    bool valueFound = false;

                    // Find required suit and value
                    suitFound = card.lookupValue("suit", suit);
```

```cpp
                    if (suitFound)
                    {
                        int value;
                        string valueString;
                        // Search for card values formatted as Integers
                        valueFound = card.lookupValue("value", value);
                        if (valueFound)
                        {
                            Card card = makeCard(value, suit);
                            // Add valid card to hand
                            if (card.isValid())
                            {
                                hand->deal(&card);
                            }
                        }
                        else
                        {
                            // Search for card values formatted as Strings
                            valueFound = card.lookupValue("value", valueString);
                            if (valueFound)
                            {
                                Card card = makeCard(valueString, suit);
                                // Add valid card to hand
                                if (card.isValid())
                                {
                                    hand->deal(&card);
                                }
                            }
                        }
                    }
                    // Print error for bad hand
                    if (!(valueFound && suitFound))
                    {
                        cout << "Could not find 'suit' and/or 'value' in Hand " << handIdx + 1 << " Card " << cardIdx + 1 << endl;
                    }
                }
                players->push_back(hand);
            }

            // Run scorer if multiple hands present
            if (players->size() > 1)
            {
                int winnerIdx = scorer->findBestHand(players);
                for (Hand *hand : *players)
                {
                    cout << "(" << hand->printCards() << "): " << hand-
```

```
                        >printHandType() << endl;
                    }
                    printResults(winnerIdx, players);
                }
            }
            // Print error message for improperly formatted file
            catch (const SettingNotFoundException &nfex)
            {
                cout << nfex.what() << ": Could not find 'hands' in " << path << endl;
            }
        }
        return 0;
} // End function main

```

```
C:\Users\nater\source\repos\PokerHands>python test\PokerHandTest.py > doc\output.txt

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Flush1.cfg
     (K of Diamonds, J of Diamonds, 9 of Diamonds, 6 of Diamonds, 4 of Diamonds): Flush
     (Q of Clubs, J of Clubs, 7 of Clubs, 6 of Clubs, 5 of Clubs): Flush
     (K of Diamonds, J of Diamonds, 9 of Diamonds, 6 of Diamonds, 4 of Diamonds) Wins. (Q of Clubs, J of Clubs, 7 of Clubs, 6 of Clubs, 5 of Clubs) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Flush2.cfg
     (Q of Clubs, J of Clubs, 7 of Clubs, 6 of Clubs, 5 of Clubs): Flush
     (J of Hearts, 10 of Hearts, 9 of Hearts, 4 of Hearts, 2 of Hearts): Flush
     (Q of Clubs, J of Clubs, 7 of Clubs, 6 of Clubs, 5 of Clubs) Wins. (J of Hearts, 10 of Hearts, 9 of Hearts, 4 of Hearts, 2 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Flush3.cfg
     (J of Hearts, 10 of Hearts, 9 of Hearts, 4 of Hearts, 2 of Hearts): Flush
     (J of Spades, 10 of Spades, 8 of Spades, 6 of Spades, 3 of Spades): Flush
     (J of Hearts, 10 of Hearts, 9 of Hearts, 4 of Hearts, 2 of Hearts) Wins. (J of Spades, 10 of Spades, 8 of Spades, 6 of Spades, 3 of Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Flush4.cfg
     (J of Spades, 10 of Spades, 8 of Spades, 6 of Spades, 3 of Spades): Flush
     (J of Hearts, 10 of Hearts, 8 of Hearts, 4 of Hearts, 3 of Hearts): Flush
     (J of Spades, 10 of Spades, 8 of Spades, 6 of Spades, 3 of Spades) Wins. (J of Hearts, 10 of Hearts, 8 of Hearts, 4 of Hearts, 3 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Flush5.cfg
     (J of Hearts, 10 of Hearts, 8 of Hearts, 4 of Hearts, 3 of Hearts): Flush
```

```
     (J of Clubs, 10 of Clubs, 8 of Clubs, 4 of Clubs, 2 of Clubs): Flush
     (J of Hearts, 10 of Hearts, 8 of Hearts, 4 of Hearts, 3 of Hearts) Wins. (J of Clubs, 10 of Clubs, 8 of Clubs, 4 of Clubs, 2 of Clubs) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Flush6.cfg
     (10 of Diamonds, 8 of Diamonds, 7 of Diamonds, 6 of Diamonds, 5 of Diamonds): Flush
     (10 of Spades, 8 of Spades, 7 of Spades, 6 of Spades, 5 of Spades): Flush
     (10 of Diamonds, 8 of Diamonds, 7 of Diamonds, 6 of Diamonds, 5 of Diamonds), (10 of Spades, 8 of Spades, 7 of Spades, 6 of Spades, 5 of Spades) Tie for the Win.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/FourOfAKind1.cfg
     (K of Spades, K of Hearts, K of Clubs, K of Diamonds, 3 of Hearts): Four of a Kind
     (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, Q of Hearts): Four of a Kind
     (K of Spades, K of Hearts, K of Clubs, K of Diamonds, 3 of Hearts) Wins. (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, Q of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/FourOfAKind2.cfg
     (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, Q of Hearts): Four of a Kind
     (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, 10 of Spades): Four of a Kind
     (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, Q of Hearts) Wins. (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, 10 of Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/FourOfAKind3.cfg
     (4 of Clubs, 4 of Spades, 4 of Diamonds, 4 of Hearts, 9 of Clubs): Four of a Kind
     (4 of Clubs, 4 of Spades, 4 of Diamonds, 4 of Hearts, 9 of Diamonds): Four of a Kind
     (4 of Clubs, 4 of Spades, 4 of Diamonds, 4 of Hearts, 9 of Clubs), (4 of Clubs, 4 of Spades, 4 of Diamonds, 4 of Hearts, 9 of Diamonds) Tie for the Win.
Test Result: PASS
```

```

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/FullHouse1.cfg
     (8 of Spades, 8 of Diamonds, 8 of Hearts, 7 of Diamonds, 7 of Clubs): Full House
     (4 of Diamonds, 4 of Spades, 4 of Clubs, 9 of Diamonds, 9 of Clubs): Full House
     (8 of Spades, 8 of Diamonds, 8 of Hearts, 7 of Diamonds, 7 of Clubs) Wins. (4 of Diamonds, 4 of Spades, 4 of Clubs, 9 of Diamonds, 9 of Clubs) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/FullHouse2.cfg
     (4 of Diamonds, 4 of Spades, 4 of Clubs, 9 of Diamonds, 9 of Clubs): Full House
     (4 of Diamonds, 4 of Spades, 4 of Clubs, 5 of Clubs, 5 of Diamonds): Full House
     (4 of Diamonds, 4 of Spades, 4 of Clubs, 9 of Diamonds, 9 of Clubs) Wins. (4 of Diamonds, 4 of Spades, 4 of Clubs, 5 of Clubs, 5 of Diamonds) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/FullHouse3.cfg
     (K of Clubs, K of Spades, K of Diamonds, J of Clubs, J of Spades): Full House
     (K of Clubs, K of Hearts, K of Diamonds, J of Clubs, J of Hearts): Full House
     (K of Clubs, K of Spades, K of Diamonds, J of Clubs, J of Spades), (K of Clubs, K of Hearts, K of Diamonds, J of Clubs, J of Hearts) Tie for the Win.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard1.cfg
     (K of Spades, 6 of Clubs, 5 of Hearts, 3 of Diamonds, 2 of Clubs): High Card
     (Q of Spades, J of Diamonds, 6 of Clubs, 5 of Hearts, 3 of Clubs): High Card
     (K of Spades, 6 of Clubs, 5 of Hearts, 3 of Diamonds, 2 of Clubs) Wins. (Q of Spades, J of Diamonds, 6 of Clubs, 5 of Hearts, 3 of Clubs) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard2.cfg
     (Q of Spades, J of Diamonds, 6 of Clubs, 5 of Hearts, 3 of Clubs): High Card
```

```
     (Q of Spades, 10 of Diamonds, 8 of Clubs, 7 of Diamonds, 4 of Spades): High Card
     (Q of Spades, J of Diamonds, 6 of Clubs, 5 of Hearts, 3 of Clubs) Wins. (Q of Spades, 10 of Diamonds, 8 of Clubs, 7 of Diamonds, 4 of Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard3.cfg
     (Q of Spades, 10 of Diamonds, 8 of Clubs, 7 of Diamonds, 4 of Spades): High Card
     (Q of Hearts, 10 of Hearts, 7 of Clubs, 6 of Hearts, 4 of Spades): High Card
     (Q of Spades, 10 of Diamonds, 8 of Clubs, 7 of Diamonds, 4 of Spades) Wins. (Q of Hearts, 10 of Hearts, 7 of Clubs, 6 of Hearts, 4 of Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard4.cfg
     (Q of Hearts, 10 of Hearts, 7 of Clubs, 6 of Hearts, 4 of Spades): High Card
     (Q of Clubs, 10 of Clubs, 7 of Diamonds, 5 of Clubs, 4 of Diamonds): High Card
     (Q of Hearts, 10 of Hearts, 7 of Clubs, 6 of Hearts, 4 of Spades) Wins. (Q of Clubs, 10 of Clubs, 7 of Diamonds, 5 of Clubs, 4 of Diamonds) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard5.cfg
     (Q of Clubs, 10 of Clubs, 7 of Diamonds, 5 of Clubs, 4 of Diamonds): High Card
     (Q of Hearts, 10 of Diamonds, 7 of Spades, 5 of Spades, 2 of Hearts): High Card
     (Q of Clubs, 10 of Clubs, 7 of Diamonds, 5 of Clubs, 4 of Diamonds) Wins. (Q of Hearts, 10 of Diamonds, 7 of Spades, 5 of Spades, 2 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard6.cfg
     (10 of Clubs, 8 of Spades, 7 of Spades, 6 of Hearts, 4 of Diamonds): High Card
     (10 of Diamonds, 8 of Diamonds, 7 of Spades, 6 of Clubs, 4 of Clubs): High Card
     (10 of Clubs, 8 of Spades, 7 of Spades, 6 of Hearts, 4 of Diamonds), (10 of Diamonds, 8 of Diamonds, 7 of Spades, 6 of Clubs, 4 of Clubs) Tie for the Win.
```

```
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair1.cfg
     (9 of Clubs, 9 of Diamonds, Q of Spades, J of Hearts, 5 of Hearts): One Pair
     (6 of Diamonds, 6 of Hearts, K of Spades, 7 of Hearts, 4 of Clubs): One Pair
     (9 of Clubs, 9 of Diamonds, Q of Spades, J of Hearts, 5 of Hearts) Wins. (6 of Diamonds, 6 of Hearts, K of Spades, 7 of Hearts, 4 of Clubs) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair2.cfg
     (6 of Diamonds, 6 of Hearts, K of Spades, 7 of Hearts, 4 of Clubs): One Pair
     (6 of Diamonds, 6 of Hearts, Q of Hearts, J of Spades, 2 of Clubs): One Pair
     (6 of Diamonds, 6 of Hearts, K of Spades, 7 of Hearts, 4 of Clubs) Wins. (6 of Diamonds, 6 of Hearts, Q of Hearts, J of Spades, 2 of Clubs) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair3 (2 shuffle).cfg
     (4 of Clubs, 6 of Diamonds, K of Spades, 7 of Hearts, 6 of Hearts): One Pair
     (J of Spades, 6 of Diamonds, 6 of Hearts, 2 of Clubs, Q of Hearts): One Pair
     (4 of Clubs, 6 of Diamonds, K of Spades, 7 of Hearts, 6 of Hearts) Wins. (J of Spades, 6 of Diamonds, 6 of Hearts, 2 of Clubs, Q of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair4.cfg
     (6 of Diamonds, 6 of Hearts, Q of Hearts, J of Spades, 2 of Clubs): One Pair
     (6 of Diamonds, 6 of Hearts, Q of Spades, 8 of Clubs, 7 of Diamonds): One Pair
     (6 of Diamonds, 6 of Hearts, Q of Hearts, J of Spades, 2 of Clubs) Wins. (6 of Diamonds, 6 of Hearts, Q of Spades, 8 of Clubs, 7 of Diamonds) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair5.cfg
```

```
     (6 of Diamonds, 6 of Hearts, Q of Spades, 8 of Clubs, 7 of Diamonds): One Pair
     (6 of Diamonds, 6 of Hearts, Q of Diamonds, 8 of Hearts, 3 of Spades): One Pair
     (6 of Diamonds, 6 of Hearts, Q of Spades, 8 of Clubs, 7 of Diamonds) Wins. (6 of Diamonds, 6 of Hearts, Q of Diamonds, 8 of Hearts, 3 of Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair6.cfg
     (8 of Spades, 8 of Diamonds, 10 of Hearts, 6 of Clubs, 5 of Spades): One Pair
     (8 of Hearts, 8 of Clubs, 10 of Clubs, 6 of Spades, 5 of Clubs): One Pair
     (8 of Spades, 8 of Diamonds, 10 of Hearts, 6 of Clubs, 5 of Spades), (8 of Hearts, 8 of Clubs, 10 of Clubs, 6 of Spades, 5 of Clubs) Tie for the Win.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Straight1.cfg
     (J of Hearts, 10 of Hearts, 9 of Clubs, 8 of Spades, 7 of Hearts): Straight
     (10 of Spades, 9 of Spades, 8 of Clubs, 7 of Hearts, 6 of Spades): Straight
     (J of Hearts, 10 of Hearts, 9 of Clubs, 8 of Spades, 7 of Hearts) Wins. (10 of Spades, 9 of Spades, 8 of Clubs, 7 of Hearts, 6 of Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Straight2.cfg
     (10 of Spades, 9 of Spades, 8 of Clubs, 7 of Hearts, 6 of Spades): Straight
     (6 of Clubs, 5 of Spades, 4 of Hearts, 3 of Spades, 2 of Diamonds): Straight
     (10 of Spades, 9 of Spades, 8 of Clubs, 7 of Hearts, 6 of Spades) Wins. (6 of Clubs, 5 of Spades, 4 of Hearts, 3 of Spades, 2 of Diamonds) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/Straight3.cfg
     (9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Diamonds, 5 of Diamonds): Straight
     (9 of Spades, 8 of Spades, 7 of Spades, 6 of Hearts, 5 of Hearts): Straight
     (9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Diamonds, 5 of Diamonds),
```

```
          (9 of Spades, 8 of Spades, 7 of Spades, 6 of Hearts, 5 of Hearts)
          Tie for the Win.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/
       StraightFlush1.cfg
     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs):
       Straight Flush
     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts):
       Straight Flush
     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs) Wins.
       (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts)
       Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/
       StraightFlush2.cfg
     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts):
       Straight Flush
     (6 of Spades, 5 of Spades, 4 of Spades, 3 of Spades, 2 of Spades):
       Straight Flush
     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts)
       Wins. (6 of Spades, 5 of Spades, 4 of Spades, 3 of Spades, 2 of
       Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/
       StraightFlush3.cfg
     (7 of Diamonds, 6 of Diamonds, 5 of Diamonds, 4 of Diamonds, 3 of
       Diamonds): Straight Flush
     (7 of Spades, 6 of Spades, 5 of Spades, 4 of Spades, 3 of Spades):
       Straight Flush
     (7 of Diamonds, 6 of Diamonds, 5 of Diamonds, 4 of Diamonds, 3 of
       Diamonds), (7 of Spades, 6 of Spades, 5 of Spades, 4 of Spades, 3
       of Spades) Tie for the Win.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/
       ThreeOfAKind1.cfg
     (6 of Hearts, 6 of Diamonds, 6 of Spades, Q of Clubs, 4 of Spades):
       Three of a Kind
     (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of Spades):
       Three of a Kind
     (6 of Hearts, 6 of Diamonds, 6 of Spades, Q of Clubs, 4 of Spades)
       Wins. (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of
```

```
          Spades) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/ThreeOfAKind2.cfg
     (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of Spades): Three of a Kind
     (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Clubs, 7 of Hearts): Three of a Kind
     (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of Spades) Wins. (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Clubs, 7 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/ThreeOfAKind3.cfg
     (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Clubs, 7 of Hearts): Three of a Kind
     (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Spades, 5 of Diamonds): Three of a Kind
     (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Clubs, 7 of Hearts) Wins. (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Spades, 5 of Diamonds) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/ThreeOfAKind4.cfg
     (9 of Spades, 9 of Hearts, 9 of Diamonds, 10 of Diamonds, 8 of Hearts): Three of a Kind
     (9 of Clubs, 9 of Spades, 9 of Hearts, 10 of Diamonds, 8 of Diamonds): Three of a Kind
     (9 of Spades, 9 of Hearts, 9 of Diamonds, 10 of Diamonds, 8 of Hearts), (9 of Clubs, 9 of Spades, 9 of Hearts, 10 of Diamonds, 8 of Diamonds) Tie for the Win.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/ThreeWayTie.cfg
     (10 of Diamonds, 9 of Diamonds, 8 of Diamonds, 6 of Diamonds, 7 of Diamonds): Straight Flush
     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs): Straight Flush
     (10 of Hearts, 9 of Hearts, 8 of Hearts, 6 of Hearts, 7 of Hearts): Straight Flush
     (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts): Straight Flush
```

```
     (10 of Diamonds, 9 of Diamonds, 8 of Diamonds, 6 of Diamonds, 7 of Diamonds), (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs), (10 of Hearts, 9 of Hearts, 8 of Hearts, 6 of Hearts, 7 of Hearts) Tie for the Win. (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/TripleTest1.cfg
     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs): Straight Flush
     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts): Straight Flush
     (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts): Straight Flush
     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs) Wins. (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts) and (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts) Lose.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/TripleTest2.cfg
     (10 of Diamonds, 9 of Diamonds, 8 of Diamonds, 6 of Diamonds, 7 of Diamonds): Straight Flush
     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs): Straight Flush
     (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts): Straight Flush
     (10 of Diamonds, 9 of Diamonds, 8 of Diamonds, 6 of Diamonds, 7 of Diamonds), (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs) Tie for the Win. (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair1.cfg
     (10 of Diamonds, 10 of Spades, 2 of Spades, 2 of Clubs, K of Clubs): Two Pair
     (5 of Clubs, 5 of Spades, 4 of Diamonds, 4 of Hearts, 10 of Hearts): Two Pair
     (10 of Diamonds, 10 of Spades, 2 of Spades, 2 of Clubs, K of Clubs) Wins. (5 of Clubs, 5 of Spades, 4 of Diamonds, 4 of Hearts, 10 of Hearts) Loses.
Test Result: PASS

Output:
     C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair2.cfg
     (5 of Clubs, 5 of Spades, 4 of Diamonds, 4 of Hearts, 10 of Hearts):
```

```
        Two Pair
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades):
        Two Pair
    (5 of Clubs, 5 of Spades, 4 of Diamonds, 4 of Hearts, 10 of Hearts)
        Wins. (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of
        Spades) Loses.
Test Result: PASS

Output:
    C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair3.cfg
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades):
        Two Pair
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, J of Spades):
        Two Pair
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades)
        Wins. (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, J of
        Spades) Loses.
Test Result: PASS

Output:
    C:\Users\nater\source\repos\PokerHands\test\../Input/
        TwoPair3Flipped.cfg
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades):
        Two Pair
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, J of Spades):
        Two Pair
    (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades)
        Wins. (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, J of
        Spades) Loses.
Test Result: PASS

Output:
    C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair4.cfg
    (K of Diamonds, K of Spades, 7 of Diamonds, 7 of Hearts, 8 of
        Hearts): Two Pair
    (K of Clubs, K of Spades, 7 of Clubs, 7 of Hearts, 8 of Clubs): Two
        Pair
    (K of Diamonds, K of Spades, 7 of Diamonds, 7 of Hearts, 8 of
        Hearts), (K of Clubs, K of Spades, 7 of Clubs, 7 of Hearts, 8 of
        Clubs) Tie for the Win.
Test Result: PASS
```