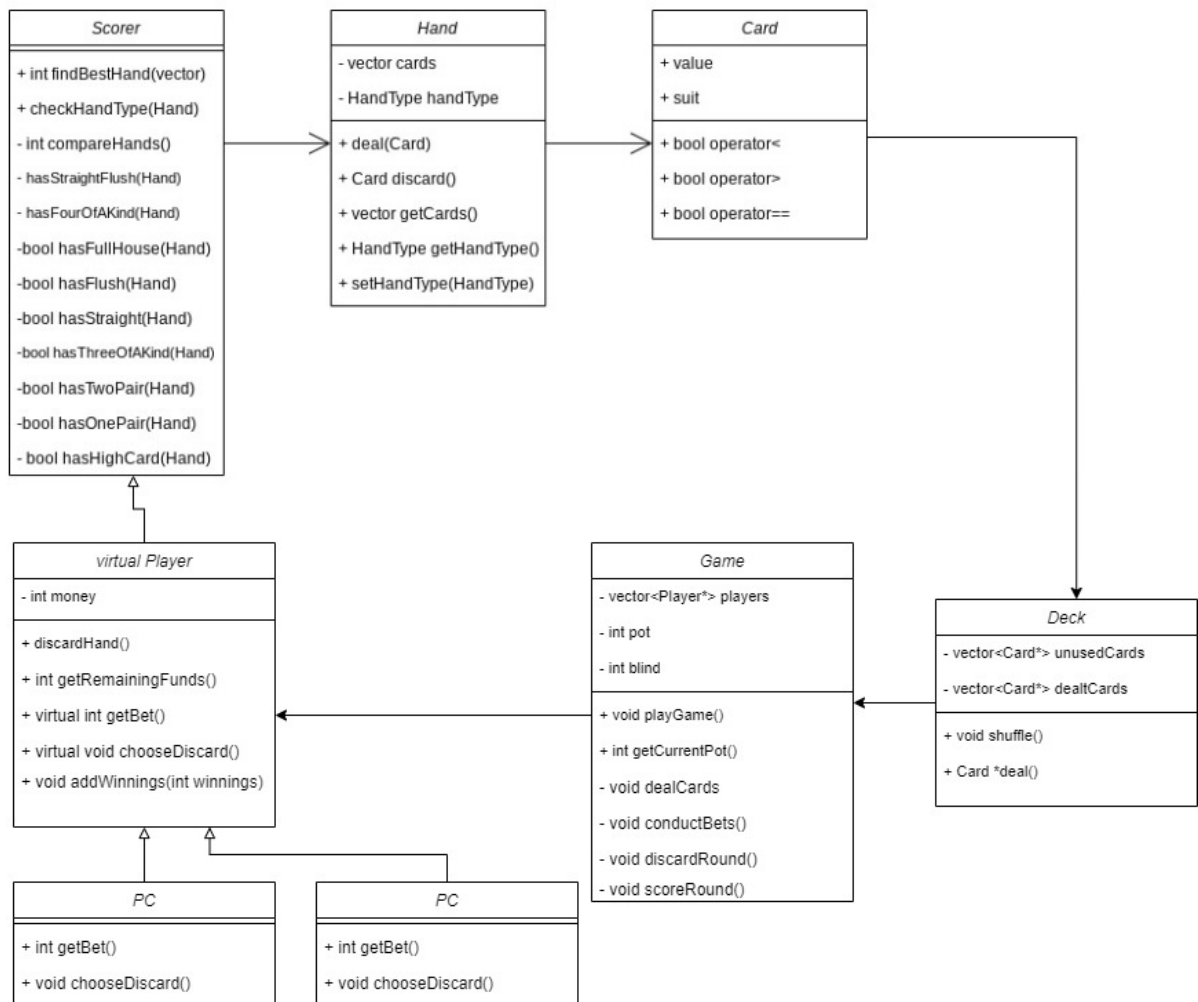


Module 12 Assignment: Poker Game

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



```
1  /// Run a 5-Card Draw Poker Game
2  #include <iostream>
3  #include "deck.h"
4  #include "game.h"
5
6  /// Run main game loop, with compile-time parameters
7  int main()
8  {
9      // Set number of Human/Computer Players
10     const int NUM_COMPUTERS = 2;
11     const int NUM_PLAYERS = 1;
12     // Set starting money amount
13     const int BUY_IN = 100;
14     // Set minimum bet values
15     const int BLIND = 3;
16
17     // Run Poker Game
18     Game *game = new Game(NUM_PLAYERS, NUM_COMPUTERS, BUY_IN, BLIND);
19     game->playGame();
20 } // End function main
21
```

```
1  /// @file
2  /// @author Nathan Roe
3  #pragma once
4  #include <stdexcept>
5  #include <string>
6  #include <vector>
7
8  class Player;
9  class Deck;
10 class Scorer;
11
12 using namespace std;
13
14 /// Game object for playing a game of 5-Card Draw Poker
15 ///
16 /// Main game loop is played using playGame(), and there
17 /// can be a total of 7 players, either PCs or NPCs
18 class Game
19 {
20 public:
21     /// Game constructor
22     /// @param numPlayers - number of Human players (PCs)
23     /// @param numComputers - number of Computer players (NPCs)
24     /// @param buyIn - starting money amount for each player
25     /// @param blind - Min starting bet for each round
26     Game(int numPlayers, int numComputers, int buyIn, int blind);
27
28     /// Error thrown when too many or too few players required
29     class InvalidGameException : public std::runtime_error
30     {
31     public:
32         InvalidGameException(int playerCount)
33             : std::runtime_error("NumPlayers must be between 2 and 7, "
34                                 + to_string(playerCount)
35                                 + " were given") {}
36     };
37
38     /// Start main game loop (Bet, Draw, Bet, Score)
39     ///
40     /// Plays a game of Poker on loop until only 1
41     /// player remains with money
42     void playGame();
43
44     /// Getter for current amount of money in the pot
45     /// Returns the amount of money in the pot
46     int getCurrentPot() { return this->pot; };
47
48 private:
49     const int MIN_PLAYERS = 2;
```

```
50     const int MAX_PLAYERS = 7;
51
52     enum class BetType
53     {
54         NONE,
55         FOLD,
56         CALL,
57         RAISE,
58         ALL_IN
59     };
60
61     vector<Player*> players;
62     vector<Player*> remainingPlayers;
63     vector<BetType> betTypes;
64     vector<int> amountIn;
65
66     Deck *deck;
67     Scorer *scorer;
68
69     int pot = 0;
70     int blind = 0;
71     int blindIdx = 0;
72
73     /// Deal 5 cards to each player
74     void dealCards();
75
76     /// Run a round of bets
77     ///
78     /// Runs betting until all players have called, folded,
79     /// or are all in
80     /// @param requireBlinds - indicates whether minimum
81     ///     bet values should be used in the round (True
82     ///     during the first round
83     void conductBets(bool requireBlinds);
84
85     /// Remove players from the round who have folded
86     void dropPlayers();
87
88     /// Handle round for redraws
89     void discardRound();
90
91     /// Score players who are still in, and distribute winnings
92     void scoreRound();
93 };
94
95
```

```
1 #include "game.h"
2 #include "deck.h"
3 #include "player.h"
4 #include "pc.h"
5 #include "npc.h"
6 #include "Scorer.h"
7 #include <iostream>
8
9 /// Game object for playing a game of 5-Card Draw Poker
10 Game::Game(int numHumans, int numComputers, int buyIn, int blind)
11 {
12     int numPlayers = numHumans + numComputers;
13     if (numPlayers < MIN_PLAYERS ||
14         numPlayers > MAX_PLAYERS)
15     {
16         throw InvalidGameException(numPlayers);
17     }
18
19     // Add PCs
20     for (int num = 0; num < numHumans; ++num)
21     {
22         Player *player = new PC(this, buyIn);
23         this->players.push_back(player);
24     }
25     // Add NPCs
26     for (int num = 0; num < numComputers; ++num)
27     {
28         Player *player = new NPC(this, buyIn);
29         this->players.push_back(player);
30     }
31
32     this->blind = blind;
33     this->scorer = new Scorer();
34     this->deck = new Deck();
35 } // End Game constructor
36
37 /// Start main game loop (Bet, Draw, Bet, Score)
38 void Game::playGame()
39 {
40     while (players.size() > 1)
41     {
42         // Setup Current Players
43         vector<Player*>::iterator iter = this->players.begin();
44         while (iter != this->players.end())
45         {
46             this->remainingPlayers.push_back(*iter++);
47         }
48         vector<BetType> startBet(this->remainingPlayers.size(),
                                BetType::NONE);
```

```
49     this->betTypes = startBet;
50     vector<int> startCash(this->remainingPlayers.size(), 0);
51     this->amountIn = startCash;
52
53     // Deal out initial cards
54     dealCards();
55
56     // Accept first round of bets
57     conductBets(true);
58     dropPlayers();
59
60     // Let players redraw
61     discardRound();
62
63     // Betting round, no blinds
64     vector<BetType> newBets(this->remainingPlayers.size(),      ↗
        BetType::NONE);
65     this->betTypes = newBets;
66     conductBets(false);
67     dropPlayers();
68
69     // Score round
70     scoreRound();
71
72     // Discard hands at end of round
73     for (Player *player : this->players)
74     {
75         player->discardHand();
76     }
77     this->remainingPlayers.clear();
78
79     // Remove Players who lost
80     iter = this->players.begin();
81     while (iter != this->players.end())
82     {
83         Player *p = *iter;
84         if (p->isBroke())
85         {
86             cout << "Player " << p << " is out of funds." << endl;
87             iter = this->players.erase(iter);
88         }
89         else
90         {
91             ++iter;
92         }
93     }
94
95     // Increment Blind position player
96     ++this->blindIdx;
```

```
97         this->blindIdx %= this->players.size();
98     }
99 } // End function playGame
100
101 // Remove players from the round who have folded
102 void Game::dropPlayers()
103 {
104     vector<Player*>::iterator iter = this->remainingPlayers.begin();
105     vector<BetType>::iterator betIter = this->betTypes.begin();
106     vector<int>::iterator cashIter = this->amountIn.begin();
107     while (iter != this->remainingPlayers.end())
108     {
109         BetType b = *betIter;
110         // Remove Players who folded
111         if (b == BetType::FOLD)
112         {
113             iter = this->remainingPlayers.erase(iter);
114             betIter = this->betTypes.erase(betIter);
115             cashIter = this->amountIn.erase(cashIter);
116         }
117         else
118         {
119             ++iter;
120             ++betIter;
121             ++cashIter;
122         }
123     }
124 } // End function dropPlayers
125
126 // Deal 5 cards to each player
127 void Game::dealCards()
128 {
129     this->deck->shuffle();
130     const int NUM_CARDS = 5;
131     for (int n = 0; n < NUM_CARDS; n++)
132     {
133         for (Player *player : this->players)
134         {
135             player->deal(this->deck->deal());
136         }
137     }
138 } // End function dealCards
139
140 // Run a round of bets
141 void Game::conductBets(bool requireBlinds)
142 {
143     // Exit if too few players
144     if (this->remainingPlayers.size() < 2)
145     {
```

```
146     return;
147 }
148
149 int playerId = this->blindIdx;
150 playerId %= this->remainingPlayers.size();
151 int bet = 0;
152 int baseBet = this->amountIn[playerId];
153 int amountToCall = 0;
154 Player *p;
155
156 if (requireBlinds)
157 {
158     // Handle Small Blind
159     p = this->remainingPlayers[playerId];
160     cout << "Player " << p << " is the Small Blind" << endl;
161     bet = p->getBet(0, this->blind);
162     cout << "Player " << p << " bet $" << bet << endl << endl;
163     pot += bet;
164     this->amountIn[playerId] += bet;
165     // Handle scenario when player can't afford small blind
166     if (bet < this->blind)
167     {
168         this->betTypes[playerId] = BetType::ALL_IN;
169     }
170     else
171     {
172         this->betTypes[playerId] = BetType::RAISE;
173     }
174     ++playerId;
175     playerId %= this->remainingPlayers.size();
176     baseBet = bet;
177
178     // Handle Big Blind
179     p = this->remainingPlayers[playerId];
180     cout << "Player " << p << " is the Big Blind" << endl;
181     amountToCall = baseBet - this->amountIn[playerId];
182     bet = p->getBet(amountToCall, 2 * this->blind);
183     cout << "Player " << p << " bet $" << bet << endl << endl;
184     pot += bet;
185     if (bet >= amountToCall)
186     {
187         baseBet += (bet - amountToCall);
188     }
189     this->amountIn[playerId] += bet;
190     // Handle scenario when player can't afford big blind
191     if (bet < 2 * this->blind)
192     {
193         this->betTypes[playerId] = BetType::ALL_IN;
194     }
```



```
195     else
196     {
197         this->betTypes[playerIdx] = BetType::RAISE;
198     }
199     ++playerIdx;
200     playerIdx %= this->remainingPlayers.size();
201 }
202
203 // Main betting loop
204 while (true)
205 {
206     // Check whether player needs to bet
207     if (!(this->betTypes[playerIdx] == BetType::FOLD ||
208         this->betTypes[playerIdx] == BetType::ALL_IN))
209     {
210         p = this->remainingPlayers[playerIdx];
211         // Get required bet amount to avoid folding
212         amountToCall = baseBet - this->amountIn[playerIdx];
213         bet = p->getBet(amountToCall);
214         pot += bet;
215         // See if bet is a raise or a call
216         if (bet >= amountToCall)
217         {
218             baseBet += (bet - amountToCall);
219         }
220         this->amountIn[playerIdx] += bet;
221         // Handle Calls
222         if (bet == amountToCall)
223         {
224             this->betTypes[playerIdx] = BetType::CALL;
225             cout << "Player " << p << " Calls" << endl << endl;
226         }
227         // Handle All In
228         else if (p->getRemainingFunds() == 0)
229         {
230             this->betTypes[playerIdx] = BetType::ALL_IN;
231             cout << "Player " << p << " is All In" << endl << endl;
232         }
233         // Handle Folds
234         else if (bet == 0)
235         {
236             this->betTypes[playerIdx] = BetType::FOLD;
237             cout << "Player " << p << " Folds" << endl << endl;
238         }
239         // Other bets are raises
240         else
241         {
242             this->betTypes[playerIdx] = BetType::RAISE;
243             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 available 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" << "\n";
        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
```

```
291 void Game::scoreRound()
292 {
293     int winIdx = -1;
294     vector<Hand*> temp(this->remainingPlayers.begin(), this-
295         >remainingPlayers.end());
296     // Print player hands
297     for (Player *p : this->remainingPlayers)
298     {
299         cout << "Player " << p << "'s hand: " << p->printCards() << endl;
300     }
301     // Find number of winning hands, and bring winners to front
302     if (this->remainingPlayers.size() > 1)
303     {
304         winIdx = this->scorer->findBestHand(&temp);
305     }
306     // If only one player is left, give pot to player
307     if (this->remainingPlayers.size() == 1)
308     {
309         int idx = 0;
310         Player* p = static_cast<Player*>(temp[idx]);
311         p->addWinnings(this->pot);
312         cout << "Player " << p << " Wins $" << this->pot << endl << endl;
313         this->pot = 0;
314     }
315     // If multiple players left, distribute winnings amongst winners
316     else if (winIdx >= 0)
317     {
318         int winnings = this->pot / (winIdx + 1);
319         for (int idx = 0; idx <= winIdx; ++idx)
320         {
321             Player* p = static_cast<Player*>(temp[idx]);
322             p->addWinnings(winnings);
323             this->pot -= winnings;
324             cout << "Player " << p << " Wins $" << winnings << endl <<
325                 endl;
326         }
327     }
328 } // End function scoreRound
329
```

```
1  /// @file
2  /// @author Nathan Roe
3  #pragma once
4  #include <vector>
5  #include <exception>
6  #include "card.h"
7
8  using namespace std;
9
10 // Deck object for responsible for tracking all cards
11 //
12 // Allows for shuffling and dealing of cards. Dealing
13 // from an empty deck throws and error.
14 class Deck
15 {
16 public:
17     /// Deck Constructor
18     Deck();
19
20     /// Error thrown when dealing from empty deck
21     class EmptyDeck : public std::exception
22     {
23     public:
24         virtual const char* what() const throw()
25         {
26             return "Cannot deal, deck is empty";
27         }
28     };
29
30     /// Shuffle deck; returns previously dealt cards to deck
31     void shuffle();
32
33     /// Deal a card
34     ///
35     /// Returns a pointer to the dealt card
36     Card *deal();
37
38     /// Getter for cards left in deck
39     /// Return the number of cards remaining in the deck
40     int cardsRemaining() { return this->unusedCards.size(); };
41
42 private:
43     vector<Card> unusedCards;
44     vector<Card> dealtCards;
45 };
46
```

```
1  #include "deck.h"
2  #include "card.h"
3  #include <array>
4  #include <list>
5  #include <algorithm>
6  #include <stdlib.h>
7
8  using namespace std;
9
10 // Deck object for responsible for tracking all cards
11 Deck::Deck()
12 {
13     const int RAND_SEED = 7;
14     srand(RAND_SEED);
15
16     array<Card::Value, 13> values = { Card::Value::Two, Card::Value::Three, ↵
17                                     Card::Value::Four,
18                                     Card::Value::Five, Card::Value::Six, ↵
19                                     Card::Value::Seven,
20                                     Card::Value::Eight, ↵
21                                     Card::Value::Nine, Card::Value::Ten, ↵
22                                     Card::Value::Jack, ↵
23                                     Card::Value::Queen, Card::Value::King,
24                                     Card::Value::Ace };
25     array<Card::Suit, 4> suits = { Card::Suit::Clubs, Card::Suit::Diamonds, ↵
26                                  Card::Suit::Hearts, ↵
27                                  Card::Suit::Spades };
28
29     // Create one card of each value/suit combination
30     for (Card::Suit suit : suits)
31     {
32         for (Card::Value value : values)
33         {
34             Card *card = new Card(value, suit);
35             this->unusedCards.push_back(*card);
36         }
37     }
38 } // End Deck constructor
39
40 // Shuffle deck; returns previously dealt cards to deck
41 void Deck::shuffle()
42 {
43     // Return dealt cards to deck
44     while (!this->dealtCards.empty())
45     {
46         this->unusedCards.push_back(this->dealtCards.back());
47         this->dealtCards.pop_back();
48     }
49     sort(this->unusedCards.begin(), this->unusedCards.end());
```

```
45
46     // Shuffle using Fisher-Yates (https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle)
47     for (unsigned int idx = 0; idx < this->unusedCards.size(); ++idx)
48     {
49         int swapIdx = rand() % this->unusedCards.size();
50         swap(this->unusedCards[idx], this->unusedCards[swapIdx]);
51     }
52 } // End function shuffle
53
54 // Deal a card
55 Card *Deck::deal()
56 {
57     if (!this->unusedCards.empty())
58     {
59         // Shift card from unused to dealt
60         this->dealtCards.push_back(this->unusedCards.back());
61         this->unusedCards.pop_back();
62         // Return pointer to dealt card
63         return &this->dealtCards.back();
64     }
65     // Throw error if deck is empty
66     else
67     {
68         throw EmptyDeck();
69     }
70 } // End function deal
71
```

```
1  /// @file
2  /// @author Nathan Roe
3  #pragma once
4  #include "hand.h"
5
6  class Game;
7
8  /// Abstract Player class representing Poker player
9  ///
10 /// Handles money, betting and discards for 5-Card
11 /// draw poker. Card information inherited from Hand
12 class Player : public Hand
13 {
14 public:
15     /// Player constructor
16     /// @param *game - Pointer to poker Game object
17     /// @param buyIn - Starting money value
18     Player(Game *game, int buyIn);
19
20     /// Player destructor
21     virtual ~Player();
22
23     /// Remove all cards from hand
24     void discardHand();
25
26     /// Check whether player has funds
27     /// Returns True if player has no money, false otherwise
28     bool isBroke() { return this->money <= 0; };
29
30     /// Getter for amount of money remaining
31     /// Returns an int of remaining money
32     int getRemainingFunds() { return this->money; };
33
34     /// Get a bet for a betting round in Poker
35     /// @param prevRaise - Last raise value, min for calling
36     /// @param minBet - Minimum allowed bet, used for blinds
37     /// Returns an int of the Player's bet
38     virtual int getBet(int prevRaise, int minBet = 0) = 0;
39
40     /// Adds Winnings to the Player's pool of funds
41     /// @param winnings - amount of money won in round
42     void addWinnings(int winnings) { this->money += winnings; };
43
44     /// Choose which cards to discard for draw phase of 5-Card draw
45     /// @param maxDiscard - Maximum cards that are available to draw
46     /// Returns the number of cards discarded
47     virtual int chooseDiscard(int maxDiscard) = 0;
48
49 protected:
```

```
50     Game *game;  
51     int money = 0;  
52 };  
53  
54
```



```
1 #include "player.h"
2 #include "card.h"
3
4 /// Abstract Player class representing Poker player
5 Player::Player(Game *game, int buyIn)
6 {
7     this->game = game;
8     this->money = buyIn;
9 } // End Player constructor
10
11 /// Default Player destructor
12 Player::~Player()
13 {
14     return;
15 } // End Player destructor
16
17 /// Discard all cards in hand
18 void Player::discardHand()
19 {
20     while (!this->cards.empty())
21     {
22         const Card card = this->cards.front();
23         discard(&card);
24     }
25 } // End function discardHand
26
```

```
1  /// @file
2  /// @author Nathan Roe
3  #pragma once
4  #include "player.h"
5
6  class Game;
7
8  /// Player class representing Human Poker player
9  class PC : public Player
10 {
11 public:
12     /// Human Player constructor
13     /// @param *game - Pointer to poker Game object
14     /// @param buyIn - Starting money value
15     PC(Game *game, int buyIn) : Player(game, buyIn) {}
16
17     /// Get a bet for a betting round in Poker
18     /// @param prevRaise - Last raise value, min for calling
19     /// @param minBet - Minimum allowed bet, used for blinds
20     /// Returns an int of the Player's bet
21     int getBet(int prevRaise, int minBet = 0);
22
23     /// Choose which cards to discard for draw phase of 5-Card draw
24     /// @param maxDiscard - Maximum cards that are available to draw
25     /// Returns the number of cards discarded
26     int chooseDiscard(int maxDiscard);
27 };
28
```





```
1  #include "pc.h"
2  #include <iostream>
3
4  using namespace std;
5
6  // Get a bet for a betting round in Poker
7  int PC::getBet(int prevRaise, int minBet)
8  {
9      int bet = 0;
10     cout << "Player " << this << "'s Turn" << endl;
11     cout << "Your Cards: " << this->printCards() << endl;
12     while (true)
13     {
14         if (minBet > 0)
15         {
16             cout << "Minimum Bet is $" << minBet << endl;
17         }
18         cout << "The previous raise was $" << prevRaise << endl;
19         cout << "You have $" << this->money << endl;
20         cout << "Enter your bet, or $0 to fold: ";
21         cin >> bet;
22
23         // Verify bet is a valid bet when a required bet is needed
24         if (bet >= minBet && bet >= prevRaise && bet <= this->money)
25         {
26             break;
27         }
28         // Accept All In bets
29         else if (bet == this->money)
30         {
31             break;
32         }
33         // Accept fold/calls when no min bet is needed
34         else if (bet == 0 && minBet == 0)
35         {
36             break;
37         }
38         // Catch error for betting unavailable money
39         else if (bet > this->money)
40         {
41             cout << "Please bet an amount of $" << this->money << " or
42                 less" << endl;
43         }
44         // Catch error for betting under required value
45         else if (bet < minBet)
46         {
47             cout << "Please bet at least $" << minBet << ", or go All In" <<
48                 endl;
```

```
48     // Catch error for not Calling or Folding
49     else
50     {
51         cout << "Bet at least $" << prevRaise << " to Call/Raise" << endl;
52     }
53 }
54
55 this->money -= bet;
56 return bet;
57 } // End function getBet
58
59 // Choose which cards to discard for draw phase of 5-Card draw
60 int PC::chooseDiscard(int maxDiscard)
61 {
62     // Find max possible discards
63     if (maxDiscard > this->cards.size())
64     {
65         maxDiscard = this->cards.size();
66     }
67     cout << "Player " << this << "'s Turn to Discard" << endl;
68     cout << "You may discard up to " << maxDiscard << " cards." << endl;
69
70     int cardSelection = 0;
71     int numDiscarded = 0;
72     // Discard up to max cards, or until player exits loop
73     while (numDiscarded < maxDiscard)
74     {
75         cout << this->printCards() << endl;
76         cout << "Choose a card to discard (1-" << this->cards.size() << "), or 0 to Stop: ";
77         cin >> cardSelection;
78         // Exit Loop condition
79         if (cardSelection == 0)
80         {
81             break;
82         }
83         // Player selects valid card for discard
84         if (cardSelection > 0 && cardSelection <= this->cards.size())
85         {
86             // Discard indexed from 1-N, not by 0
87             this->cards.erase(this->cards.begin() + cardSelection - 1);
88             ++numDiscarded;
89         }
90     }
91     return numDiscarded;
92 } // End function chooseDiscard
93
```

```
1  /// @file
2  /// @author Nathan Roe
3  #pragma once
4  #include "player.h"
5
6  class Game;
7
8  /// Player class representing Computer Poker player
9  class NPC : public Player
10 {
11 public:
12     /// Computer Player constructor
13     /// @param *game - Pointer to poker Game object
14     /// @param buIn - Starting money value
15     NPC(Game *game, int buyIn) : Player(game, buyIn) {};
16
17     /// Get a bet for a betting round in Poker
18     /// @param prevRaise - Last raise value, min for calling
19     /// @param minBet - Minimum allowed bet, used for blinds
20     /// Returns an int of the Player's bet
21     int getBet(int prevRaise, int minBet = 0);
22
23     /// Choose which cards to discard for draw phase of 5-Card draw
24     /// @param maxDiscard - Maximum cards that are available to draw
25     /// Returns the number of cards discarded
26     int chooseDiscard(int maxDiscard);
27 };
28
```

```
1 #include "npc.h"
2 #include <algorithm>
3
4 // Get a bet for a betting round in Poker
5 int NPC::getBet(int prevRaise, int minBet)
6 {
7     // If required bet, bet minimum amount
8     if (minBet > 0)
9     {
10         // Bet required value
11         if (this->money > minBet)
12         {
13             this->money -= minBet;
14             return minBet;
15         }
16         // Bet remaining money if less than required
17         else
18         {
19             int prevFunds = this->money;
20             this->money = 0;
21             return prevFunds;
22         }
23     }
24
25     // Call if funds are available
26     if (prevRaise < this->money)
27     {
28         this->money -= prevRaise;
29         return prevRaise;
30     }
31     // Otherwise, fold
32     else
33     {
34         return 0;
35     }
36 } // End function getBet
37
38 // Choose which cards to discard for draw phase of 5-Card draw
39 int NPC::chooseDiscard(int maxDiscard)
40 {
41     if (maxDiscard > this->cards.size())
42     {
43         maxDiscard = this->cards.size();
44     }
45     int numDiscard = rand() % maxDiscard;
46     // Sort cards High to Low
47     sort(this->cards.begin(), this->cards.end());
48     // Randomly select some number of cards to discard
49     for (int count = 0; count < numDiscard; ++count)
```

```
50     {  
51         // Discard lowest n cards  
52         this->cards.erase(this->cards.end() - 1);  
53     }  
54     return numDiscard;  
55 } // End function chooseDiscard  
56
```

1 Player 000001476AC3F8E0 is the Small Blind
2 Player 000001476AC3F8E0's Turn
3 Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of 
Diamonds
4 Minimum Bet is \$3
5 The previous raise was \$0
6 You have \$100
7 Enter your bet, or \$0 to fold: 3
8 Player 000001476AC3F8E0 bet \$3
9
10 Player 000001476AC39E40 is the Big Blind
11 Player 000001476AC39E40 bet \$6
12
13 Player 000001476AC37E50 Calls
14
15 Player 000001476AC3F8E0's Turn
16 Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of 
Diamonds
17 The previous raise was \$3
18 You have \$97
19 Enter your bet, or \$0 to fold: 6
20 Player 000001476AC3F8E0 Raises \$3
21
22 Player 000001476AC39E40 Calls
23
24 Player 000001476AC37E50 Calls
25
26 Player 000001476AC3F8E0's Turn
27 Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of 
Diamonds
28 The previous raise was \$0
29 You have \$91
30 Enter your bet, or \$0 to fold: 5
31 Player 000001476AC3F8E0 Raises \$5
32
33 Player 000001476AC39E40 Calls
34
35 Player 000001476AC37E50 Calls
36
37 Player 000001476AC3F8E0's Turn
38 Your Cards: 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of 
Diamonds
39 The previous raise was \$0
40 You have \$86
41 Enter your bet, or \$0 to fold: 0
42 Player 000001476AC3F8E0 Calls
43
44 Player 000001476AC3F8E0's Turn to Discard
45 You may discard up to 5 cards.

46 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts, 3 of Diamonds
47 Choose a card to discard (1-5), or 0 to Stop: 5
48 8 of Hearts, 7 of Spades, J of Clubs, K of Hearts
49 Choose a card to discard (1-4), or 0 to Stop: 1
50 7 of Spades, J of Clubs, K of Hearts
51 Choose a card to discard (1-3), or 0 to Stop: 1
52 J of Clubs, K of Hearts
53 Choose a card to discard (1-2), or 0 to Stop: 0
54 Player 000001476AC3F8E0 discarded 3 cards
55
56 Player 000001476AC39E40 discarded 0 cards
57
58 Player 000001476AC37E50 discarded 4 cards
59
60 Player 000001476AC3F8E0's Turn
61 Your Cards: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of [↗](#)
Diamonds
62 The previous raise was \$0
63 You have \$86
64 Enter your bet, or \$0 to fold: 10
65 Player 000001476AC3F8E0 Raises \$10
66
67 Player 000001476AC39E40 Calls
68
69 Player 000001476AC37E50 Calls
70
71 Player 000001476AC3F8E0's Turn
72 Your Cards: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of [↗](#)
Diamonds
73 The previous raise was \$0
74 You have \$76
75 Enter your bet, or \$0 to fold: 5
76 Player 000001476AC3F8E0 Raises \$5
77
78 Player 000001476AC39E40 Calls
79
80 Player 000001476AC37E50 Calls
81
82 Player 000001476AC3F8E0's Turn
83 Your Cards: J of Clubs, K of Hearts, J of Spades, K of Diamonds, J of [↗](#)
Diamonds
84 The previous raise was \$0
85 You have \$71
86 Enter your bet, or \$0 to fold: 0
87 Player 000001476AC3F8E0 Calls
88
89 Player 000001476AC3F8E0's hand: J of Clubs, K of Hearts, J of Spades, K of [↗](#)
Diamonds, J of Diamonds
90 Player 000001476AC39E40's hand: 4 of Clubs, 5 of Spades, 7 of Diamonds, Q [↗](#)





of Spades, Q of Hearts
91 Player 000001476AC37E50's hand: 2 of Spades, J of Hearts, 9 of Spades, 7 of Clubs, 2 of Diamonds
92 Player 000001476AC3F8E0 Wins \$87
93
94 Player 000001476AC39E40 is the Small Blind
95 Player 000001476AC39E40 bet \$3
96
97 Player 000001476AC37E50 is the Big Blind
98 Player 000001476AC37E50 bet \$6
99
100 Player 000001476AC3F8E0's Turn
101 Your Cards: J of Diamonds, 8 of Hearts, J of Clubs, 9 of Hearts, 10 of Clubs
102 The previous raise was \$6
103 You have \$158
104 Enter your bet, or \$0 to fold: 6
105 Player 000001476AC3F8E0 Calls
106
107 Player 000001476AC39E40 Calls
108
109 Player 000001476AC37E50 Calls
110
111 Player 000001476AC39E40 discarded 4 cards
112
113 Player 000001476AC37E50 discarded 3 cards
114
115 Player 000001476AC3F8E0's Turn to Discard
116 You may discard up to 5 cards.
117 J of Diamonds, 8 of Hearts, J of Clubs, 9 of Hearts, 10 of Clubs
118 Choose a card to discard (1-5), or 0 to Stop: 2
119 J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs
120 Choose a card to discard (1-4), or 0 to Stop: 0
121 Player 000001476AC3F8E0 discarded 1 cards
122
123 Player 000001476AC39E40 Calls
124
125 Player 000001476AC37E50 Calls
126
127 Player 000001476AC3F8E0's Turn
128 Your Cards: J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs, 6 of Diamonds
129 The previous raise was \$0
130 You have \$152
131 Enter your bet, or \$0 to fold: 5
132 Player 000001476AC3F8E0 Raises \$5
133
134 Player 000001476AC39E40 Calls
135




136 Player 000001476AC37E50 Calls
137
138 Player 000001476AC3F8E0's Turn
139 Your Cards: J of Diamonds, J of Clubs, 9 of Hearts, 10 of Clubs, 6 of [↗](#)
Diamonds
140 The previous raise was \$0
141 You have \$147
142 Enter your bet, or \$0 to fold: 0
143 Player 000001476AC3F8E0 Calls
144
145 Player 000001476AC3F8E0's hand: J of Diamonds, J of Clubs, 9 of Hearts, 10 [↗](#)
of Clubs, 6 of Diamonds
146 Player 000001476AC39E40's hand: 4 of Hearts, A of Spades, K of Diamonds, 4 [↗](#)
of Diamonds, 3 of Clubs
147 Player 000001476AC37E50's hand: 7 of Hearts, 8 of Diamonds, K of Clubs, A [↗](#)
of Clubs, J of Spades
148 Player 000001476AC3F8E0 Wins \$33
149
150 Player 000001476AC37E50 is the Small Blind
151 Player 000001476AC37E50 bet \$3
152
153 Player 000001476AC3F8E0 is the Big Blind
154 Player 000001476AC3F8E0's Turn
155 Your Cards: Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of [↗](#)
Spades
156 Minimum Bet is \$6
157 The previous raise was \$3
158 You have \$180
159 Enter your bet, or \$0 to fold: 6
160 Player 000001476AC3F8E0 bet \$6
161
162 Player 000001476AC39E40 Calls
163
164 Player 000001476AC37E50 Calls
165
166 Player 000001476AC3F8E0's Turn
167 Your Cards: Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of [↗](#)
Spades
168 The previous raise was \$0
169 You have \$174
170 Enter your bet, or \$0 to fold: 3
171 Player 000001476AC3F8E0 Raises \$3
172
173 Player 000001476AC39E40 Calls
174
175 Player 000001476AC37E50 Calls
176
177 Player 000001476AC3F8E0's Turn
178 Your Cards: Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of [↗](#)

Spades

179 The previous raise was \$0
180 You have \$171
181 Enter your bet, or \$0 to fold: 0
182 Player 000001476AC3F8E0 Calls
183
184 Player 000001476AC37E50 discarded 1 cards
185
186 Player 000001476AC3F8E0's Turn to Discard
187 You may discard up to 5 cards.
188 Q of Spades, 3 of Hearts, 10 of Spades, 5 of Spades, 7 of Spades
189 Choose a card to discard (1-5), or 0 to Stop: 2
190 Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades
191 Choose a card to discard (1-4), or 0 to Stop: 0
192 Player 000001476AC3F8E0 discarded 1 cards
193
194 Player 000001476AC39E40 discarded 1 cards
195
196 Player 000001476AC37E50 Calls
197
198 Player 000001476AC3F8E0's Turn
199 Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades [↗](#)
200 The previous raise was \$0
201 You have \$171
202 Enter your bet, or \$0 to fold: 10
203 Player 000001476AC3F8E0 Raises \$10
204
205 Player 000001476AC39E40 Calls
206
207 Player 000001476AC37E50 Calls
208
209 Player 000001476AC3F8E0's Turn
210 Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades [↗](#)
211 The previous raise was \$0
212 You have \$161
213 Enter your bet, or \$0 to fold: 10
214 Player 000001476AC3F8E0 Raises \$10
215
216 Player 000001476AC39E40 Calls
217
218 Player 000001476AC37E50 Calls
219
220 Player 000001476AC3F8E0's Turn
221 Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades [↗](#)
222 The previous raise was \$0
223 You have \$151

224 Enter your bet, or \$0 to fold: 10
225 Player 000001476AC3F8E0 Raises \$10
226
227 Player 000001476AC39E40 Calls
228
229 Player 000001476AC37E50 Calls
230
231 Player 000001476AC3F8E0's Turn
232 Your Cards: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades [↗](#)
233 The previous raise was \$0
234 You have \$141
235 Enter your bet, or \$0 to fold: 0
236 Player 000001476AC3F8E0 Calls
237
238 Player 000001476AC3F8E0's hand: Q of Spades, 10 of Spades, 5 of Spades, 7 of Spades, J of Spades [↗](#)
239 Player 000001476AC39E40's hand: 2 of Clubs, 4 of Diamonds, 6 of Spades, J of Clubs, A of Hearts [↗](#)
240 Player 000001476AC37E50's hand: 2 of Hearts, 4 of Spades, 7 of Hearts, J of Hearts, K of Hearts [↗](#)
241 Player 000001476AC3F8E0 Wins \$117
242
243 Player 000001476AC3F8E0 is the Small Blind
244 Player 000001476AC3F8E0's Turn
245 Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds [↗](#)
246 Minimum Bet is \$3
247 The previous raise was \$0
248 You have \$258
249 Enter your bet, or \$0 to fold: 3
250 Player 000001476AC3F8E0 bet \$3
251
252 Player 000001476AC39E40 is the Big Blind
253 Player 000001476AC39E40 bet \$6
254
255 Player 000001476AC37E50 Calls
256
257 Player 000001476AC3F8E0's Turn
258 Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds [↗](#)
259 The previous raise was \$3
260 You have \$255
261 Enter your bet, or \$0 to fold: 3
262 Player 000001476AC3F8E0 Calls
263
264 Player 000001476AC39E40 Calls
265
266 Player 000001476AC3F8E0's Turn to Discard

267 You may discard up to 5 cards.
268 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds
269 Choose a card to discard (1-5), or 0 to Stop: 0
270 Player 000001476AC3F8E0 discarded 0 cards
271
272 Player 000001476AC39E40 discarded 3 cards
273
274 Player 000001476AC37E50 discarded 4 cards
275
276 Player 000001476AC3F8E0's Turn
277 Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds 
278 The previous raise was \$0
279 You have \$252
280 Enter your bet, or \$0 to fold: 8
281 Player 000001476AC3F8E0 Raises \$8
282
283 Player 000001476AC39E40 Calls
284
285 Player 000001476AC37E50 Calls
286
287 Player 000001476AC3F8E0's Turn
288 Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds 
289 The previous raise was \$0
290 You have \$244
291 Enter your bet, or \$0 to fold: 8
292 Player 000001476AC3F8E0 Raises \$8
293
294 Player 000001476AC39E40 Folds
295
296 Player 000001476AC37E50 Folds
297
298 Player 000001476AC3F8E0's Turn
299 Your Cards: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds 
300 The previous raise was \$0
301 You have \$236
302 Enter your bet, or \$0 to fold: 0
303 Player 000001476AC3F8E0 Calls
304
305 Player 000001476AC3F8E0's hand: 9 of Spades, A of Hearts, 4 of Spades, 9 of Diamonds, 8 of Diamonds 
306 Player 000001476AC3F8E0 Wins \$0
307
308 Player 000001476AC39E40 is the Small Blind
309 Player 000001476AC39E40 bet \$3
310
311 Player 000001476AC37E50 is the Big Blind

312 Player 000001476AC37E50 bet \$6
313
314 Player 000001476AC3F8E0's Turn
315 Your Cards: 2 of Diamonds, J of Clubs, K of Hearts, 5 of Hearts, 7 of 
Diamonds
316 The previous raise was \$6
317 You have \$286
318 Enter your bet, or \$0 to fold: 6
319 Player 000001476AC3F8E0 Calls
320
321 Player 000001476AC39E40 Calls
322
323 Player 000001476AC37E50 Calls
324
325 Player 000001476AC39E40 discarded 4 cards
326
327 Player 000001476AC37E50 discarded 2 cards
328
329 Player 000001476AC3F8E0's Turn to Discard
330 You may discard up to 5 cards.
331 2 of Diamonds, J of Clubs, K of Hearts, 5 of Hearts, 7 of Diamonds
332 Choose a card to discard (1-5), or 0 to Stop: 1
333 J of Clubs, K of Hearts, 5 of Hearts, 7 of Diamonds
334 Choose a card to discard (1-4), or 0 to Stop: 4
335 J of Clubs, K of Hearts, 5 of Hearts
336 Choose a card to discard (1-3), or 0 to Stop: 3
337 J of Clubs, K of Hearts
338 Choose a card to discard (1-2), or 0 to Stop: 0
339 Player 000001476AC3F8E0 discarded 3 cards
340
341 Player 000001476AC39E40 Calls
342
343 Player 000001476AC37E50 Calls
344
345 Player 000001476AC3F8E0's Turn
346 Your Cards: J of Clubs, K of Hearts, K of Clubs, 4 of Diamonds, Q of 
Diamonds
347 The previous raise was \$0
348 You have \$280
349 Enter your bet, or \$0 to fold: 5
350 Player 000001476AC3F8E0 Raises \$5
351
352 Player 000001476AC39E40 Folds
353
354 Player 000001476AC37E50 Folds
355
356 Player 000001476AC3F8E0's Turn
357 Your Cards: J of Clubs, K of Hearts, K of Clubs, 4 of Diamonds, Q of 
Diamonds

358 The previous raise was \$0
359 You have \$275
360 Enter your bet, or \$0 to fold: 0
361 Player 000001476AC3F8E0 Calls
362
363 Player 000001476AC3F8E0's hand: J of Clubs, K of Hearts, K of Clubs, 4 of [↗](#)
Diamonds, Q of Diamonds
364 Player 000001476AC3F8E0 Wins \$23
365
366 Player 000001476AC37E50 is the Small Blind
367 Player 000001476AC37E50 bet \$1
368
369 Player 000001476AC3F8E0 is the Big Blind
370 Player 000001476AC3F8E0's Turn
371 Your Cards: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of [↗](#)
Hearts
372 Minimum Bet is \$6
373 The previous raise was \$1
374 You have \$298
375 Enter your bet, or \$0 to fold: 6
376 Player 000001476AC3F8E0 bet \$6
377
378 Player 000001476AC39E40 Folds
379
380 Player 000001476AC3F8E0's Turn
381 Your Cards: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of [↗](#)
Hearts
382 The previous raise was \$0
383 You have \$292
384 Enter your bet, or \$0 to fold: 0
385 Player 000001476AC3F8E0 Calls
386
387 Player 000001476AC3F8E0's Turn to Discard
388 You may discard up to 5 cards.
389 A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
390 Choose a card to discard (1-5), or 0 to Stop: 0
391 Player 000001476AC3F8E0 discarded 0 cards
392
393 Player 000001476AC37E50 discarded 2 cards
394
395 Player 000001476AC3F8E0's Turn
396 Your Cards: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of [↗](#)
Hearts
397 The previous raise was \$0
398 You have \$292
399 Enter your bet, or \$0 to fold: 0
400 Player 000001476AC3F8E0 Calls
401
402 Player 000001476AC37E50 is All In

403
404 Player 000001476AC3F8E0's hand: A of Clubs, 2 of Clubs, J of Diamonds, 2 of Diamonds, 9 of Hearts
405 Player 000001476AC37E50's hand: 5 of Diamonds, 5 of Clubs, 10 of Hearts, A of Hearts, Q of Diamonds
406 Player 000001476AC37E50 Wins \$7
407
408 Player 000001476AC3F8E0 is the Small Blind
409 Player 000001476AC3F8E0's Turn
410 Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 2 of Hearts, 6 of Diamonds
411 Minimum Bet is \$3
412 The previous raise was \$0
413 You have \$292
414 Enter your bet, or \$0 to fold: 3
415 Player 000001476AC3F8E0 bet \$3
416
417 Player 000001476AC39E40 is the Big Blind
418 Player 000001476AC39E40 bet \$1
419
420 Player 000001476AC37E50 Calls
421
422 Player 000001476AC3F8E0's Turn
423 Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 2 of Hearts, 6 of Diamonds
424 The previous raise was \$0
425 You have \$289
426 Enter your bet, or \$0 to fold: 0
427 Player 000001476AC3F8E0 Calls
428
429 Player 000001476AC3F8E0's Turn to Discard
430 You may discard up to 5 cards.
431 8 of Clubs, 10 of Hearts, A of Clubs, 2 of Hearts, 6 of Diamonds
432 Choose a card to discard (1-5), or 0 to Stop: 4
433 8 of Clubs, 10 of Hearts, A of Clubs, 6 of Diamonds
434 Choose a card to discard (1-4), or 0 to Stop: 4
435 8 of Clubs, 10 of Hearts, A of Clubs
436 Choose a card to discard (1-3), or 0 to Stop: 0
437 Player 000001476AC3F8E0 discarded 2 cards
438
439 Player 000001476AC39E40 discarded 4 cards
440
441 Player 000001476AC37E50 discarded 4 cards
442
443 Player 000001476AC3F8E0's Turn
444 Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 9 of Spades, 3 of Hearts
445 The previous raise was \$0
446 You have \$289
447 Enter your bet, or \$0 to fold: 2

448 Player 000001476AC3F8E0 Raises \$2
449
450 Player 000001476AC39E40 is All In
451
452 Player 000001476AC37E50 Calls
453
454 Player 000001476AC3F8E0's Turn
455 Your Cards: 8 of Clubs, 10 of Hearts, A of Clubs, 9 of Spades, 3 of Hearts
456 The previous raise was \$0
457 You have \$287
458 Enter your bet, or \$0 to fold: 0
459 Player 000001476AC3F8E0 Calls
460
461 Player 000001476AC3F8E0's hand: 8 of Clubs, 10 of Hearts, A of Clubs, 9 of ♠
Spades, 3 of Hearts
462 Player 000001476AC39E40's hand: 2 of Spades, 7 of Hearts, 7 of Diamonds, 9 ♠
of Hearts, Q of Diamonds
463 Player 000001476AC37E50's hand: 5 of Clubs, 6 of Hearts, 10 of Diamonds, Q ♠
of Spades, 8 of Spades
464 Player 000001476AC39E40 Wins \$11
465
466 Player 000001476AC39E40 is the Small Blind
467 Player 000001476AC39E40 bet \$3
468
469 Player 000001476AC37E50 is the Big Blind
470 Player 000001476AC37E50 bet \$2
471
472 Player 000001476AC3F8E0's Turn
473 Your Cards: 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of ♠
Clubs
474 The previous raise was \$3
475 You have \$287
476 Enter your bet, or \$0 to fold: 3
477 Player 000001476AC3F8E0 Calls
478
479 Player 000001476AC39E40 Calls
480
481 Player 000001476AC39E40 discarded 3 cards
482
483 Player 000001476AC37E50 discarded 4 cards
484
485 Player 000001476AC3F8E0's Turn to Discard
486 You may discard up to 5 cards.
487 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
488 Choose a card to discard (1-5), or 0 to Stop: 0
489 Player 000001476AC3F8E0 discarded 0 cards
490
491 Player 000001476AC39E40 Calls
492

493 Player 000001476AC37E50 is All In
494
495 Player 000001476AC3F8E0's Turn
496 Your Cards: 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
497 The previous raise was \$0
498 You have \$284
499 Enter your bet, or \$0 to fold: 0
500 Player 000001476AC3F8E0 Calls
501
502 Player 000001476AC3F8E0's hand: 8 of Diamonds, 7 of Diamonds, 10 of Clubs, 5 of Hearts, Q of Clubs
503 Player 000001476AC39E40's hand: 7 of Spades, 9 of Clubs, 8 of Spades, 4 of Diamonds, 4 of Clubs
504 Player 000001476AC37E50's hand: 3 of Clubs, K of Diamonds, 10 of Hearts, K of Spades, J of Spades
505 Player 000001476AC37E50 Wins \$8
506
507 Player 000001476AC37E50 is the Small Blind
508 Player 000001476AC37E50 bet \$3
509
510 Player 000001476AC3F8E0 is the Big Blind
511 Player 000001476AC3F8E0's Turn
512 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
513 Minimum Bet is \$6
514 The previous raise was \$3
515 You have \$284
516 Enter your bet, or \$0 to fold: 6
517 Player 000001476AC3F8E0 bet \$6
518
519 Player 000001476AC39E40 Calls
520
521 Player 000001476AC37E50 Calls
522
523 Player 000001476AC3F8E0's Turn
524 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
525 The previous raise was \$0
526 You have \$278
527 Enter your bet, or \$0 to fold: 1
528 Player 000001476AC3F8E0 Raises \$1
529
530 Player 000001476AC39E40 Calls
531
532 Player 000001476AC37E50 Calls
533
534 Player 000001476AC3F8E0's Turn
535 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
536 The previous raise was \$0
537 You have \$277

538 Enter your bet, or \$0 to fold: 1
539 Player 000001476AC3F8E0 Raises \$1
540
541 Player 000001476AC39E40 Folds
542
543 Player 000001476AC37E50 Folds
544
545 Player 000001476AC3F8E0's Turn
546 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
547 The previous raise was \$0
548 You have \$276
549 Enter your bet, or \$0 to fold: 1
550 Player 000001476AC3F8E0 Raises \$1
551
552 Player 000001476AC3F8E0's Turn
553 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
554 The previous raise was \$0
555 You have \$275
556 Enter your bet, or \$0 to fold: 1
557 Player 000001476AC3F8E0 Raises \$1
558
559 Player 000001476AC3F8E0's Turn
560 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
561 The previous raise was \$0
562 You have \$274
563 Enter your bet, or \$0 to fold: 1
564 Player 000001476AC3F8E0 Raises \$1
565
566 Player 000001476AC3F8E0's Turn
567 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
568 The previous raise was \$0
569 You have \$273
570 Enter your bet, or \$0 to fold: 1
571 Player 000001476AC3F8E0 Raises \$1
572
573 Player 000001476AC3F8E0's Turn
574 Your Cards: 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
575 The previous raise was \$0
576 You have \$272
577 Enter your bet, or \$0 to fold: 0
578 Player 000001476AC3F8E0 Calls
579
580 Player 000001476AC3F8E0's Turn to Discard
581 You may discard up to 5 cards.
582 8 of Clubs, A of Clubs, 6 of Clubs, A of Diamonds, 7 of Spades
583 Choose a card to discard (1-5), or 0 to Stop: 0
584 Player 000001476AC3F8E0 discarded 0 cards
585
586 Player 000001476AC3F8E0's hand: 8 of Clubs, A of Clubs, 6 of Clubs, A of

Diamonds, 7 of Spades
587 Player 000001476AC3F8E0 Wins \$26
588
589 Player 000001476AC3F8E0 is the Small Blind
590 Player 000001476AC3F8E0's Turn
591 Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
592 Minimum Bet is \$3
593 The previous raise was \$0
594 You have \$298
595 Enter your bet, or \$0 to fold: 3
596 Player 000001476AC3F8E0 bet \$3
597
598 Player 000001476AC39E40 is the Big Blind
599 Player 000001476AC39E40 bet \$1
600
601 Player 000001476AC37E50 Folds
602
603 Player 000001476AC3F8E0's Turn
604 Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
605 The previous raise was \$0
606 You have \$295
607 Enter your bet, or \$0 to fold: 1
608 Player 000001476AC3F8E0 Raises \$1
609
610 Player 000001476AC3F8E0's Turn
611 Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
612 The previous raise was \$0
613 You have \$294
614 Enter your bet, or \$0 to fold: 0
615 Player 000001476AC3F8E0 Calls
616
617 Player 000001476AC3F8E0's Turn to Discard
618 You may discard up to 5 cards.
619 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
620 Choose a card to discard (1-5), or 0 to Stop: 0
621 Player 000001476AC3F8E0 discarded 0 cards
622
623 Player 000001476AC39E40 discarded 4 cards
624
625 Player 000001476AC3F8E0's Turn
626 Your Cards: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs
627 The previous raise was \$0
628 You have \$294
629 Enter your bet, or \$0 to fold: 0
630 Player 000001476AC3F8E0 Calls

631
632 Player 000001476AC39E40 is All In
633
634 Player 000001476AC3F8E0's hand: 10 of Diamonds, Q of Diamonds, A of Spades, K of Hearts, 9 of Clubs [↗](#)
635 Player 000001476AC39E40's hand: 4 of Clubs, 8 of Hearts, 8 of Clubs, 7 of Spades, 5 of Clubs [↗](#)
636 Player 000001476AC39E40 Wins \$5
637
638 Player 000001476AC39E40 is the Small Blind
639 Player 000001476AC39E40 bet \$3
640
641 Player 000001476AC37E50 is the Big Blind
642 Player 000001476AC37E50 bet \$1
643
644 Player 000001476AC3F8E0's Turn
645 Your Cards: Q of Spades, 2 of Clubs, K of Clubs, 4 of Diamonds, A of Spades [↗](#)
646 The previous raise was \$3
647 You have \$294
648 Enter your bet, or \$0 to fold: 3
649 Player 000001476AC3F8E0 Calls
650
651 Player 000001476AC39E40 Calls
652
653 Player 000001476AC39E40 discarded 4 cards
654
655 Player 000001476AC37E50 discarded 0 cards
656
657 Player 000001476AC3F8E0's Turn to Discard
658 You may discard up to 5 cards.
659 Q of Spades, 2 of Clubs, K of Clubs, 4 of Diamonds, A of Spades
660 Choose a card to discard (1-5), or 0 to Stop: 2
661 Q of Spades, K of Clubs, 4 of Diamonds, A of Spades
662 Choose a card to discard (1-4), or 0 to Stop: 3
663 Q of Spades, K of Clubs, A of Spades
664 Choose a card to discard (1-3), or 0 to Stop: 0
665 Player 000001476AC3F8E0 discarded 2 cards
666
667 Player 000001476AC39E40 Calls
668
669 Player 000001476AC37E50 is All In
670
671 Player 000001476AC3F8E0's Turn
672 Your Cards: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
673 The previous raise was \$0
674 You have \$291
675 Enter your bet, or \$0 to fold: 1
676 Player 000001476AC3F8E0 Raises \$1

677
678 Player 000001476AC39E40 Calls
679
680 Player 000001476AC3F8E0's Turn
681 Your Cards: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
682 The previous raise was \$0
683 You have \$290
684 Enter your bet, or \$0 to fold: 1
685 Player 000001476AC3F8E0 Raises \$1
686
687 Player 000001476AC39E40 Folds
688
689 Player 000001476AC3F8E0's Turn
690 Your Cards: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
691 The previous raise was \$0
692 You have \$289
693 Enter your bet, or \$0 to fold: 0
694 Player 000001476AC3F8E0 Calls
695
696 Player 000001476AC3F8E0's hand: Q of Spades, K of Clubs, A of Spades, J of Spades, 10 of Clubs
697 Player 000001476AC37E50's hand: 2 of Diamonds, 4 of Clubs, 9 of Hearts, K of Hearts, A of Clubs
698 Player 000001476AC3F8E0 Wins \$10
699
700 Player 000001476AC37E50 is out of funds.
701 Player 000001476AC3F8E0 is the Small Blind
702 Player 000001476AC3F8E0's Turn
703 Your Cards: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
704 Minimum Bet is \$3
705 The previous raise was \$0
706 You have \$299
707 Enter your bet, or \$0 to fold: 3
708 Player 000001476AC3F8E0 bet \$3
709
710 Player 000001476AC39E40 is the Big Blind
711 Player 000001476AC39E40 bet \$1
712
713 Player 000001476AC3F8E0's Turn
714 Your Cards: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
715 The previous raise was \$0
716 You have \$296
717 Enter your bet, or \$0 to fold: 0
718 Player 000001476AC3F8E0 Calls
719
720 Player 000001476AC3F8E0's Turn to Discard
721 You may discard up to 5 cards.

722 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
723 Choose a card to discard (1-5), or 0 to Stop: 0
724 Player 000001476AC3F8E0 discarded 0 cards
725
726 Player 000001476AC39E40 discarded 1 cards
727
728 Player 000001476AC3F8E0's Turn
729 Your Cards: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
730 The previous raise was \$0
731 You have \$296
732 Enter your bet, or \$0 to fold: 0
733 Player 000001476AC3F8E0 Calls
734
735 Player 000001476AC39E40 is All In
736
737 Player 000001476AC3F8E0's hand: 5 of Spades, Q of Spades, K of Spades, 10 of Diamonds, A of Hearts
738 Player 000001476AC39E40's hand: 3 of Spades, 7 of Diamonds, 10 of Spades, K of Hearts, 10 of Clubs
739 Player 000001476AC39E40 Wins \$4
740
741 Player 000001476AC39E40 is the Small Blind
742 Player 000001476AC39E40 bet \$3
743
744 Player 000001476AC3F8E0 is the Big Blind
745 Player 000001476AC3F8E0's Turn
746 Your Cards: 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
747 Minimum Bet is \$6
748 The previous raise was \$3
749 You have \$296
750 Enter your bet, or \$0 to fold: 6
751 Player 000001476AC3F8E0 bet \$6
752
753 Player 000001476AC39E40 Folds
754
755 Player 000001476AC3F8E0's Turn
756 Your Cards: 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
757 The previous raise was \$0
758 You have \$290
759 Enter your bet, or \$0 to fold: 0
760 Player 000001476AC3F8E0 Calls
761
762 Player 000001476AC3F8E0's Turn to Discard
763 You may discard up to 5 cards.
764 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
765 Choose a card to discard (1-5), or 0 to Stop: 0
766 Player 000001476AC3F8E0 discarded 0 cards
767

768 Player 000001476AC3F8E0's hand: 3 of Clubs, K of Spades, J of Spades, 5 of Hearts, J of Clubs
769 Player 000001476AC3F8E0 Wins \$9
770
771 Player 000001476AC3F8E0 is the Small Blind
772 Player 000001476AC3F8E0's Turn
773 Your Cards: 4 of Clubs, 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
774 Minimum Bet is \$3
775 The previous raise was \$0
776 You have \$299
777 Enter your bet, or \$0 to fold: 3
778 Player 000001476AC3F8E0 bet \$3
779
780 Player 000001476AC39E40 is the Big Blind
781 Player 000001476AC39E40 bet \$1
782
783 Player 000001476AC3F8E0's Turn
784 Your Cards: 4 of Clubs, 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
785 The previous raise was \$0
786 You have \$296
787 Enter your bet, or \$0 to fold: 0
788 Player 000001476AC3F8E0 Calls
789
790 Player 000001476AC3F8E0's Turn to Discard
791 You may discard up to 5 cards.
792 4 of Clubs, 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
793 Choose a card to discard (1-5), or 0 to Stop: 1
794 6 of Clubs, 7 of Hearts, 2 of Clubs, 10 of Diamonds
795 Choose a card to discard (1-4), or 0 to Stop: 1
796 7 of Hearts, 2 of Clubs, 10 of Diamonds
797 Choose a card to discard (1-3), or 0 to Stop: 1
798 2 of Clubs, 10 of Diamonds
799 Choose a card to discard (1-2), or 0 to Stop: 1
800 10 of Diamonds
801 Choose a card to discard (1-1), or 0 to Stop: 1
802 Player 000001476AC3F8E0 discarded 5 cards
803
804 Player 000001476AC39E40 discarded 4 cards
805
806 Player 000001476AC3F8E0's Turn
807 Your Cards: 9 of Spades, Q of Clubs, 3 of Clubs, 9 of Diamonds, J of Clubs
808 The previous raise was \$0
809 You have \$296
810 Enter your bet, or \$0 to fold: 0
811 Player 000001476AC3F8E0 Calls
812
813 Player 000001476AC39E40 is All In

814

815 Player 000001476AC3F8E0's hand: 9 of Spades, Q of Clubs, 3 of Clubs, 9 of
Diamonds, J of Clubs ↗

816 Player 000001476AC39E40's hand: 6 of Hearts, 4 of Diamonds, 2 of Hearts,
10 of Hearts, 8 of Clubs ↗

817 Player 000001476AC3F8E0 Wins \$4

818

819 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.

```
1 #pragma once
2 #include <vector>
3
4 class Card;
5 class Hand;
6
7 using namespace std;
8
9 // @file
10 // @author Nathan Roe
11 // Class to assess and find winner for set of Poker Hands.
12 //
13 // Given a set of hands, will evaluate the Poker Hand Type
14 // and determine the winner or winners.
15 class Scorer
16 {
17 public:
18     // Evaluates a set of hands to determine winner(s)
19     //
20     // Accepts a vector of Hands and moves winner or
21     // winners to the front of the vector. The return
22     // value indicates the last index of a winner.
23     // One winner returns 0, two-way tie returns 1, etc.
24     // Returns -1 for error
25     // @param *hands - pointer to vector of Hands to evaluate and rank
26     // @return the index of the final winner in the partially
27     // sorted vector
28     int findBestHand(vector<Hand*> *hands);
29
30     // Determines and sets the HandType for a given poker hand
31     //
32     // @param *hand - Hand for which to set HandType
33     // @return vector of Cards sorted for comparison based on HandType
34     vector<Card> checkHandType(Hand *hand);
35
36 private:
37     const int HAND_SIZE = 5;
38
39     // Compare two Hands to determine which is better
40     //
41     // @param hand1 - First Hand for comparing
42     // @param hand2 - Second Hand for comparing
43     // @return 1 for hand1, 2 for hand2, 0 for tie,
44     // or -1 for error
45     int compareHands(Hand *hand1, Hand *hand2);
46
47     // Check hand for presence of Straight Flush
48     //
49     // If type is present, sort cards for comparison to
```

```
50 // similar sets of cards
51 // @param *cards - pointer to vector of cards to check
52 // @return true if hand type is present, false otherwise
53 bool hasStraightFlush(vector<Card> *cards);
54
55 // Check hand for presence of Four of a Kind
56 //
57 // If type is present, sort cards for comparison
58 // @param *cards - pointer to vector of cards to check
59 // @return true if hand type is present, false otherwise
60 bool hasFourOfAKind(vector<Card> *cards);
61
62 // Check hand for presence of Full House
63 //
64 // If type is present, sort cards for comparison
65 // @param *cards - pointer to vector of cards to check
66 // @return true if hand type is present, false otherwise
67 bool hasFullHouse(vector<Card> *cards);
68
69 // Check hand for presence of Flush
70 //
71 // If type is present, sort cards for comparison
72 // @param *cards - pointer to vector of cards to check
73 // @return true if hand type is present, false otherwise
74 bool hasFlush(vector<Card> *cards);
75
76 // Check hand for presence of Straight
77 //
78 // If type is present, sort cards for comparison
79 // @param *cards - pointer to vector of cards to check
80 // @return true if hand type is present, false otherwise
81 bool hasStraight(vector<Card> *cards);
82
83 // Check hand for presence of Three of a Kind
84 //
85 // If type is present, sort cards for comparison
86 // @param *cards - pointer to vector of cards to check
87 // @return true if hand type is present, false otherwise
88 bool hasThreeOfAKind(vector<Card> *cards);
89
90 // Check hand for presence of Two Pair
91 //
92 // If type is present, sort cards for comparison
93 // @param *cards - pointer to vector of cards to check
94 // @return true if hand type is present, false otherwise
95 bool hasTwoPair(vector<Card> *cards);
96
97 // Check hand for presence of One Pair
98 //
```

```
99     // If type is present, sort cards for comparison
100    // @param *cards - pointer to vector of cards to check
101    // @return true if hand type is present, false otherwise
102    bool hasOnePair(vector<Card> *cards);
103
104    // Check hand for presence of High Card
105    //
106    // If type is present, sort cards for comparison
107    // @param *cards - pointer to vector of cards to check
108    // @return true if hand type is present, false otherwise
109    bool hasHighCard(vector<Card> *cards);
110 };
111
```

```
1 #include "scorer.h"
2 #include "hand.h"
3 #include "card.h"
4 #include <vector>
5 #include <algorithm>
6
7 using namespace std;
8
9 // Evaluates a set of hands to determine winner(s)
10 int Scorer::findBestHand(vector<Hand*> *hands)
11 {
12     // Return error if not enough hands to score
13     if (hands->size() < 2)
14     {
15         return -1;
16     }
17
18     int numTied = 0;
19     // Begin with first hand as current best
20     Hand *bestHand = hands->at(0);
21     // Compare remaining hands to the current best hand
22     for (int idx = 1; idx < hands->size(); ++idx)
23     {
24         int result = compareHands(bestHand, hands->at(idx));
25         // If hands tie, move new hand to front section of vector
26         if (result == 0)
27         {
28             ++numTied;
29             Hand *hand = hands->at(idx);
30             hands->erase(hands->begin() + idx);
31             hands->insert(hands->begin() + numTied, hand);
32         }
33         // If current best hand is better, leave vector unchanged
34         else if (result == 1)
35         {
36             continue;
37         }
38         // If new hand is better, move to front and reset number of ties
39         else if (result == 2)
40         {
41             numTied = 0;
42             Hand *hand = hands->at(idx);
43             hands->erase(hands->begin() + idx);
44             hands->insert(hands->begin(), hand);
45         }
46         // Return error for unexpected value
47         else
48         {
49             return -1;
```

```
50     }
51 }
52 return numTied;
53 } // End function findBestHand
54
55 // Compare two Hands to determine which is better
56 int Scorer::compareHands(Hand *hand1, Hand *hand2)
57 {
58     // If hands are not sets of 5 cards, return error
59     if (!(hand1->size() == HAND_SIZE) ||
60         !(hand2->size() == HAND_SIZE))
61     {
62         return -1;
63     }
64
65     // Check type of each hand, and prep cards for comparison
66     vector<Card> cards1 = checkHandType(hand1);
67     vector<Card> cards2 = checkHandType(hand2);
68
69     // Verify that each hand is a valid Poker Hand Type
70     if (hand1->getHandType() == Hand::HandType::None ||
71         hand2->getHandType() == Hand::HandType::None)
72     {
73         return -1;
74     }
75
76     int result = -1;
77
78     // Check to see if one Hand is a higher rank than the other
79     if (hand1->getHandType() > hand2->getHandType())
80     {
81         result = 1;
82     }
83     else if (hand1->getHandType() < hand2->getHandType())
84     {
85         result = 2;
86     }
87     // If hands are of same rank, compare cards to find best hand
88     else
89     {
90         // Since hands are pre-sorted, iterate through cards until
91         // a higher card is found
92         for (int idx = 0; idx < HAND_SIZE; ++idx)
93         {
94             if (cards1.at(idx) > cards2.at(idx))
95             {
96                 result = 1;
97                 break;
98             }
99         }
100     }
101 }
```



```
99         else if (cards1.at(idx) < cards2.at(idx))
100         {
101             result = 2;
102             break;
103         }
104         // If all cards are of same value, return tie
105         else
106         {
107             result = 0;
108         }
109     }
110 }
111 return result;
112 } // End function compareHands
113
114 // Determines and sets the HandType for a given poker hand
115 vector<Card> Scorer::checkHandType(Hand *hand)
116 {
117     vector<Card> cards = hand->getCards();
118     // Verify that all cards in the hand are valid
119     for (Card card : cards)
120     {
121         if (!card.isValid())
122         {
123             return cards;
124         }
125     }
126
127     // Test hand against Poker Hand Types from Best to Worst
128     if (hasStraightFlush(&cards))
129     {
130         hand->setHandType(Hand::HandType::StraightFlush);
131     }
132     else if (hasFourOfAKind(&cards))
133     {
134         hand->setHandType(Hand::HandType::FourOfAKind);
135     }
136     else if (hasFullHouse(&cards))
137     {
138         hand->setHandType(Hand::HandType::FullHouse);
139     }
140     else if (hasFlush(&cards))
141     {
142         hand->setHandType(Hand::HandType::Flush);
143     }
144     else if (hasStraight(&cards))
145     {
146         hand->setHandType(Hand::HandType::Straight);
147     }
```

```
148     else if (hasThreeOfAKind(&cards))
149     {
150         hand->setHandType(Hand::HandType::ThreeOfAKind);
151     }
152     else if (hasTwoPair(&cards))
153     {
154         hand->setHandType(Hand::HandType::TwoPair);
155     }
156     else if (hasOnePair(&cards))
157     {
158         hand->setHandType(Hand::HandType::OnePair);
159     }
160     else if (hasHighCard(&cards))
161     {
162         hand->setHandType(Hand::HandType::HighCard);
163     }
164     // If no valid hand type is found, assign None
165     else
166     {
167         hand->setHandType(Hand::HandType::None);
168     }
169
170     // Return vector of cards sorted for comparison based on HandType
171     return cards;
172 } // End function checkHandType
173
174 // Check hand for presence of Straight Flush
175 bool Scorer::hasStraightFlush(vector<Card> *cards)
176 {
177     // Return false if vector is not 5 cards
178     if (!(cards->size() == HAND_SIZE))
179     {
180         return false;
181     }
182
183     // Sort cards from high to low
184     sort(cards->begin(), cards->end());
185     reverse(cards->begin(), cards->end());
186
187     // Special case for Ace-Low
188     if (cards->at(0).value == Card::Value::Ace &&
189         cards->at(1).value == Card::Value::Five)
190     {
191         // Set starting card value/suit to the 5
192         int startVal = static_cast<typename
193             std::underlying_type<Card::Value>::type>(cards->at(1).value);
194         Card::Suit startSuit = cards->at(0).suit;
195         // Iterate through remaining cards, and exit if not in
196         // descending order and matching suit
```

```
196     for (int idx = 2; idx < cards->size(); ++idx)
197     {
198         int cardVal = static_cast<typename
                                std::underlying_type<Card::Value>::type>(cards->at
                                (idx).value);
199         Card::Suit cardSuit = cards->at(idx).suit;
200         if (!(cards->at(idx).isValid()) ||
201             !(cardSuit == startSuit) ||
202             !(cardVal == startVal-- - 1))
203         {
204             return false;
205         }
206     }
207     // Move ace to end
208     Card ace = cards->front();
209     cards->erase(cards->begin());
210     cards->push_back(ace);
211 }
212 // Check Cases with no Aces
213 else
214 {
215     // Set starting card value/suit
216     int startVal = static_cast<typename
                                std::underlying_type<Card::Value>::type>(cards->at(0).value);
217     Card::Suit startSuit = cards->at(0).suit;
218     // Iterate through remaining cards, and exit if not in
219     // descending order and matching suit
220     for (int idx = 1; idx < cards->size(); ++idx)
221     {
222         int cardVal = static_cast<typename
                                std::underlying_type<Card::Value>::type>(cards->at
                                (idx).value);
223         Card::Suit cardSuit = cards->at(idx).suit;
224         if (!(cards->at(idx).isValid()) ||
225             !(cardSuit == startSuit) ||
226             !(cardVal == startVal-- - 1))
227         {
228             return false;
229         }
230     }
231 }
232 return true;
233 } // End function hasStraightFlush
234
235 // Check hand for presence of Four of a Kind
236 bool Scorer::hasFourOfAKind(vector<Card> *cards)
237 {
238     // Return false if vector is not 5 cards
239     if (!(cards->size() == HAND_SIZE))
```

```
240     {
241         return false;
242     }
243
244     // Sort cards from high to low
245     sort(cards->begin(), cards->end());
246     reverse(cards->begin(), cards->end());
247
248     int FOAK = 4;
249     int foundFOAK = false;
250     Card::Value startVal = Card::Value::Invalid;
251     // Test cards in sets (1-4) and (2-5) to see if all values match
252     for (int idx = 0; idx <= cards->size() - FOAK; ++idx)
253     {
254         foundFOAK = true;
255         startVal = cards->at(idx).value;
256         // Iterate over 3 following cards and check for match
257         for (int innerIdx = idx + 1; innerIdx < idx + FOAK; ++innerIdx)
258         {
259             Card::Value cardVal = cards->at(innerIdx).value;
260             // If different value is found, no FOAK
261             if (!(cards->at(innerIdx).isValid()) ||
262                 !(startVal == cardVal))
263             {
264                 foundFOAK = false;
265                 break;
266             }
267         }
268         // If first four cards are FOAK, break loop
269         if (foundFOAK)
270         {
271             break;
272         }
273     }
274
275     // If cards have FOAK, sort matching cards to the front
276     if (foundFOAK)
277     {
278         vector<Card> unusedCards;
279         for (int idx = 0; idx < cards->size(); ++idx)
280         {
281             if (!(cards->at(idx).value == startVal))
282             {
283                 unusedCards.push_back(cards->at(idx));
284                 cards->erase(cards->begin() + idx);
285             }
286         }
287         // Sort non-FOAK cards low to high
288         sort(unusedCards.begin(), unusedCards.end());
```

```
289     // Put unused cards at the end of card set from
290     // high to low
291     for (int idx = 0; idx < unusedCards.size(); ++idx)
292     {
293         cards->push_back(unusedCards.back());
294         unusedCards.pop_back();
295     }
296 }
297
298 return foundFOAK;
299 } // End function hasFourOfAKind
300
301 // Check hand for presence of Full House
302 bool Scorer::hasFullHouse(vector<Card> *cards)
303 {
304     // Return false if vector is not 5 cards
305     if (!(cards->size() == HAND_SIZE))
306     {
307         return false;
308     }
309
310     sort(cards->begin(), cards->end());
311     reverse(cards->begin(), cards->end());
312
313     bool foundFF = false;
314     Card::Value pair = Card::Value::Invalid;
315     Card::Value triple = Card::Value::Invalid;
316     // Check whether first two cards match
317     if (cards->at(0).value == cards->at(1).value)
318     {
319         // If first two cards match, check whether
320         // first three cards make a set of 3
321         if (cards->at(1).value == cards->at(2).value)
322         {
323             triple = cards->at(0).value;
324         }
325         // Otherwise, set pair value as value of first card
326     else
327     {
328         pair = cards->at(0).value;
329     }
330
331     if (!(pair == Card::Value::Invalid))
332     {
333         // If first two cards are a pair, check remaining
334         // cards to see if they make a set of 3
335         if (cards->at(2).value == cards->at(3).value &&
336             cards->at(3).value == cards->at(4).value)
337         {
```

```
338         foundFF = true;
339         triple = cards->at(2).value;
340         // Move set of 3 to the front
341         cards->push_back(cards->front());
342         cards->erase(cards->begin());
343         cards->push_back(cards->front());
344         cards->erase(cards->begin());
345     }
346 }
347 else if (!(triple == Card::Value::Invalid))
348 {
349     // If first three cards make a set of 3, check
350     // remaining cards to see if they are a pair
351     if (cards->at(3).value == cards->at(4).value)
352     {
353         foundFF = true;
354         pair = cards->at(3).value;
355     }
356 }
357 }
358 return foundFF;
359 } // End function hasFullHouse
360
361 // Check hand for presence of Flush
362 bool Scorer::hasFlush(vector<Card> *cards)
363 {
364     // Return false if vector is not 5 cards
365     if (!(cards->size() == HAND_SIZE))
366     {
367         return false;
368     }
369
370     // Sort cards from high to low
371     sort(cards->begin(), cards->end());
372     reverse(cards->begin(), cards->end());
373
374     Card::Suit startSuit = cards->at(0).suit;
375     // Iterate through cards to check whether suits match
376     for (int idx = 1; idx < cards->size(); ++idx)
377     {
378         Card::Suit cardSuit = cards->at(idx).suit;
379         // If different suit is found, return false
380         if (!(cards->at(idx).isValid()) ||
381             !(cardSuit == startSuit))
382         {
383             return false;
384         }
385     }
386 }
```

```
387     return true;
388 } // End function hasFlush
389
390 // Check hand for presence of Straight
391 bool Scorer::hasStraight(vector<Card> *cards)
392 {
393     // Return false if vector is not 5 cards
394     if (!(cards->size() == HAND_SIZE))
395     {
396         return false;
397     }
398
399     // Sort cards high to low
400     sort(cards->begin(), cards->end());
401     reverse(cards->begin(), cards->end());
402
403     // Special case for Ace-Low
404     if (cards->at(0).value == Card::Value::Ace &&
405         cards->at(1).value == Card::Value::Five)
406     {
407         // Set highest card as the 5
408         int startVal = static_cast<typename
409             std::underlying_type<Card::Value>::type>(cards->at(1).value);
410         for (int idx = 2; idx < cards->size(); ++idx)
411         {
412             int cardVal = static_cast<typename
413                 std::underlying_type<Card::Value>::type>(cards->at
414                     (idx).value);
415             // Iterate through remaining cards to see if they are
416             // consecutive
417             if (!(cards->at(idx).isValid()) ||
418                 !(cardVal == startVal-- - 1))
419             {
420                 return false;
421             }
422         }
423         // Move Ace to the back
424         Card ace = cards->front();
425         cards->erase(cards->begin());
426         cards->push_back(ace);
427     }
428     // Check Cases with no Aces
429     else
430     {
431         // Set starting card value
432         int startVal = static_cast<typename
433             std::underlying_type<Card::Value>::type>(cards->at(0).value);
434         for (int idx = 1; idx < cards->size(); ++idx)
```

```
431         int cardVal = static_cast<typename  
            std::underlying_type<Card::Value>::type>(cards->at  
            (idx).value);  
432         // Iterate through remaining cards to see if they are  
            consecutive  
433         if (!(cards->at(idx).isValid()) ||  
434             !(cardVal == startVal-- - 1))  
435         {  
436             return false;  
437         }  
438     }  
439 }  
440 return true;  
441 } // End function hasStraight  
442  
443 // Check hand for presence of Three of a Kind  
444 bool Scorer::hasThreeOfAKind(vector<Card> *cards)  
445 {  
446     // Return false if vector is not 5 cards  
447     if (!(cards->size() == HAND_SIZE))  
448     {  
449         return false;  
450     }  
451  
452     // Sort cards high to low  
453     sort(cards->begin(), cards->end());  
454     reverse(cards->begin(), cards->end());  
455  
456     int TOAK = 3;  
457     int foundTOAK = false;  
458     Card::Value startVal = Card::Value::Invalid;  
459     // Check (1-3), (2-4), (3-5) for three of a kind  
460     for (int idx = 0; idx <= cards->size() - TOAK; ++idx)  
461     {  
462         foundTOAK = true;  
463         startVal = cards->at(idx).value;  
464         // Compare three consecutive cards for matching values  
465         for (int innerIdx = idx + 1; innerIdx < idx + TOAK; ++innerIdx)  
466         {  
467             Card::Value cardVal = cards->at(innerIdx).value;  
468             // If different value is found, move to next set of 3  
469             if (!(cards->at(innerIdx).isValid()) ||  
470                 !(startVal == cardVal))  
471             {  
472                 foundTOAK = false;  
473                 break;  
474             }  
475         }  
476         // End search if three of a kind is found
```



```
477         if (foundTOAK)
478         {
479             break;
480         }
481     }
482
483     // If three of a kind is found, sort cards for comparison
484     if (foundTOAK)
485     {
486         vector<Card> unusedCards;
487         int cardsMoved = 0;
488         // Remove cards that are not part of TOAK
489         for (int idx = 0; idx < HAND_SIZE; ++idx)
490         {
491             int cardIdx = idx - cardsMoved;
492             if (!(cards->at(cardIdx).value == startVal))
493             {
494                 unusedCards.push_back(cards->at(cardIdx));
495                 cards->erase(cards->begin() + cardIdx);
496                 ++cardsMoved;
497             }
498         }
499         // Place non-TOAK cards from high-low and end of cards
500         sort(unusedCards.begin(), unusedCards.end());
501         size_t startSize = unusedCards.size();
502         for (int idx = 0; idx < startSize; ++idx)
503         {
504             cards->push_back(unusedCards.back());
505             unusedCards.pop_back();
506         }
507     }
508
509     return foundTOAK;
510 } // End function hasThreeOfAKind
511
512 // Check hand for presence of Two Pair
513 bool Scorer::hasTwoPair(vector<Card> *cards)
514 {
515     // Return false if vector is not 5 cards
516     if (!(cards->size() == HAND_SIZE))
517     {
518         return false;
519     }
520
521     // Sort cards from high to low
522     sort(cards->begin(), cards->end());
523     reverse(cards->begin(), cards->end());
524
525     bool foundTP = false;
```

```
526     vector<Card> pair1;
527     vector<Card> pair2;
528
529     // Search cards for a pair
530     for (int idx = 1; idx < cards->size(); ++idx)
531     {
532         int prevIdx = idx - 1;
533         Card *card = &cards->at(idx);
534         Card *prevCard = &cards->at(prevIdx);
535         // If pair is found, store cards
536         if (card->value == prevCard->value)
537         {
538             foundTP = true;
539             pair1.push_back(*prevCard);
540             pair1.push_back(*card);
541             cards->erase(cards->begin() + prevIdx);
542             cards->erase(cards->begin() + prevIdx);
543             break;
544         }
545     }
546
547     // If pair is found, search remaining cards for second pair
548     if (foundTP)
549     {
550         foundTP = false;
551         for (int idx = 1; idx < cards->size(); ++idx)
552         {
553             int prevIdx = idx - 1;
554             Card *card = &cards->at(idx);
555             Card *prevCard = &cards->at(prevIdx);
556             // If pair is found, store cards
557             if (card->value == prevCard->value)
558             {
559                 foundTP = true;
560                 pair2.push_back(*prevCard);
561                 pair2.push_back(*card);
562                 cards->erase(cards->begin() + prevIdx);
563                 cards->erase(cards->begin() + prevIdx);
564             }
565         }
566     }
567
568     // If two-pair found, sort cards highPair-lowPair-spareCard
569     if (foundTP)
570     {
571         // Place pairs in set highest to lowest
572         if (pair1.front().value > pair2.front().value)
573         {
574             cards->push_back(pair1.front());
```

```
575         cards->push_back(pair1.back());
576         cards->push_back(pair2.front());
577         cards->push_back(pair2.back());
578     }
579     else
580     {
581         cards->push_back(pair2.front());
582         cards->push_back(pair2.back());
583         cards->push_back(pair1.front());
584         cards->push_back(pair1.back());
585     }
586     // Move spare card to the end
587     cards->push_back(cards->front());
588     cards->erase(cards->begin());
589 }
590 // Place unused pair back into card set
591 else if (pair1.size() > 0)
592 {
593     cards->push_back(pair1.front());
594     cards->push_back(pair1.back());
595 }
596
597 return foundTP;
598 } // End function foundTwoPair
599
600 // Check hand for presence of One Pair
601 bool Scorer::hasOnePair(vector<Card> *cards)
602 {
603     // Return false if vector is not 5 cards
604     if (!(cards->size() == HAND_SIZE))
605     {
606         return false;
607     }
608
609     sort(cards->begin(), cards->end());
610     reverse(cards->begin(), cards->end());
611
612     bool foundPair = false;
613     vector<Card> pair;
614
615     // Search cards for pair of equal value
616     for (int idx = 1; idx < cards->size(); ++idx)
617     {
618         int prevIdx = idx - 1;
619         Card *card = &cards->at(idx);
620         Card *prevCard = &cards->at(prevIdx);
621         // If pair is found, store pair
622         if (card->value == prevCard->value)
623         {
```

```
624         foundPair = true;
625         pair.push_back(*prevCard);
626         pair.push_back(*card);
627         cards->erase(cards->begin() + prevIdx);
628         cards->erase(cards->begin() + prevIdx);
629         break;
630     }
631 }
632
633 // If pair is found, place pair at front of cards
634 if (foundPair)
635 {
636     cards->insert(cards->begin(), pair.front());
637     cards->insert(cards->begin(), pair.back());
638 }
639
640 return foundPair;
641 } // End function hasOnePair
642
643 // Check hand for presence of High Card
644 bool Scorer::hasHighCard(vector<Card> *cards)
645 {
646     // Return false if vector is not 5 cards
647     if (!(cards->size() == HAND_SIZE))
648     {
649         return false;
650     }
651
652     // Sort cards from high to low
653     sort(cards->begin(), cards->end());
654     reverse(cards->begin(), cards->end());
655
656     return true;
657 } // End function hasHighCard
658
```

```
1 #include "card.h"
2 #include "hand.h"
3 #include "scorer.h"
4 #include <iostream>
5 #include <vector>
6 #include <libconfig.h++>
7
8 using namespace std;
9 using namespace libconfig;
10 // @file
11 // @author Nathan Roe
12 // Compares Poker Hands provided in file to find winners
13 //
14 // Given a set of hands, will evaluate the Poker Hand Type
15 // and determine the winner or winners.
16
17 // Creates Card object using integer value and string suit
18 Card makeCard(int value, string suit)
19 {
20     Card::Value cardValue = Card::Value::Invalid;
21     Card::Suit cardSuit = Card::Suit::Invalid;
22     // Set card value
23     try
24     {
25         cardValue = static_cast<Card::Value>(value);
26     }
27     catch (const exception &e)
28     {
29         cout << e.what() << endl;
30         cout << "Invalid Card Value: " << value << endl;
31         cardValue = Card::Value::Invalid;
32     }
33
34     // Set card suit
35     if (suit.size() > 0)
36     {
37         char cardChar = suit[0];
38         switch (cardChar)
39         {
40             case 'C':
41                 cardSuit = Card::Suit::Clubs;
42                 break;
43             case 'D':
44                 cardSuit = Card::Suit::Diamonds;
45                 break;
46             case 'H':
47                 cardSuit = Card::Suit::Hearts;
48                 break;
49             case 'S':
```

```
50         cardSuit = Card::Suit::Spades;
51         break;
52     default:
53         cardSuit = Card::Suit::Invalid;
54         break;
55     }
56 }
57
58 return Card(cardValue, cardSuit);
59 } // End function makeCard
60
61 // Creates Card object using string value and suit
62 Card makeCard(string value, string suit)
63 {
64     Card::Value cardValue = Card::Value::Invalid;
65     Card::Suit cardSuit = Card::Suit::Invalid;
66
67     // Set value for Ten through Ace
68     if (value == "Ten" || value == "T")
69     {
70         cardValue = Card::Value::Ten;
71     }
72     else if (value == "Jack" || value == "J")
73     {
74         cardValue = Card::Value::Jack;
75     }
76     else if (value == "Queen" || value == "Q")
77     {
78         cardValue = Card::Value::Queen;
79     }
80     else if (value == "King" || value == "K")
81     {
82         cardValue = Card::Value::King;
83     }
84     else if (value == "Ace" || value == "A")
85     {
86         cardValue = Card::Value::Ace;
87     }
88     // If card is not 10-Ace, try to set based on numerical value
89     else
90     {
91         try
92         {
93             cardValue = static_cast<Card::Value>(stoi(value));
94         }
95         catch (const invalid_argument &e)
96         {
97             cout << e.what() << ": Invalid Card Value: " << value << endl;
98             cardValue = Card::Value::Invalid;
```

```
99     }
100     catch (const exception &e)
101     {
102         cout << e.what() << ": Invalid Card Value: " << value << endl;
103         cardValue = Card::Value::Invalid;
104     }
105 }
106
107 // Set suit value
108 if (suit.size() > 0)
109 {
110     char cardChar = suit[0];
111     switch (cardChar)
112     {
113     case 'C':
114         cardSuit = Card::Suit::Clubs;
115         break;
116     case 'D':
117         cardSuit = Card::Suit::Diamonds;
118         break;
119     case 'H':
120         cardSuit = Card::Suit::Hearts;
121         break;
122     case 'S':
123         cardSuit = Card::Suit::Spades;
124         break;
125     default:
126         cardSuit = Card::Suit::Invalid;
127         break;
128     }
129 }
130
131 return Card(cardValue, cardSuit);
132 } // End function makeCard
133
134 // Format and print test results for input files
135 void printResults(int lastWinIdx, vector<Hand*> *players)
136 {
137     // Print out error message if error value returned
138     if (lastWinIdx == -1)
139     {
140         cout << "ERROR IN SCENARIO" << endl;
141         return;
142     }
143
144     // Print out winning hands
145     cout << "(" << players->at(0)->printCards() << ")";
146     for (int idx = 1; idx <= lastWinIdx; ++idx)
147     {
```

```
148         cout << ", "
149         << "(" << players->at(idx)->printCards() << ")";
150     }
151     // Format string based on whether ties are present
152     if (lastWinIdx > 0)
153     {
154         cout << " Tie for the Win.";
155     }
156     else
157     {
158         cout << " Wins.";
159     }
160
161     // Print out losing hands
162     bool multiLoser = false;
163     for (int idx = lastWinIdx + 1; idx < players->size(); ++idx)
164     {
165         cout << " "
166         << "(" << players->at(idx)->printCards() << ")";
167         if (idx < players->size() - 1)
168         {
169             multiLoser = true;
170             cout << " and";
171         }
172     }
173     // Format string based on presence of multiple losing hands
174     if (multiLoser)
175     {
176         cout << " Lose.";
177     }
178     else if (lastWinIdx < players->size() - 1)
179     {
180         cout << " Loses.";
181     }
182     cout << endl;
183 } // End function printResults
184
185 // Main function; runs Poker Hand Scorer for each input file provided
186 int main(int argc, char **argv)
187 {
188     Scorer *scorer = new Scorer();
189
190     // Iterate through input args, and read as paths
191     for (int idx = 1; idx < argc; ++idx)
192     {
193         Config *cfg = new Config();
194         char *path = argv[idx];
195
196         // Attempt to parse Config files
```



```
197     try
198     {
199         cout << path << endl;
200         cfg->readFile(path);
201     }
202     // Catch file path errors
203     catch (const FileIOException &fioex)
204     {
205         cerr << fioex.what() << " while reading file " << path <<      ↗
206             endl;
207         return (EXIT_FAILURE);
208     }
209     // Catch config file format errors
210     catch (const ParseException &pex)
211     {
212         cerr << "Parse error at " << pex.getFile() << ":" <<      ↗
213             pex.getLine()
214             << " - " << pex.getError() << endl;
215         return (EXIT_FAILURE);
216     }
217
218     vector<Hand*> *players = new vector<Hand*>;
219
220     const Setting &root = cfg->getRoot();
221     // Find "hands" data set in config file
222     try
223     {
224         const Setting &hands = root["hands"];
225         int numHands = hands.getLength();
226
227         // Create a Hand object for each hand in Config
228         for (int handIdx = 0; handIdx < numHands; ++handIdx)
229         {
230             Hand *hand = new Hand();
231
232             // Read Cards list in config
233             const Setting &cards = hands[handIdx];
234             int numCards = cards.getLength();
235             for (int cardIdx = 0; cardIdx < numCards; ++cardIdx)
236             {
237                 const Setting &card = cards[cardIdx];
238
239                 string suit;
240                 bool suitFound = false;
241                 bool valueFound = false;
242
243                 // Find required suit and value
244                 suitFound = card.lookupValue("suit", suit);
```

```
244         if (suitFound)
245         {
246             int value;
247             string valueString;
248             // Search for card values formatted as Integers
249             valueFound = card.lookupValue("value", value);
250             if (valueFound)
251             {
252                 Card card = makeCard(value, suit);
253                 // Add valid card to hand
254                 if (card.isValid())
255                 {
256                     hand->deal(&card);
257                 }
258             }
259             else
260             {
261                 // Search for card values formatted as Strings
262                 valueFound = card.lookupValue("value",
263 valueString);
264                 if (valueFound)
265                 {
266                     Card card = makeCard(valueString, suit);
267                     // Add valid card to hand
268                     if (card.isValid())
269                     {
270                         hand->deal(&card);
271                     }
272                 }
273             }
274             // Print error for bad hand
275             if (!(valueFound && suitFound))
276             {
277                 cout << "Could not find 'suit' and/or 'value' in
278 Hand " << handIdx + 1 << " Card " << cardIdx + 1 <<
279 endl;
280             }
281         }
282         players->push_back(hand);
283     }
284     // Run scorer if multiple hands present
285     if (players->size() > 1)
286     {
287         int winnerIdx = scorer->findBestHand(players);
288         for (Hand *hand : *players)
289         {
290             cout << "(" << hand->printCards() << "): " << hand-
```

```
                >printHandType() << endl;
290            }
291            printResults(winnerIdx, players);
292        }
293    }
294    // Print error message for improperly formatted file
295    catch (const SettingNotFoundException &nfex)
296    {
297        cout << nfex.what() << ": Could not find 'hands' in " << path << endl;
298    }
299 }
300 return 0;
301 } // End function main
302
```

```
C:\Users\nater\source\repos\PokerGame\doc\handsOutput.txt 1
1 C:\Users\nater\source\repos\PokerHands>python test\PokerHandTest.py > doc
  \output.txt
2
3 Output:
4 C:\Users\nater\source\repos\PokerHands\test\../Input/Flush1.cfg
5 (K of Diamonds, J of Diamonds, 9 of Diamonds, 6 of Diamonds, 4 of
  Diamonds): Flush
6 (Q of Clubs, J of Clubs, 7 of Clubs, 6 of Clubs, 5 of Clubs): Flush
7 (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.
8 Test Result: PASS
9
10 Output:
11 C:\Users\nater\source\repos\PokerHands\test\../Input/Flush2.cfg
12 (Q of Clubs, J of Clubs, 7 of Clubs, 6 of Clubs, 5 of Clubs): Flush
13 (J of Hearts, 10 of Hearts, 9 of Hearts, 4 of Hearts, 2 of Hearts):
  Flush
14 (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.
15 Test Result: PASS
16
17 Output:
18 C:\Users\nater\source\repos\PokerHands\test\../Input/Flush3.cfg
19 (J of Hearts, 10 of Hearts, 9 of Hearts, 4 of Hearts, 2 of Hearts):
  Flush
20 (J of Spades, 10 of Spades, 8 of Spades, 6 of Spades, 3 of Spades):
  Flush
21 (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.
22 Test Result: PASS
23
24 Output:
25 C:\Users\nater\source\repos\PokerHands\test\../Input/Flush4.cfg
26 (J of Spades, 10 of Spades, 8 of Spades, 6 of Spades, 3 of Spades):
  Flush
27 (J of Hearts, 10 of Hearts, 8 of Hearts, 4 of Hearts, 3 of Hearts):
  Flush
28 (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.
29 Test Result: PASS
30
31 Output:
32 C:\Users\nater\source\repos\PokerHands\test\../Input/Flush5.cfg
33 (J of Hearts, 10 of Hearts, 8 of Hearts, 4 of Hearts, 3 of Hearts):
  Flush
```

```
34      (J of Clubs, 10 of Clubs, 8 of Clubs, 4 of Clubs, 2 of Clubs): Flush
35      (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.
36 Test Result: PASS
37
38 Output:
39      C:\Users\nater\source\repos\PokerHands\test\../Input/Flush6.cfg
40      (10 of Diamonds, 8 of Diamonds, 7 of Diamonds, 6 of Diamonds, 5 of  ↗
      Diamonds): Flush
41      (10 of Spades, 8 of Spades, 7 of Spades, 6 of Spades, 5 of Spades):  ↗
      Flush
42      (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.
43 Test Result: PASS
44
45 Output:
46      C:\Users\nater\source\repos\PokerHands\test\../Input/FourOfAKind1.cfg
47      (K of Spades, K of Hearts, K of Clubs, K of Diamonds, 3 of Hearts):  ↗
      Four of a Kind
48      (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, Q of Hearts):  ↗
      Four of a Kind
49      (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.
50 Test Result: PASS
51
52 Output:
53      C:\Users\nater\source\repos\PokerHands\test\../Input/FourOfAKind2.cfg
54      (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, Q of Hearts):  ↗
      Four of a Kind
55      (7 of Hearts, 7 of Diamonds, 7 of Spades, 7 of Clubs, 10 of Spades):  ↗
      Four of a Kind
56      (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.
57 Test Result: PASS
58
59 Output:
60      C:\Users\nater\source\repos\PokerHands\test\../Input/FourOfAKind3.cfg
61      (4 of Clubs, 4 of Spades, 4 of Diamonds, 4 of Hearts, 9 of Clubs):  ↗
      Four of a Kind
62      (4 of Clubs, 4 of Spades, 4 of Diamonds, 4 of Hearts, 9 of Diamonds):  ↗
      Four of a Kind
63      (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.
64 Test Result: PASS
```

```
65
66 Output:
67     C:\Users\nater\source\repos\PokerHands\test\../Input/FullHouse1.cfg
68     (8 of Spades, 8 of Diamonds, 8 of Hearts, 7 of Diamonds, 7 of Clubs): ↗
        Full House
69     (4 of Diamonds, 4 of Spades, 4 of Clubs, 9 of Diamonds, 9 of Clubs): ↗
        Full House
70     (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.
71 Test Result: PASS
72
73 Output:
74     C:\Users\nater\source\repos\PokerHands\test\../Input/FullHouse2.cfg
75     (4 of Diamonds, 4 of Spades, 4 of Clubs, 9 of Diamonds, 9 of Clubs): ↗
        Full House
76     (4 of Diamonds, 4 of Spades, 4 of Clubs, 5 of Clubs, 5 of Diamonds): ↗
        Full House
77     (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.
78 Test Result: PASS
79
80 Output:
81     C:\Users\nater\source\repos\PokerHands\test\../Input/FullHouse3.cfg
82     (K of Clubs, K of Spades, K of Diamonds, J of Clubs, J of Spades): ↗
        Full House
83     (K of Clubs, K of Hearts, K of Diamonds, J of Clubs, J of Hearts): ↗
        Full House
84     (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.
85 Test Result: PASS
86
87 Output:
88     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard1.cfg
89     (K of Spades, 6 of Clubs, 5 of Hearts, 3 of Diamonds, 2 of Clubs): ↗
        High Card
90     (Q of Spades, J of Diamonds, 6 of Clubs, 5 of Hearts, 3 of Clubs): ↗
        High Card
91     (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.
92 Test Result: PASS
93
94 Output:
95     C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard2.cfg
96     (Q of Spades, J of Diamonds, 6 of Clubs, 5 of Hearts, 3 of Clubs): ↗
        High Card
```

C:\Users\nater\source\repos\PokerGame\doc\handsOutput.txt	4
97 (Q of Spades, 10 of Diamonds, 8 of Clubs, 7 of Diamonds, 4 of Spades): High Card	↗
98 (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.	↗
99 Test Result: PASS	
100	
101 Output:	
102 C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard3.cfg	
103 (Q of Spades, 10 of Diamonds, 8 of Clubs, 7 of Diamonds, 4 of Spades): High Card	↗
104 (Q of Hearts, 10 of Hearts, 7 of Clubs, 6 of Hearts, 4 of Spades): High Card	↗
105 (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.	↗
106 Test Result: PASS	
107	
108 Output:	
109 C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard4.cfg	
110 (Q of Hearts, 10 of Hearts, 7 of Clubs, 6 of Hearts, 4 of Spades): High Card	↗
111 (Q of Clubs, 10 of Clubs, 7 of Diamonds, 5 of Clubs, 4 of Diamonds): High Card	↗
112 (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.	↗
113 Test Result: PASS	
114	
115 Output:	
116 C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard5.cfg	
117 (Q of Clubs, 10 of Clubs, 7 of Diamonds, 5 of Clubs, 4 of Diamonds): High Card	↗
118 (Q of Hearts, 10 of Diamonds, 7 of Spades, 5 of Spades, 2 of Hearts): High Card	↗
119 (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.	↗
120 Test Result: PASS	
121	
122 Output:	
123 C:\Users\nater\source\repos\PokerHands\test\../Input/HighCard6.cfg	
124 (10 of Clubs, 8 of Spades, 7 of Spades, 6 of Hearts, 4 of Diamonds): High Card	↗
125 (10 of Diamonds, 8 of Diamonds, 7 of Spades, 6 of Clubs, 4 of Clubs): High Card	↗
126 (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.	↗

127 Test Result: PASS

128

129 Output:

130 C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair1.cfg

131 (9 of Clubs, 9 of Diamonds, Q of Spades, J of Hearts, 5 of Hearts): [↗](#)
One Pair

132 (6 of Diamonds, 6 of Hearts, K of Spades, 7 of Hearts, 4 of Clubs): [↗](#)
One Pair

133 (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.

134 Test Result: PASS

135

136 Output:

137 C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair2.cfg

138 (6 of Diamonds, 6 of Hearts, K of Spades, 7 of Hearts, 4 of Clubs): [↗](#)
One Pair

139 (6 of Diamonds, 6 of Hearts, Q of Hearts, J of Spades, 2 of Clubs): [↗](#)
One Pair

140 (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.

141 Test Result: PASS

142

143 Output:

144 C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair3 (2 [↗](#)
shuffle).cfg

145 (4 of Clubs, 6 of Diamonds, K of Spades, 7 of Hearts, 6 of Hearts): [↗](#)
One Pair

146 (J of Spades, 6 of Diamonds, 6 of Hearts, 2 of Clubs, Q of Hearts): [↗](#)
One Pair

147 (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.

148 Test Result: PASS

149

150 Output:

151 C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair4.cfg

152 (6 of Diamonds, 6 of Hearts, Q of Hearts, J of Spades, 2 of Clubs): [↗](#)
One Pair

153 (6 of Diamonds, 6 of Hearts, Q of Spades, 8 of Clubs, 7 of Diamonds): [↗](#)
One Pair

154 (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.

155 Test Result: PASS

156

157 Output:

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


```

159      (6 of Diamonds, 6 of Hearts, Q of Spades, 8 of Clubs, 7 of Diamonds): One Pair ↗
160      (6 of Diamonds, 6 of Hearts, Q of Diamonds, 8 of Hearts, 3 of Spades): One Pair ↗
161      (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. ↗
162 Test Result: PASS
163
164 Output:
165      C:\Users\nater\source\repos\PokerHands\test\../Input/OnePair6.cfg
166      (8 of Spades, 8 of Diamonds, 10 of Hearts, 6 of Clubs, 5 of Spades): One Pair ↗
167      (8 of Hearts, 8 of Clubs, 10 of Clubs, 6 of Spades, 5 of Clubs): One Pair ↗
168      (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. ↗
169 Test Result: PASS
170
171 Output:
172      C:\Users\nater\source\repos\PokerHands\test\../Input/Straight1.cfg
173      (J of Hearts, 10 of Hearts, 9 of Clubs, 8 of Spades, 7 of Hearts): Straight ↗
174      (10 of Spades, 9 of Spades, 8 of Clubs, 7 of Hearts, 6 of Spades): Straight ↗
175      (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. ↗
176 Test Result: PASS
177
178 Output:
179      C:\Users\nater\source\repos\PokerHands\test\../Input/Straight2.cfg
180      (10 of Spades, 9 of Spades, 8 of Clubs, 7 of Hearts, 6 of Spades): Straight ↗
181      (6 of Clubs, 5 of Spades, 4 of Hearts, 3 of Spades, 2 of Diamonds): Straight ↗
182      (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. ↗
183 Test Result: PASS
184
185 Output:
186      C:\Users\nater\source\repos\PokerHands\test\../Input/Straight3.cfg
187      (9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Diamonds, 5 of Diamonds): Straight ↗
188      (9 of Spades, 8 of Spades, 7 of Spades, 6 of Hearts, 5 of Hearts): Straight ↗
189      (9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Diamonds, 5 of Diamonds), ↗

```

```

C:\Users\nater\source\repos\PokerGame\doc\handsOutput.txt 7
    (9 of Spades, 8 of Spades, 7 of Spades, 6 of Hearts, 5 of Hearts) 7
    Tie for the Win.
190 Test Result: PASS
191
192 Output:
193     C:\Users\nater\source\repos\PokerHands\test\../Input/ 7
        StraightFlush1.cfg
194     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs): 7
        Straight Flush
195     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts): 7
        Straight Flush
196     (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs) Wins. 7
        (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts) 7
        Loses.
197 Test Result: PASS
198
199 Output:
200     C:\Users\nater\source\repos\PokerHands\test\../Input/ 7
        StraightFlush2.cfg
201     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts): 7
        Straight Flush
202     (6 of Spades, 5 of Spades, 4 of Spades, 3 of Spades, 2 of Spades): 7
        Straight Flush
203     (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts) 7
        Wins. (6 of Spades, 5 of Spades, 4 of Spades, 3 of Spades, 2 of 7
        Spades) Loses.
204 Test Result: PASS
205
206 Output:
207     C:\Users\nater\source\repos\PokerHands\test\../Input/ 7
        StraightFlush3.cfg
208     (7 of Diamonds, 6 of Diamonds, 5 of Diamonds, 4 of Diamonds, 3 of 7
        Diamonds): Straight Flush
209     (7 of Spades, 6 of Spades, 5 of Spades, 4 of Spades, 3 of Spades): 7
        Straight Flush
210     (7 of Diamonds, 6 of Diamonds, 5 of Diamonds, 4 of Diamonds, 3 of 7
        Diamonds), (7 of Spades, 6 of Spades, 5 of Spades, 4 of Spades, 3 7
        of Spades) Tie for the Win.
211 Test Result: PASS
212
213 Output:
214     C:\Users\nater\source\repos\PokerHands\test\../Input/ 7
        ThreeOfAKind1.cfg
215     (6 of Hearts, 6 of Diamonds, 6 of Spades, Q of Clubs, 4 of Spades): 7
        Three of a Kind
216     (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of Spades): 7
        Three of a Kind
217     (6 of Hearts, 6 of Diamonds, 6 of Spades, Q of Clubs, 4 of Spades) 7
        Wins. (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of 7

```

Spades) Loses.

218 Test Result: PASS

219

220 Output:

221 C:\Users\nater\source\repos\PokerHands\test\../Input/
ThreeOfAKind2.cfg

222 (3 of Diamonds, 3 of Spades, 3 of Clubs, K of Spades, 2 of Spades):
Three of a Kind

223 (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Clubs, 7 of Hearts):
Three of a Kind

224 (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.

225 Test Result: PASS

226

227 Output:

228 C:\Users\nater\source\repos\PokerHands\test\../Input/
ThreeOfAKind3.cfg

229 (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Clubs, 7 of Hearts):
Three of a Kind

230 (3 of Diamonds, 3 of Spades, 3 of Clubs, J of Spades, 5 of Diamonds):
Three of a Kind

231 (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.

232 Test Result: PASS

233

234 Output:

235 C:\Users\nater\source\repos\PokerHands\test\../Input/
ThreeOfAKind4.cfg

236 (9 of Spades, 9 of Hearts, 9 of Diamonds, 10 of Diamonds, 8 of
Hearts): Three of a Kind

237 (9 of Clubs, 9 of Spades, 9 of Hearts, 10 of Diamonds, 8 of
Diamonds): Three of a Kind

238 (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.

239 Test Result: PASS

240

241 Output:

242 C:\Users\nater\source\repos\PokerHands\test\../Input/ThreeWayTie.cfg

243 (10 of Diamonds, 9 of Diamonds, 8 of Diamonds, 6 of Diamonds, 7 of
Diamonds): Straight Flush

244 (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs):
Straight Flush

245 (10 of Hearts, 9 of Hearts, 8 of Hearts, 6 of Hearts, 7 of Hearts):
Straight Flush

246 (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts):
Straight Flush

```

C:\Users\nater\source\repos\PokerGame\doc\handsOutput.txt 9
247      (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.
248 Test Result: PASS
249
250 Output:
251      C:\Users\nater\source\repos\PokerHands\test\../Input/TripleTest1.cfg
252      (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs):
          Straight Flush
253      (8 of Hearts, 7 of Hearts, 6 of Hearts, 5 of Hearts, 4 of Hearts):
          Straight Flush
254      (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts):
          Straight Flush
255      (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.
256 Test Result: PASS
257
258 Output:
259      C:\Users\nater\source\repos\PokerHands\test\../Input/TripleTest2.cfg
260      (10 of Diamonds, 9 of Diamonds, 8 of Diamonds, 6 of Diamonds, 7 of
          Diamonds): Straight Flush
261      (10 of Clubs, 9 of Clubs, 8 of Clubs, 7 of Clubs, 6 of Clubs):
          Straight Flush
262      (3 of Hearts, 5 of Hearts, 2 of Hearts, A of Hearts, 4 of Hearts):
          Straight Flush
263      (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.
264 Test Result: PASS
265
266 Output:
267      C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair1.cfg
268      (10 of Diamonds, 10 of Spades, 2 of Spades, 2 of Clubs, K of Clubs):
          Two Pair
269      (5 of Clubs, 5 of Spades, 4 of Diamonds, 4 of Hearts, 10 of Hearts):
          Two Pair
270      (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.
271 Test Result: PASS
272
273 Output:
274      C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair2.cfg
275      (5 of Clubs, 5 of Spades, 4 of Diamonds, 4 of Hearts, 10 of Hearts):

```

```
Two Pair
276 (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades):  ↗
Two Pair
277 (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.
278 Test Result: PASS
279
280 Output:
281 C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair3.cfg
282 (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades):  ↗
Two Pair
283 (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, J of Spades):  ↗
Two Pair
284 (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.
285 Test Result: PASS
286
287 Output:
288 C:\Users\nater\source\repos\PokerHands\test\../Input/  ↗
TwoPair3Flipped.cfg
289 (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, Q of Spades):  ↗
Two Pair
290 (5 of Clubs, 5 of Spades, 3 of Clubs, 3 of Diamonds, J of Spades):  ↗
Two Pair
291 (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.
292 Test Result: PASS
293
294 Output:
295 C:\Users\nater\source\repos\PokerHands\test\../Input/TwoPair4.cfg
296 (K of Diamonds, K of Spades, 7 of Diamonds, 7 of Hearts, 8 of  ↗
Hearts): Two Pair
297 (K of Clubs, K of Spades, 7 of Clubs, 7 of Hearts, 8 of Clubs): Two  ↗
Pair
298 (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.
299 Test Result: PASS
300
301
```