Poker Hands

How to run

Tested and built under Ubuntu with GCC 5.4. Command: `g++ -std=c++11 -Wall -Wextra PokerHands.cpp`

Steps Performed in the Program

1. For each line read, split it into white and black poker hands.
2. For each poker hand with 5 cards, construct the bit representation using C++ STL library `bitset<64>`. There will be several maps to keep track of information when we are looping through the cards:
    - `suit_count`: Keep track of # of cards in each suit in the poker hand.
    - `value_count`: Keep track of # of cards with the same value in the poker hand.
    - `highest_card`: A integer for keeping the highest value in the poker hand.
    - `pair_count`: Keep track of # of pairs presented in the poker hand.
3. There will be information bits like `0000 0000 0000` in the most significant bits part of the return value. The first 4 bits represent the ranking which is a binary number from 1~8 determined using the following criteria.

Type	# of Consecutive Value Cards	# of Same Suit Cards	# of Same Value Cards	# of Pairs
Straight flush	5	5		
4 of a kind			4	
Full house			3	2
Flush		5		
Straight	5			

Type	# of Consecutive Value Cards	# of Same Suit Cards	# of Same Value Cards	# of Pairs
3 of a kind			3	
2 pairs			2	2
Pair			2	

4. The second 4 bits represent the highest card. In case of the same ranking between the 2 poker hands, the highest card is used to determine who wins.
5. The will be further comparison if the highest card is the same as well. In this case, the program will construct the "or" bits with 4 suits ($C[i] \mid D[i] \mid H[i] \mid S[i], i = 1, ..., 13$) which will tell us which value in the 13 values does the poker hand have and then compare the number. The poker hand which has the higher number of "or" bits will win.

Check the Check

How to run

Tested and built under Ubuntu with GCC 5.4. Command: `g++ -std=c++11 -Wall -Wextra CheckTheCheck.cpp`

Steps Performed in the Program

1. Read console inputs and then build an 8x8 2-dimentional vector to store the information.
2. Find the coordinates of both white and black kings by going through the data structure.
3. **Starting from a king's coordinate to the edge of the data structure**, check horizontally, vertically and diagonally to see whether there is any piece along the way. If yes, then:
    - If the element is empty, ignore it and continue the search along the way.
    - If the first element we encountered belongs to the same team as the king (i.e. black king with black piece and vice versa), this means we are safe along this way. Set `blocked` to 1 to indicate this.

    - If the first element we encountered belongs to the oponnet team as the king (i.e. black king with white piece and vice versa), we need to check this piece's jumping. If it can reach the king in one legal move, the king will be under check.
4. Knight's jumping will be checked separately because it doesn't follow any specific horizontal, vertical or diagonal move.