PROJECT 1

BLACKJACK

Abstract

This project implements a one-on-one Blackjack game in C++, featuring a player versus dealer with standard rules, using STL containers and algorithms to manage game mechanics efficiently. Developed to demonstrate proficiency in object-oriented programming and STL usage, the code is organized into classes for cards, deck, hand, and game logic, hosted on GitHub.

Jorge Carbajal

CIS-17C 42513 (05/07/25)

INTRODUCTION

Why are you coding this game and why?

I built a one-on-one Blackjack game in C++ because I've always liked casino games like Blackjack and poker, where you can use strategy and probability to tilt the odds a bit in your favor, assuming the game's fair. I picked it to dive into C++'s STL, using containers like std::list, std::deque, std::set, std::map, std::stack, and std::queue to manage cards, hands, and game flow. It was also a great chance to use algorithms like std::shuffle, std::for_each, std::copy, and std::max_element for shuffling, dealing, and keeping track of stats.

How long did you spend, how many lines, classes, etc.....

I spent 3 days working on this Blackjack game, clocking in over 12 hours each day, totaling at least 36 hours. The code is 450 lines long, including comments and whitespace, and consists of four main classes: Cards, Deck, Hand, and Blackjack. While I didn't reach the 750-line goal, I focused on meeting all required STL container and algorithm specifications rather than adding features like multiplayer support, which could've pushed the line count higher by including additional player hands and turn management logic. Time constraints kept me from expanding the code further.

Where on github is it located?

GitHub

APPROACH TO DEVELOPMENT

For version control I used my home whiteboard to map the big picture, a phase1 version getting all the functions/methods on the table, phase2 was the implementation into classes, and phase3 was the conversion utilizing the STL library. I also created a .txt file that, after reviewing the code, located areas of implementation along with the ideas to implement.

GAME RULES

Everyone knows how to play blackjack.

SAMPLE INPUT/OUTPUT

```
The dealers hand: [Hidden]... Queen of Spades...

Your hand: 7 of Hearts... 8 of Clubs...

Hit or Stand? (h/s) h

You add: 7 of Hearts... 8 of Clubs... 4 of Diamonds...

Hit or Stand? (h/s) s

Awesome your total card value is: 19

Dealers hand: 6 of Hearts... Queen of Spades...

Dealer adds: 6 of Hearts... Queen of Spades... 3 of Clubs...

You win!

Game Statistics:

Player: 1 wins

Dealer: 0 wins

Tie: 0 wins

Leading: Player with 1 wins

Play again? (y/n): n
```

CHECKOFF SHEET

1. Containers (Where in code did you put each of these Concepts and how were they used?)

- Sequences (At least 1)
  - (LISTS) I utilized the std::list sequence container in Hand::cards to store each player's and dealer's cards, allowing dynamic insertion of new cards during gameplay. It was also used in Deck::backupDeck to maintain a copy of the deck, copied back when the deck runs out, ensuring smooth reshuffling.
- Associative Containers (At least 2)
  - (SETS) I utilized the std::set associative container in Deck::dealtCards to keep track of cards that have been dealt during the game. It ensures no duplicate cards are dealt by storing unique Cards objects, organized by their suit and value for efficient lookup and insertion.
  - (MAPS) I utilized the std::map associative container in Cards::valueNames and Cards::suitNames to map card values (e.g., 1 to "Ace") and suits (e.g., 'H' to "Hearts") for display, and in Blackjack::stats to track game outcomes (Player, Dealer, Tie wins). It allowed efficient key-value lookups to print card details and update/display win statistics during gameplay.
- Container Adapters (At least 2)
  - (STACK) I utilized the std::stack container adaptor in Deck::cardsStack to create an alternate deck for dealing cards in the game. It was used to pop cards from the top during Deck::deal(), mimicking a real deck's last-in, first-out behavior, and was refilled from the shuffled std::deque when empty.
  - (QUEUE) I utilized the std::queue container adaptor in Blackjack::turnOrder to manage the sequence of player and dealer turns in the game. It was used to push pointers to Hand objects (&player, &dealer) and maintain a first-in, first-out order, though the game's fixed turn structure limited its active use.

2. Iterators (Describe the iterators utilized for each container)

- Concepts
  - (INPUT ITERATOR) This was utilized in std::map for traversing Blackjack::stats to display game statistics and in Cards::valueNames/suitNames for looking up card display names.
  - (OUTPUT ITERATOR) This was utilized in std::copy for Deck::backupDeck, using std::back_inserter to insert cards into the std::list backup deck.
  - (BYDIRECTIONAL ITERATOR) This was utilized in std::list for Hand::cards during Hand::display and Hand::getValue to iterate through cards and in Deck::backupDeck for copying.
  - (RANDOM ACCESS ITERATOR) This was utilized in std::deque for Deck::cards during std::shuffle to randomly access and reorder cards efficiently.

3. Algorithms (Choose one from each)

- Non-mutating algorithms
  - (FIND_IF) I utilized the non-mutating algorithm std::find_if in Hand::getValue to locate Aces in the std::list of cards. It was used to iterate through the card list, checking each card's value with a lambda to identify Aces (value == 1) for proper scoring adjustments.
- Mutating algorithms
  - (RANDOM) I utilized the mutating algorithm std::shuffle in Deck::shuffle to randomize the order of cards in the std::deque deck. It was used with a std::mt19937 random number generator seeded by std::random_device to ensure a unique, fair shuffle each time the deck is reset.
- Organization
  - (MAX_ELEMENT) I utilized the organization algorithm std::max_element in Blackjack::play to find the key-value pair with the highest win count in the std::map of game statistics.

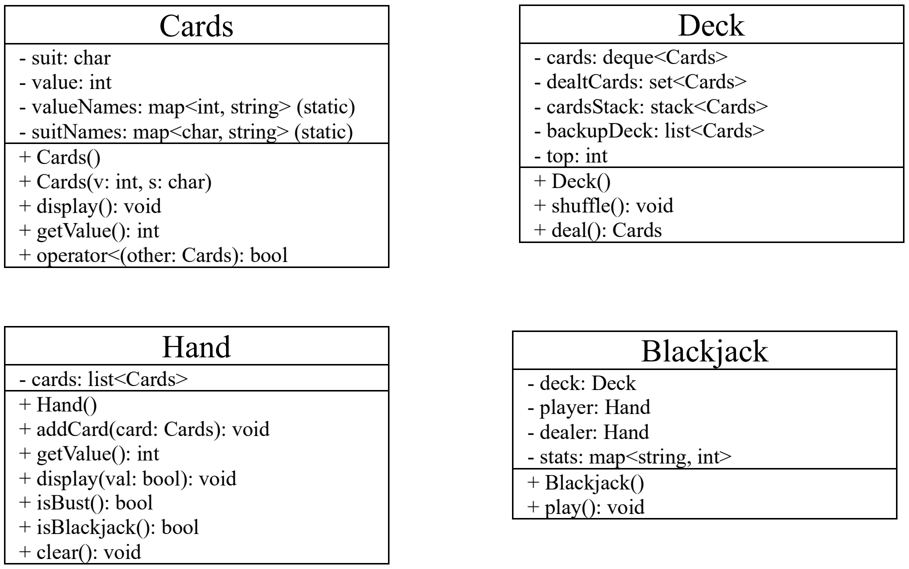
DOCUMENTATOIN OF CODE

2. Pseudo-Code

```
Initialize Blackjack game
  Create Deck object
  Create Hand objects for player and dealer
  Initialize stats map for Player, Dealer, Tie wins
  Shuffle deck
Function PlayGame:
  While player wants to play:
    Clear player and dealer hands
    Create list of hands for initial deal: [player, dealer, player, dealer]
    For each hand in list:
      Deal card from deck to hand
    Display dealer's hand (hide first card)
    Display player's hand
    If player does not have Blackjack:
      While player chooses to hit and not bust:
        Deal card to player
        Display player's hand
        If player busts:
          Output "Bust, You lost!"
          Increment Dealer wins
          Break
        If player has Blackjack:
          Break
    If player not bust:
      Display dealer's full hand
      While dealer's hand value < player's or < 17:
        Deal card to dealer
        Display dealer's hand
      If dealer busts:
        Output "Dealer busts! You win!"
        Increment Player wins
      Else if player has Blackjack and dealer does not:
        Output "Blackjack! You win!"
        Increment Player wins
      Else if dealer has Blackjack and player does not:
        Output "Dealer has Blackjack! You lose!"
        Increment Dealer wins
      Else if player's hand value > dealer's:
        Output "You win!"
        Increment Player wins
      Else if player's hand value < dealer's:
        Output "You lose!"
        Increment Dealer wins
      Else:
        Output "Push! It's a tie"
        Increment Tie wins

    Display stats (Player, Dealer, Tie wins)
    Find and display leading player using max_element
    Ask if player wants to play again
```

3. UML



```
Cards
- suit: char
- value: int
- valueNames: map<int, string> (static)
- suitNames: map<char, string> (static)
+ Cards()
+ Cards(v: int, s: char)
+ display(): void
+ getValue(): int
+ operator<(other: Cards): bool
```

```
Deck
- cards: deque<Cards>
- dealtCards: set<Cards>
- cardsStack: stack<Cards>
- backupDeck: list<Cards>
- top: int
+ Deck()
+ shuffle(): void
+ deal(): Cards
```

```
Hand
- cards: list<Cards>
+ Hand()
+ addCard(card: Cards): void
+ getValue(): int
+ display(val: bool): void
+ isBust(): bool
+ isBlackjack(): bool
+ clear(): void
```

```
Blackjack
- deck: Deck
- player: Hand
- dealer: Hand
- stats: map<string, int>
+ Blackjack()
+ play(): void
```