Project 1

Blackjack

Christian Villanueva
CSC 17C - 43324
6 June 2021

Introduction:

I have created a program that runs a simple version of the card game Blackjack from scratch. This version of Blackjack is the card aspect only. I have not yet implemented a chip system for betting, nor have I included the ability to split when a pair is drawn. Blackjack is my favorite card game, and it seemed fitting to make, given the requirements for the project, and my prerequisite knowledge of the game's rules. The entire program was created from the ground up, and no reference code was used until the final stage of the project, implementing trees, graphs, and heap sort.

The player is dealt two cards and one of the dealer cards is shown. The player is then given the choice to "hit" or "stay". If the player chooses to hit, then the player is given another card. If the player's card total (each card is given a number value, and is added up to find the total) hits or exceeds 21, they automatically stay. The dealer then reveals their second card. After the dealer reveals their second card, if their total is less than or equal to sixteen, they draw until their total exceeds sixteen. The card total of the dealer and the player are compared, and whichever total is closer to 21 wins, given that the total does not exceed 21 (if the player or dealer exceeds a total of 21, they lose the game). If the difference from 21 is the same or if both the dealer and player exceed 21 or "bust" then the game results in a draw. If the player chooses to stay, then their total remains, and the dealer continues the same as previously described.

Summary:

Total Lines: 1,678

Lines of Comments: 211

Number of Classes: 8

Number of Variables: 58

This project iteration took me about three days to complete, building off of my own Blackjack game from the past. Roughly 10 hours were put into the project including the documentation. The most challenging aspect of the project was implementing all of the trees and graphs into the program. I ultimately had to settle on replacing already existing containers and creating a distance problem at the end. Although I functionally understand how they work, implementing them into my project was difficult, and I ultimately had to come up with places to put them, despite the fact that the game is functional without them.

Github Link:

https://github.com/cv2808089/Villanueva_Christian_CIS_17/tree/master/CSC%2017C/Projects/Project%201

Description:

The game initially started as just a program that drew cards and output them. The card values and the ability to total them were then added, as well as the conditions for winning the game. Next came checks to ensure that repeat cards did not show up. After this came the rest of the concepts from the class up until this point such as pointers and binary files. After this came the splitting of the project into multiple source files, followed by conversion into classes. After converting into classes, I added exceptions, static variables and utilized a part of the STL. The next version added the copy constructor, and other minor tweaks. Following this was the addition of lists to the project. The next version added stacks, maps, and queues, which ultimately made the game more efficient, removing the need to check for repeat cards every time someone drew a card. The next concepts implemented were sets, and a sorting algorithm to find the min and max values played. After this came recursive sorts, where I replaced another sorting function I had with heap sort. The following version implemented AVL trees, where I took the existing program from the class github and the modifications made to it for homework, and used it to replace another container in the program.
The final version added graphs and hashing. Graphs were used to determine the minimum distance to travel home after leaving the casino. Hashing was used to check if an array was working correctly, by providing a means to check for a certain string quickly.

References:

AVL Tree:

https://github.com/ml1150258/2021_Spring_CSC_CIS_17c/tree/master/Class/AVLTreeTest

Heap Sort:

https://www.geeksforgeeks.org/heap-sort/

Graphs:

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-set-in-stl/

```
The dealers shown card is: Four Spades

Your cards are:
Ace Diamonds
King Hearts

Would you like the Ace to be worth 1 or 11?
11
Would you like to hit or stay?
stay

The Dealer's second card is:
Three Hearts

The dealer draws a card
The next card is
Five Clubs

The dealer draws a card
The next card is
Ten Hearts

Dealer's 22 vs Player's 21
Dealer busts
You win

The winning hand is:
King Hearts
Ace Diamonds

6 cards were played this game
The highest card value played this game was: Eleven
The lowest card value played this game was: One

Hope you enjoyed blackjack

Now its time to go home, finding the shortest distance home using the 5 airports available
Distance:       City:
0               Las Vegas, Nevada
737             Helena, Montana
2644            Quebec, Canada
986             Oklahoma City, Oklahoma
2230            New York, New York

RUN SUCCESSFUL (total time: 5s)
```

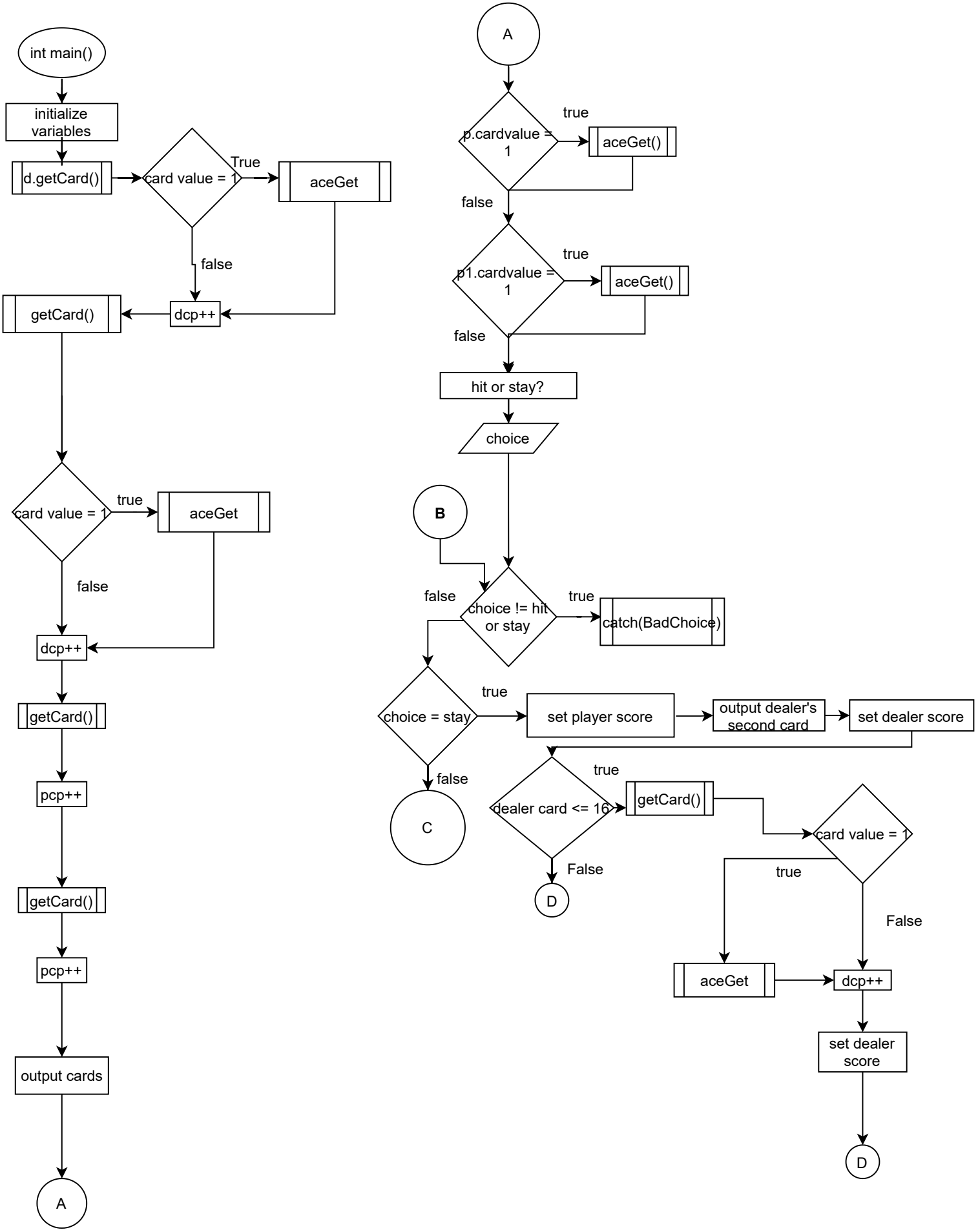
```
The dealers shown card is: Ace Diamonds

Your cards are:
Ten Hearts
Six Diamonds

Would you like to hit or stay?
hit
You drew Jack Spades

The Dealer's second card is:
Eight Spades

Dealer's 19 vs Player's 26
You bust
You lose

The winning hand is:
Ace Diamonds
Eight Spades

5 cards were played this game
The highest card value played this game was: Eleven
The lowest card value played this game was: One

Hope you enjoyed blackjack

Now its time to go home, finding the shortest distance home using the 5 airports available
Distance:       City:
0               Las Vegas, Nevada
737             Helena, Montana
2644            Quebec, Canada
986             Oklahoma City, Oklahoma
2230            New York, New York

RUN SUCCESSFUL (total time: 3s)
```

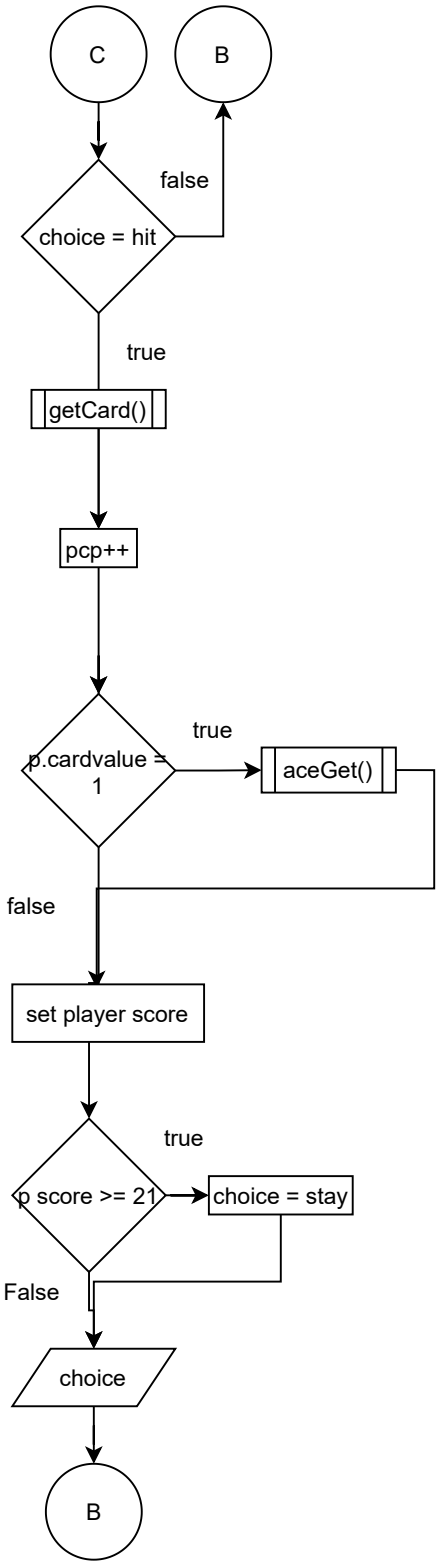
Containers	File	Line
List	hand.h	16
Map	main.cpp	359
Stack	blackjack_imp	346
Queue	main.cpp	415
Set	blackjack_imp	357
Iterators		
Input	hand.h	71
Output	hand.h	57
Bidirectional	main.cpp	398
Random Access	main.cpp	364
Algortihms		
Count	main.cpp	449
Random_shuffle	blackjack_imp	347
Sort	blackjack_imp	398
Concepts		
Recursion	AVLTree.h	137
Recursive Sort	blackjack_imp	438
Hashing	main.cpp	462
Tree	main.cpp	423
Graph	main.cpp	461

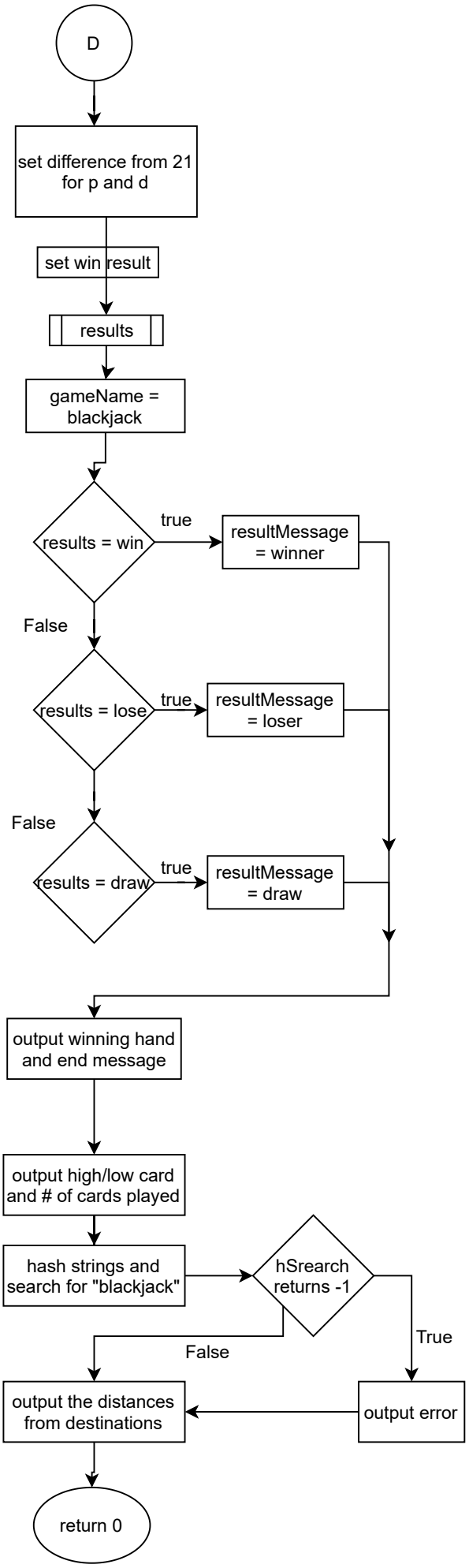
Variable Type	Name	Line	NOTE: LINE IS IN MAIN UNLESS OTHER FILE IS STATED
int	cardValue	card.h, 12	
	cip	hand.h, 12	
	score	hand.h, 14	
	df21	hand.h, 18	
	aceResult	player.h, 9	
	chk	43	
	dcp	51	
	pcp	53	
	aceChoice	55	
	ac	player.cpp, 8	
	hsh	457	
	largest	357	<- in blackjack_implementation.cpp
	l	358	<- in blackjack_implementation.cpp
	r	359	<- in blackjack_implementation.cpp
	h	481	<- in blackjack_implementation.cpp
	l_height	AVLTree.h: 48, 61	
	r_height	AVLTree.h: 49, 62	
	max_height	AVLTree.h: 50, 63	
	bal_factor	AVLTree.h: 118	
	V	Graph.h: 11	
	weight	Graph.h: 78	
unsigned int	hash	469	<- in blackjack_implementation.cpp
string	cardName	card.h, 11	
	suit	card.h, 13	
	choice	49	
enum	gameResult	21	
gameResult	winc	20	<- in blackjack_implementation.cpp
bool	res	hand.cpp: 16, 29, 42	
	lt	card.h: 41, 50	
Card	hold	77	<- in blackjack_implementation.cpp
	hold	hand.cpp, 7	
Hand	Dealer	47	
player	p1	45	
list<Card>	cards	hand.h, 16	
	played	393	<- in blackjack_implementation.cpp
list<string>	arr	459	
list<pair<int, int>>	adj	Graph.h: 15	
vector<Card>	deck	76	<- in blackjack_implementation.cpp
	gameName	367	
vector<int>	dist	Graph.h: 49	
stack<Card>	deck2	346	<- in blackjack_implementation.cpp
	deck	59	
map<int, char>	resultMessage	364	

queue<Card>	winningHand	432				
set<int>	played	397		<- in blackjack_implementation.cpp		
set<paor<int, int>	setds	Graph.h: 45				
iterator	it	hand.h: 27, 34, 42, 49, 57, 64, 71, 81, 92				
	i	364	Graph.h: 72			
AVLTree<Card>	winningHand	425				
string	arr2	458				
	val	466				
	cities	Graph.h: 100				
Graph	g	473				
BNTnode<T>	root	AVLTree.h: 20				
	temp	AVLTree.h: 72, 84, 107				
	left	BNTnode.h: 16				
	right	BNTnode.h: 17				
T	data	BNTnode.h: 15				

Note: I use abbreviations for variable names




C
B
false
choice = hit
true
getCard()
pcp++
p.cardvalue = 1
true
aceGet()
false
set player score
p score >= 21
true
choice = stay
False
choice
B


D
set difference from 21 for p and d
set win result
results
gameName = blackjack
results = win
true
resultMessage = winner
False
results = lose
true
resultMessage = loser
False
results = draw
true
resultMessage = draw
output winning hand and end message
output high/low card and # of cards played
hash strings and search for "blackjack"
hSrearch returns -1
True
False
output the distances from destinations
output error
return 0

Version 11 Pseudocode

Set random number seed
Define classes and variables

Draw dealer cards
If an ace is drawn, make its value 11 if it doesn't bust the dealer
Increment number of cards in play

Draw player cards
Increment number of cards in play

Output dealer's first card and player's two cards
If the player draws a(n) ace(s) gets their choice for the aces value(only allows 1 or 11)
Gets the player's choice to hit or stay with input validation

If stay is chosen,
    Calculates player and dealer card total
    Reveals dealer's second card
    If dealer's card total is <= 16,
        draws cards, iterates cards in play
        repeats until a total of >16 is achieved
        If dealer draws ace, sets it to 11 if it doesn't bust the dealer
        Calculates dealer card total
        Output dealer's cards

If hit is chosen, loops until stay is chosen
    Calculates player and dealer card total

Draw card and iterate cards in play

Display card drawn
If ace is drawn, get choice for ace value(only allows 1 or 11)
Add card value to player card total

If player card total >= 21, set choice to stay
If player card total < 21, get choice to hit or stay(with input validation)

Reveals dealer's second card
If dealer's card total is <= 16,
    draws cards, iterates cards in play, and checks
    repeats until a total of >16 is achieved
    If dealer draws ace, sets it to 11 if it doesn't bust the dealer
    Calculates dealer card total
    Output dealer's cards

Find difference of dealer and player total from 21
Find and output game results based on difference

Output the game results

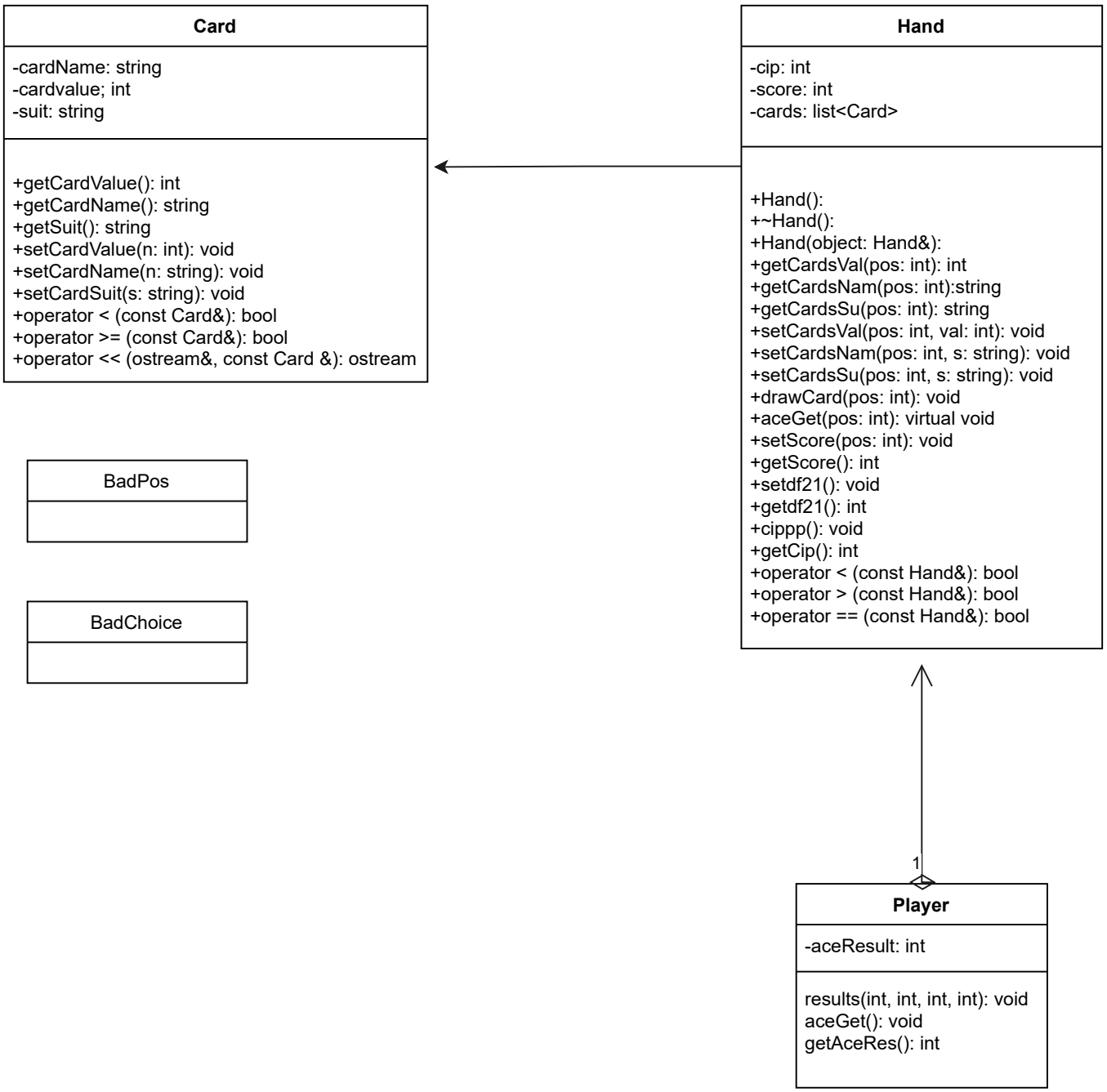
Define map and vector to hold result message and game name
If win result is win, message is winner
If win result is lose, message is loser
If win result is draw, message is draw

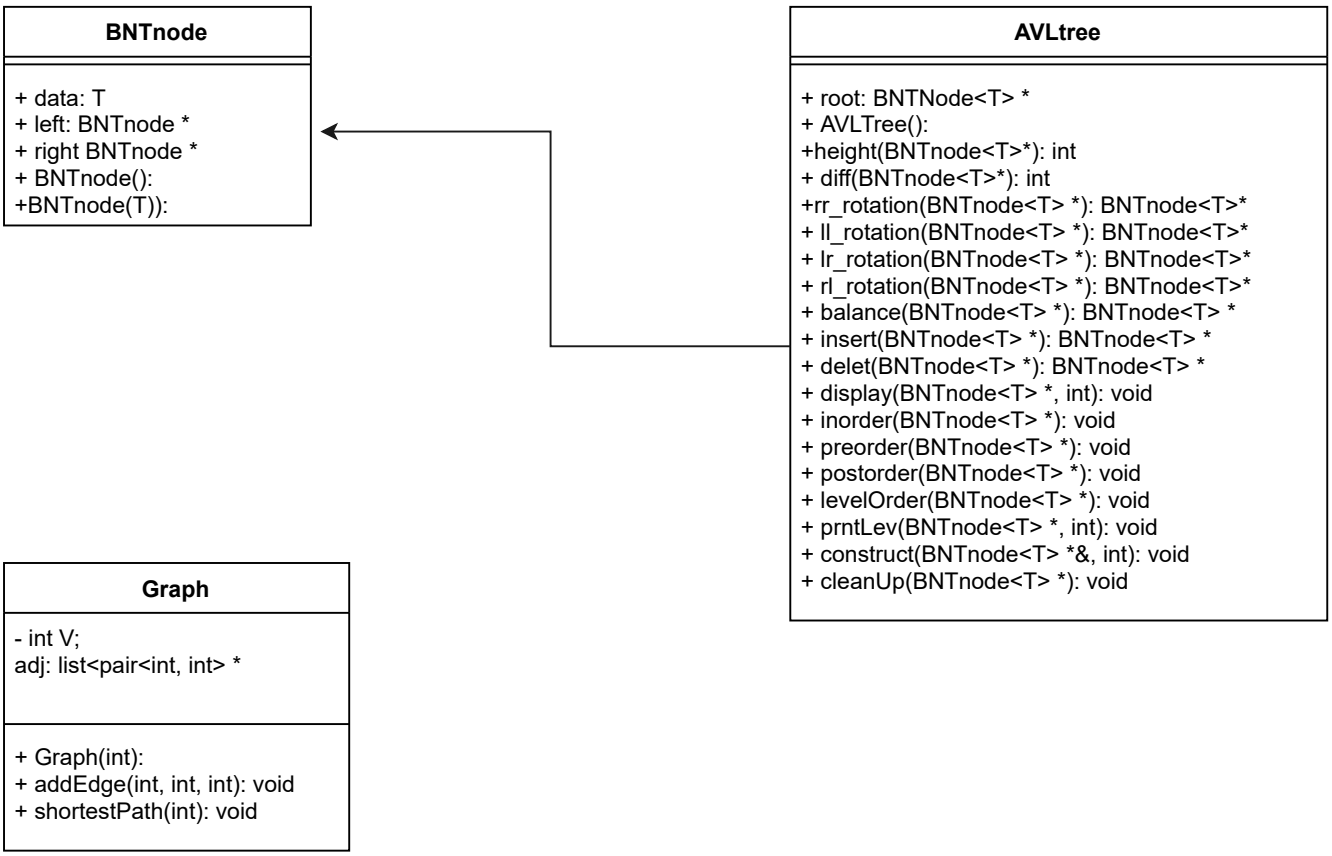
Write the result message to a binary file

Outputs queue of winning hand
Output the number of cards used in the game
Outputs the highest and lowest values played
Checks for "blackjack" in array

Finds and displays shortest distance from destination
Output exit message

Free up the used memory and exit the program


Card
-cardName: string
-cardvalue; int
-suit: string
+getCardValue(): int
+getCardName(): string
+getSuit(): string
+setCardValue(n: int): void
+setCardName(n: string): void
+setCardSuit(s: string): void
+operator < (const Card&): bool
+operator >= (const Card&): bool
+operator << (ostream&, const Card &): ostream
Hand
-cip: int
-score: int
-cards: list<Card>
+Hand():
+~Hand():
+Hand(object: Hand&):
+getCardsVal(pos: int): int
+getCardsNam(pos: int):string
+getCardsSu(pos: int): string
+setCardsVal(pos: int, val: int): void
+setCardsNam(pos: int, s: string): void
+setCardsSu(pos: int, s: string): void
+drawCard(pos: int): void
+aceGet(pos: int): virtual void
+setScore(pos: int): void
+getScore(): int
+setdf21(): void
+getdf21(): int
+cippp(): void
+getCip(): int
+operator < (const Hand&): bool
+operator > (const Hand&): bool
+operator == (const Hand&): bool
BadPos
BadChoice
1
Player
-aceResult: int
results(int, int, int, int): void
aceGet(): void
getAceRes(): int


BNTnode
+ data: T
+ left: BNTnode *
+ right BNTnode *
+ BNTnode():
+BNTnode(T)):
AVLtree
+ root: BNTNode<T> *
+ AVLTree():
+height(BNTnode<T>*): int
+ diff(BNTnode<T>*): int
+rr_rotation(BNTnode<T> *): BNTnode<T>*
+ ll_rotation(BNTnode<T> *): BNTnode<T>*
+ lr_rotation(BNTnode<T> *): BNTnode<T>*
+ rl_rotation(BNTnode<T> *): BNTnode<T>*
+ balance(BNTnode<T> *): BNTnode<T> *
+ insert(BNTnode<T> *): BNTnode<T> *
+ delet(BNTnode<T> *): BNTnode<T> *
+ display(BNTnode<T> *, int): void
+ inorder(BNTnode<T> *): void
+ preorder(BNTnode<T> *): void
+ postorder(BNTnode<T> *): void
+ levelOrder(BNTnode<T> *): void
+ prntLev(BNTnode<T> *, int): void
+ construct(BNTnode<T> *&, int): void
+ cleanUp(BNTnode<T> *): void
Graph
- int V;
adj: list<pair<int, int> *
+ Graph(int):
+ addEdge(int, int, int): void
+ shortestPath(int): void