Appendix B

Faculty of Science and Technology

2017/2018

Level 4

Introduction To Programming

Assignment 2

Black Jack Program

Analysis, Design, and Implementation

Software Report

Template

Student number: 4922675
Maciej Legas

1. Self-Assessment of Performance 5%

Tutor : Andrew Watson

Student ID	4922675

Circle the appropriate response:

Did I submit the assignment on time?	**Yes**	**No**		
Did I complete the assignment?	**Yes**	**No**		
If not, approx. how much did I complete?	**%**			
How happy am I with what I submitted?	**Very happy**	**Satisfied**	**Disappointed**	**Ashamed**
What mark do I expect?	**65%**			
Did I spend enough time on the assignment?	**Yes**	**No**		
Did I get it proof-read by someone else?	**Yes**	**No**		
Have I properly 'referenced' it?	**Yes**	**No**		
Could I improve the presentation?	**Yes**	**No**		

Answer the following questions:

The best part of my performance was:	The code
The worst part of my performance was:	The documentation about the code
One way in which I could improve the content of my assignment is:	Create a save/load mechanic for the player's balance
One way in which I could improve the presentation of my assignment is:	Increase the amount of flowcharts
One thing I will do to improve my performance in my next assignment is:	To try to modulate and optimise written code further
Another thing I will do to improve my performance in my next assignment is:	Analyse the problem more thoroughly in the document

2. Analysis : Marks 20%

The program is supposed to allow one human player to play a game of Blackjack against a computer-processed Dealer. I began by creating pseudocode of the whole program. In order to do that, I initially needed to have game logic written down. As a base for improvement, I took the game loop mechanic from *Beginning C++ Through Game Programming, Second Edition* (Dawson 2007), and for comparison I paste it below.

```
Deal players and the house two initial cards
Hide the house's first card
Display players' and house's hands
Deal additional cards to players
Reveal house's first card
Deal additional cards to house

If house is busted
    Everyone who is not busted wins

Otherwise
    For each player
        If player isn't busted
            If player's total is greater than the house's total
                Player wins
            Otherwise if player's total is less than house's total
                Player loses
            Otherwise
                Player pushes
Remove everyone's cards
```

Of course, this game loop contains only the most important and basic aspects of the game, and therefore needs to be enhanced as I only need one player present in the game except the dealer, and there is no bet and balance system implemented which is required. I firstly wrote down the entire list of operations the program has to do to see how can it be modulated:

```
BEGIN
OUTPUT A welcome message and the rules of the "casino"
```


Student number: 4922675
Maciej Legas

OUTPUT Query the player to input any key to continue
INPUT A key press from the player
INITIALIZE A deck of cards
Fill the deck with cards
Shuffle the deck of cards
INITIALIZE An initial balance of $10000 for the player
Start the game loop
WHILE The game is played
    OUTPUT Query the player how much does he wish to bet
    INPUT The bet from the player
    OUTPUT A confirmation of taking the bet from the balance
    INITIALIZE The player's hand and the dealer's hand
    Draw two cards each for the dealer's and player's hand and add the values of cards to each hand's strength
        IF All cards have been used from the deck
            Shuffle the deck before drawing the card
        END IF
        IF An ace has been drawn for the first time in a hand
            Store the location of the first ace in a hand
        END IF
    OUTPUT An interface showing:
        The dealer's hand, with its strength (except for the hidden card)
        The player's hand, with its strength
        Balance and bet
    IF The player has a natural blackjack (two initial cards' strength resulting in 21)
        Show the dealer's hidden card and compare the dealer's hand strength with the player's
        IF The dealer has a blackjack as well
            Output a push outcome
        ELSE
            Output a win outcome
        END IF
    END IF
    WHILE The player is not busted, is deciding whether to hit or stand, or his hand strength is not equal to 21
        OUTPUT Query the player if he wants to hit or stand
        INPUT Player's decision

Student number: 4922675
Maciej Legas

IF The player decides to hit
Draw a card for the player and add its value to the player's hand strength
IF All cards have been used from the deck
Shuffle the deck before drawing the card
END IF
IF The player has an ace which has a value of 11, and will go over 21 with drawing the next card
Set the value of that ace to 1
END IF
IF The drawn card is an ace, and the player had no aces in his hand before
Save the location of that ace
IF The hand strength is over 10
Set the value of the drawn ace to 1
END IF
END IF
ELSE
IF The player decides to stand
Reveal the dealer's hidden card
ELSE
OUTPUT Query the player to input again
END IF
END WHILE
IF The player busts
OUTPUT A lose outcome
END IF
WHILE The dealer hand strength is less than 17, or if the dealer has an ace with a value of 11 and his hand strength is equal to 17
Draw a card for the dealer
IF All cards have been used from the deck
Shuffle the deck before drawing the card
END IF
IF The dealer has an ace which has a value of 11, and will go over 21 with drawing the next card
Set the value of that ace to 1
END IF
IF The drawn card is an ace, and the dealer had no aces in his hand before
Save the location of that ace

Student number: 4922675
Maciej Legas

```
                IF The hand strength is over 10
                    Set the value of the drawn ace to 1
                END IF
            END IF
            Add the card's value to the dealer's hand strength
        END WHILE
        IF The dealer busts
            OUTPUT A win outcome
        ELSE
        IF The player has the same hand strength as the dealer
            OUTPUT A push outcome
        ELSE
        IF The player has a higher hand strength than the dealer and is not busted
            OUTPUT A win outcome
        ELSE
        IF The dealer has a higher hand strength than the player and is not busted
            OUTPUT A lose outcome
        END IF
        IF The outcome is win
            OUTPUT A winning message
            Add the won amount to balance
        ELSE
        IF The outcome is push
            OUTPUT A draw message
            Return the bet amount to balance
        ELSE
        IF The outcome is lose
            OUTPUT A lose message
        ELSE
        IF The outcome is blackjack
            OUTPUT A blackjack outcome
        END IF
        IF The player has a balance above zero
            OUTPUT Query the player if he wishes to play again
            INPUT Player's answer
                IF Yes
                    Repeat the game loop
                ELSE
```

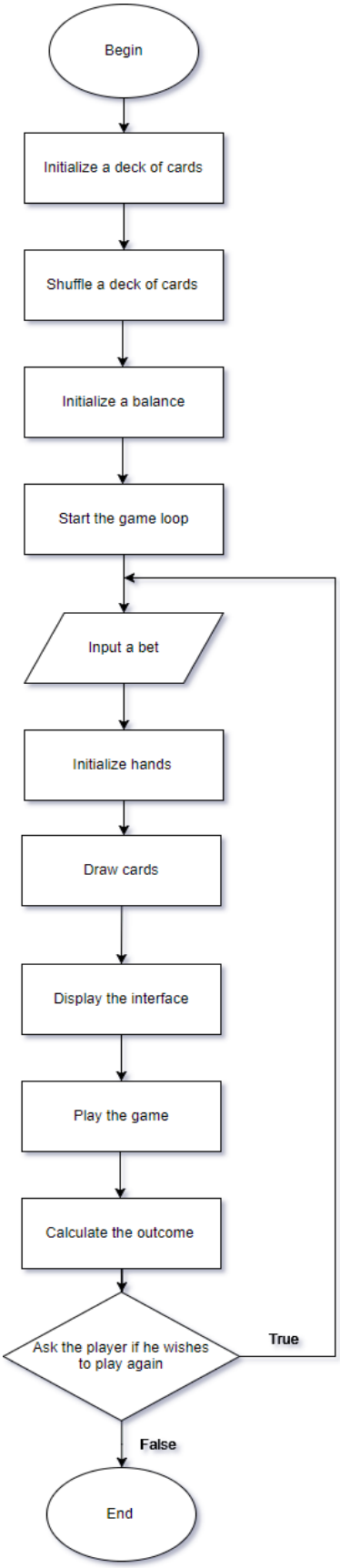

Student number: 4922675
Maciej Legas

```
                IF No
                    Exit the game loop
                ELSE
                    Ask the player again
                END IF
        ELSE
            OUTPUT Game Over
        END IF
END WHILE
END
```

From analysing the pseudocode above, it was possible to break down the program into modules. The modules are the larger operations taken from the pseudocode:

- Initializing a deck of cards
- Shuffling a deck of cards
- Game loop itself
- Betting system
- Drawing a card
- Interface system
- Calculating the outcome
- Asking the player if he wishes to play again

Here is a simplified flowchart of the whole program:



Student number: 4922675
Maciej Legas

I paste the pseudocode again, highlighting what can be modulated out of the code.

```
BEGIN
OUTPUT A welcome message and the rules of the “casino”
OUTPUT Query the player to input any key to continue
INPUT A key press from the player
INITIALIZE A deck of cards
Fill the deck with cards -> FUNCTION CreateDeck
Shuffle the deck of cards -> FUNCTION ShuffleCards
INITIALIZE An initial balance of $10000 for the player
Start the game loop -> FUNCTION GameLoop
WHILE The game is played
    OUTPUT Query the player how much does he wish to bet
    INPUT The bet from the player
    OUTPUT A confirmation of taking the bet from the balance
    INITIALIZE The player’s hand and the dealer’s hand
    Draw two cards each for the dealer’s and player’s hand and add the values of cards to each hand’s strength
        IF All cards have been used from the deck
            Shuffle the deck before drawing the card
        END IF
        IF An ace has been drawn for the first time in a hand
            Store the location of the first ace in a hand
        END IF
    OUTPUT An interface showing:
        The dealer’s hand, with its strength (except for the hidden card)
        The player’s hand, with its strength
        Balance and bet
    IF The player has a natural blackjack (two initial cards’ strength resulting in 21)
        Show the dealer’s hidden card and compare the dealer’s hand strength with the player’s
        IF The dealer has a blackjack as well
            Output a push outcome
        ELSE
            Output a blackjack outcome
        END IF
```

-> FUNCTION InputBet

-> FUNCTION DrawCard

-> FUNCTION PrintOutHUD

Student number: 4922675
Maciej Legas

END IF

WHILE The player is not busted, is deciding whether to hit or stand, or his hand strength is not equal to 21

OUTPUT Query the player if he wants to hit or stand

INPUT Player's decision

IF The player decides to hit

Draw a card for the player

IF All cards have been used from the deck

Shuffle the deck before drawing the card -> FUNCTION ShuffleCards

END IF

IF The player has an ace which has a value of 11, and will go over 21 with drawing the next card

Set the value of that ace to 1

END IF

IF The drawn card is an ace, and the player had no aces in his hand before

Save the location of that ace

IF The hand strength is over 10

Set the value of the drawn ace to 1

END IF

END IF

Add the card's value to the player's hand strength

-> FUNCTION DrawCard

END WHILE

IF The player decides to stand

Reveal the dealer's hidden card

ELSE

OUTPUT Query the player to input his decision again

END IF

IF The player busts

OUTPUT A lose outcome

END IF

WHILE The dealer hand strength is less than 17, or if the dealer has an ace with a value of 11 and his hand strength is equal to 17

Draw a card for the dealer

IF All cards have been used from the deck

Shuffle the deck before drawing the card -> FUNCTION ShuffleCards

END IF

Student number: 4922675
Maciej Legas

IF The dealer has an ace which has a value of 11, and will go over 21 with drawing the next card

Set the value of that ace to 1

END IF

IF The drawn card is an ace, and the dealer had no aces in his hand before

Save the location of that ace

IF The hand strength is over 10

Set the value of the drawn ace to 1

END IF

END IF

Add the card's value to the dealer's hand strength

-> FUNCTION DrawCard

END WHILE

IF The dealer busts

OUTPUT A win outcome

ELSE

IF The player has the same hand strength as the dealer

OUTPUT A push outcome

ELSE

IF The player has a higher hand strength than the dealer and is not busted

OUTPUT A win outcome

ELSE

IF The dealer has a higher hand strength than the player and is not busted

OUTPUT A lose outcome

END IF

IF The outcome is win

OUTPUT A winning message

Add the won amount to balance

ELSE

IF The outcome is push

OUTPUT A draw message

Return the bet amount to balance

-> FUNCTION CalculateOutcome

ELSE

IF The outcome is lose

OUTPUT A lose message

IF The outcome is blackjack

OUTPUT A winning blackjack message

Add a blackjack payout rate to balance

IF The player has a balance above zero

Student number: 4922675
Maciej Legas

```
            OUTPUT Query the player if he wishes to play again
            INPUT Player's answer
                IF Yes
                    Repeat the game loop
                ELSE
                IF No
                    Exit the game loop
                ELSE
                    OUTPUT Query the player to input his decision again
                END IF
        ELSE
            OUTPUT Game Over
        END IF
END WHILE
END
```

-> FUNCTION PlayAgain

Here is the created list of functions, which will be helpful in designing each module in the Design section:

- FUNCTION CalculateOutcome
- FUNCTION CreateDeck
- FUNCTION DrawCard
- FUNCTION GameLoop
- FUNCTION InputBet
- FUNCTION PlayAgain
- FUNCTION PrintOutHUD
- FUNCTION ShuffleCards

Let's try to investigate further into the code and have an orientation of how the two main functions, main and GameLoop, should look like, to have an easier way of designing the program.

```
FUNCTION main
BEGIN
OUTPUT A welcome message and the rules of the "casino"
OUTPUT Query the player to input any key to continue
INPUT A key press from the player
INITIALIZE A deck of cards
CALL CreateDeck
```


Student number: 4922675
Maciej Legas

```
CALL ShuffleCards
INITIALIZE An initial balance of $10000 for the player
CALL GameLoop
End
```

Automatically, I can see that I need to pass the initialized deck of cards as a parameter to two first functions. The deck of cards and the balance needs to be passed to the game loop for the game loop to operate on them. This information will be useful for designing the program.

```
FUNCTION GameLoop
BEGIN
WHILE The game is played
        CALL InputBet
        INITIALIZE The player's hand and the dealer's hand
        CALL DrawCard
        CALL DrawCard
        CALL DrawCard
        CALL DrawCard

        CALL PrintOutHUD
        IF The player has a natural blackjack (two initial cards' strength resulting in 21)
                Show the dealer's hidden card and compare the dealer's hand strength with the
                player's
                IF The dealer has a blackjack as well
                        Output a push outcome
                ELSE
                        Output a blackjack outcome
                END IF
        END IF
        WHILE The player is not busted, is deciding whether to hit or stand, or his hand
        strength is not equal to 21
            OUTPUT Query the player if he wants to hit or stand
            INPUT Player's decision

            IF The player decides to hit
                CALL DrawCard
            ELSE
            IF The player decides to stand
                Reveal the dealer's hidden card
```


Student number: 4922675
Maciej Legas

```
        ELSE
            OUTPUT Query the player to input his decision again
        END IF
    END WHILE

    IF The player busts
            OUTPUT A lose outcome
    END IF
    WHILE The dealer hand strength is less than 17, or if the dealer has an ace with a
    value of 11 and his hand strength is equal to 17
            CALL DrawCard
    END WHILE
    IF The dealer busts
            OUTPUT A win outcome
    ELSE
    IF The player has the same hand strength as the dealer
            OUTPUT A push outcome
    ELSE
    IF The player has a higher hand strength than the dealer and is not busted
            OUTPUT A win outcome
    ELSE
    IF The dealer has a higher hand strength than the player and is not busted
            OUTPUT A lose outcome
    END IF
    CALL CalculateOutcome
    CALL PlayAgain
END WHILE
END
```

Having the main functions planned out, with the basic inputs, it's time to move onto the design and expand on the functions.

Student number: 4922675
Maciej Legas

3. Design : Marks 30%

To see the needed inputs and outputs, I needed to describe the usage of each function initially.

FUNCTION CalculateOutcome

Depending on the outcome of the game, outputs an appropriate message and makes changes to player's balance.

FUNCTION CreateDeck

Fills a deck of cards with cards.

FUNCTION DrawCard

Draws a card from the deck to the passed hand.

FUNCTION GameLoop

Loops the game as long as the player wishes to play, making use of the player's balance and the deck of cards.

FUNCTION InputBet

Allows the player to input a bet from the balance.

FUNCTION PlayAgain

Asks the player if he wishes to play again.

FUNCTION PrintOutHUD

Outputs the interface showing:

The dealer's hand, with its strength (except for the hidden card if the player is not standing)

The player's hand, with its strength

Balance and bet.

FUNCTION ShuffleCards

Shuffles the cards.

Let's go into detail with each module except the game loop and main, so there is a clear sight of which variables will be needed in those functions.

```
FUNCTION CalculateOutcome
Pass in: integer _bet, integer *_balanceptr, Result _outcome

CASE Based on _outcome
    CASE m_win
        OUTPUT A message stating that the player won and stating his
        winnings
        *_balanceptr = *_balanceptr + (_bet * 2)
        OUTPUT *_balanceptr
    CASE m_push
        OUTPUT A message stating that the player drew with the dealer
```

```
                and that his bet is returned
                *_balanceptr = *_balanceptr + _bet
                                OUTPUT *_balanceptr
        CASE m_lose
                OUTPUT A message stating that the player lost his bet
                OUTPUT *_balanceptr
        CASE m_blackjack
                OUTPUT A message stating that the player won and stating his
                winnings
                *_balanceptr = *_balanceptr + (_bet * 1.5)
                OUTPUT balance
        CASE default
                OUTPUT “An exception has been found”
END CASE

Pass out: Nothing
END FUNCTION
```

As seen from the module, it is possible to return balance back by the function, but just to ensure that there is a change to the balance I decided to use a pointer for it. For the outcome, I decided to create an enum to have a controlled and readable way of setting the result instead of using an integer.

```
        FUNCTION CreateDeck
        Pass in: Card _deck[]
        INITIALIZE integer suit = 0
        INITIALIZE integer card = 0

        WHILE suit < 52
                WHILE card < 13

                IF card == 0
                        _deck[suit + card].m_cardValue = 11
                ELSE
                IF card > 9
                        _deck[suit + card].m_cardValue = 10
                ELSE
                IF card > 0 && card < 10
                        _deck[suit + card].m_cardValue = card + 1

                CASE Based on suit
                        CASE 0
                                Set _deck[suit + card].m_cardSuit as “Clubs”
                        CASE 13
                                Set _deck[suit + card].m_cardSuit as “Hearts”
                        CASE 26
                                Set _deck[suit + card].m_cardSuit as “Diamonds”
                        CASE 39
```

```
                    Set _deck[suit + card].m_cardSuit as “Spades”
                CASE default
                    OUTPUT An exception has been found
            END CASE

            CASE Based on card
                CASE 0
                    Set _deck[suit + card].m_cardName as “Ace”
                CASE 1
                    Set _deck[suit + card].m_cardName as “Two”
                CASE 2
                    Set _deck[suit + card].m_cardName as “Three”
                CASE 3
                    Set _deck[suit + card].m_cardName as “Four”
                CASE 4
                    Set _deck[suit + card].m_cardName as “Five”
                CASE 5
                    Set _deck[suit + card].m_cardName as “Six”
                CASE 6
                    Set _deck[suit + card].m_cardName as “Seven”
                CASE 7
                    Set _deck[suit + card].m_cardName as “Eight”
                CASE 8
                    Set _deck[suit + card].m_cardName as “Nine”
                CASE 9
                    Set _deck[suit + card].m_cardName as “Ten”
                CASE 10
                    Set _deck[suit + card].m_cardName as “Jack”
                CASE 11
                    Set _deck[suit + card].m_cardName as “Queen”
                CASE 12
                    Set _deck[suit + card].m_cardName as “King”
                CASE default
                    OUTPUT An exception has been found
            END CASE

            card +=1
            END WHILE
        suit +=13
        END WHILE

        Pass out: Nothing
        END FUNCTION
```

The _deck array requires to use a struct type, allowing it to contain such needed properties of a card like its name, suit and value. Aces are set to a value of 11 by default, but the value of the aces actually varies in-game (1 or 11). The code has been inspired by code created on one of the programming workshops.

```
FUNCTION DrawCard
Pass in: Card _deck[], Card _hand[], integer *_deckCardCount, integer
*_handCardCount, integer *_handStrength, integer *_firstAceLocation,
boolean *_isFirstAceSet

IF *deckCardCount >= 51
        OUTPUT A message stating that all cards have been used
        CALL ShuffleCards(_deck)
        *_deckCardCount = 0
END IF

_hand[*_handCardCount].m_cardValue =
_deck[*_deckCardCount].m_cardValue

Set _hand[*_handCardCount].m_cardName to
_deck[*_deckCardCount].m_cardName

Set _hand[*_handCardCount].m_cardSuit to
_deck[*_deckCardCount].m_cardSuit

IF _hand[*_handCardCount].m_cardValue == 11
        IF *_isFirstAceSet == false && *_handStrength < 10
                                        *_firstAceLocation = *_handCardCount
                                        *_isFirstAceSet = true
        ELSE
        IF *_handStrength > 10
                _hand[*_handCardCount].m_cardValue = 1
        END IF
END IF

IF *_handStrength + _hand[*_handCardCount].m_cardValue) > 21 &&
*_isFirstAceSet == true && _hand[*_firstAceLocation].m_cardValue != 1
                _hand[*_firstAceLocation].m_cardValue = 1
                *_handStrength -= 10
END IF

*_handStrength += _hand[*_handCardCount].m_cardValue
*_deckCardCount = *_deckCardCount + 1
*_handCardCount = *_handCardCount + 1

Pass out: Nothing
END FUNCTION
```

This module actually does quite a few things instead of just drawing cards, such as:

- Shuffles the deck again if the deck runs out of cards
- Assigns a correct value of an ace depending on the hand's strength

- If the passed hand has an ace and its value is 11, and the drawn card will make the passed hand go bust, it changes the value of that ace to 1
- Adds the card's value to the hand's strength

All of these operations are based in this module as the module requires these parameters for the drawing card process by itself, therefore it is optimal for processing the code.

```
FUNCTION InputBet
Pass in: integer *_balanceptr
INITIALIZE integer bet = 0
INITIALIZE boolean correctBetGiven = false

WHILE correctBetGiven == false
    OUTPUT *_balanceptr
    OUTPUT Query the player how much does he wish to bet
    INPUT bet

    IF bet > *_balanceptr
        OUTPUT A message stating that the bet is higher than the
        balance and query the player to input again
    ELSE
    IF bet <= 0
        OUTPUT A message stating that the bet is an invalid value
        and query the player to input again
    ELSE
    IF bet > 0 && bet <= *_balanceptr
        OUTPUT bet
        *_balanceptr -= bet
        OUTPUT A message stating that the bet has been taken
        from the balance
        correctBetGiven = true
    END IF
END WHILE

Pass out: bet
END FUNCTION
```

In this module I decided to return bet by value, as the function will be called every time at the start of the game loop. It makes it easier to debug in case of invalid values of the bet.

```
FUNCTION PlayAgain
Pass in: integer *_balanceptr, boolean *_quitGame
INITIALIZE char choice
INITIALIZE correctOption = false

IF *_balanceptr <= 0
```

```
            OUTPUT A message stating that the player lost all of his money
            OUTPUT A message stating Game Over
            *_quitGame = true

        ELSE
            WHILE correctOption == false
                OUTPUT A query asking if the player wishes to play again
                INPUT choice

                CASE Based on choice
                    CASE Y
                    CASE y
                        correctOption = true
                    CASE N
                    CASE n
                        OUTPUT A message thanking the player for
                        playing the game
                        correctOption = true
                        *_quitGame = true
                    CASE default
                        OUTPUT A message asking the player to
                        input again
                END CASE
            END WHILE
        END IF

        Pass out: Nothing
        END FUNCTION
```

The PlayAgain module is responsible for the decision whether to repeat the game loop or to quit the game, by operating on the passed boolean *_quitGame.

```
        FUNCTION PrintOutHUD
        Pass in: Card _dealersHand[], Card _playersHand[], integer
*_dealerHandStrength, integer *_playerHandStrength, integer _bet, integer
*_balanceptr, boolean _standCheck

        IF _standCheck == false
            OUTPUT Display the dealer's hand and its strength, with the first
            card hidden using "???" symbols
            OUTPUT Display player's hand and its strength, using a while
            loop to show each card's strength, and then the sum of them
            using *_playerHandStrength
            OUTPUT Display the player's _bet and *_balanceptr
        ELSE
        IF _standCheck == true
```

```
            OUTPUT Display dealer's hand and its strength, using a while
            loop to show each card's strength, and then the sum of them
            using *_dealerHandStrength
            OUTPUT Display player's hand and its strength, using a while
            loop to show each card's strength, and then the sum of them
            using *_playerHandStrength
            OUTPUT Display the player's _bet and *_balanceptr
    END IF

    Pass out: Nothing
    END FUNCTION
```

This module displays the interface for the player, depending whether the player is deciding to hit or stand, is still hitting, or standing. This choice is important since the dealer's card can only be uncovered if it is the dealer's turn.

```
    FUNCTION ShuffleCards
    Pass in: Card _deck[]
    INITIALIZE Card temp = { 0 }
    INITIALIZE integer i = 0
    INITIALIZE integer j = 0
    INITIALIZE integer random

    WHILE i < 10
        WHILE j < 52
            temp = _deck[j]
            Set the integer random to a random integer between
            (including) 0 and 51
            _deck[j] = _deck[random]
            _deck[random] = temp
        END WHILE
    END WHILE

    OUTPUT A message stating that the cards have been shuffled

    Pass out: Nothing
    END FUNCTION
```

This module is quite self-explanatory, as it shuffles the passed deck of cards. I decided to have cards be shuffled ten times.

As I have now all other modules detailed, I have now the needed variables for main and game loop.

Looking back at the pseudocode of the whole program, it seems that these variables used in other modules will be initialized in the main function:

```
Card deck[52]
integer *balanceptr
```

And these in the game loop function:

```
boolean *quitGame
integer *playerHandStrength
integer *dealerHandStrength
integer *deckCardCount
integer *playerHandCount
integer *dealerHandCount
integer *playerFirstAceLocation
integer *dealerFirstAceLocation
boolean *playerIsFirstAceSet
boolean *dealerIsFirstAceSet
boolean standCheck
Card playersHand[11]
Card dealersHand[10]
Result outcome
```

As seen, the modules use a struct called Card and an enumeration called Result. Before heading into writing down the main function as well as the game loop function, let's write them down first.

```
STRUCT Card
    char m_cardSuit[9]
    char m_cardName[6]
    integer m_cardValue
END STRUCT

ENUM Result
    m_null, m_win, m_push, m_lose, m_blackjack
END ENUM

    FUNCTION main
    Pass in: Nothing
    INITIALIZE char anykey

    OUTPUT A welcome message and the rules of the "casino"
    OUTPUT Query the player to input any key to continue
    INPUT anykey

    INITIALIZE integer balance = 10000
    INITIALIZE integer *balanceptr = &balance
    INITIALIZE Card deck[52]

    CALL CreateDeck(deck)
    CALL ShuffleCards(deck)
```

```
CALL GameLoop(balanceptr, deck)

Pass out: Nothing
```

The main function initializes the initial game variables and starts the game loop. After the game loop ends, the program quits.

```
FUNCTION GameLoop
Pass in: integer *_balanceptr, Card _deck[]
INITIALIZE choice
INITIALIZE boolean *quitGame
INITIALIZE integer *playerHandStrength
INITIALIZE integer *dealerHandStrength
INITIALIZE integer *deckCardCount = 0
INITIALIZE integer *playerHandCount
INITIALIZE integer *dealerHandCount
INITIALIZE integer *playerFirstAceLocation
INITIALIZE integer *dealerFirstAceLocation
INITIALIZE boolean *playerIsFirstAceSet
INITIALIZE boolean *dealerIsFirstAceSet

WHILE *quitGame == false
    INITIALIZE Card playersHand[11] = { 0 }
    INITIALIZE Card dealersHand[10] = { 0 }
    INITIALIZE Result outcome = m_null

    CALL InputBet(_balanceptr)
    INITIALIZE boolean playerIsDeciding = true
    INITIALIZE boolean standCheck = false
    INITIALIZE boolean roundOver = false

    *playerHandStrength = 0
    *dealerHandStrength = 0
    *playerHandCount = 0
    *dealerHandCount = 0
    *playerFirstAceLocation = 0
    *dealerFirstAceLocation = 0
    *playerIsFirstAceSet = false
    *dealerIsFirstAceSet = false

    CALL DrawCard(_deck, dealersHand, deckCardCount,
    dealerHandCount, dealerHandStrength, dealerFirstAceLocation,
    dealerIsFirstAceSet)
    CALL DrawCard(_deck, dealersHand, deckCardCount,
    dealerHandCount, dealerHandStrength, dealerFirstAceLocation,
    dealerIsFirstAceSet)
    CALL DrawCard(_deck, playersHand, deckCardCount,
    playerHandCount, playerHandStrength, playerFirstAceLocation,
    playerIsFirstAceSet)
```

```
CALL DrawCard(_deck, playersHand, deckCardCount,
playerHandCount, playerHandStrength, playerFirstAceLocation,
playerIsFirstAceSet)

Clear the screen
OUTPUT A message stating that the initial cards have been
drawn

CALL PrintOutHUD(dealersHand, playersHand,
dealerHandStrength, playerHandStrength, bet, _balanceptr,
standCheck)

IF *playerHandStrength == 21
        OUTPUT A message stating that the player has a natural
        blackjack
        standCheck = true
        CALL PrintOutHUD(dealersHand, playersHand,
        dealerHandStrength, playerHandStrength, bet,
        _balanceptr, standCheck)

        IF *dealerHandStrength == *playerHandStrength
                OUTPUT A message stating that the dealer has a blackjack
                as well
                outcome = m_push
        ELSE
                OUTPUT A message stating that the dealer did not have a
                blackjack
                outcome = m_blackjack
        END IF
        roundOver = true
END IF

WHILE *playerHandStrength != 21 && *playerHandStrength < 22 &&
playerIsDeciding == true
        OUTPUT Query the player whether he would like to hit or stand
        INPUT choice

        CASE Based on choice
                CASE H
                CASE h
                        CALL DrawCard(_deck, playersHand,
                        deckCardCount, playerHandCount,
                        playerHandStrength, playerFirstAceLocation,
                        playerIsFirstAceSet)

                        Clear the screen

                        OUTPUT A message stating what card
                        has been drawn

                        standCheck = false
```

```
                    CALL PrintOutHUD(dealersHand, playersHand,
                    dealerHandStrength, playerHandStrength, bet,
                    _balanceptr, standCheck)
            CASE S
            CASE s
                    playerIsDeciding = false

                    Clear the screen

                    OUTPUT A message stating that the player stands

                    standCheck = true

                    CALL PrintOutHUD(dealersHand, playersHand,
                    dealerHandStrength, playerHandStrength, bet,
                    _balanceptr, standCheck)
            CASE default
                    OUTPUT A message asking the player to input
                    again
        END CASE
END WHILE

IF *playerHandStrength == 21 && roundOver == false
        Clear the screen
        OUTPUT A message stating that the player has hit 21 and
        automatically stands

        standCheck = true
        CALL PrintOutHUD(dealersHand, playersHand,
        dealerHandStrength, playerHandStrength, bet, _balanceptr,
        standCheck)
ELSE
IF *playerHandStrength > 21 && roundOver == false
        OUTPUT A message stating that the player busts
        outcome = m_lose
        roundOver = true
END IF

WHILE (*dealerHandStrength < 17 || *dealerIsFirstAceSet &&
(*dealerHandStrength == 17 &&
dealersHand[*dealerFirstAceLocation].m_cardValue == 11))
&& !roundOver
        Clear the screen

        OUTPUT A message stating that the dealer hits

        CALL DrawCard(_deck, dealersHand, deckCardCount,
        dealerHandCount, dealerHandStrength, dealerFirstAceLocation,
        dealerIsFirstAceSet)

        CALL PrintOutHUD(dealersHand, playersHand,
        dealerHandStrength, playerHandStrength, bet, _balanceptr,
        standCheck)
END WHILE

IF *dealerHandStrength == 21 && !roundOver
```

```
            OUTPUT A message stating that the dealer has 21
        END IF

        IF *dealerHandStrength > 21 && !roundOver
            OUTPUT A message stating that the dealer busts
            outcome = m_win
            roundOver = true
        ELSE
        IF *playerHandStrength == *dealerHandStrength && !roundOver
            OUTPUT A message stating that the player draws with the dealer
            outcome = m_push
            roundOver = true
        ELSE
        IF *playerHandStrength > *dealerHandStrength && !roundOver
            OUTPUT A message stating that the player has a stronger hand
            than the dealer
            outcome = m_win
            roundOver = true
        ELSE
        IF *playerHandStrength < *dealerHandStrength && !roundOver
            OUTPUT A message stating that the player has a weaker hand
            than the dealer
            outcome = m_lose
            roundOver = true
        END IF

        CALL CalculateOutcome(_balanceptr, bet, outcome)
        CALL PlayAgain(_balanceptr, quitGame)
END WHILE
END FUNCTION
```


Student number: 4922675
Maciej Legas

4. Implementation : Marks 45%

Complete this section using Arial font 12pts

```
#include <iostream>
#include <cstring>
#include <ctime>
#include <cstdlib>
#include "Windows.h"

struct Card
{
	char m_cardSuit[9];
	char m_cardName[6];
	int m_cardValue;
};

enum Result
{
	m_null, m_win, m_push, m_lose, m_blackjack
};

void CalculateOutcome(long long *_balanceptr, long long _bet, Result _outcome);
void CreateDeck(Card _deck[]);
void DrawCard(Card _deck[], Card _hand[], short *_deckCardCount, short *_handCardCount,
short *_handStrength, short *_firstAceLocation, bool *_isFirstAceSet);
void GameLoop(long long *_balanceptr, Card _deck[]);
long long InputBet(long long *_balanceptr);
void PlayAgain(long long *_balanceptr, bool *_quitGame);
void PrintOutHUD(Card _dealersHand[], Card _playersHand[], short *_dealerHandStrength,
short *_playerHandStrength, long long _bet, long long *_balanceptr, bool _standCheck);
void ShuffleCards(Card _deck[]);

int main()
{
	std::cout << "Welcome to... Blackjack!\n\n\n";
	std::cout << "== Basic rules of the casino ==\n\n";
	std::cout << "There is one deck of cards used.\n";
	std::cout << "The payout rate is 2:1.\n";
	std::cout << "A blackjack pays 3:2.\n";
	std::cout << "The dealer hits until his hand strength is above 16.\n";
	std::cout << "The dealer hits on a soft 17. (e.g. when the dealer has an Ace and a
Six)\n";
	std::cout << "The value of aces changes automatically to 1 from 11 in case of a bust
scenario.\n\n\n";
	system("PAUSE");

	long long balance = 10000;
	long long *balanceptr = &balance;

	Card deck[52];

	CreateDeck(deck);
	ShuffleCards(deck);
```


Student number: 4922675
Maciej Legas

```
	GameLoop(balanceptr, deck);

	system("PAUSE");
	return 0;
}

void CalculateOutcome(long long *_balanceptr, long long _bet, Result _outcome)
{
	switch (_outcome)
	{
		case m_win:
		{
			*_balanceptr = *_balanceptr + (_bet * 2);
			std::cout << "Your winnings: $" << (_bet * 2) << "\n\n";
			std::cout << "Your balance: $" << *_balanceptr << "\n\n";
			break;
		}

		case m_push:
		{
			*_balanceptr += _bet;
			std::cout << "Your bet of $" << _bet << " has been returned.\n\n";
			std::cout << "Your balance: $" << *_balanceptr << "\n\n";
			break;
		}

		case m_lose:
		{
			std::cout << "You lost your bet of $" << _bet << ".\n\n";
			std::cout << "Your balance: $" << *_balanceptr << "\n\n";
			break;
		}

		case m_blackjack:
		{
			*_balanceptr = *_balanceptr + ((double)_bet * 1.5);
			std::cout << "Your winnings: $" << ((double)_bet * 1.5) << "\n\n";
			std::cout << "Your balance: $" << *_balanceptr << "\n\n";
			break;
		}

		default:
		{
			std::cout << "Exception found!\n";
			break;
		}
	}
}

void CreateDeck(Card _deck[])
{
	for (int suit = 0; suit < 52; suit += 13)
	{
		for (int card = 0; card < 13; card++)
		{
			if (card == 0)
```


Student number: 4922675
Maciej Legas

```
{
	_deck[suit + card].m_cardValue = 11;
}

if (card > 9)
{
	_deck[suit + card].m_cardValue = 10;
}

if (card > 0 && card < 10)
{
	_deck[suit + card].m_cardValue = card + 1;
}

switch (suit)
{
	case 0:
	{
		strcpy_s(_deck[suit + card].m_cardSuit, "Clubs");
		break;
	}

	case 13:
	{
		strcpy_s(_deck[suit + card].m_cardSuit, "Hearts");
		break;
	}

	case 26:
	{
		strcpy_s(_deck[suit + card].m_cardSuit, "Diamonds");
		break;
	}

	case 39:
	{
		strcpy_s(_deck[suit + card].m_cardSuit, "Spades");
		break;
	}

	default:
	{
		std::cout << "Exception found!\n";
		break;
	}
}

switch (card)
{
	case 0:
	{
		strcpy_s(_deck[suit + card].m_cardName, "Ace");
		break;
	}

	case 1:
```

```
{
        strcpy_s(_deck[suit + card].m_cardName, "Two");
        break;
}

case 2:
{
        strcpy_s(_deck[suit + card].m_cardName, "Three");
        break;
}

case 3:
{
        strcpy_s(_deck[suit + card].m_cardName, "Four");
        break;
}

case 4:
{
        strcpy_s(_deck[suit + card].m_cardName, "Five");
        break;
}

case 5:
{
        strcpy_s(_deck[suit + card].m_cardName, "Six");
        break;
}

case 6:
{
        strcpy_s(_deck[suit + card].m_cardName, "Seven");
        break;
}

case 7:
{
        strcpy_s(_deck[suit + card].m_cardName, "Eight");
        break;
}

case 8:
{
        strcpy_s(_deck[suit + card].m_cardName, "Nine");
        break;
}

case 9:
{
        strcpy_s(_deck[suit + card].m_cardName, "Ten");
        break;
}

case 10:
{
        strcpy_s(_deck[suit + card].m_cardName, "Jack");
```

```
                                        break;
                                }

                                case 11:
                                {
                                        strcpy_s(_deck[suit + card].m_cardName, "Queen");
                                        break;
                                }

                                case 12:
                                {
                                        strcpy_s(_deck[suit + card].m_cardName, "King");
                                        break;
                                }

                                default:
                                {
                                        std::cout << "Exception found!\n";
                                        break;
                                }
                        }
                }
        }
}

void DrawCard(Card _deck[], Card _hand[], short *_deckCardCount, short *_handCardCount,
short *_handStrength, short *_firstAceLocation, bool *_isFirstAceSet)
{
        if (*_deckCardCount >= 51)
        {
                std::cout << "All cards have been used.\n";
                ShuffleCards(_deck);
                *_deckCardCount = 0;
        }

        _hand[*_handCardCount].m_cardValue = _deck[*_deckCardCount].m_cardValue;
        strcpy_s(_hand[*_handCardCount].m_cardName,
_deck[*_deckCardCount].m_cardName);
        strcpy_s(_hand[*_handCardCount].m_cardSuit, _deck[*_deckCardCount].m_cardSuit);

        if (_hand[*_handCardCount].m_cardValue == 11)
        {
                if (*_isFirstAceSet == false && *_handStrength < 10)
                {
                        *_firstAceLocation = *_handCardCount;
                        *_isFirstAceSet = true;
                }

                if (*_handStrength > 10)
                {
                        _hand[*_handCardCount].m_cardValue = 1;
                }
        }

        if ((*_handStrength + _hand[*_handCardCount].m_cardValue) > 21 && *_isFirstAceSet
== true && _hand[*_firstAceLocation].m_cardValue != 1)
```

```
	{
		_hand[*_firstAceLocation].m_cardValue = 1;
		*_handStrength -= 10;
	}

	*_handStrength += _hand[*_handCardCount].m_cardValue;
	*_deckCardCount = *_deckCardCount + 1;
	*_handCardCount = *_handCardCount + 1;
}

void GameLoop(long long *_balanceptr, Card _deck[])
{
	char choice;
	bool *quitGame = new bool;
	*quitGame = false;
	short *playerHandStrength = new short;
	short *dealerHandStrength = new short;
	short *deckCardCount = new short;
	*deckCardCount = 0;
	short *playerHandCount = new short;
	short *dealerHandCount = new short;
	short *playerFirstAceLocation = new short;
	short *dealerFirstAceLocation = new short;
	bool *playerIsFirstAceSet = new bool;
	bool *dealerIsFirstAceSet = new bool;

	while (!*quitGame)
	{
		Card playersHand[11] = { 0 };
		Card dealersHand[10] = { 0 };
		Result outcome = m_null;

		long long bet = InputBet(_balanceptr);
		bool playerIsDeciding = true;
		bool standCheck = false;
		bool roundOver = false;

		*playerHandStrength = 0;
		*dealerHandStrength = 0;
		*playerHandCount = 0;
		*dealerHandCount = 0;
		*playerFirstAceLocation = 0;
		*dealerFirstAceLocation = 0;
		*playerIsFirstAceSet = false;
		*dealerIsFirstAceSet = false;

		DrawCard(_deck, dealersHand, deckCardCount, dealerHandCount,
dealerHandStrength, dealerFirstAceLocation, dealerIsFirstAceSet);
		DrawCard(_deck, dealersHand, deckCardCount, dealerHandCount,
dealerHandStrength, dealerFirstAceLocation, dealerIsFirstAceSet);
		DrawCard(_deck, playersHand, deckCardCount, playerHandCount,
playerHandStrength, playerFirstAceLocation, playerIsFirstAceSet);
		DrawCard(_deck, playersHand, deckCardCount, playerHandCount,
playerHandStrength, playerFirstAceLocation, playerIsFirstAceSet);

		system("CLS");
```

```
		std::cout << "Drawing initial cards for the dealer and the player...\n\n";

		PrintOutHUD(dealersHand, playersHand, dealerHandStrength,
playerHandStrength, bet, _balanceptr, standCheck);

		if (*playerHandStrength == 21)
		{
			system("CLS");
			std::cout << "Natural blackjack!\n\n\n";

			standCheck = true;
			PrintOutHUD(dealersHand, playersHand, dealerHandStrength,
playerHandStrength, bet, _balanceptr, standCheck);

			if (*dealerHandStrength == *playerHandStrength)
			{
				std::cout << "And the dealer has a blackjack as well! Push!\n\n";
				outcome = m_push;
			}

			else
			{
				std::cout << "The dealer did not have a blackjack! You win!\n\n";
				outcome = m_blackjack;
			}
			roundOver = true;
		}

		while (*playerHandStrength != 21 && *playerHandStrength < 22 &&
playerIsDeciding)
		{
			std::cout << "Would you like to [H]it or [S]tand?\n";
			std::cin >> choice;

			switch (choice)
			{
				case 'H':
				case 'h':
				{
					DrawCard(_deck, playersHand, deckCardCount,
playerHandCount, playerHandStrength, playerFirstAceLocation, playerIsFirstAceSet);
					system("CLS");

					std::cout << "You draw a card.\n\n";
					std::cout << "You drew a(n) " <<
playersHand[*playerHandCount - 1].m_cardName << " of " <<
playersHand[*playerHandCount - 1].m_cardSuit << ".\n\n";

					standCheck = false;
					PrintOutHUD(dealersHand, playersHand,
dealerHandStrength, playerHandStrength, bet, _balanceptr, standCheck);
					break;
				}

				case 'S':
				case 's':
```

```
                        {
                                playerIsDeciding = false;
                                system("CLS");

                                std::cout << "You stand.\n\n";
                                std::cout << "The dealer reveals his first card.\n\n\n";

                                standCheck = true;
                                PrintOutHUD(dealersHand, playersHand,
dealerHandStrength, playerHandStrength, bet, _balanceptr, standCheck);
                                Sleep(2500);
                                break;
                        }

                        default:
                        {
                                std::cout << "Sorry, I did not understand. Try again.\n\n";
                                break;
                        }
                }
        }

        if (*playerHandStrength == 21 && !roundOver)
        {
                system("CLS");
                std::cout << "21 hand strength!\n\n";
                std::cout << "You automatically stand.\n\n";
                std::cout << "The dealer reveals his first card.\n\n\n";

                standCheck = true;
                PrintOutHUD(dealersHand, playersHand, dealerHandStrength,
playerHandStrength, bet, _balanceptr, standCheck);
                Sleep(2500);
        }

        if (*playerHandStrength > 21 && !roundOver)
        {
                std::cout << "You bust!\n\n";
                outcome = m_lose;
                roundOver = true;
        }

        while ((*dealerHandStrength < 17 || *dealerIsFirstAceSet &&
(*dealerHandStrength == 17 && dealersHand[*dealerFirstAceLocation].m_cardValue == 11))
&& !roundOver)
        {
                system("CLS");
                std::cout << "The dealer hits.\n\n";

                DrawCard(_deck, dealersHand, deckCardCount, dealerHandCount,
dealerHandStrength, dealerFirstAceLocation, dealerIsFirstAceSet);
                PrintOutHUD(dealersHand, playersHand, dealerHandStrength,
playerHandStrength, bet, _balanceptr, standCheck);

                Sleep(2000);
        }
```

```
		if (*dealerHandStrength == 21 && !roundOver)
		{
			std::cout << "The dealer has 21!\n\n";
		}

		if (*dealerHandStrength > 21 && !roundOver)
		{
			std::cout << "The dealer busts!\n\n";
			outcome = m_win;
			roundOver = true;
		}

		if (*playerHandStrength == *dealerHandStrength && !roundOver)
		{
			std::cout << "The dealer has the same card strength! Push!\n\n";
			outcome = m_push;
			roundOver = true;
		}

		if (*playerHandStrength > *dealerHandStrength && !roundOver)
		{
			std::cout << "Your hand is stronger than the dealer's!\n\n";
			outcome = m_win;
			roundOver = true;
		}

		if (*playerHandStrength < *dealerHandStrength && !roundOver)
		{
			std::cout << "Your hand is weaker than the dealer's.\n\n";
			outcome = m_lose;
			roundOver = true;
		}
		CalculateOutcome(_balanceptr, bet, outcome);
		PlayAgain(_balanceptr, quitGame);
	}
	delete quitGame, playerHandStrength, dealerHandStrength, deckCardCount,
playerHandCount, dealerHandCount, playerFirstAceLocation, dealerFirstAceLocation,
playerIsFirstAceSet, dealerIsFirstAceSet;
}

long long InputBet(long long *_balanceptr)
{
	system("CLS");

	long long bet = 0;
	bool correctBetGiven = false;

	while (correctBetGiven == false)
	{
		std::cout << "Your balance: $" << *_balanceptr << "\n";
		std::cin >> bet;

		if (bet > *_balanceptr)
		{
```

```
                system("CLS");
                std::cout << "Your bet is higher than your balance! Try again.\n\n";
            }

            if (bet <= 0)
            {
                system("CLS");
                std::cout << "Your bet is an invalid value! Try again.\n\n";
            }

            if (bet > 0 && bet <= *_balanceptr)
            {
                system("CLS");
                std::cout << "Your bet is: $" << bet << "\n";

                *_balanceptr -= bet;

                std::cout << "The amount of $" << bet << " has been taken from your
balance.\n\n";
                correctBetGiven = true;
            }
            Sleep(1000);
        }
        return bet;
}

void PlayAgain(long long *_balanceptr, bool *_quitGame)
{
        char choice;
        bool correctOption = false;

        if (*_balanceptr <= 0)
        {
            std::cout << "You have lost all of your money!\n\n";
            std::cout << "G A M E  O V E R\n\n";
            *_quitGame = true;
        }

        else
        {
            while (correctOption == false)
            {
                std::cout << "Would you like to play again? [Y/N] (Inputting N will quit the
game)\n";
                std::cin >> choice;

                switch (choice)
                {
                    case 'Y':
                    case 'y':
                    {
                        correctOption = true;
                        break;
                    }

                    case 'N':
```


Student number: 4922675
Maciej Legas

```
                                case 'n':
                                {
                                        std::cout << "\nThank you for playing! Goodbye!\n\n";
                                        correctOption = true;
                                        *_quitGame = true;
                                        break;
                                }

                                default:
                                {
                                        std::cout << "\nSorry, I did not understand.\n\n";
                                        break;
                                }
                        }
                }
        }
}

void PrintOutHUD(Card _dealersHand[], Card _playersHand[], short *_dealerHandStrength,
short *_playerHandStrength, long long _bet, long long *_balanceptr, bool _standCheck)
{
        if (_standCheck == false)
        {
                std::cout << "Dealer's Hand:\n\n";
                std::cout << "??? of ??? || " << _dealersHand[1].m_cardName << " of " <<
_dealersHand[1].m_cardSuit << "\n\n";
                std::cout << "Strength: ??? + " << _dealersHand[1].m_cardValue << "\n\n\n";

                std::cout << "Your Hand:\n\n";
                std::cout << _playersHand[0].m_cardName << " of " <<
_playersHand[0].m_cardSuit;

                for (short i = 1; _playersHand[i].m_cardValue != 0; i++)
                {
                        std::cout << " || " << _playersHand[i].m_cardName << " of " <<
_playersHand[i].m_cardSuit;
                }

                std::cout << "\n\nStrength: " << _playersHand[0].m_cardValue;

                for (short j = 1; _playersHand[j].m_cardValue != 0; j++)
                {
                        std::cout << " + " << _playersHand[j].m_cardValue;
                }

                std::cout << " = " << *_playerHandStrength;

                std::cout << "\n\n\nYour bet: $" << _bet << "\t Your balance: $" << *_balanceptr
<< "\n\n\n";
        }

        if (_standCheck == true)
        {
                std::cout << "Dealer's Hand:\n\n";
                std::cout << _dealersHand[0].m_cardName << " of " <<
_dealersHand[0].m_cardSuit;
```

```
		for (short k = 1; _dealersHand[k].m_cardValue != 0; k++)
		{
			std::cout << " || " << _dealersHand[k].m_cardName << " of " <<
_dealersHand[k].m_cardSuit;
		}

		std::cout << "\n\nStrength: " << _dealersHand[0].m_cardValue;

		for (short l = 1; _dealersHand[l].m_cardValue != 0; l++)
		{
			std::cout << " + " << _dealersHand[l].m_cardValue;
		}

		std::cout << " = " << *_dealerHandStrength;

		std::cout << "\n\n\nYour Hand:\n\n";
		std::cout << _playersHand[0].m_cardName << " of " <<
_playersHand[0].m_cardSuit;

		for (short m = 1; _playersHand[m].m_cardValue != 0; m++)
		{
			std::cout << " || " << _playersHand[m].m_cardName << " of " <<
_playersHand[m].m_cardSuit;
		}

		std::cout << "\n\nStrength: " << _playersHand[0].m_cardValue;

		for (short n = 1; _playersHand[n].m_cardValue != 0; n++)
		{
			std::cout << " + " << _playersHand[n].m_cardValue;
		}

		std::cout << " = " << *_playerHandStrength;

		std::cout << "\n\n\nYour bet: $" << _bet << "\t Your balance: $" << *_balanceptr
<< "\n\n\n";
	}
}

void ShuffleCards(Card _deck[])
{
	srand(time(NULL));
	Card temp = { 0 };

	for (short i = 0; i < 10; i++)
	{
		for (short j = 0; j < 52; j++)
		{
			temp = _deck[j];
			short random = rand() % 52;
			_deck[j] = _deck[random];
			_deck[random] = temp;
		}
	}
	std::cout << "Cards have been shuffled.\n";
```

```
}
```

Appendices .References

Complete this section using Arial font 12pts

Dawson, M., 2007. *Chapter 10 - Inheritance and Polymorphism—Blackjack.* In: *Beginning C++ Through Game Programming, Second Edition* [online]. Cengage Course Technology, Boston.

Busbee, K., 2018. *Pseudocode Examples for Functions* [online]. Cnx.org. Available from: https://cnx.org/contents/6Uvbhb3A@7/Pseudocode-Examples-for-Functi [Accessed 10 Jan 2018].

Anon., 2018. *Case Structure Pseudo code* [online]. Virtual.parkland.edu. Available from: http://virtual.parkland.edu/kcouch/CIS122/Week8/case_psuedocode.htm [Accessed 10 Jan 2018].

Anon., 2018. *PSEUDOCODE STANDARD* [online]. http://users.csc.calpoly.edu. Available from: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html [Accessed 10 Jan 2018].

Munsch, J., 2009. *Is there an elegant way to deal with the Ace in Blackjack?* [online]. Stackoverflow.com. Available from: https://stackoverflow.com/questions/837951/is-there-an-elegant-way-to-deal-with-the-ace-in-blackjack [Accessed 8 Jan 2018].