**CIS 2275 C++ Programming Part II NAME:**

**Program 8 – Black Jack Game 150 pts DUE: Thursday, March 31 By 11:59pm**

**Objective: a program that simulates a BlackJack card game**

**This is our group project assignment. You and your partner are responsible for submitting one project. You will both receive the same grade on this project. Make sure you agree with what is submitted for grading. Put your own name on top of the code you contribute. If either one of you is unable to contribute code, that person will receive a zero.**

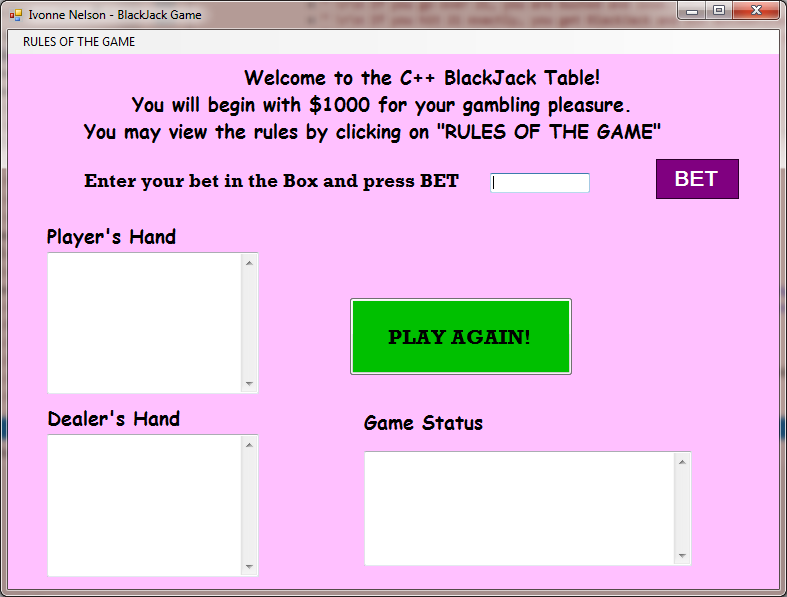
**Turn In Requirements:**

1. **5 pts Name your Visual C++ 2015 project LastnameP8, such as NelsonP8.**
2. **5 pts Upload one project to Visual Studio Online in your Team Workspace.**
3. **5 pts If you upload to Blackboard, print out the \*.h and \*.cpp files, staple this page to the front of your printed source code when you turn it in for grading.**
4. **5 pts If you upload to Blackboard, remove BOTH debug folders and the .sdf file from your project before uploading it to Blackboard.**

**Program Requirements:**

1. **3 pts Write your name, email address and file name at the top of your source code in a comment.**
2. **5 pts Your main function should have cout statements that write “header” information to the screen. The header info includes your name, course and program information, as well as a 1-2 line description of the program.**
3. **5 pts. Use good C++ programming style and formatting for your program. Use appropriate comments to explain what you are doing.**

Yay! You get to write a game simulation, of a sort. There will be five classes and, of course, your form, which will also contain the MyForm.h class. Here is an example of my initial form:



There is a menu bar on the form with a single item, Rules of the Game. The play begins when the player enters a bet and presses the “Place Bet” button. Once the bet is made, the initial hand is dealt, the Hit or Stay buttons are made visible, and there is no more betting:

The player has the option of hitting until he/she busts or chooses to stay. Each time the player hits, the **Player’s Hand** Text Box is updated. When the player stays, the dealer plays his hand. The dealer’s hidden card is shown and each time the dealer hits, the **Dealer’s Hand** Text Box is updated. When the dealer either busts, or stops because his points are 17 or over, the game is evaluated as to who won. The winner is announced in the Game Status textbox. There is an opportunity to Play Again by pressing that button. The game keeps track of the money held by the player.

There will be five classes to make up this game:

1. **Card.h** The card class is used to translate the integer value of cards, maintained by objects of the deck class, into objects of type string. It declares **three private variables**,
   1. int iValue; //numeric value corresponding to the card
   2. string value; //”Ace” “2” through “9”, “Ten”, “Jack”, “Queen”, “King”
   3. string suit; //”S”, “H”, “C”, “D”

The iValue variable is used to keep track of the number of points associated with a card.

The value variable references a text string that is used to describe the face value of a card.

The suit variable is used to identify the suit of a card.

There is a default constructor and **an overloaded constructor**: public Card(int n) {…}, which is the heart of the card class. It receives a value of 0 through 51 from the Deck class. Card() first determines the suit of the Card identified by the n parameter by dividing by 13 and assigning the result to a local integer named iSuit. It determines the point value of the card as iValue = n%13 + 1. This value is further adjusted for the face cards by stating that if iValue > 10, iValue = 10.

Card () then uses a switch to assign the correct text to the suit variable by comparing the iSuit value to the values identified in each of the four case labels (0-3). It then uses another switch to assign the string variable values to each card. That is, if iValue == 1, value = “Ace” and if iValue == 13, value = “King”, etc for the face cards, and for the number cards, value = the string representation of iValue. You may use stringstream to convert from int to string.

When completed, the playing cards should be mapped as follows:

0 – 12 Ace of Spades through the King of Spades

13 – 25 Ace of Hearts through the King of Hearts

26 – 38 Ace of Clubs through the King of Clubs

39 – 51 Ace of Diamonds through the King of Diamonds

There are **four public class methods**:

1. void SetIValue(int val){iValue = val;}
2. string GetValue(){ return value;}
3. int GetIValue(){return iValue;}
4. string GetSuit(){return suit;}
5. **Deck.h** The Deck class declares **three private** **class members** :
   1. array <int, 52> cards; //array of 52 cards
   2. int topCard; //0-51, index of the next card to be dealt
   3. default\_random\_engine engine; //Creates an object of the random engine

The **constructor** is default. It seeds the random engine object, then initializes the cards[] array from 0 to 51. topCard is initialized at 0 and the method Shuffle() is called.

There are **three public class methods**:

1. **void** **Shuffle()** method shuffles the deck by randomly switching two cards in the deck 52 times using the RandomCard() method. Check that the two random numbers generated are different. If not, go back and generate two new random numbers before you shuffle.
2. **int** **RandomCard()** method returns a random integer by using the uniform\_int\_distribution in the range of 0 - 51.
3. **void Deal(Card &c)** method is used to deal a card off the top of the deck. It passes a Card object by reference. It uses the topCard variable as an index of the cards[] array for instantiating a new Card object., then setting the reference passed in to the new card object. The topCard variable starts at 0 and is incremented until it is greater than 34, indicating that the remaining 17 cards could possibly use up the rest of the deck on the next hand. In this case, the deck is re-shuffled and topCard is again set to 0.
4. **Hand.h** The Hand class is used to implement a hand of cards by both the player and the dealer. It declares **three private class members**:
   1. int numCards: //number of cards in the hand
   2. string showHand; //a string describing the hand, generated in the Show method
   3. array <Card , MAX\_CARDS> cards; //array of cards, not the same as the array in Deck

Either #define or declare const int MAX\_CARDS; //max number of cards in a hand

The Hand() **constructor** is default and initialized numCards = 0.

There are **seven public methods**:

1. **void AddCard(Card c)** method adds cards to a hand. It takes an object of the Card class as an argument and adds it to the first available position within the cards[] array. It then increments numCards .
2. **string Show(bool isdealer, bool hideFirstCard)** displays either the dealer’s or the player’s hand. The first card of the dealer’s hand should be hidden until the player “stays” and the dealer plays his hand. The method returns a string which contains the title of Dealer or Player and lists the cards in the hand.
3. **bool BlackJack()** checks if the hand is BlackJack and returns true or false. It also checks if the number of cards in the hand = 2. It has to be 2 for BlackJack. The player wins 1.5 \* bet.
4. **bool Under(int n)** checks whether the number of points in a hand is less than the number n, that is passed in as an argument. It sums the points in the hand and adds up the iValue of each card.
5. **int BestScore()** returns an integer identifying the best possible score for the hand. It adjusts the value associated with aces to either 1 or 11, depending on whether it causes the hand to go over 21 points.

It uses a variable, haveAce, of type bool, to identify whether the hand contains an ace. It uses a for statement to calculate the minimum number of points in the hand and to determine if any aces are present. If an ace is found, it determines whether it is better to use the 11 or 1 point value of the ace.

1. **bool MustHit()** checks the dealer’s hand to see if the hand is less than 17. If it is, the dealer must hit. If it is 17 or higher, the dealer must stay.
2. **bool Busted()** checks if the number of points in a hand is under 22. If not, the hand is busted and returns true.
3. **void ClearHand()** sets showHand to “” and numCards = 0.
4. **Game.h** The game class is rather long. It directs the play of the game and provides methods for the interface with the form. It declares **five private class members**:
   1. int bet; //There is one bet for each hand.
   2. double money; //This is the player’s money balance.
   3. int wins, losses, ties, numberOfBets (or hands); //keeps running totals for summary
   4. Deck deck; //The deck object of cards
   5. Hand playersHand; //The player’s Hand object
   6. Hand dealersHand; //The dealer’s Hand object
   7. Logger log; //The Logger object

The **constructor** is default and initializes the bet to 0, money to 1000.

There are **many public methods**:

1. bool **SetBet**(int b); //sets the bet into the class, checks that the bet !< 0 and bet !>money. If either case is true, the method returns false. The event handler displays a message in the Status box and the play does not continue. A new bet must be entered and the BET button pressed again.
2. **void InitialDeal()** Using a short for loop, deal both player and dealer two cards.
3. **string ShowPlayerHand()** returns playersHand.Show(false, false);
4. **string ShowDealersHand(bool hide)** returns dealersHand.Show(true, hide);
5. **bool IsBlackJack()** returns playersHand.BlackJack(); The player wins 1.5 \* bet.
6. **bool PlayerBusted()** returns playersHand.Busted();
7. **bool PlayerContinues()** checks if the playersHand is busted and if the playersHand is under(22) to return true. Else return false.
8. **void PlayerHits()** adds a Card to the playersHand
9. **string PlayerWins()** returns a description that starts with “Player wins:” and adds the amount of the bet and tells how much money the player has now. If the win was from a BlackJack, the win is 1.5 \* bet.
10. **bool DealerContinues()** checks if the dealer MustHit. If so, adds a Card to the dealersHand and returns true. Else returns false.
11. **string DealerWins()** Actually returns a description that starts with “Player loses:” and shows the amount of the bet and tells how much money the player has now.
12. **string Tie()** returns a description starting with “tie” and tells how much money the player has now.
13. **string NoResults()**  A place holder, returns “”;
14. **string ShowResults()** determines the disposition of the game and calls one of the methods above for the description. The choices are:
    1. **Both player and dealer are busted -- Tie()**
    2. **Player busted -- DealerWins()**
    3. **Dealer busted – PlayerWins()**
    4. **Player’s best score > dealer’s best score – PlayerWins()**
    5. **Player’s best score < dealer’s best score – DealerWins()**
    6. **Player’s best score =dealer’s best score – PlayerWins()**
    7. **Or Else – NoResults()**
15. **void ClearHands()** calls the ClearHand method for both hands.
16. **bool** **IsLogOpened(){**return log.IsLogOpen();}
17. **void EndGame( );** creates the summary string and calls the Logger’s CloseLog with the string as the argument.
18. **Logger.h**  The Logger class receives strings from the BlackJack Game and writes them into the log file. There are five private member variables and two private methods:
    1. string filename; // The Filename method will create the filename. It will make the filename with a date and time stamp.
    2. bool bLogOpen; //to ensure the log is opened.
    3. char timeRightNow[20]; //a string to write into the header of the file
    4. int hour, min, sec, day, month, year; //ints representing the date/time now
    5. ofstream output; //will be a class member, since the file will be written into repeatedly
    6. void Time(); // gets system time for timeRightNow
    7. void FileName(); //Creates the filename using the current system date in this format: Log\_month\_day\_year\_hour\_minute\_second.txt

**The constructor** is default and does a lot in this class. The private method FileName is called, building the log filename using the title Log\_ , the current date and the extension “.txt”. Then the log file is opened using the output file stream output and bLogOpen is set. The first line of the log file is a time stamp. Call the Time method to get the current time.

There are **four public methods**:

1. void StartLog(double initialBal); writes the initial lines of the log file. A double is passed in.
2. void WriteLog(string s)writes into the log file any time a game is completed.
3. void CloseLog(string s) writes the message from the EndGame() function and closes the file.
4. bool IsLogOpen() returns bLogOpen so the log success can be checked by the Game class.

**MyForm.h**

Put a Game class object in the MyForm.h class. Instantiate the Game class above the MyForm class declaration and start with the Hit or Stay group of label and two buttons not visible. Do this in the Form constructor using this code:

lblHitOrStay->Visible = false;

btnHit->Visible = false;

btnStay-> Visible = false;

Also check that the logFile was opened in the constructor and if the log file was not opened, report that in the Status box. But keep playing.

You need a MenuStrip event handler for the Rules of the Game. Display these in a MessageBox.

You will have event handlers for the five buttons. The Bet button will initiate the game. It sets the bet into the class and if the return is false, displays a message in the Status box and the play does not continue. A new bet must be entered and the BET button pressed again. If the return is true, it calls for the initial deal. It checks for BlackJack and handles it appropriately. If there is no BlackJack, the Bet group of (label, textbox and button) become not visible, and the Hit or Stay group become visible, so the play can continue.

When the player clicks the Hit button, the event handler checks if the player may continue. If yes, the player hits and the new player hand is shown. If the player is busted as a result, that is shown. The player can survey his hand and decide whether to hit again or stay.

When the player hits the stay button, the event handler checks if the dealer may continue. If so, the new cards are shown and when the dealer must stop, the results of the game are shown.

The PlayAgain button will re-initialize the Bet and Hit or Stay groups and clear the text Boxes.

This is a fun program. I hope you enjoy it!