Riley Quijano (ID: 2686839)

Dr. Mark Lehr

CIS5 (45549)

20 July 2017

Project 1 Write-Up

**Brief Explanation of Program**

The Blackjack game created for project 1 takes the user through a classic game of Blackjack. In the beginning of the program, there are several variables declared to account for the possibility of 32 unique cards. The number 32 was chosen due to the fact that the user can have a total of 24 cards given that they split the maximum number of three times and, in their four resulting hands, hold every lowest card in the deck. The other eight cards come from the fact that the dealer can almost never have more than eight cards in their hand, as the probability of them having eight in one hand at all is lower than 1 in 10 trillion. So, 32 unique cards had to be accounted for, as the program is designed to have the dealer draw from cards 25 through 32, and the user draw from cards 1 through 24. This does not affect the random nature of the game, and simplifies it in terms of coding. An important note to make for this version of the project, however, is that the ability to split when the two cards in a hand have the same face is not yet implemented. So, as of now, the program may seem to have an unnecessary amount of declared card variables.

Following the declaration of all of the unique card variables including variables for their names and values, the declaration of other variables pertaining to bet amounts, and the opening of a file to be written to, a for loop is opened. This for loop is used to allow the user to choose how many games they want to play in a row and to “shuffle” the cards after each game to ensure unrealistic dealing of cards does not occur. After the for loop is opened, all 32 cards get assigned a name and a value. Each assignment is done using a do-while loop. Within the “do” portion of the loop, a card is initialized to have a random value from 1 to 52. A switch is then opened that uses the modulus operator to divide the random value by 13 and use its remainder to determine the face and value of the card; depending on the remainder, a face and value is assigned to the card. Following the switch statement, the card is assigned a suit using a series of if-else if statements. This assignment depends on which fourth of 52 the card was given when assigned a random value (1-13, 14-26, etc.). The face and suit of the card are then added together to determine its name. The “while” portion of the loop forces the assignment of the name and value of the card to repeat so long as the name of the card is the same as one of the previous cards. This technique is used to assign names and values to cards 2 through 32.

Just like in classic Blackjack, the user places a bet. Specifically in this program, the user always starts out the first game with $500. Round 1 is then played out, and the user’s cards and dealer’s one card are displayed in the output. If the dealer’s cards (card25 and card26) are 21 or the user’s, the game immediately comes to an end, all the cards are revealed, and the proper payouts are given (just as all of these steps would occur in classic blackjack). The program only differs from classic Blackjack in that splits are not yet implemented. Just like in classic Blackjack, the user can place an insurance bet if the dealer’s showing card is an ace. The possibilities of the outcome of round 1 (dealer’s Blackjack, player’s Blackjack, or player’s and dealer’s Blackjack) are played out by if/else if statements. If no Blackjack occurs, the user can continue to the next round by either choosing to hit, stand, or double.

The user’s choice to stand, hit, or double decides the next course of action the program takes. The different paths of the program, which are determined by this user’s choice, are put into effect using a switch statement nested inside of the “else” portion of round 1. If the user decides to hit, the board of cards is redisplayed with a third card. The value of the cards is counted up; if the user is over 21 or at 21, they are no longer allowed to hit, but if they are under, they can choose to hit or stand at the end of the second round (just as this would occur in classic Blackjack). If the user hits, they continue to round 3 (which is played out using another switch nested inside of the switch nested inside of the else of round 1), where the same exact events as round 2 occur except with a fourth card. If the user continues to hit, this same process also occurs for all of the following rounds (meaning the program keeps going into another nested switch statement for each round) until they bust, stand, or reach 21 exactly. In each of these three cases, the outer-most switch automatically puts the user into another case that forces the user to stop playing and allows the dealer to start drawing. Another important note to make is that had the user chosen to double after round 1, they would have been taken to a third “case” by the outer-most switch that carries out the same exact steps that were carried out when the user chose to hit; however, the bet amount is doubled and the user cannot choose to draw cards any longer.

Once the user’s turns are over, the dealer flips over their first card. This is simulated by redisplaying the board except with the dealer’s cards (card25 and card26) both revealed. Just like in regular Blackjack, the dealer must hit if they are 16 or under or must stand if they are not. Whether the dealer hits or stands is decided by an if/else statement. If they hit, they continue to their draw, where, just like with the flipping of their card, the board is redisplayed except with all three of their cards showing. If they are under 17, they must hit again; if they are over 17, they must stand; and if they are over 21, they bust. As the dealer continues to hit, the program continues to enter into more and more nested if statements until the dealer no longer draws.

Once the player and dealer are done drawing cards, the earnings of the player are determined by the total amounts of the their own and the dealer’s hands. The amount of money won off of the bet follows the same exact rules as classic Blackjack, and in this program, a series of if/else-if statements are used to determine these winnings. Once the winnings are determined, game 1 is over, and the program prompts the user to either continue to their next game (if they chose to play more than 1) or to continue to the next part of the program. If the user plays another game, everything is repeated because of the for loop in the beginning, into which the user put the amount of games they wanted to play. Another important aspect of the program to note is that the amount of the money the user has at the end of each game is written to the file that was opened in the very beginning of the program.

Once the actual games of Blackjack are over, the program shows the user what percentage the amount of money they end with is of the amount of money they started with in game 1. If the user chooses, the program also displays the average amount of money the user had at the end of each game and the standard deviation of the different money amounts at the end of these games. After displaying these statistics, the program ends.

**Summary of Program**

**-** Play Blackjack with the user for however many games in a row the user decides

\*Repeat the games using for loop

- At the start of each game, give every card that will be used a unique name and the appropriate coinciding value using switch statements nested inside the “do” portions of do-while loops

-Play round 1; use several if/else if’s to determine what happens next in the program if anyone starts with 21; use the else portion of the statements to continue the program if no one has 21

-Have the user choose to hit, stand, or bust; each choice leads to a case of a switch statement

-For each hit user does, keep displaying rounds of the game using if’s nested within if’s; once the user can no longer draw or if they chose to double, exit the switch containing all the if’s nested in the if’s and have the dealer start drawing

-What the dealer does is determined by if/elses nested within the else of an initial if/else statement

-When the dealer is done, appropriate payout is given to the user

-Once all of the games are over, display average money the user had at the end of each game and the standard deviation of their money amounts by reading from a file that was saving the end game amounts

-**Variable Count:** 122

-**Line Count:** 2388

**Personal Notes**

Overall, the composition of all of this project’s components took five days to complete. The first portion of this project I completed was the portion that assigns unique names and appropriate coinciding values to each of the 32 necessary cards; for me this was the most difficult part to figure out how to execute. While randomly assigning names and values to the cards was not much of a struggle, I was unable to grasp how to ensure all of the cards had a unique name for a reasonable amount of time. The next and most time consuming portion of the project was playing out all of the rounds for the player by utilizing several nested switch statements and if statements. Prior to actually beginning coding of this portion of the program, I had not thoroughly thought out how to play out multiple rounds given the knowledge I currently have, so as I coded each round I began to become more and more cognizant of just how much time this method of developing the game would take. Even given my knowledge, this may have been a very inefficient way of coding the program, as the amount of lines and time this program took were unreasonable with regard to what was actually assigned. Since I had done the player’s portion of gameplay utilizing all of this nesting, I figured I had to do the same with the dealer except without the user having to input a choice. Thus, this portion of the program, which is the third “chunk” I did, took an immense amount of time as well. Given that there were no switches to incorporate, however, the total invested time was slightly less. The final chunk of the program I completed was the portion which implements the statistics by writing to and reading from a file. This portion took by far the least amount of time with respect to the previous three potions. The portions that the coding was completed in can be seen on GitHub. Ultimately, this project took immensely more time than I had anticipated, likely due to the fact that I used an extremely inefficient means to execute each individual round of Blackjack by just using repeated nested ifs and switches.

**Notes About the Check-off List**

Within the check-off list, I either included all of the instances of the topic contained within the program or included as many examples as I could until the “line number” box of the list was full. For example, for the “cout” topic, there were many more instances than could fit within the one line provided, so I only listed until there was no more room to type. For topics where this was not the case, such as “type casting,” every single instance where it occurs in the program is listed in the check-off box. In addition to this note, notes should be made about the different \*’s adjacent to certain line or line ranges:

\* - This appears only next to some line or line ranges that deal with chapter 2 topics; every line or line range this is next to is one where the variable that the line is cited to contain is either initialized, inputted into, or used in some way; for example, line 865\* is a line where something is inputted into a character variable, but line 857 is where that variable is actually declared; so, lines without \*’s contain instances of a certain variable type being declared, while lines with \*’s contain instances of those variable types being worked with in same way

\*\* - This appears at the end of the “nesting” topic, so each instance of nesting can be explained

Lines: 243-324 = Switch nested in do-while

328-409 = Switch nested in do-while

413-494 = Switch nested in do-while

828-834 = If/else-if nested in if

858-877 = If/else nested in if

886-910 = If nested in if

986-988 = If nested in if

It is also important to note that many of the branching statements and loops can be considered to be nested within other extremely large branching or loop statements that

are less obvious. For example, every branching and loop statement from 145 to 2183 can

be considered to be nested in the large for loop.

\*\*\* - This appears at the end of 1101-1684 because this is an unusually large switch statement; there are several other switch statements nested within this switch statement

\*\*\*\* - This appears at the end of 145-2183 because this is an extremely large for loop