ARM BlackJack

Project 2

Amul Bham

CSC 11 :48982

12/17/2015

INTRODUCTION

BlackJack is a simple card game in which the player competes against the dealer to get the higher hand but not over 21. Whoever goes over 21 loses. The player receives their first two cards and the dealer receives one. Based on the player's card total, they must decide to “hit” or “stay”, trying not to go over 21 but still attempting to have a higher hand than the dealer. The dealer is set up in a way where they never “hit” if they are at 17 or higher, therefore it is the player’s decision to hit or stay depending on if they think the dealer will bust or have a low card total.

For example: If the player has a hand of 12, they risk the chance of going over 21 on their next card draw (Any value of 10 or greater provides the chance of busting). So they might decide to stay, despite having a low card total, in hopes that the dealer will ultimately bust. If they dealer has a 9 for their first card, they will keep drawing cards till they are at 17 or higher. It is up to the player to balance risk and reward so outwit the dealer and use their aggressive play to their advantage.

The dealer is always playing a risky and aggressive and it is important to remember that having a high card total or being as close to 21 as possible is not always the best play and often the player will win regardless of card total due to the dealer busting. Although the complexities and limitations of assembly language prevented the game from incorporating some of the more finesse rules and strategies of a true game of BlackJack (rules such as splitting, doubling up, placing bets, storing user data, etc.), I believe it catches the essence of blackjack and one could certainly at least learn the rules and flow of how to play blackjack as most of the logic in how the hands play out are the same.

In addition, I love the idea of a simple game on the surface but can have infinite levels of depth; which describes blackjack perfectly. It is a simple enough game to learn quickly however it is far more difficult to develop a natural sense of how to play each hand. In addition, adding ideas such as betting, player information storage, multiplayer, can add varying degrees of depth to the game. This leaves a game like blackjack open for tons of improvements and modifications down the line.

SUMMARY

Project Size : About 570 lines

Number of Functions: 4

Number of Variables : About 20

Number of Methods : 15-20

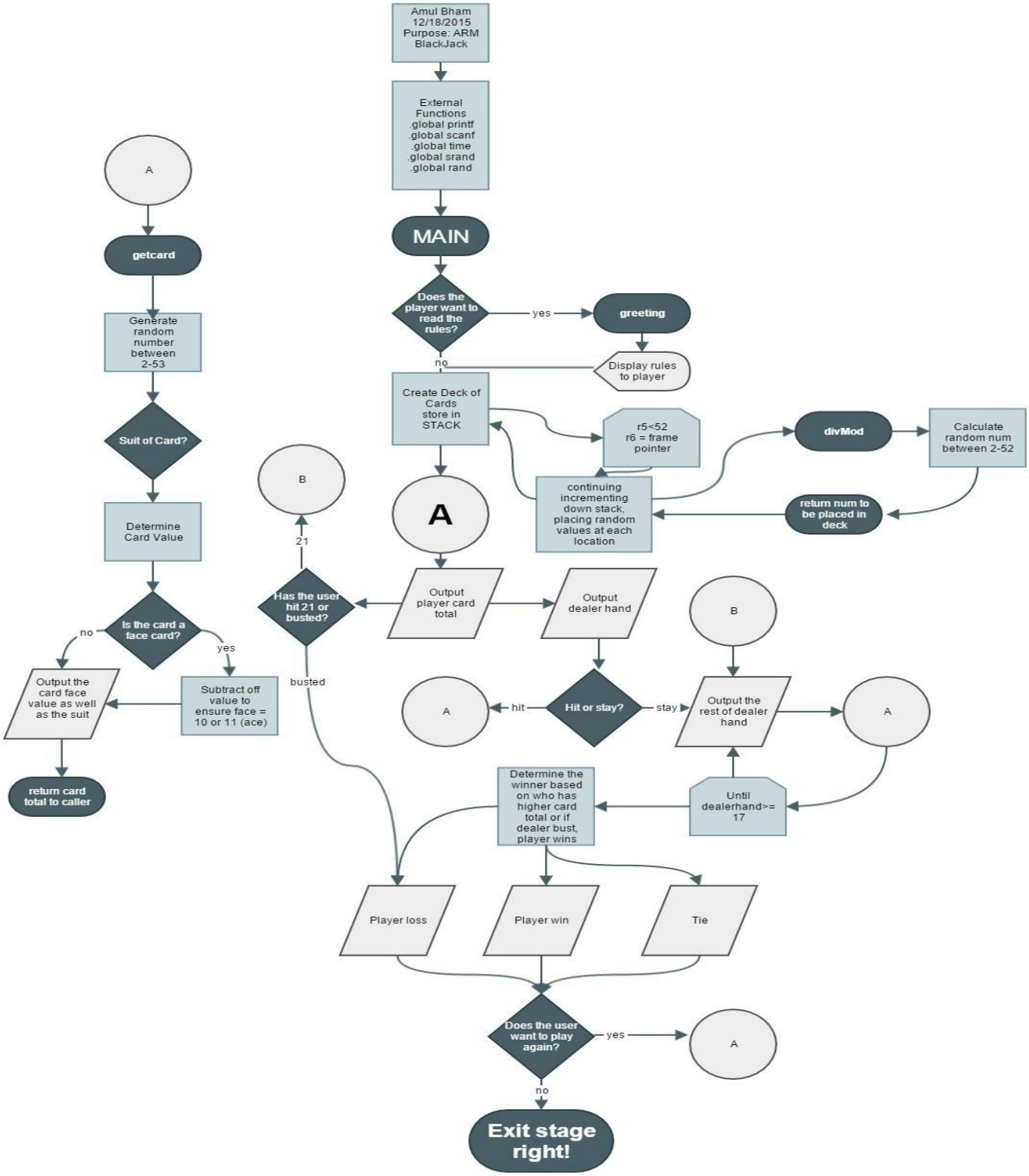
Repository:

https://github.com/amulbham/ASSEMBLY_CSC11/tree/master/proj/BlackJack_Proj_2

My project ended up being much larger than I originally anticipated coming in at about 600 lines of code between all the separate functions. This occurred simply because I under anticipated how difficult it would be to program all the different logic points in Assembly Language, something that proves to be simple in C++ can be exponentially more difficult in Assembly. For example, emulating a deck of cards proved to very difficult. At first, I tried to create an array and store random values between 2-53, and then have a player “draw” from this pool of random values to determine their card. However, I realized that I didn’t quite understand arrays fully to effectively implement this concept. So instead, I used a random number generator to give a value between 2-54 which was assigned to the player drawing. Even though their is bound to be duplicates, I figured it would suffice as Casino decks typically are made of four decks making duplicates a part of the game. Stimulating the concept of randomly drawing from a deck, as well as assigning this value a “face” and a card value proved to be the most difficult part of the game to program. This work can all be found within my get card function which took me around 6-8 hours to code and debug.

I also utilized open source code provided by Dr. Lehr to generate a true random number based on time. In addition, I used my previous C++ BlackJack project as a reference and “road map” for all the different logic points in the game. This proved to be crucial as it also allowed me to more easily visualize what needed to be done as C++ is far easier to interpret. Any time I found myself stuck or not knowing what to do, I would glance over the C++ code to give me some guidance. Although Dr Lehr constantly emphasizes the importance of coding assembly projects in C first, this project truly made me appreciate why as the project would have taken me far longer to code without C++ reference code.

Flowchart


Amul Bham
12/18/2015
Purpose: ARM
BlackJack
External
Functions
.global printf
.global scanf
.global time
.global srand
.global rand
MAIN
Does the player want to read the rules?
yes
greeting
Display rules to player
no
Create Deck of Cards store in STACK
r5<52
r6 = frame pointer
divMod
Calculate random num between 2-52
return num to be placed in deck
continuing incrementing down stack, placing random values at each location
A
Output player card total
Output dealer hand
Has the user hit 21 or busted?
21
busted
B
Hit or stay?
hit
stay
Output the rest of dealer hand
Until dealerhand>= 17
Determine the winner based on who has higher card total or if dealer bust, player wins
Player loss
Player win
Tie
Does the user want to play again?
yes
no
Exit stage right!
getcard
Generate random number between 2-53
Suit of Card?
Determine Card Value
Is the card a face card?
no
yes
Output the card face value as well as the suit
Subtract off value to ensure face = 10 or 11 (ace)
return card total to caller

Sample Input/Output

The executable file is called project1 -> type ./project1 in the working directory of the project to run the game, if makefile is corrupt, enter the following to play the game

1. gcc -o blackJack main.o divmod.o getcard.o greeting.o
2. ./blackjack

```
pi@raspberrypi ~/Desktop/ASSEMBLY_CSC11/proj/BlackJack_Proj_1/asm_files $ ./project1
Welcome to Amul Bham's BlackJack!
Would you like to hear the rules to play? 1 for yes, 2 for no
```

Determine user's experience with blackjack

```
Basically the point of the game is to get to 21
or as close as possible without going over
I will deal 2 cards from a regular deck of cards
depending on the total value of these cards, you must
decide if you want another card or stay where you are
Also, note that my job as a dealer is to always hit if my card
value is below 17, if i bust (go over 21) then you automatically win
```

The rules displayed for new players

```
Now dealing your first two cards...
Queen of hearts
4 of cloves
card total: 14


Now dealing my first card...
Ace of diamonds
card total: 11


Would you like to hit(1) or stay (2)
```

After the greeting and the rules, the first two cards of the player are outputted along with the first card of the dealer hand

The player must decide if they would like to hit(1) or stay(2) based on their current card value and the first card of the dealer.

```
Would you like to hit(1) or stay (2)
1
King of diamonds
card total: 24


You busted! Better luck next time!
Would you like to play again? 1 - yes / 2- no
```

If the player busts (goes over 21), then the program skips to the _lose function, the rest of the dealer hand is skipped, leaving the player to decide if they would like to play again

```
Now dealing your first two cards...
4 of cloves
5 of cloves
card total: 9


Now dealing my first card...
Ace of spades
card total: 11


Would you like to hit(1) or stay (2)
1
Ace of cloves
card total: 20


Would you like to hit(1) or stay (2)
2
And now the rest of my hand...
3 of hearts
card total: 14


Ace of spades
card total: 25


Well played! You win the hand!
Would you like to play again? 1 - yes / 2- no
```

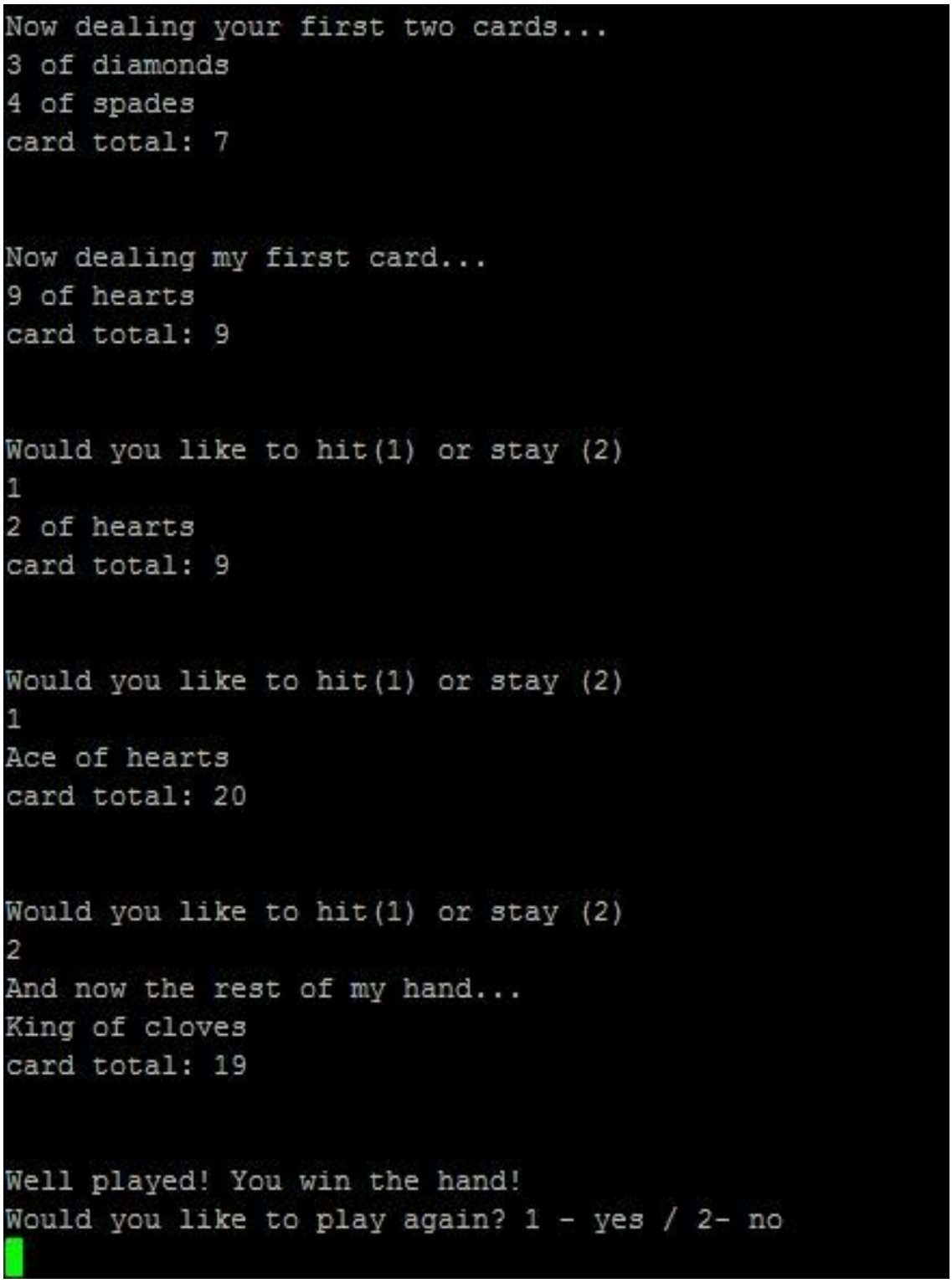
If the player decides to stay, or hit but not go over 21, then the rest of the dealer hand is displayed. The dealer will keep drawing until their card value is >= 17

At this point, the winner is determined based on who had the higher card total but not over 21

In this example, the dealer busted so the player automatically wins

The player is prompted to play another game or exit the program

Full Input/Output


Now dealing your first two cards...
3 of diamonds
4 of spades
card total: 7
Now dealing my first card...
9 of hearts
card total: 9
Would you like to hit(1) or stay (2)
1
2 of hearts
card total: 9
Would you like to hit(1) or stay (2)
1
Ace of hearts
card total: 20
Would you like to hit(1) or stay (2)
2
And now the rest of my hand...
King of cloves
card total: 19
Well played! You win the hand!
Would you like to play again? 1 - yes / 2- no

Pseudocode:

First the user decides if they want to hear the rules of or not

if so, branch to the greeting function to display the rules

else, branch to the start of the game

The player then receives their first two cards

Cards are randomly selected using random number generator, assignment a suit, a card value and determines if the card is a face. Branch to he get card function

The dealer receives their first card

branch to the get card function

Player must now decide if they want to hit or stay

Program calculates the value of each player's hand after each card draw -> get card

if the user decides to hit, they will continue to receive cards till they are over 21, they decide to stay, or hit a BlackJack.

else, the dealer receives their next cards, and continues until they are at 17 or over.

If player or dealer bust, opposing player wins -> the calculations are ignored if either player busts and the game skips to the output

else, if both players have not gone over 21, a winner is determined.

Program calculates winner depending on who had the higher card total

if user has a higher card total, they win

else if the dealer has a higher card total, or the player has gone over 21, the dealer wins

User is prompted to play again or walk away

if user walks away -> exit the program

else, game loops to the start of the game loop, registers are reset for next hand

Major Variables:

Variable Name	Description	Location
pcard_total	stores the value of the players current card	.global main
dcard_total	stores the value of the dealer's current card	.global main
address_of_message address_of_finish address_of_lose address_of_win address_of_tie address_of_plose21 address_of_plose21	displayed various messages regarding the outcome of the game, informs the user if they won, lost, tie, hit a blackjack etc.	.global main
address_of_return: address_of_opening: address_of_opening2: address_of_continue: address_of_continueA: address_of_hitorstay: address_of_firstcards: address_of_dfirstcards:	displayed various messages throughout the game, including the intro message, informing the user of what point the game currently is at, if they want to hit or stay, etc	.global main
output_hearts output_diamonds output_cloves output_spades	stored the string text of each of the four suits that the card could possibly be	.global getcard,
output_jack output_queen output_king output_ace	stored the string text for each of the face cards in the case that the player or dealer draws a face	.global getcard,

address_of_return	used to store the link register	.global main

Checklist of Concepts:

Concept	Implementation
Using link register	Stored the lr upon entering a function and then popped back the lr upon leaving the function Can be seen in my greeting function, I also utilized push and pop commands in my get card function
Passing registers as Parameters for a function	Before entering a function, I made sure beforehand that the proper variables were stored in R0-R3 to ensure they were passed to the get card and dcard functions appropriately for modification, can also be seen upon entering printf and scanf, I loaded the registers with the appropriate outputs, or format for scanf, prior to using them .
Branching : bge, bl, blt, beq, bx,b compare: cmp	Utilized branching like second nature all throughout my program, used it to make decisions, act like if else statements, switches, basically anywhere I needed the program to make a decision based on the input. I also used the bl command to branch to other functions while also passing the link register. Based on certain circumstances, I used branching to move fluidly and effectively around my program
str and ldr - store and load register	Used str command to preserve my link register upon entering a function. Used the ldr command all throughout my program to store memory addresses in certain registers for modification or passing to functions.
Looping using labels and branching	I used labels at the start of major points

	in the program to save writing excess code, for example the hit or miss part of the program can be looped until the player decides to stay or bust, instead of just rewriting the code, I used a label and a branch statement to continuously reuse this code given the user wants to keep hitting. I used this concept in many other parts of my program including the end when the user decides if they would like to play again, I simply created a label at the start of the game, and branch to that label if the user would like to play again.
Functions	Utilized functions to separate logic points in my program to streamline the code and make it more modular. For example, I kept the greeting function separated since it is essentially a wall of text explaining the rules, so instead of having that in my main function, where it takes up space and gets in the way of readability. Also calculating the card value and card total required a ton of code and logic that I kept separated from the main function, this allowed me to focus on getting the logic correct in that block of code instead of having it all jumbled up together in my main function. This allowed my main function to simply guide the program while my outside functions did the more complicated tasks. It also allowed me to more easily detect where a bug was arising from as I could individually debug functions.
External Functions	I used scanf and printf to read user input and output text to the screen all throughout my program. In addition, I also used the srand and time functions to bring in the current time and then plant the random number seed inside of the rand function which I used to generate a random number representing a card draw

Setting Flags using cmp	I used the cmp to compare numbers against register values in order to guide my program to the next step. For example, the dealer keeps drawing cards till their card total is at or above 17 so to ensure this, I used a cmp R1, #17 to set the flag, if the value was greater than or equal (bge) the program would go to the next step, else blt was executed and the program looped . Or when determining the winner of the game, I simply used cmp R1, R2, in other words I just directly compared the card values of the dealer and player, whoever was high was the winner
Addition, subtraction of values in registers	Used arm subtraction and addition all over my program to keep a counter, add the card values together, set loop conditions etc.

References:

1.) I utilized the DivMod function as well as parts of the random number generator graciously provided by Dr. Lehr. I used the divMod function to calculate the random number between 2-53, in addition I utilized the structure of the Random Generator function to initialize my deck of cards randomly.

   a.) https://github.com/ml1150258/LehrMark_CSC11_48982/blob/master/Class/Week9/randTest.s

   b.) https://github.com/ml1150258/LehrMark_CSC11_48982/blob/master/Class/Week9/mainDivModFuncV2.s

2.) I previously coded BlackJack in C++ in a previous course; I used this source code as the basis for my project and an outline of sorts when dealing with the logic of the game. I used the deck class as a model for my getCard class in this program as it proved to be very efficient and I was proud of the way it was structured.

a.) https://github.com/amulbham/CSC-17A/tree/master/proj/BlackJack_Project_

3.) This project was a modified version of Project 1 we created early on in the semester. I used parts of the original code however most of it was rewritten to incorporate cleaner and more efficient assembly techniques (push, pull, stack storage, etc). Although the output and appearance of the game appear similar, major modifications were made under the hood to streamline the original code.

a.) https://github.com/amulbham/ASSEMBLY_CSC11/tree/master/proj/BlackJack_Proj_1