Name: Jacob Isbell

Date: 2/29/2012

Assignment: Program 2 – Dr. Bill Eberle

Pseudo-code of the Algorithm:

The game begins in the main.cpp file.

- Welcome statement is displayed
- New instance of class "Game" is created as "blackjack"
- The default Game constructor asks for number of players and checks for that a valid range was entered
- That number of players are then instantiated by using the "Player" class
    - The default constructor for the Player class sets the number of cards = 0 and prompts user to enter a name for the players and stores this value in a var
- The game file then calls "InitDeck" to initialize the deck and create the cards in it complete with their suite, value, and type of card.
- The game file then calls "RandomizeDeck" which uses srand and time to randomly shuffle the cards in the deck.
- The game then gives each player a two cards by using a for loop
- The game will then output the players' name, two cards and their total point value by calling the players[i].getName and players[i].printCards functions, respectively
- The game will also print out the House's cards while hiding the first card and not outputting the total point value of the house cards
- LogicLoop() is then called which is the main logic of the game and marks the end of most of the initialization of the program types.
- Inside LogicLoop(), ask users if they want another hit. It true then give them another card by calling the ".addCard" function and then print their hand again. Also, if the players busts in the processes of getting a hit set that player's ".busted" data type to true.
- Once each player has gotten a chance to take a hit or stand, the house then decides if it wants to take the chance on taking a hit. If the house's point total is less than or equal to 16 the house will take another hit.
- Once the house's turn is completed the program then loops through all of the players finding the one with the closest score to 21 by subtracting the player's total from 21. The player with the smallest difference is the closest to 21 and hence will be compared with the house to see if they beat the house.
- There is handling for if players tie each other. In this case neither wins.

- The winners and losers will then have their names printed with [playerName] wins/loses respectively.
- This concludes the logic loop function at which point the program returns to the main function in main.cpp.
- The game is then deleted to free memory and prevent memory leaks from becoming a major problem.
- The program then enters a while loop which has an argument of “true” to create an infinite loop until broken out of.
- The while loops asks if the user would like to player again and if so, calls the blackjack function constructor again and repeats the process.

Summary:

Design Decisions

I decided to implement the minimum number of class three, of types Game, Player, and Card. This was an effort to cut down on program complexity and make future maintenance and updates easier. I first thought about creating a class named “Deck” to handle the deck, however, I decided that the deck would be better implemented in the Game function as another private data member.

To provide a simple, clutter-free program, I implemented no special rules in the game and no betting.

In the class card, I decided to make three data members. The first two are rank and suit as suggested and the third is the value of the cards. This is especially important for the cards that are not of numerical value in their name (i.e. A, K, Q, J).

I also decided to make the house a type of Player. To account for the various differences in the way the house interacts with the program I had to add a few functions to handle the printing of the house’s hand and decision logic to determine if the house wants another hit.

I intentionally left nearly as much logical code out of the main.cpp as possible. This way it is very easy to see what function gets called and follow its subsequent steps throughout the program. I could have put the “Play again” statements in another function but this did not seem necessary for the code to be “beautiful”.

Issues

At first, there were many issues related to how to handle the house. At first, I thought I would merely take the number of players and add one to that value and set that last player equal to house. While I probably could of done it this way, it certainly would have been much more inefficient than what I ended up implemented – a completely separate declaration of a player with specific functions.

Another issue was how to handle the creation and randomization of the deck. As stated above, my first thought was to create a separate deck class to handle these functions but I ended up merely including this functionality in the Game class. The deck consists of an array of 52 variables of type Card. One of the hard parts about this was figuring out how to assign the card values, suits, and types to each card. I ended up using a 'for' loop with a subsequent series of if and else if statements to set each card to the types in a deck of cards. Another method of doing this would have been to manually set each card to its type, value, and suit but that seemed a bit more inefficient and "messy".

To handle the randomization of the deck I created another function that is called before the players get their cards. The function uses srand(time(0)) to create a truly random number generator.

In the printCards function I had a problem where the program would not output the correct values. This was found to being caused by outputting ASCII characters. This was easily fixed with an if statement that handles such exception.

Notes

Suit characters were outputted with a cout statement in a nested if/else if loop condition so that depending on the value returned from the suit. The program was developed in Ubuntu using the standard g++ compiler.

I wanted to implement some statistical analysis for two various functions. The first would be to determine if the player would prefer an ace to count as 1 or 11 points based on various factors such as the player's current score, the score they would have if they took 11 points, and what the probability of getting another card that would allow them to get closer to 21 without busting would be. The second function I would like to implement a similar logic to is the house hit/stand decision. Instead of the function being a mere "if less than this take hit" I would like for it to take into account what players have busted and what cards they held to determine the probability of the chance of getting a better score without busting.

I would say the hardest part of this program was figuring out how to fit various ideas of how to create the code with what the specified classes and data members were. I still believe that there are much easier ways to create the same program with simpler code in a much cleaner, readable format.