codeJack: The Blackjack Simulator

Submission Details

Due: `Oct 4, 2024 11:59 PM CST`

- *The due date for different sections may vary. Please verify the exact due date with your instructor to ensure timely submission.*

Total Possible Points: 100 points

Documentation Header Reminder

Before you start your assignment, you will need to add documentation similar to what we demonstrated in the first few lectures.

```
// Programmer: San Yeung
// Date: 9/4/1002
// File: fahr2celc.cpp
// Assignment: HW2
// Purpose: this file contains the main function of the program which
//   will input Fahrenheit temps from the user, then convert and output
//   the corresponding Celcius temperature.
```

Function Documentation Reminder

Having proper documentation for C++ functions is important. Make sure that each function includes the following:

- General Description: A brief description of what the function does.
- Precondition: Describe any requirements that must be met before calling the function, e.g., function parameter conditions.
- Postcondition: Explain what the function accomplishes as the aftermath and report any significant changes it makes to persistent variables or data.

Multiple File Compilation

For this assignment (and onward), you will submit **multiple** C++ compilable files containing a program written in C++. Name your files a meaningful name and give it a proper extension (.h, .hpp, .cpp). To compile multiple files in the same directory, simply execute "fg++

*.cpp -o your_executable_file_name" on the CS Linux virtual machine's command line environment. The syntax *.cpp will grab all the .cpp files to be compiled (header files need NOT to be included). Remember, fg++ is just a shortcut for invoking the g++ command with some useful flags turned on, i.e. g++ -Wall -W -s -pedantic-errors, to ensure the quality of the program follows the C++ standard. Objective
The objective of this assignment is to reinforce key C++ programming concepts, including the use of random number generation, default arguments in functions declarations, function overloading, and template functions.

Simulation Specification

Your task is to create a basic version of the Blackjack card game in C++. This simulation will involve a single player (the user) playing against the dealer (the computer).

Responsible Gaming: The intention of this assignment is to foster learning and creativity, not to promote gambling[1]. Please be responsible for gaming. Gambling can lead to serious financial problems, addiction, and other negative social and personal consequences. You have been warned!

1. **Starting Money**
    a. Upon starting the simulation, the program will randomly assign a starting balance to the player AFTER prompting the player to enter their name. This balance is to be used for placing bets within the game. The generated random balance should **always** include two decimal places to reflect cents, with a range within $12.00 to $7000.00, inclusively.
2. **Game Setup**
    a. Usually in Blackjack, a standard card deck of 52 cards is divided into four suits (hearts, diamonds, clubs, and spades), each with 13 cards: Ace through 10, and the three face cards, which are the Jack, Queen, and King. However, in this simplified version of Blackjack, it is safe to assume that each card has an **INFINITE** number of copies in the deck.

1. Please make sure that no actual wallets were harmed in your making of this assignment.

b. The values of the cards are as follows: 2-10 are valued at their face value, Jack/Queen/King are valued at 10, and Aces can be either 1 or 11.

c. **Prompt for Wager:** At the beginning of each round, prompt the player to enter the amount of money they would like to wager on the upcoming hand. Ensure the player input is a valid amount (not more than their current balance).

d. **Hand Initialization:** Each round of the new game begins by the program dealing two random cards to the player and the dealer. Each card's value can be *randomly* generated in the range [1, 13]. While the suit of a card doesn't affect its value in this Blackjack simulation, you are still required to *randomly* assign one of the four suits to each card drawn. Since the uniqueness of each card in this simulation is irrelevant, there's *no need* to check if a card has already been dealt. An example output right after dealing might look like this:

```
Player's Hand: 10 of Diamonds, 3 of Hearts
Dealer's Hand: King of Spades, [Hidden Card]
```

e. **Check for Blackjack:** After the initial dealing, check if either has a hand of Blackjack. A Blackjack is the highest hand and consists of an Ace and a 10-valued card (10, Jack, Queen, or King). This hand is superior to other hands, including those that total 21 but with more than two cards.

i. If only the player has Blackjack, they win.

ii. If only the dealer has Blackjack, the player loses.

iii. If both have Blackjack, it's a tie.

3. Player's Turn

i. **Basic Actions:** After the player's hand and the dealer's up card[2] are dealt, proceed to allow the player to choose "Hit" (get another card) or "Stand" (end their turn). The player can hit multiple times if desired as long as their hand total does not exceed 21; otherwise, the hand is considered to be a bust and the game ends.

2. The dealer's "up card" is the first card dealt to the dealer that is visible to the player and serves as insight into the potential hand strength of the dealer. The second card dealt remains to be faced down.

ii. **The Double Down Rule:** Besides the standard "Hit" option, the player can also choose a special strategic choice known as "Double Down" which allows the player to **double** their wager on their hand in exchange for receiving **ONLY one additional** card. After receiving this one additional card, the player is NOT ALLOWED to take any further actions; they must stand, regardless of the total of their hand. Note that the player can only choose this option if they have enough balance to double down.

1. Note that this option is **mutually exclusive** with the regular "Hit" option. This means that in a round where the player opts to double down after receiving the initial hand, they forfeit the opportunity to "Hit" normally. A player must decide between taking a regular "Hit" to possibly receive multiple additional cards without altering their wager, or "Double Down" for a single card with the chance to double their potential winnings.

4. Dealer's Turn

a. First, reveal the dealer's hidden card. The dealer **must** hit if their total is 16 or less and stand on 17 or higher. If the dealer busts, the player gets the win.

5. Determining the Winner

a. If neither the player nor the dealer has busted, compare the total values of the hands. The winner is the one who has the total point closest to 21.

b. In the event of a tie (when both hand values are equal), neither the player nor the dealer is declared the winner. In addition, the player's wager needs to be returned to them.

6. **Continuous Play and Exit Condition**
    a. As each round concludes, the simulation should then prompt the player to decide whether they wish to play another round. If the player decides not to play, then the program should end gracefully after displaying a summary of the player's performance.
    b. The game should also automatically terminate if the player's balance depletes, signifying that they can no longer place any wagers.

Function Requirements

The following specific, named functions are required. However, the choice of return type and parameters is left to your discretion. It will be your responsibility to decide on the most suitable function signatures based on your own implementation and design choices. So, be creative and effective in your solutions!

1. `generateRandomCard()`: Generate a random card value within an acceptable range.
    a. **Suggested Signature:** `int generateRandomCard(const int min, const int max);`
2. `generateRandomCard()`(**Overloaded**): This special version not only generates a random card value but also incorporates a "luck factor" that occasionally increases the likelihood of drawing a 10-valued card.
    a. Suggested Signature: `int generateRandomCard(const int min, const int max, const int probability);`
    b. In each round, there is a 38% chance[3] that the simulation will favor the player by **increasing** the probability of guaranteeing a 10-valued card. If the player is lucky, then this overloaded function should be used to generate the random card values for the initial hand, passing in a random probability. If such a lucky probability occurs inside the function, then a random 10-valued card is **guaranteed** to be drawn. Please also note that the luck factor influences both

cards drawn in a round, with each draw having its own probability of receiving a favorable outcome.

   c. **Example:** Let's say that in this round the player hits the 38% chance of luck, the game then generates a random probability – for instance 70% – to be passed into the probability parameter of this overloaded function. This means that, when this function is called, there's a 70% chance that a random 10-valued card will be generated for the **first** card draw. When it's time to draw the **second** card, the game then generates *another* random probability – for instance 50% – to be passed into the function.

3. `isBlackjack()`: Determine whether a hand is a Blackjack.
4. `updatePlayerBalance()`: Update the player's balance after each round, taking into account the outcome of the game.
   a. **Payout Schemes:** The payout for a player winning a hand against the dealer is 1:1, meaning the player wins an amount equal to their wager. For example, if the player wagers $20 and wins the hand, they would receive their original wager back **plus** an additional $20 in winnings. The payout for winning a hand with Blackjack is 1.5:1 and double-down is 2:1.
5. `displayOutcome()`: Display a message indicating the game outcome of the current round and any changes to the player's balance. An example output might look something like this:

```
Congratulations! You hit Blackjack!
Balance Update: +$30
Your new balance is $1030.
```

6. displayGameSummary(): Provide a detailed summary of the player's entire game session upon exiting the program, including **total wins, losses, ties, final balance,** and any other relevant statistics to offer additional insights into their performance. An example output might look something like this:

```
=== Game Summary ===
Total Rounds Played: 15
Total Wins (Regular): 5
Total Wins (Blackjack): 2
Total Losses (Regular): 4
Total Losses (Blackjack): 2
Total Ties (Regular): 2
Total Ties (Blackjack): 0
Final Balance: $1030
Net Gain/Loss: +$30

Thank you for playing CodeJack: The Blackjack Simulator!
We hope to see you again soon.
```

Example Expected Input/Output Format

The following example is provided to illustrate the overall workflow and input/output format of the program. Please be aware that the actual outputs from your program will vary.

IMPORTANT!!: Please follow exactly the specific input (*not output*) and action sequence as doing so will simplify the grading process.

- Ensure all inputs during each round are taken in the specified order: **wager input, choice of action(s),** and **decision to replay**.
- Failure to comply may result in the rejection of your submission.

```
Welcome to CodeJack: The Blackjack Simulator!
Player, please enter your name: John

Hi Alex, I'm pleased to notify you that your starting balance is
$1283.45.

Round 1
*******
Player balance: $1283.45
```

```
Please enter your wager: $100

Player Hand: 5 of Hearts, 10 of Diamonds
Dealer's Hand: King of Spades, [Hidden Card]

Player Total: 15

// Note that the option to Double Down is available only on the
player's initial action of the hand.
Player, please choose your action: Hit(H), Stand(S), Double
Down(D): H

Player Hand: 5 of Hearts, 10 of Diamonds, 3 of Clubs
Player Total: 18

// As the player has already chosen to Hit, the option to Double
Down is no longer available.
Player, please choose your action: Hit(H), Stand(S): S

Dealer's Hand: King of Spades, 7 of Diamonds
Dealer's Total: 17
Dealer stands.

Congratulations! You win this round!
Balance Update: +$100
Player new balance is $1383.45.

Play another round? (Y/N): Y

Round 2
*******
Player balance: $1383.45
Please enter your wager: $500

Player Hand: Ace of Hearts, Queen of Spades
Dealer's Hand: 9 of Clubs, [Hidden Card]

Congratulations, player! You have Blackjack!
Dealer's Hand: 9 of Clubs, 6 of Diamonds

Congratulations! You win with Blackjack!
Balance Update: +$750
Player new balance is $2133.45.

Play another round? (Y/N): Y

Round 3
*******
Player balance: $2133.45
Please enter your wager: $1000
```

```
Player Hand: 9 of Diamonds, 2 of Spades
Dealer's Hand: 6 of Hearts, [Hidden Card]

Player Total: 11

Player, please choose your action: Hit(H), Stand(S), Double
Down(D): H

Player Hand: 9 of Diamonds, 2 of Spades, Ace of Hearts
// The player total is 12 and not 22 because otherwise it'd be
busted.
Player Total: 12

Player, please choose your action: Hit(H), Stand(S): H

Player Hand: 9 of Diamonds, 2 of Spades, Ace of Hearts, 7 of Clubs
Player Total: 19

Player, please choose your action: Hit(H), Stand(S): S

Dealer's Hand: 6 of Hearts, 10 of Diamonds
Dealer's Total: 16
Dealer hits.
Dealer's Hand: 6 of Hearts, 10 of Diamonds, 5 of Clubs
Dealer's Total: 21

Sorry, you lose this round.
Balance Update: -$1000
Player new balance is $1133.45.

Play another round? (Y/N): Y

Round 4
*******
Player balance: $1133.45
Please enter your wager: $133.45

Player Hand: Ace of Diamonds, 7 of Hearts
Dealer's Hand: King of Spades, [Hidden Card]

Player Total: 8 or 18

Player, please choose your action: Hit(H), Stand(S), Double
Down(D): D

Player Hand: Ace of Diamonds, 7 of Hearts, 9 of Clubs
Player Total: 17

Dealer's Hand: King of Spades, 6 of Diamonds
Dealer's Total: 16
Dealer hits.
```

```
Dealer's Hand: King of Spades, 6 of Diamonds, 4 of Hearts
Dealer's Total: 20

Sorry, you lose this round.
// Double loss due to the Double Down selection.
Balance Update: -$266.90
Player new balance is $866.55

Play another round? (Y/N): N

=== Game Summary ===
Total Rounds Played: 4
Total Wins (Regular): 1
Total Wins (Blackjack): 1
Total Losses (Regular): 2
Total Losses (Blackjack): 0
Total Ties (Regular): 0
Total Ties (Blackjack): 0
Final Balance: $866.55
Net Gain/Loss: -$416.90

Thank you for playing CodeJack: The Blackjack Simulator!
We hope to see you again soon.
```

Important Notes

- For this assignment, use 5300 to seed the random number generation process.
- If the parameters and return type of a function is not specified, then it is your responsibility to determine the most appropriate function signatures for them.
- You may create additional custom functions for this assignment as you see fit.
- You may use functions from other C++ libraries if 1) they are explicitly stated in the assignment or 2) the usage of the library functions has been thoroughly introduced in class. Otherwise, the usage of other functions is prohibited without the permission of your instructor.
- DO include input/range validations as long as they are appropriate. While input data type validation is not required for this assignment, it is good practice to validate the data values for range.

- Use modular programming! Break down the assignment into smaller functions that perform specific tasks. This will help make your code easier to read, test, and maintain.
- Thoroughly test each function before moving on to the implementation of the next function! This is referred to as the iterative/test-based development approach. It emphasizes on testing and refining each function before moving onto the next one. This helps to identify and fix errors early on, and ensure that the final program is reliable and robust.
- Test the program with different inputs. This will help you ensure that the program is functioning as expected and will also help you identify any bugs or errors.
- Read the error messages carefully if the program is not working as expected. Understanding the error messages will help you to identify and fix the problem.
- Don’t forget to use an adequate amount of comments to explain your code and make it easy to understand.
- You must use proper indentation (two whitespaces) to make the code more readable and easy to understand.
- Don’t forget to have fun and be creative with your program design and implementation! Programming can be both challenging and interesting at the same time!
- Finally, don't hesitate to ask for help from the instructor or TA if you are having trouble with the assignment.

Grading Criteria:

1. **Problem-Solving and Logical Accuracy** (25%)
    - The solution effectively addresses the problem.
    - Logic used in the code is sound and free of major errors.
    - Program behaves as expected under different conditions.
2. **Completeness** (20%)
    - All components of the assignment are addressed.
    - Edge cases or unique scenarios are considered and handled appropriately.
    - The program handles invalid inputs gracefully.
    - The program meets basic function requirements and compiles successfully without any major errors or warnings.
    - **Non-Compiling Work Deduction**
        - If your submission does not compile, an initial deduction of 50% of the total points will be applied.
        - Students whose submissions fail to compile due to **minor issues** are eligible to contact their assigned grader for feedback and may resubmit corrected work.
3. **Optimization and Efficiency** (20%)
    - The solution takes advantage of efficient methods, algorithms, and data structures where applicable.
    - Redundant or unnecessary code or computations are minimized.
4. **Syntax and Structure** (20%)
    - Code follows a consistent format and adheres to C++ standards.
    - Proper indentation is used to enhance code readability.
    - Use of appropriate C++ conventions, functions, and constructs.
    - Demonstrates the ability to organize code across multiple files for modularity.
5. **Clarity and Understandability** (10%)
    - Variables and functions are appropriately named and clearly defined.
    - Logical flow is evident and coherent from start to finish.
6. **Documentation and Comments** (5%)
    - Important steps are commented for clarity.
    - Complex sections of code have accompanying explanations.
    - Each function is accompanied by clear documentation.