ME 493: HW 1

Shane Lawson

February 7, 2017

Disclaimer

If you're looking for information on the `testA` and `testB` functions, skip to sections 2.2 and 2.3, respectively. They are member functions of the `Shoe` class (section 2.1.3). How the user changes the number of decks shuffled is detailed in section 2.1.4. Writing to a text file uses the `fstream` library and consists of functions in the `Card` and `Shoe` classes, sections 2.1.1 and 2.1.3, respectively. I successfully completed all parts of the assignment.

1 Introduction

Some introductory notes to how the program works. I haven't programmed anything of significant size or difficulty in quite a while, so I hoped to use this homework to jump back into things. With that being said, I chose to implement all my classes in separate (.h, .cpp) files, keeping data members and functions private where they should be and public where necessary. I also used const functions and overloaded operators of my classes where these would be beneficial. Anything I came across that sparked a thought of, "I remember doing that before," I went ahead and implemented. Also, this extensive documentation is all because *LaTeX* is fun.

2 Specifics

2.1 Program

In addition to the main program file, I made three classes for this project. A `Card` class, a `Deck` class, and a `Shoe` class.

2.1.1 Card

The `Card` class is simple, and as one might expect, contains an `int` which holds the rank of the card, and a `string` which holds the suit. This class contains a constructor which allows for creating a card with a specified rank and suit (defaults of `0` and `""`), a function to output a card's info to datafile using the fstream library, and an overloaded comparison operator (==) which compares the data of two cards and returns true if they match.

2.1.2 Deck

The `Deck` class contains only a vector of cards, and a function that returns a copy of the cards vector it contains. A constructor is also used here to construct a perfect, unshuffled deck of 52 cards.

2.1.3 Shoe

The `Shoe` class is where the main functionality of the program exists. It contains a vector of cards and an `int` that keeps track of the number of decks contained in the shoe. A constructor creates and fills a shoe with a specified number of decks. Member functions of this class are:

- `shuffle()` : This function shuffles the cards by picking two random indices of the card vector and swapping the cards located at those positions. This function is capable of handling variable size shoes by using the member function `.size()` of the vector class. The number of swaps made is also variable by multiplying the vector size by 20. This produces about 10,000 swaps per deck of 52 cards.
- `printToFile()` : This function opens a file and calls the `.display()` function of the `Card` class for each card in the shoe to print each line of the output file.
- `runTest()` : Depending on the number of decks in the shoe, this function calls either `.testA()` (the test for single decks) or `.testB()` (the test for multiple decks), which are discussed further in sections 2.2 and 2.3, respectively.

2.1.4 Main

The main program consists of a single function which prompts the user for a number of decks to shuffle and makes sure the user inputs a value that is numeric and positive, changing the prompt to ask for a positive integer if it is not. Once this number is captured a shoe is declared with the specified number of decks. The `.shuffle()` function of the `Shoe` class is called, which shuffles all the cards in the shoe. Once the shoe is shuffled, a test is run using the `.runTest()` function of the `Shoe` class to ensure that the cards have been successfully shuffled and no errors have occurred. Next, the `.printToFile()` function of the `Shoe` class is called, creating an output file containing the shuffled cards, which is named "singledeck.txt" or "multideck.txt" depending on the number of decks in the shoe.

2.2 testA Function

This function, a member of the `Shoe` class, `.testA()`, tests if any card in a single deck appears more than once. It starts by declaring a temp `Card` and a `bool` variable to keep track of whether or not a duplicate has been found. A `for` loop is used from `i = 0` to `i < cards.size()` which will loop through once for every card in the shoe. Inside this loop, the card at the `cards` vector index `i` is stored in the temp `Card`. Another `for` loop is used (again from `i = 0` to `i < cards.size()`) to loop through all the cards in the shoe. Each card in the shoe is compared to the temp `Card` (using the overlaoded == operator) and

if a card exists AND is not the same as the temp `Card` (`i != j`) then the `if` statement sets `found` to `true`. After checking each card against every other in the shoe (when the `j` loop is complete), an `assert` is used to verify that `found` is equal to `false` meaning a duplicate has not been found. This repeats for every card in the shoe (the `i` loop), then terminates the function. This function has been tested and proven to catch cases where any card is duplicated in the single deck case, thus meeting the requirements specified in section 2 of the *Homework 1* document.

2.3 testB Function

This function, a member of the `Shoe` class, `.testB()`, tests if the first card of n decks appears more or less than n times. It starts by declaring an `int` variable called `instances` (initialized to `0`) to keep track of how many times a card occurs. An instance of `Card`, called `firstCard`, is also declared and set to the card at the `cards` vector index `0`. Next, a `for` loop from `i = 0` to `i < cards.size()` is used to loop through every card in the shoe. Every time a `Card` at index `i` is equal to `firstCard` (checked using the overloaded `Card` == operator), the `instance` variable is incremented. Once through the loop, the `instance` variable should contain the number of times a card has appeared in the shoe. An `assert` is then used at this point to verify that `instances` is equal to the `numDecks` variable of the `Shoe` class which keeps track of the number of decks in the shoe. This function has been tested and proven to catch cases where the first card of n decks has occurred more or less than n times, thus meeting the requirements specified in section 2 of the *Homework 1* document.