CMPS109 Assignment 2 Report
Max Zhao mlzhao 1466223

Problem 1
Build and run:

```
g++ main1.cpp Problem1.cpp -o main1
./main1
```

Design decisions

With this assignment being my first actual exposure to C++, I took a while to review and understand all the concepts before deciding on the best implementation for each problem. For Problem 1, I simply decided to use an array of unsigned shorts, which take up 2 bytes compared to the 4 an int does.

Following the guidelines of the assignment, the implementation of the problem is relatively simple - I have 1 constructor and destructor for the allocation of the seats array, 1 private variable to store the total number of open seats at any given time, and the required public methods. To track whether or not seats were taken, I simply changed the values of each respective seat accordingly, with '1' indicating that the seat is taken, and '0' indicating that seat is empty. I track the total number of seats available at any given time with a counter, 'openSeats' - intialized to the total number of seats available in the classroom, and decrementing/incrementing respectively as students reserve or make seats available.

Unusual issues

Upon review, I realized that using unsigned_int8, bool, or char would have been more optimized, so I would've made that change given more time.

Problem 2
Build and run:

```
g++ main2.cpp Problem2.cpp -o main2
./main2
```

Design decisions

The design of Problem 2 was fairly similar to the design of Problem 1, with the main differences being that each element in the array had a distinct value, indiciating the stock of each product. With that, I chose to stick to using an integer array, for I ran into some overflow issues when I tried using shorts. Similarly to Problem 1, a single constructor and destructor for the array, a private variable to track the total products available, and the required public methods. I account for the minimum and maximum capacity of stock by setting stock to 256 or 0 whenever the user tries to overflow or underflow the stock. I also have a counter to keep track of the total products available, intialized to zero in the beginning since no items have been entered, incrementing and decrementing respectively as items go into and out of stock.

Problem 3
Build and run:

```
g++ main3.cpp Problem3.cpp -o main3
./main3
```

Design decisions

The design of problem 3 required some thinking - the given pseudocode wasn't the clearest so it took some time to implement. There were a few syntax and range issues that I had to fix. Following the requirements of the assignment, I first started by building a copy constructor, where I use calloc to assign the same memory to the array, and iterate through and copy each respective element. Then, I moved onto the operator overloading functions. For many of the functions, I had to use the 'friend' keyword in order to private variables.

For overloading the + operator, I had to make two separate functions, one that takes input of a heap and integer, and another that takes input of two heaps. These were relatively trivial, for I simply created a new heap that would be able to encompass both inputs, then inserted elements from each heap until the new heap contained all of the elements.

Moving on to overloading the [] operator, I also created a new heap that copied elements from the input heap, so that changes I made during the function wouldn't affect my original heap. I used std's sort library function in order to be able to access individual elements of the heap as if they were in sorted arrays.

For overloading the = operator assignments, I created a custom swap helper function to swap members of the two heaps so that they became equal.

I ran into some issues with overloading the += operator, I found it difficult reallocating/resizing memory, which is why I decided to create a new heap instead of just altering the original one.

The << operator was relatively simple, connecting to C++'s ostream, I simply iterated through each element of the heap's array, and stored them into my returned ostream.

Unusual issues

The given pseudocode was not very concise/well documented, so it was really difficult understanding what each part did. Also, I didn't know how to reallocate or resize memory, which gave me some trouble when trying to optimize my functions.