

Emphasis on: **Stacks and Queues**

This programming assignment actually consists of two parts, one part dealing with a modification of the Stack class and the second modifying the Queue class. You will be given the original source code as discussed by Nyhoff in our textbook of the implementations of these classes. In each part you will modify the class to do something new, then be asked to write a client that uses the modified version of the Stack/Queue to solve a problem.

Part A: Stack

Nyhoff [Stack.h](#)
[Stack.cpp](#)

One variation of a Stack data type is when the stack is not allowed to hold duplicate values. If an item is pushed on the stack that already appears somewhere lower in the stack, the item should be removed from the lower location and placed at the top of the stack. So in effect, a push of an item that already occurs in the stack simply causes the item to be jumped to the top of the stack. Implement the Stack of unique items by modifying the push operation.

With your unique stack implementation, write a client that accepts a string of integers (separated by spaces). The client should push the integers onto the unique stack and, once all integers have been pushed proceed to pop all of them off. This should have the effect of reversing the list of numbers but only printing out any duplicate numbers one time. For example the input:

```
4 5 3 8 9 5 7 3 8 2 5
```

Results in the output

```
5 2 8 3 7 9 4
```

Requirements Part A

- 1) Of course make sure that you implement your unique push correctly. There should never be duplicate values of any items on your unique queue.

Extra Credit Opportunity A

Pushing an item on a unique stack does require that you must look through all of the items to see if it is already on the stack, there is nothing more efficient that you can do besides this sequential search. However, when you remove the item to put it on the top of the queue you could implement this by shifting. How might you instead use a sentinel value (e.g. a dummy value) so that you could avoid doing the shifting. Hint, you would have to modify the pop and top implementations as well. Worth an extra 5%.

Part B: Random Queue

Nyhoff [Queue.h](#)
[Queue.cpp](#)
[QueueTestDriver.cpp](#)

As we mentioned in class there are many variations on the basic queue data type, including priority queues, dequeues, etc. Another example of a queue variation is the *random queue*. In a random queue, items are added to the back of the queue as normal. However, when an item is removed from the queue, instead of taking the item at the front of the queue we instead remove

a random item from the queue (with equal probability of removing any item that is currently on the queue). Modify the Queue class remove method to implement a random queue. Make sure, however, that your remove operation still takes constant time (e.g. don't do shifting of elements when you remove an item, hint: think about doing a swap instead...). You should use a random function, like the rand() function in the C standard library, to choose the item to remove.

With the random queue class you just implemented, write a client that picks numbers for a lottery. Your client should put the numbers 1 through 99 on your random queue. Then it should print the results of the lottery by removing five items and displaying the winning numbers.

Requirements Part B

1) You must modify the remove function of the Queue class in an efficient way. You should not use shifting to fix up holes after you randomly remove an item.

Extra Credit Opportunity A & B

For an extra 10%, use your unique stack and random queues to implement another client to do the following. In this client we are playing a funny sort of card game. The card game consists of 4 players (who will be represented using a unique stack for each player) and a shuffled deck of cards. This game uses 4 decks of cards. Use integer values to represent a deck of cards:

1-13: A, 2, 3, ... J, Q, K of clubs

14-26: A, 2, 3, ... J, Q, K of diamonds

27-39: A, 2, 3, ... J, Q, K of hearts

40-52: A, 2, 3, ... J, Q, K of spaces

Therefore to begin the game you need to push 4 series of the numbers 1-52 onto a random queue (to represent having 4 decks of cards all shuffled into one pack). Each player draws 13 cards from the pack. However, each player should only have a single instance of any card, which is why they are represented by a unique stack. Make sure that each player draws cards and ends up with 13 unique cards in their hands.

Have the program create the deck of cards, the 4 players and then deal the hands to the players. Display the hands of the 4 players after the deal.

For even more credit, how might you go about creating a class to represent a playing card, instead of interpreting the integers 1-52 as particular cards?