

## HW12 -Templates (Queues) [100 pts]

A queue is a first-in-first-out data-storage technique (like a line at a bank teller's window). If one puts in 1, 2, and 3, one gets back 1, 2, and 3. (Remember that a stack is a last-in-first-out data-storage technique. If one puts in 1, 2, 3, one gets back 3, 2, 1.)

Implement an *array-based* queue (i.e. elements are stored internally in a dynamic array). A stack needs only needs to keep track of one index for an array. A queue must keep track of two indices for an array. One index keeps track of the tail of the queue where new items are added. The other index keeps track of the head of the queue where old items are removed.

Use a unique pointer (rather than a raw pointer) data member to represent ownership of queue's dynamic array.

Develop and test the copy constructor and overloaded copy assignment operator. Implement both copy constructor and overloaded copy assignment.

Develop and test the move constructor and overloaded move assignment operator (as needed). Move operations must be noexcept. Use the default move constructor. Implement overloaded move assignment.

Develop and test the following methods:

### ■ Main queue operations:

- `enqueue(object)`: inserts an element at the end of the queue
- `object dequeue()`: removes and returns the element at the front of the queue
- `object front()`: returns the element at the front without removing it

## HW12 -Templates (Queues) [100 pts]

- integer `size()`: returns the number of elements stored
- boolean `isEmpty()`: indicates whether no elements are stored
- boolean `isFull()`: indicates whether queue is full

Write a class template for a queue class. Define several queues of different data types (int, double, string) and insert and remove data from them. Write a member function to print the queues. Use this function to verify the inserts and deletes.

Perform six additions, four deletions, five additions, three deletions. Print the queue after each operation (there should be about 25 output statements per data type).

Test your program by

1. Adding to a full queue and removing from an empty queue
2. Testing the `isEmpty()` and `IsFull()` methods for both the pass and fail conditions
3. Testing the `front()` method

Run *valgrind* to check for memory leaks

Extra Credit [+10 pts]

Implement a *node-based* queue class (suggested name *linkedQueue*). This time queue elements are stored internally as a *singly linked list* of nodes. Perform all addition & deletion operations, tests as outlined above using *linkedQueue*. Run *valgrind* to check for memory leaks.

Use the command script to capture your interaction compiling and running the program, including all operations, as shown below:

## HW12 -Templates (Queues) [100 pts]

CS1C Spring 2023 TTH HW12 100pts **Due: Tu 4/18/2023**

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ script hw12.scr
```

```
Script started, file is hw12.scr
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ date
```

```
...
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ ls -l
```

```
...
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ make all
```

```
...
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ ls -l
```

```
...
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ ./hw12
```

```
... // print queue output after each addition, deletion operation  
above
```

```
... // print output from queue tests 1 thru 3 above
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ exit
```

```
Script done, file is hw12.scr
```

```
cs1c@cs1c-VirtualBox ~/cs1c/hw/12 $ make tar
```

```
...
```

Submit the tar package file hw12.tar to canvas by Tuesday April 18, 2023.