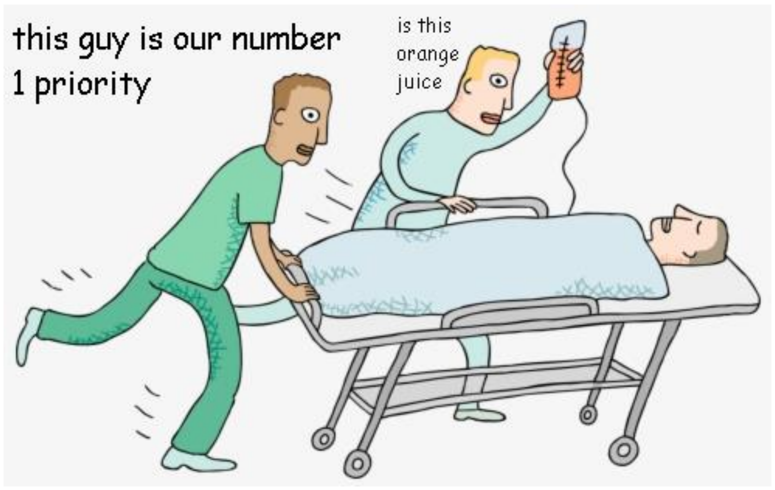
Project 5: Priority Queue ADT



Background:

When you decide to leave or afk in your League of Legends games, you get put into something called *a low priority queue*.

LOW PRIORITY QUEUE

Abandoning a match or being AFK results in a negative experience for your teammates, and is a punishable offense in League of Legends. You've been placed in a lower priority queue.

TIME REMAINING
0:09

LEAVE QUEUE

Believe it or not, the algorithm used to determine the priority might be complex, but the basic idea and data structure used to match you with other players are the same as what you will be making for this project.

If we recall what a Queue is, you can think about it as waiting in a line. The first person on the line is also the first person to come off the line (this is what we mean by First In First Out or **FIFO**). But as you'll see, this isn't necessarily always the case in a priority queue.

Your objective for this project is to implement from scratch **a Priority Queue Abstract Data Structure** using a *Linked Chain* (or linked list) as the underlying mechanism. In order to successfully complete this project, you must understand the concept of the regular **Queue** ADT. A Priorty Queue behaves exactly like a regular Queue, except that each element, when **enqued**, immediately advances past all elements with lower **priority** in front of it. The automatic effect of this is that elements will always **dequeue** in order of their priority, rather than the order in which they were **enqueued**. The order in which they were **enqueued** will only matter for elements with the same priority. Think of like having a long list of tasks, you have task 1, task 2 and task 3 but what if you get a new task, task 4 which is so important that you have to do that before every other task?

Implementation:

- ***You will also be given the hpp files for a PriorityNode and a PriorityQueue***
- You will need to implement each method that resides in the hpp file into a cpp file for the respective class.

Additional resources:

• ***Queue:*** ○ GeeksForGeeks ○ Wikipedia	• ***Doubly-linked list review:*** ○ Implementation and how it works	• ***Priority Queue*** ○ Wikipedia ○ Programiz

Task: Implementing the `PriorityNode` functions!

Just like all of your linked chain projects we have to have a node to link with.

This class has nothing you haven't seen before, you just have to note that in the private members you now have a priority associated with each node.

In this case ***a lower number means a higher priority,*** just think of it like "this is priority number 1" meaning that this should go into the front of the queue.

If nodes are ***tied with priority*** then you can treat it like a normal queue, meaning if node1 has a priority of 3 and node 2 has a priority of 3 then node 2 will just come after node 1 as it normally would.

Everything in this class is just constructors, getters and setter functions so this should be pretty quick to implement.

Task: Implementing the `PriorityQueue` functions!

The public interface for the **Priority Queue** will be essentially the same as the regular **Queue**. In fact, the implementation of all functions but one will be similar as well. The only function that you will need to change is the **void enqueue(const ItemType& new_entry)** function. Again, you must write this function from scratch and not use external libraries to assist you.

You'll see that **enqueue** this time takes in 2 parameters, one is the item itself and the other one being the priority. The priority part was already explained earlier but here's a concrete example:

priorityQueue.enqueue("C",1);
priorityQueue.enqueue("F",3);
priorityQueue.enqueue("E",2);
priorityQueue.enqueue("A",0);
priorityQueue.enqueue("B",0);
priorityQueue.enqueue("D",1);

This sequence should get you A B C D E F

Testing

How to compile:

```
g++ <test main file> -std=c++17
```

You must always implement and test your programs **INCREMENTALLY!!!** What does this mean? Implement and test one method at a time.

- Implement one function/method and test it thoroughly (multiple test cases + edge cases if applicable).
- Implement the next function/method and test in the same fashion. **How do you do this?** Write your own `main()` function to test your class. In this course you will never submit your test program, but you must always write one to test your classes. Choose the order in which you implement your methods so that you can test incrementally: i.e. implement mutator functions before accessor functions. Sometimes functions depend on one another. If you need to use a function you have not yet implemented, you can use stubs: a dummy implementation that always returns a single value for testing. Don't forget to go back and implement the stub!!! If you put the word STUB in a comment, some editors will make it more visible.

Grading Rubric

Correctness 80% (distributed across unit testing of your submission) **Documentation 10% Style and Design 10%** (proper naming, modularity, and organization)

Important: You must start working on the projects as soon as they are assigned to detect any problems with submitting your code and to address them with us **well before** the deadline so that we have time to get back to you **before** the deadline. This means that you must submit and resubmit your project code **early** and **often** in order to resolve any issues that might come up **before** the project deadline.

There will be no negotiation about project grades after the submission deadline.

Submission:

You will submit the following file(s)**:** **`PriorityNode.cpp PriorityQueue.cpp`**

Your project must be submitted on Gradescope. Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You **MUST** test and debug your program locally. Before submitting to Gradescope you **MUST** ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope. "But it ran on my machine!" is not a valid argument for a submission that does not compile. Once you have done all the above you submit it to Gradescope.