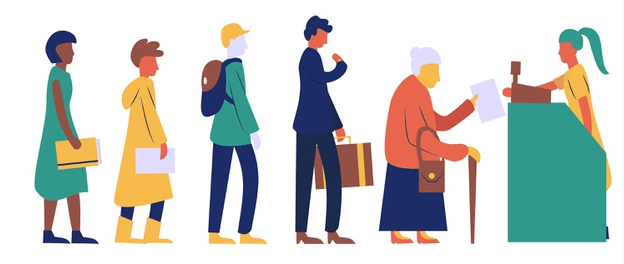
**Project 6   Priority Queue: Linked Chain Implementation**



Your objective for this project is to implement from scratch a **Priority Queue** Abstract Data Structure using a **Linked Chain** 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.

Admitedly, a **Linked Chain** implementation of the **Priority Queue** ADT will result in **O(n)** **enqueque** performance.  You will have a chance to re-implemenmt **Priority Queue** using a **Binary Search Tree** in the **optional extra credit** project.  This immensely superior implementation will result in a much better **O(log n)** **enqueue** performance.  If you choose to do this additional project, it will replace the project for which you received the lowest score.

Also, congratulations for making it to the end of the semester! I hope that you have learned a lot over the last three mnonths and that you have done work that you are proud of. We are aware of how straining it can be to learn about and to work on technical material in a limited timeframe. If you have made it this far, you should feel confident to consider yourself a budding computer scientist, because we certainly do. And good luck on the final!

**Required Files**

[PriorityNode.hpp](https://maryash.github.io/235/projects/project-6/PriorityNode.hpp)  
[PriorityQueue.hpp](https://maryash.github.io/235/projects/project-6/PriorityQueue.hpp)

**Task**

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 the same 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**.

#ifndef PRIORITYQUEUE\_H\_

#define PRIORITYQUEUE\_H\_

template<typename ItemType>

class PriorityQueue

{

public:

PriorityQueue();

PriorityQueue(const PriorityQueue<ItemType>& a\_priority\_queue);

~PriorityQueue();

void enqueue(const ItemType& new\_entry, int priority); //adds by priority

void dequeue(); // removes element from front of the queue

ItemType front() const; // returns a copy of the front element

PriorityNode<ItemType>\* getFrontPtr() const; //returns front\_ pointer

int size() const; // returns the number of elements in the queue

bool isEmpty() const; // returns true if no elements in the queue

private:

PriorityNode<ItemType>\* back\_; // Pointer to back of the queue

PriorityNode<ItemType>\* front\_; // Pointer to front of the queue

int item\_count;

}; //end PriorityQueue

#include "PriorityQueue.cpp"

#endif // PRIORITYQUEUE\_H\_

You will also need to use a new Node<ItemType> class, which will now have three data members:Node<ItemType>\* next\_, ItemType item\_, and int priority\_.

#ifndef PRIORITY\_NODE\_H\_

#define PRIORITY\_NODE\_H\_

template<typename ItemType> class PriorityNode

{

public:

PriorityNode();

PriorityNode(const ItemType& an\_item);

PriorityNode(const ItemType& an\_item, int priority);

PriorityNode(const ItemType& an\_item, int priority,

PriorityNode<ItemType>\* next\_node\_ptr);

void setItem(const ItemType& an\_item);

void setPriority(const int priority);

void setNext(PriorityNode<ItemType>\* next\_node\_ptr);

ItemType getItem() const;

int getPriority() const;

PriorityNode<ItemType>\* getNext() const;

private:

ItemType item\_; // A data item

int priority\_; // The item's priority

PriorityNode<ItemType>\* next\_; // Pointer to next node

}; // end PriorityNode

#include "PriorityNode.cpp"

#endif // PRIORITYNODE\_H\_

**Implementation**

**Work incrementally!** Work sequentially (implement and test). Only move forward, when you are positive that the previous one has been completed correctly. Remember that the names of function prototypes and member variables must exactly match those declared in the respective header file when implementing a class.

**Submission:**

You will submit the following files:  
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.

**Testing**

How to compile:

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

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

* 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 classes. 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 Rubrics**

**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.**