

Assignment 9 - Dynamic Priority Queue

[Submit Assignment](#)

Due Dec 5 by 11:59pm **Points** 0 **Submitting** a file upload **File Types** hpp
Available Nov 22 at 12am - Dec 5 at 11:59pm 14 days

The purpose of this assignment is to have you learn to create your own C++ container and have it work like other containers within the standard library. You'll gain experience in developing an iterator that enables the nice integration into the language and the standard library.

The subject of this assignment is the creation of a `usu::priority_queue` class that allows items to be added with some priority and then have their priority changed while in the container, with the container properly updating based on the updated priority.

This assignment will challenge you to break down the problem into small pieces that you slowly implement and build up over time. Before writing any code, create a plan the details the individual pieces and the order you are going to write them. When asking me for help on this assignment, if I think you haven't done this or are trying to do too much at once, I'll ask you for this development plan.

Assignment

Write a templated `priority_queue` class, contained within a `usu` namespace according to the following specifications.


- An item in the priority queue is composed of a `value` and a `priority`. The `priority_queue` template should set the default priority data type to be `unsigned int`, but allow it to be any data type the programmer chooses (you can assume the type has comparison operators).
 - The priority queue is a max heap, larger priority values represent higher priority.
- The number of elements in the priority queue must be able to dynamically grow (note default constructor below). When the storage size of the queue needs to increase, use this formula: $\text{new size} = (\text{current size}) * 1.25 + 1$
 - Consider using an `std::vector` to store the queue. It has a `.resize` method on it.
- Constructors
 - Default constructor that initializes an empty queue, with an initial reserved storage of 0.
 - Overloaded constructor that takes an `std::initializer_list` of items and initializes the queue with those items.
- Methods
 - `void enqueue(T value, priority_type priority)` : Adds the new value/priority to the queue.
 - `auto dequeue()` : Removes the highest priority item from the queue. If nothing is in the queue, it throws `std::exception`.
 - `iterator find(T value)` : Returns an `iterator` to the item, if it doesn't exist the `.end()` iterator is returned.
 - `void update(iterator i, priority_type priority)` : Change the priority for the item to `priority`, updating

the queue as necessary.

- `bool empty()` : Returns `true` if empty, `false` otherwise.
- `size_type size()` : Returns the number of items in the queue (not the reserved storage size).
- `iterator begin()` : Returns an iterator to the first item in the queue (or `.end()` if nothing in the queue).
- `iterator end()` : Returns an iterator to the "end" of the queue.
- The `priority_queue` must provide an iterator, with at least the following requirements
 - Copy constructable
 - Move constructable
 - Copy assignable
 - Move assignable
 - Incrementable
 - Dereferenceable
 - When the iterator is dereferenced, it returns an object (of your design) that has `value` (of type `T`) and `priority` (of type `priority_type`) members. You'll notice I use structured bindings (destructuring) in the `reportPQ` method to grab the values from this object. As long as the object you return can property bind to this code, however you implement that object is fine with me.
 - Overload both the `*` and `->` operators.
 - Relational operators to test for equality and inequality
- You'll need to define the following type aliases
 - `priority_type` : The data type of the priority for the value is added to the queue.
 - `value_type` : The data type of the value that is added to the queue.
 - `size_type` : The data type returned for the number of items in the queue; probably should be `std::size_t`.

I'm leaving a lot of the detail for you to figure out and decide on your own. The provided main driver code and unit tests provide the strict interpretation of what must be done, but they follow the standard library conventions. How you decided to store the items in the queue is up to you, how you create the iterator is up to you (because there are a lot of ways to implement one, but they all must meet certain requirements).

Example Usage & Unit Tests

An example [main.cpp](#)  is provided that exercises some (definitely not all) of the capabilities of the code you need to write. Additionally you are provided a set of unit tests that exercise the capabilities of the priority queue; it is fairly comprehensive.

Files

- [main.cpp](#) 
- [TestPriorityQueue.cpp](#) 

Implementation & Other Notes



A few items to help you out with this project:

- Even though it is not supposed to be required (by the updated C++ standard), g++ still needs an iterator to derive from the standard library iterator to define the iterator tag; MSVC doesn't require it, but in order

to compile on both platforms, you need to do it. Here is the class declaration to do that...

- `class iterator : public std::iterator<std::forward_iterator_tag, priority_queue*>`
- The following references may help you with the priority queue
 - <https://opensa-server.cs.vt.edu/ODSA/Books/Everything/html/Heaps.html> [\(https://opensa-server.cs.vt.edu/ODSA/Books/Everything/html/Heaps.html\)](https://opensa-server.cs.vt.edu/ODSA/Books/Everything/html/Heaps.html) (I like this implementation, my recommendation for the basis of your code)
 - <https://robin-thomas.github.io/min-heap/> [\(https://robin-thomas.github.io/min-heap/\)](https://robin-thomas.github.io/min-heap/)
 - <https://algorithms.tutorialhorizon.com/binary-min-max-heap/> [\(https://algorithms.tutorialhorizon.com/binary-min-max-heap/\)](https://algorithms.tutorialhorizon.com/binary-min-max-heap/)
- The following reference may help you with iterators, in addition to my notes and sample code
 - <https://en.cppreference.com/w/cpp/iterator> [\(https://en.cppreference.com/w/cpp/iterator\)](https://en.cppreference.com/w/cpp/iterator)

Submission Notes

- Turn in only the following file: **priority_queue.hpp**
- Your code must compile without any warnings or compiler errors; against the provided [main.cpp](#)  and [TestPriorityQueue.cpp](#)  provided code. See syllabus regarding code that has compiler errors.
- Your code must adhere to the CS 3460 coding standard: [link](#)
- Your code must be formatted through the use of the following clang-format configuration file: [link](#)

Example Run

The following is the expected output from the "simpleExample" function in the provided code. I was shooting for a moon lander game, but it just wasn't working out well, too complex and distracting from the main purpose of the assignment...the priority queue.

```
--- Found item using std::find_if(...): c(3)

--- Before ---
e(4) c(3) b(2) d(1) a(1) f(2)
--- After Updating a ---
a(5) e(4) b(2) d(1) c(3) f(2)
--- After Updating b ---
a(5) e(4) b(3) d(1) c(3) f(2)
--- After Updating b ---
a(5) e(4) b(4) d(1) c(3) f(2)
--- After Updating b ---
a(5) e(4) b(5) d(1) c(3) f(2)
--- After Updating b ---
b(6) e(4) a(5) d(1) c(3) f(2)
--- Emptying The Queue ---
a(5) e(4) f(2) d(1) c(3)
e(4) c(3) f(2) d(1)
c(3) d(1) f(2)
f(2) d(1)
d(1)

--- Initialized from std::initializer_list ---
e(4) c(3) b(2) d(1) a(1) f(2)
```