Programming Assignment #1, Phase 2

The Program:

For this assignment, you will write two implementations of a *Priority Queue*. For this ADT, removal operations always return the object in the queue of highest priority that has been in the queue the longest. That is, no object of a given priority is ever removed as long as the queue contains one or more object of a higher priority. Within a given priority First-In-First-Out (FIFO) order must be preserved.

Your implementations will be with data structures:

- Ordered Array
- Unordered Array

Both implementations must have identical behavior and must implement the `PriorityQueue` interface (provided). The implementations must have two constructors, a default constructor with no arguments that uses the `DEFAULT_MAX_CAPACITY` constant from the `PriorityQueue` interface (implemented with a pure virtual class), and a constructor that takes a single integer parameter that represents the maximum capacity of the priority queue. The `PriorityQueue` interface follows.

This assignment will be the second of two phases. Phase 1 consisted of implementing the two Array classes: `UnorderedArrayList` and `OrderedArrayList`. Phase 2 consists of implementing a Priority Queue for each array class type `UnorderedPQ` and `OrderedPQ`.

Phase 2 instructions

To begin, read *Separate Compilation*. If you skip this step, you may struggle to understand how the program files work together.

Class prototypes defined in `UnorderedPQ.h` and `OrderedPQ.h` are provided. Each of these classes are subclasses of the pure virtual `PriorityQueue` class defined in its header file. This file is also provided; you may not modify `PriorityQueue .h`

Your project Phase 2 will consist of the following files *in addition to Phase 1 files*. You must use exactly these filenames.

`PriorityQueue.h`	The ADT interface (provided).
`OrderedPQ.cpp`	The ordered array implementation, based on `OrderedPQ.h` (provided).
`UnorderedPQ.cpp`	The unordered array implementation, based on `UnorderedPQ.h` (provided).

Download the header files and makefile from this assignment *zip* file.

Your task is to write *your own code* for the required public functions in `UnorderedPQ.h` and `OrderedPQ.h`. You may add private members as needed. Some public functions may be useful for debugging purposes, but these must not interfere with the required functions.

`UnorderedPQ.cpp` and `OrderedPQ.cpp` should be implemented by calling methods from the Phase 1 list classes you implemented. If you are rewriting the same code as `OrderedArrayList.cpp` or `UnorderedArrayList.cpp`, then you're on the wrong track. Focus on how to use the Phase 1 objects to implement Phase 2 functions with very little new code.

Testing your code is essential and may require more time than writing the classes. Be sure to test combinations of functions.

Additional Details

Each method must be as efficient as possible. That is, a O(n) is unacceptable if the method could be written with O(log n) complexity. Accordingly, the ordered array implementation **must** use binary search where possible, such as `contains()`, `delete(int obj)`, and to identify the correct insertion point for new additions.

By convention, a **lower number=higher priority**. If there are five priorities for a given object, 1 .. 5, then 1 is the highest priority, and 5 the lowest priority.

Your project must consist of only the files specified (including the provided interfaces), no additional source code files are permitted.

You may not make any modifications to the PriorityQueue interface provided. I will grade your project with my copy of this file.

All source code files must have your name and class account number at the beginning of the file.

Your implementations may only include `"OrderedArrayList.h"`, `"UnorderedArrayList.h"`, and `<stdexcept>`. While you are debugging, you can include `<iostream>`, but remove any printed output before turning in your program. You are expected to write all the code yourself, and you may not use the C++ Standard Library API for any containers.

Your code must not print anything.

Your code should never crash but must handle any/all error conditions gracefully, i.e. if the user attempts to call the clear() method on an empty PQ, or remove an item from an empty PQ, the program should not crash. Be sure to follow the specifications for all methods.

You must write generic code according to the interface provided. You may not add any public methods to the implementations, but you may add private ones, if needed.

Your code may generate unchecked cast warnings when compiled, but it must compile and run correctly on Gradescope to receive any credit. Testing on edoras may prove to be a helpful exercise.

Minimal tester/driver programs will be provided to help you test your code. You should add more tests as you think of more scenarios.

Allowing sufficient time for testing on the grading platform is essential. If testing on edoras, use your ~/sandbox/p1/ with the following files:

```
YourTesterForPriorityQueue.cpp
AbstractList.h
PriorityQueue.h
OrderedArrayList.h
OrderedArrayList.cpp
UnorderedArrayList.h
UnorderedArrayList.cpp
OrderedPQ.h
OrderedPQ.cpp
UnorderedPQ.h
UnorderedPQ.cpp

makefile
```

Turning in your project

To submit your project, you must upload the following C++ source code files to your `Gradescope` account through Canvas.

```
OrderedArrayList.h
UnorderedArrayList.h
OrderedPQ.h
UnorderedPQ.h
OrderedArrayList.cpp
UnorderedArrayList.cpp
OrderedPQ.cpp
UnorderedPQ.cpp
```

Cheating Policy

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. During the grading process I will examine your code carefully. Anyone caught cheating on a programming assignment (or on an exam) will receive an "F" in the course, and a referral to Judicial Procedures.