31251 Data Structures and Algorithms
Assignment 2

1 Overview

For this assignment you will be implementing a data structure that we have mentioned in class, but not examined in detail, the priority queue. A priority queue is similar to a queue, except that elements are inserted into the queue with an associated priority value, and are removed from the queue according to those priority values, rather than the order the were inserted.

Priority queues are integral to many scheduling algorithms for many tasks. They are also used as fast ordered containers for algorithms where it is important which options are examined first. They appear in implementations of many searching algorithms in this context.

The priority queue you will implement will use `int`s to establish the priority of the elements, however it will use templates for the element type. For the assignment, the priority order will be *lowest value first*. So an element with priority 10 will come before an element with priority 23. All priorities should be non-negative integers (*i.e.* 0 and above).

There are many ways to achieve the functionality of a priority queue, you may want to do some basic research to decide which approach is appropriate for your goals. For those aiming for the highest marks, you will have to use a particular approach, detailed below.

2 The Code

You are provided with 2 C++ files:

- `PriorityQueue.h`
- `Assignment2Test.h`

The only file you should modify is `PriorityQueue.h`. That's right, you'll be putting source code in a header file![1] This will also be the only file you submit.

[1] For those interested in the structure of C++, what would prompt this behaviour?

As before, you may add any functions you wish to your code, but you are not allowed to `#include` any further headers (and you shouldn't need them). In particular, you can't just use the `std::priority_queue` class.

Note that the as yet unseen class `std::pair<T1, T2>` is used. The documentation is at `http://en.cppreference.com/w/cpp/utility/pair`, but the only components you will probably need are `.first` and `.second` (note that they are variables, not functions, so no `()` at the end).

As before, carefully read the skeleton code before beginning your implementation, including the tests. The details may affect the choices you make.

3 Goals

For those aiming for anything less than an HD, your goal is to implement the functionality of the priority queue according to its external interface (the comments in `PriorityQueue.h` outline how the functions behave). What internal representation you employ, and how the priority order is implemented is up to you. There are several "easy" methods.

For those aiming for an HD, you are required to implement a more restricted version. You must maintain your internal representation of the queue as a min-heap. This data structure has been briefly mentioned in class, but part of your task is to research heaps and how to implement them[2]. You will also have to consider the tests that assess this property, and produce suitable output.

3.1 Order of "Difficulty"

While it is hard to predict what different coders find difficult and what they don't, my suggested order of difficulty (easiest to hardest) is as follows:

1. `size()`
2. `empty()`
3. `get_all_elements()`
4. `get_all_priorities()`
5. `insert(int, E)`
6. `insert_all(std::vector<std::pair<int, E> >)`
7. `peek()`
8. `contains(E)`
9. `get_priority(E)`
10. `remove_front()`
11. `change_priority(E, int)`

Due to the nature of the data structure, the functions are not as independent as with the first assignment. To pass most of the tests, you will need working `insert(int, E)`,

[2]This should not actually be difficult, they are covered in great detail in every data structures text book and certainly on the internet.

`size()` and `empty()` functions at least, so it should be a priority in your implementation. However, you can complete this assignment without any pointers whatsoever.

It should also be pointed out that the choice of internal data structure has been left up to you. Two library container classes have been included if you wish to use them (`std::vector` and `std::list`), however, if you wish, you may use arrays, build your own linked list, or any other approach that doesn't require further `#includes`.

As with the first assignment functions will be divided amongst the four grade levels, however you may complete them in whichever order you see fit. The rubric below is designed to outline judgement of your achievement level, not to limit your marks.

4 The Test File

For the functionality testing, we will use CxxTest. You are provided with a copy of the test file so you can check your progress as you complete the assignment. To make the scoring work correctly, the test file needs to be built with the additional options we used in Assignment 1. It may work without these options, but the scoring will be inaccurate, and some assertions guarding against crashes might fail.

To get the scoring working:

```
>cxxtestgen --have-eh --abort-on-fail --error-printer -o Assignment2Tests.cpp Assignment2Tests.h
```

Of course you will have to run `cxxtestgen` as appropriate given your local setup. After this however, the compilation works as before (and you can pick whatever output name you want).

5 Marking

The overall mark, as beforeconsists of three parts, functionality, design and style.

5.1 Functionality

Functionality will be assessed by the automated test file and is worth 70% of the total mark. The marks will be assigned according to the following rubric:

Pass	Full completion of `size()`, `empty()`, `get_all_elements()`, `get_all_priorities()`, `insert(int, E)`, `insert_all(std::vector<std::pair<int, E> >)` and `peek()`.
Credit	Pass level requirements, plus full completion of `contains(E)` and `get_priority(E)`.
Distinction	Full completion of all functions in `PriorityQueue.h`.
High Distinction	Full complete of all functions, plus employment of a min-heap as the internal data structure.

5.2 Design

The design of your assignment will be assessed by your tutor during the in-class demonstration and is worth 20%. The marks will be assigned according to the following rubric:

Pass	The code shows basic understanding of how to employ data structures to achieve a goal. The design should avoid unnecessary data structures and should make reasonable use of iteration and recursion when appropriate.
Credit	The design shows a solid understanding of data structures and demonstrate effective use of control structures to achieve the program's goals.
Distinction	The design shows a high degree of understanding of how to use data structures to achieve a goal efficiently, and demonstrate some evidence that the design does not use unnecessary resources. The design should be clean and efficient.
High Distinction	The design demonstrates a high degree of understanding of data structures and how to efficiently employ them to build algorithms that not only meet technical goals, but support maintenance and future development.

5.3 Coding Style

Coding style will also be marked by your tutor during the in-class demonstration, it is worth 10% of the total mark and will be marked against the following rubric:

Pass	The code mostly uses some formatting standard and is somewhat readable.
Credit	The code adheres well to a formatting standard and variables are well named.
Distinction	At least as well formatted as for Credit standards, along with sufficient inline commenting to explain the code.
High Distinction	Excellent formatting and variable naming. Excellent, judiciously employed comments that explain the code without just repeating the code.

5.4 Marking Schedule

If you submit your assignment on time, and no other issues arise, we expect to return your marks within a week after your in-class demonstration. We do however reserve the right to delay this schedule if necessary.

6 Plagiarism Detection Software

You assignment may be submitted to plagiarism detection software at the course coordinator's discretion.

7 Submission

You will submit your `PriorityQueue.h` file using UTSOnline. As UTSOnline has issues with submitting code files, please zip you submission and name the resulting file using your student number as before. For example, if your student number is 123456, please name the file `123456Assignment2.zip`.

8 Due Date

The assignment is due by 5pm, Friday, June 2nd. See the subject outline for late submission penalties.