Assignment Priority Queues and Heaps

COSC 2336: Data Structures and Algorithms

Spring 2022

Objectives

- Implement Priority Queue API functions.
- Better understand using heap data structure to implement priority queue functions.
- More examples and practice of using inheritance and class abstractions / ADT in C++.

Description

In this assignment, you will implement a heap based priority queue.

Overview and Setup

For this assignment you will be given the following files that you will be using and adding code to for this assignment.

File Name	Description
`src/test-APriorityQueue.cpp`	Unit tests of the array based `APriorityQueue` implementation of the `Queue` API
`include/Queue.hpp`	Header file of the ADT base `Queue` class that defines the `Queue` interface / abstraction
`include/AQueue.hpp`	Header file of the concrete array based implementation of the `Queue` abstract data type
`include/APriorityQueue.hpp`	Header file of the concrete array based implementation of a priority queue for the `Queue` abstract data type
`src/Queue.cpp`	Implementation file of common methods of the `Queue` base class
`src/APriorityQueue.cpp`	Implementation file for the `APriorityQueue` member functions that implement the concrete priority queue array based `Queue`
`src/LPriorityQueue.cpp`	Implementation file for the `LPriorityQueue` member functions that implement the concrete priority queue linked list based `Queue`

This week you will be adding all of your functions to the `APriorityQueue.[hpp|cpp]` source files as member functions of an array based (heap) priority queue class.

As usual, before starting on the assignment tasks proper, you should make sure you have completed the following setup steps.

1. Copy the assignment repository on GitHub using the provided assignment invitation link for 'Assignment Recursion' for our current class semester and section.
2. Clone the repository using the SSH URL to your local class DevBox development environment.
3. Configure the project by running the `configure` script from a terminal.
4. Confirm that the project builds and runs, though no tests will be defined or run initially. If the project does not build on the first checkout, please inform the instructor.
5. You should create the issue for Task 1 and/or for all tasks for the assignment now before beginning the first task. On your GitHub account, go to issues, and create it/them from the issue templates for the assignment. Also make sure you link the issues to the `Feedback` pull request after creating them.

Assignment Tasks

Task 1: Implement the `APriorityQueue` `fixUp()` member method.

As usual make sure that you have created Task 1 on your GitHub repository for this assignment and have linked the Task 1 issue to the open `Feedback` pull request in your GitHub assignment repository.

Add in the `fixUp()` member method for our priority queue. This method will be implemented as a private member method of the `APriorityQueue` class. This method can be implemented in mostly the same was as shown in our textbook, but modified to use our `APriorityQueue` class array of values and member variables.

An `exchange()` method has already been defined for you to use in the `APriorityQueue` class. This method simply takes the index of the heap value where you need to start the bottom-up reheapify operation. The `values` array of values that holds the heap is a member variable (inherited from the `AQueue` parent base class.

Uncomment the test case for task 1 and implement this function. When you are satisfied with the funciton, commit it and push your work to your GitHub class repository.

Task 2: Implement the `AProirotyQueue` `fixDown()` member method

Start task 2 by uncommenting the next unit test in the `test-APriorityQueue.cpp` and creating the function declaration and a stub function to make sure the tests run.

This task is similar to task 1, but you will implement the `fixDown()` re-heapify method to fix a heap from a node down to the leaf. This member method will again be a private member method. And likewise it will be a void function that takes a single integer value as its input parameter, which again is the index of the value in the heap array of values to start the top-down re-heapify process at.

You will need to use the `values` member array and also the `size` member variable in this method.

Uncomment the test case for task 2 and implement this function. When you are satisfied with the funciton, commit it and push your work to your GitHub class repository.

Task 3: Implement the `APriorityQueue` `enqueue()` member method

Do the same as previous 2 tasks to start task 3, uncomment the next test case in `test-APriorityQueue.cpp`, add in the declaration and stub function for the `enqueue()` method.

Implementing the enqueue method will be simpler than what we had to do to maintain an ordered array of items in our previous assignment. For `enqueue()` the steps (as shown in our textbook) are:

1. Call `growQueueIfNeeded()` to ensure the `values[]` backing storage array for the heap is large enough to add another value to the end.
2. Increase the size of the queue by 1. Since we are not using index 0 of the `values` array, the size is also the index of the last item in the array being maintained as a heap. By increasing the size by 1 first, the `size` now also indicates the index location to add in the new value being enqueued.
3. Add the new indicated value being enqueued into the backing array at the end of the array.
4. Call your `fixUp()` method to do a bottom-up re-heapify operation, starting at the new value index you just added.

Uncomment the test case for task 3 and implement this function. When you are satisfied with the funciton, commit it and push your work to your GitHub class repository.

Task 4: Implement the `APriorityQueue` `dequeue()` member method

Uncomment the next test case 4 in `test-APriorityQueue.cpp`, add in the declaration and stub function for the `enqueue()` method.

The `dequeue()` method uses a similar approach to `enqueue()`, and we will reuse your `fixDown()` method for dequeue. The general algorithm to implement the dequeue operation is:

1. First test if a `dequeue()` is being attempted on an empty queue. If the queue is empty, throw a `QueueEmptyException` instead of trying to perform the operation on an empty queue.

2. Otherwise, exchange the value at the top (or root) of the heap with the last value in the heap array. The root or top of the heap is always at index 1 of the array. Likewise the `size` of the queue will be the index of the last item in the array, since we are treating the array as a 1-based indexed array for this heap.
3. After exchanging the root value, which is the highest priority item we are dequeuing, you can reduce the size of the queue by 1.
4. Then we need to do a bottom-down re-heapify operation. Call your `fixDown()` method on the heap root to re-heapify the heap.

Uncomment the test case for task 3 and implement this function. When you are satisfied with the funciton, commit it and push your work to your GitHub class repository.

Assignment Submission

For this class, the submission process is to correctly create pull request(s) with changes committed and pushed to your copied repository for grading and evaluation. For the assignments, you may not be able to complete all tasks and have all of the tests successfully finishing. This is ok. However, you should endeavor to have as many of the tasks completed before the deadline for the assignment as possible. Also, try and make sure that you only push commits that are building and able to run the tests. You may loose points for pushing a broken build, especially if the last build you submit is not properly compiling and running the tests.

In this problem, up to 25 points will be given for having at least 1 commit that compiles and runs the tests (and at least some attempt was made to work on the first task). Thereafter 15 points are awarded for completing each of the 5 tasks. However you should note that the autograder awards either all points for passing all tests, or no points if any test is failing for one of the tasks. Also note that even if you pass all tests, when the instructor evaluates your assignment, they may remove points if you don't follow the requirements for implementing the code (e.g. must reuse functions here as described, need to correctly declare parameters or member functions as `const` where needed, must have function documentation correct). You may also loose points for style issues. The instructor may give back comments in pull requests and/or create new issues for you if you have issues such as these, so it is good to have work committed early before the due date, so that the instructor may give feedback requesting you to fix issues with your current submission.

Program Style

At some point you will be required to follow class style and formatting guidelines. The VSCode environment has been set up to try and format your code for some of these guidelines automatically to conform to class style requirements. But not all style issues can be enforced by the IDE/Editor. The instructor may give you feedback in your pull comments and/or create issues for you for the assignment that you need to address and fix. You should address those if asked, and push a new commit that fixes the issue (or ask for clarification if you don't understand the request). In general the following style/formatting issues will be required for programs for this class:

1. All programs must be properly indented. All indentation must be consistent and lined up correctly. Class style requires 2 spaces with no embedded tabs for all code indentation levels. The editor style checker should properly indent your code when you save it, but if not you may need to check or correct this if code is misaligned or not properly indented.
2. Variable and function names must use `camelCaseNameingNotation`. All variable and function names must begin with a lowercase letter. Do not use underscores between words in the variable or function name. Often function names will be given to you, but you will need to create variables, and maybe some functions, that conform to the naming conventions.
    - Global constants should be used instead of magic numbers. Global constants are identified using `ALL_CAPS_UNDERLINE_NAMING`.
    - User defined types, such as classes, structures and enumerated types should use camel case notation, but should begin with an initial upper case letter, thus `MyUserDefinedClass`.
3. You are required to use meaningful variable and function names. Choosing good names for code items is an important skill. The code examples and starting code tries to give examples of good and meaningful names. In general, do not use abbreviations. Single variable names should be avoided, except maybe for generic loop index variables `i`, `j`, etc. Make your code readable, think of it as writing a document to communicate with other developers (and with your instructor who will be evaluating your code).

4. There are certain white space requirements. In general there should usually never be more than 1 blank line in a row in your code. Likewise there should usually not be more than 1 blank space on a line. There should be 1 blank space before and after all binary operators like `+`, `*`, `=`, `or`.
5. Function documentation is required for all regular functions and all class member functions. You need to follow the correctly formatted Doxygen function documentation format. We will use function documentation generation, and you should be sure your documentation can be built without emitting warnings or errors. Likewise all files should have a file header documentation at the top. You should edit the file header of files where you add in new code (not simply uncommenting existing code). Make sure the information has your correct name, dates, and other information.
6. Practice using proper Git commit messages. You should refer to issues and tasks correctly in commit messages.

Additional Information

The following are suggested online materials you may use to help you understand the tools and topics we have introduced in this assignment.

- Lecture U10-1 Implementing Queues
- Data Structures - Full Course Using C and C++ The videos 4:32:17 Introduction to Queues through 4:56:33 Linked List implementation of Queue are all excellent introductions to the fundamentals of implementing Queues as arrays and linked lists.
- C++ Classes and Objects
- C++ Inheritance
- C++ Templates