# Project 3: Priority Queue

CS271: Data Structures

January 2024

## 1 Learning Goals

In this project we learn how to implement a heap data structure and then use that heap to implement a priority queue which will complete operations in $O(n \lg n)$ time.

## 2 Heap

We will implement a max heap as described in Chapter 6 of our textbook. You are to build a complete `Heap` class. This implementation should be entirely in a `Heap.h` file. The class should use dynamically allocated arrays as the underlying heap data structure. The class should be templated where the datatype must supply the less-than and greater-than operators (necessary for heap operations). You might want to refer back to the CS173 assignment where you created a `List` class using a dynamically allocated array. You will likely need an array, a capacity variable (size of array including used and unused slots), and size variable (current number of items in the array). You will likely need to be able to re-allocate the array in the event it fills up.

You should implement the following methods:

- Constructors: default, copy, int parameter (which creates an empty heap with the specified capacity), and with array/size parameters (so that the heap can be initialized from an array).

```
Heap<int>   h1 (int a[], int size);
```

- Destructor, assignment operator.
- heapify as specified in the book.
- buildHeap as specified in the book.
- heapSort (implement this even though we may not use it).
- increaseKey as specified in the book.
- insert as specified in the book.

- length – returns the number of items in the heap.
- empty – true if empty, false otherwise
- max – returns the max item in the heap without removing it.
- extract – removes and returns the max item in the heap.
- Also overload the `cout <<` operator as a friend function so that you can print out the heap (print the items in the array, separated by spaces, on one line).

```
Heap<string>   h1;
...
cout << "h1 = " << h1 << endl;
```

You should throw an exception or print an error and terminate the program if you attempt to max() or extract() from an empty heap or if you provide an index that is invalid in one of the other methods.

# 3 Priority Queue

You are to implement a `PQueue` class as a priority queue. The priority is decided using the > operator where items that are greater are given higher priority. Your PQueue class should be templated (allowing any datatype that implements the > and < methods). You should use your Heap class as the private member of your priority queue which will provide all the implementation functionality. That is, the heap class is used by composition to build the priority queue.

Your priority queue must implement the following methods:

- Constructors: default, copy, and with array/size parameters (so that the priority queue can be initialized from an array).

```
PQueue<int>   pq (int a[], int size);
```

- Destructor, assignment operator.
- enqueue – inserts a new item by priority.
- length – returns the number of items in the priority queue.
- empty – true if empty, false otherwise
- peek – returns the first item in the priority queue without removing it.
- dequeue – removes and returns the first item in the priority queue.
- Also overload the `cout <<` operator as a friend function so that you can print out the priority queue.

```
PQueue<string>   pq1;
...
cout << "pq1 = " << pq1 << endl;
```

You should throw an exception or print an error and terminate the program if you attempt to peek() or dequeue() from an empty queue.

# 4 Details

## 4.1 Acceptable Resources

Use mainly your textbook as your resource. You may consult your cs173 assignment on dynamically allocated arrays. I want you to code the algorithms yourself.

## 4.2 Submission

With your project provide all the following files in a tar/zip file.

1. `Heap.h` that implements your heap class.
2. `PQueue.h` that implements your priority queue class.
3. `main1.cpp` that fully tests and demonstrates the correct operation of all your heap methods (including heapSort) using at least two different datatypes.
4. `main2.cpp` that fully tests and demonstrates the correct operation of all your priority queue methods using at least two different datatypes.
5. `makefile`. This makefile should compile two different targets. It should create a program called `main1` which runs the program for testing the heap. It should also create a program called `main2` which runs the program for testing all your priority queue operations.

## 4.3 Group Dynamics

This is a group project. You are to complete this project with your assigned partners. It is important that each person contribute something of value to the project in a reasonably balanced way. I will ask you to enumerate the contributions of each member of the group, so please keep a record of this information. At the end of the unit, you will also be asked to provide an account of the work of other team members. Each team member should be accountable for work that is of high quality and is completed in a timely manner.

## 4.4 Linux Compatibility

You can develop your project on any platform you wish, but you should absolutely test your project on the linux machines in Olin 219. I will evaluate your project on these linux boxes. It is your responsibility to ensure that your submission works perfectly on the linux machines regardless of where you develop the code. It is surprising how often code that works on one platform might not work correctly on a different platform. Usually this incompatibility is identified at compile time, but there can be execution differences too. It is best just to develop the code on the linux machines.

## 4.5 github

It may be a good idea to create github repositories for your projects and include your project partners in the access privileges for the repository. That way you can manage a group of people all developing different parts of the project, even different parts (ie different methods) of the same file. It is much easier than emailing code back and forth or using other techniques to share code development. Since you have multiple projects with the group, the overhead work of creating the repository pays dividends over multiple assignments.

## 4.6 Documentation

It is necessary to fully document all your code correctly. This includes

- A full comment header at the top of each source file that contains the filename, the developer names, the creation date, and also a description of the file's contents.
- All methods should have a full comment block including a description of what the method does, a Parameter section that lists explicit pre-conditions on the inputs to the function, and a Return section that explicitly describes the return value or any side effects of the project.
- Judicious use of inline comments to comment different blocks of logic or particularly confusing code statements where the semantic understanding isn't necessarily immediately obvious from the syntax.