# CENG 202 PROGRAMMING ASSIGNMENT 2 (Priority Queues)

In this assignment, you will be implementing a priority queue class that stores strings. You can enqueue strings in any order you wish. Whenever you extract a string from the priority queue, the priority queue will remove and return the lexicographically first string in the queue.

As you have seen in lecture, there are multiple ways to implement each of the collections classes. Your assignment is to implement a priority queue class in four different ways. Those are:

- Unsorted Vector (**pqueue-vector.h**). The elements in the priority queue are stored unsorted in a Vector.
- Sorted linked list (**pqueue-linkedlist.h**). The elements are stored in a sorted, singly-linked list.
- Unsorted linked list (**pqueue-doublylinkedlist.h**). The elements are stored in an unsorted, doubly-linked list.
- Binary heap (**pqueue-heap.h**). The elements are stored in a binary heap, a specialized data structure well-suited to priority queues. While you are free to write these implementations in any order that you wish, we strongly suggest implementing them in the above order.

You will implement the *cpp* files for the given interfaces (*h* files). You must do all of your own memory management. In the implementation you will use the The Stanford C++ Libraries (http://stanford.edu/~stepp/cppdoc/) that were originally developed by Stanford Professor **Eric Roberts,** with the assistance of Julie Zelenski**,** Keith Schwarz**,** and other Stanford colleagues**.**

A driver application **pqueue-test.cpp** is provided to test your implementation. This testing code contains two important tools. First, it contains an interactive testing environment in which you can issue individual commands to your priority queue. This is designed to let you test your priority queue on specific inputs. Once you are comfortable that your priority queue works correctly, you can run the automated test suite.

*This assignment is adapted from Stanford CS106B Programming Abstractions in C++.

**Extra Credit Opportunity: Build Your Own Priority Queue!**

The four heap implementation strategies you will implement in this assignment are only a small sampling of the myriad implementations of priority queues. For extra credit, research one of the following priority queue implementations, then implement the ExtraPriorityQueue class using one of these more complex (but more efficient!) designs:

- Binomial heap: A type of heap built out of a collection of smaller trees called binomial trees. Binomial heaps support the merge operation efficiently: given two binomial heaps, it is possible to combine them together into a single binomial heap in time O(log n), where n is the total number of nodes in the two heaps.
- Fibonacci heap: If you're up for a real challenge, try implementing the Fibonacci heap! This data structure is a modified version of a binomial heap that supports the decrease-key operation efficiently. Instead of sorting the strings alphabetically, instead you would associate each string with a priority and then sort the strings by priority. The decrease-key operation then lets you low the priority of an already-enqueued element. Fibonacci heaps have excellent theoretical properties, but tend to be slow in practice.
- Pairing heap: A pairing heap is a simpler alternative to the Fibonacci heap that is slightly slower on huge inputs but is faster on small inputs.

*This assignment is adapted from Stanford CS106B Programming Abstractions in C++.