Lab 11: Priority Queue

1.1 Information

Topics: Basic priority queue

Turn in: All source files (.cpp and .hpp).

Starter files: Download on GitHub or D2L.

```
Lab 11 - Priority Queue/
├── lab11_main.cpp
├── lab11_PriorityQueue.hpp
└── input_message.txt
```

This program does not have any tests associated with it

1.2 About: Priority Queues

"In computer science, a priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

While priority queues are often implemented with heaps, they are conceptually distinct from heaps. A priority queue is an abstract concept like "a list" or "a map"; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods such as an unordered array." [1]

[1] From Wikipedia, Priority queue, https://en.wikipedia.org/wiki/Priority_queue

1.3 Reassembling messages

For this lab, we will just work on a small program using a "Naive Priority Queue". These are relatively inefficient, and just implemented with an array.

For this program we will have an input file full of messages that have been received out of order. When working with packets over a network, packets may be received out of order and need to be reassembled. We will use a priority queue to store the messages in the correct order.

Follow the instructions to implement two functions in the priority queue.

1.3.1 About the Node

For this Priority Queue, the Node looks like this:

```
1 template <typename T>
2 struct Node
3 {
4     T data;
5     int priority;
6 };
```

The Node will keep track of the item's data and its priority. Note that a lower priority # will mean a higher priority item.

1.3.2 GetIndexOfHighestPriority

This function is responsible for returning the index of a Node with the highest priority.

Follow these steps:

1. Create variables to keep track of the highest priority item...
   (a) Create a new variable whose data type is *int* named `minPriority`. Set it equal to the first Node's priority level: `m_data[0].priority`.
   (b) Create a new variable whose data type is *int* named `minIndex`. Set it to 0.
2. Create a for loop that iterates from index 1 to the `m_itemCount` (exclusive), incrementing by 1 each time. Within this loop...
   (a) If the current item's priority, `m_data[i].priority`, is less than the current `minPriority`, then set the value of `minPriority` to this item's priority, and `minIndex` to this index (i).

1.3.3 Push

After the error check / exception throw if statement, add the following code...

1. Create an integer variable named `insertAt`, and initialize it to the `m_itemCount`.
2. Create a for-loop that starts at 0 and goes until `m_itemCount` (exclusive), incrementing by 1 each time. Within this loop...
   (a) If the priority value passed in, `priority`, is smaller than the priority of the current item, `m_data[i].priority`, then...
      i. set `insertAt` to the index (i)
      ii. call the `ShiftRight` function, passing in the index.
      iii. Then, break out of the for loop statement.
3. Outside of the for loop...
4. Set the data at the *insertAt* position (`m_data[insertAt].data` to the `newData` passed in as a parameter.
5. Set the priority at the *insertAt* position (`m_data[insertAt].priority` to the `priority` passed in as a parameter.
6. Increment `m_itemCount` by 1.

1.4 Example output

Once you're finished, the output should look like this:

```
37 items in priority queue

0.  100 Deleted code is
1.  102 debugged code.
2.  106 - Jeff Sickel
(AND SO ON...)

Highest priority: Deleted code is
```