Programming Assignment 3 (part 2)

Syed Abeer Hasan Zaidi

March 7, 2019

Program Description and Purpose of Assignment

The program is a Queue data structure implemented using the templated doubly linked list as both a guide and as a tool. The purpose of this assignment was to reinforce our knowledge of queues, and of the adapter design pattern implementation we'd learned about in class. This assignement was also meant to show us the usefulness of doubly linked lists.

The Data Structure

The Data Structure in this assignment was a Queue, called minQueue, implemented through the use of a doubly linked list. The Queue is meant to have a doubly linked list as a private data member and perform the basic queue operations using it. The class had a second private data member, capacity, which kept track of how many elements are in the queue at any time.

Compiling and Running Instructions

I made a makefile for this part of the assignment, using the ones from part 1 as a template. I compiled it in the terminal for my macbook using **make all**, and then ran it with **./run-queue**. When running the program performs a variety of tests on the functions in the class. The tests included populating the queue, dequeueing, min(), and returning the first and last elements.

Logical Exceptions and Bugs

The templated doubly linked list has one bug where it will only take in the desired input after its DListNode and constructor is edited where the empty object is indicative of the desired input. The program will compile, but when it runs we encounted a segmentation fault, this goes away when we change e = 0, header(0) and trailer(0) to e = “”, header(“”), trailer (“”). This bug carried over into minQueue, however it seems to be restricted to when we make a queue of strings.

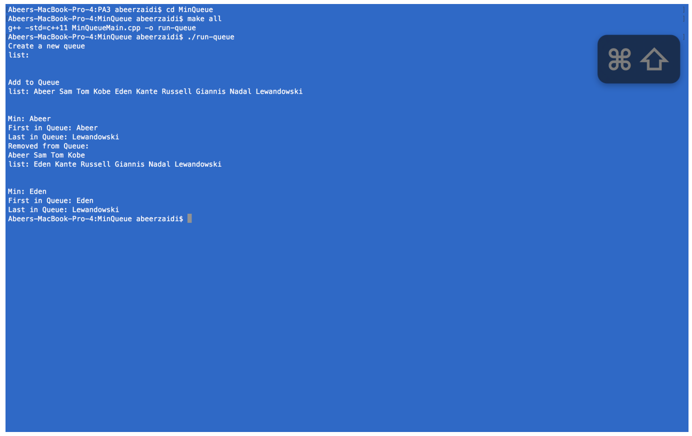

Figure 1:

Queue Implementation

The Queue was implemented with heavy utilization of the templated doubly linked list. The list was used in almost every part of the program since all the enqueued values were stored in the nodes of the linked list using the new call. The constructor and destructor for minQueue was simply calling the constructor and destructor for the doubly linked list. When a value was dequeued, it was deleted using the removeFirst() function in the doubly linked list. When a value was queued, it was added using the insertLast() function of the doubly linked list. The only function that didn't rely on the doubly linked list was size(), this used the private data member capacity that was tracking the number of elements in the queue.

Complexity Analysis

The time complexity of every function in this is data structure, except for min(), was O(1). This was because none of those functions iterated through the queue in order to return, add, or remove an element so their runtime wasn't affected by the number of elements in the queue. However, min() had to iterate through the entrire queue in order to find the element with the smallest value, and as a result had a time complexity function of O(n), where n is the number of elements in the queue.

Testing Evidence

Figure 1: MinQueue testing with the type set to string

Figure 2: MinQueue testing with the element type set to char

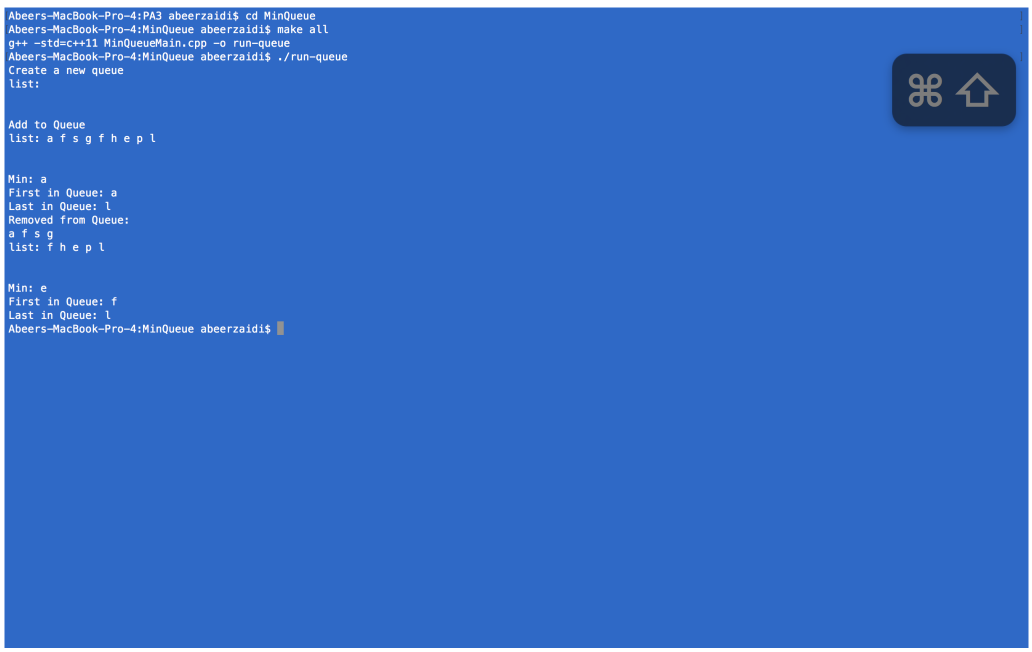
Abeers-MacBook-Pro-4:PA3 abeerzaidi$ cd MinQueue
Abeers-MacBook-Pro-4:MinQueue abeerzaidi$ make all
g++ -std=c++11 MinQueueMain.cpp -o run-queue
Abeers-MacBook-Pro-4:MinQueue abeerzaidi$ ./run-queue
Create a new queue
list:
Add to Queue
list: a f s g f h e p l
Min: a
First in Queue: a
Last in Queue: l
Removed from Queue:
a f s g
list: f h e p l
Min: e
First in Queue: f
Last in Queue: l
Abeers-MacBook-Pro-4:MinQueue abeerzaidi$

Figure 2: