CSCI 2720: Data Structures

Fall Semester, 2016

Project 1

Instructor: Dr. Eman Saleh

Due date

Part 1: 09/29/2016 at 11:30PM
Part 2: 10/02/2016 at 11:30PM

The goal of this project is to give you an opportunity to practice implementing and using queues. You should also continue to practicing good software engineering approach, as well as, using object-oriented concepts in your design and implementation.

Part1: Priority Queue

A priority queue is a queue in which items are stored in an order determined by a numeric priority

Value, that a task with priority 1 comes before a task with priority 2. This interpretation, means lower numerical value has a highesr priority. Thus, a priority queue is not a FIFO structure.

Write the implementation of the **PriorityQueue** class using **one** of the following representations:

1. **A dynamic array that stores the elements in priority order**
2. **A linked list that stores the elements in priority order**

The methods in the PriorityQueue class

For each of the implementations, the ***PriorityQueue*** class is exported through the pqueue.h header file. The operations you need to implement are always the same; what changes is the internal representation, which is of little concern to the client. The methods exported by the ***PriorityQueue*** class appear in the table below on the next page. As you can see from the table, the methods in the ***PriorityQueue*** class are similar to those in the QueueADT lectures (chapter 5 in the text book). In our previous homework we faced a complexity problem due to lack of a method that returns the item at the front of the queue without removing it from queue, in this implementation we'll add this method to the ***PriorityQueue*** class, there are two more differences:

(1) The enqueue method takes an extra argument indicating the priority of the item

(2) There is a new peekPriority method, which returns the priority associated with the queue item with highest priority (the item at the front of the queue).

PriorityQueue Functions:

`enqueue(`*value*`, `***priority*`)`**	Adds *value* at the appropriate position in the queue as determined by the numeric value *priority*. As in conventional English usage, lower numeric values correspond to higher levels of urgency, so that a task with priority 1 comes before a task with priority 2. If the *priority* parameter is missing, it defaults to priority 1.
`dequeue()`	Removes the value at the head of the queue and returns it to the caller. Calling **`dequeue`** on an empty queue causes a runtime error.
`peek()`	Returns the value of the most urgent item in the queue without removing it. Calling **`peek`** on an empty queue causes a runtime error.
`peekPriority()`	Returns the priority of the most urgent item in the queue without removing it. Calling **`peekPriority`** on an empty queue causes a runtime error.
`clear()`	Removes all the elements from a queue.
`size()`	Returns the number of elements currently on the queue.
`isEmpty()`	Returns **`true`** if the queue is empty.

Implement a test program to test the priority queue, you may add printQueue function to the PriorityQueue class call it after dequeuer and enqueue functions to make sure that your queue is working correctly.

What to submit:

1. The priorityqueue.h file
2. PriorityQueue.cpp file
3. The PriorityTest.cpp file.
4. A README.txt file telling us how to compile your program and how to execute it.
5. Submit to ***nike*** (The directory will be announced soon)

The following test case may help with understading how should your ***PriorityQueue*** work:

Test cases	Output of PrintQueue after test case	
Q.enqueue('C', 2)	**C**	
Q.enqueue('D', 4)	**C D**	
Q.enqueue('A', 1)	**A C D**	
Q.enqueue('B', 3)	**A C B D**	
Q.dequeue (?)	**C B D**	
Q.Peek()	**C B D**	**note: output of cout<< Q.peek() is C**
Q.peekPriority()	**C B D**	**note: output of cout<< Q.peekpriority() is 2**
Q.dequeue (?)	**B D**	

Part2:

For this project, you will implement a simulator that simulates customers entering and leaving a bank. You program will take as input the number of tellers, the arrival time and service time of each customer. You may use an input file that will look as follows:

```
numTellers 3
Customer1  arrival 25 service 70
Customer2  arrival 40 service 40
Customer3  arrival 60 service 124
Customer3  arrival 75 service 32
```

In this example, there are three tellers working at the bank. Customer 1 arrives at time 25. Since there are no other customers in the bank, all tellers are free and the customer can be serviced immediately. The cusomer's departure time is calculated as **95** (25+70). Customer 2 arrives at time 40, can be serviced immediately, and is scheduled for departure at time **80**. Customer 3 arrives at time 60, can be serviced immediately, and is scheduled for departure at time **184**. Customer 4 arrives at time 75. Unfortunately, all tellers are busy. At time 80, customer 2 leaves, freeing a teller and customer 4 can be serviced. Customer 4's departure time is scheduled as 112 (**80**+32).

The output of your program will be a list of the wait times and wait plus service times for all customers. You can output this information to a file or to standard output. For the previous example, the output would be the following:

```
wait 0  wait and service 95
wait 0  wait and service 80
wait 0  wait and service 184
wait 5  wait and service 112
```

Your program must use your implementation of the ***QueueType*** class preferred the circular queue which was left an exercise. You can use any implementation for the queue type. Your program must also use the ***PriorityQueue*** you have implemented in part1 of this project. The ***PriorityQueue*** enqueue function will insert items (customers) into the queue where departure time is considered as the priority of the inserted item (i.e. an item *i* has a departure time less than the departure times of all items after *i* in the queue.)

Your program must declare *two* instances of the ***QueueType*** class. The first will store information about each customer arrival that has not yet been processed. When your program starts, you should open the input file and read in the customer arrival data. For each arrival, create a QueueItem and insert it into the queue. ***You can assume that the data stored in the file will be sorted by arrival time.***

The second instance of ***QueueType*** will store information about customers waiting in line. In other words, if a customer arrives and cannot be serviced, that customer's arrival and service information is stored in the waiting queue.

Your program must also declare an instance of the ***PriorityQueue*** to store information about customers currently being serviced. When a teller is freed, the next customer in the waiting line is removed from

the waiting line and put into the line of customers being serviced. If no customers are waiting, the teller remains unoccupied until another customer arrives.

What to submit:

1. The QueueType.h and QueueType.cpp files
2. The ***PriorityQueue***.h and ***PriorityQueue*** files
3. A README.txt file telling us how to compile your program and how to execute it.
4. Submit your ***Project1*** directory to **cs2720b** On nike. You must use the submit command to submit your work, as shown below:
   $ submit Project1 cs2720b
   Important: You should execute the submit command while being in the parent directory of Project1
5. An application BankSimulationTest.cpp to test the bank simulation problem.
6. Make sure that each function is well documented. Your documentation should specify the type and function of the input paramaters, ouput and pre and post-conditions for all functions.
7. Run your program on a variety of inputs ensuring that all error conditions are handled correctly.
8. **Reminder:** *No* part of your code may be copied from any other source unless noted on this page. All other code submitted must be original code written by you.

Evaluation of the projects will include:

1) evaluating test cases using a pass or fail metric
2) programming style. In addition to ensuring your program compiles and runs, you are also responsible for proper documentation. Proper documentation includes proper function commenting (i.e. purpose, pre-, and post- conditions) and explicit directions on how to compile and run your programs.

Preparation:

You must include the following comment at the top of the program file. Copy and agree to the entirety of the text below, and fill in the class name of your .cpp or .h file, your name, submission date, and the program's purpose.

```
/*
 [File name here].cpp
 Author: [Your name here]
```

Submission Date: [Submission date here]

Purpose: A brief paragraph description of the program. What does it do?

Statement of Academic Honesty:

The following code represents my own work. I have neither received nor given inappropriate assistance. I have not copied or modified code from any source other than the course webpage or the course textbook. I recognize that any unauthorized assistance or plagiarism will be handled in accordance with the University of Georgia's Academic Honesty Policy and the policies of this course. I recognize that my work is based on an assignment created by the Department of Computer Science at the University of Georgia. Any publishing or posting of source code for this project is strictly prohibited unless you have written consent from the Department of Computer Science at the University of Georgia.

*/

In the future, every file of every project you submit must have a comment such as this.

Otherwise, points will be deducted from your project.