Project 5 – CMSC 202

UMBC Scheduler

1. Overview

In this project, you will:

- Practice basic C++ syntax including branching structures
- Write classes and instantiate those classes using a constructor
- Create a templated data structure (queue or stack)
- Use simple file input
- Use overloaded operators to access templated data structure

2. Background

Many of you have had the opportunity to register for courses for next semester already. As such, you were probably able to navigate the sometimes-frustrating process of finding a course only to have it cruelly ripped out from under you when the course was full. What if you had the ability to be added to the waitlist and then when the waitlist was full, a new section of the class was offered automatically?! What a magical place that would be.

For this project, we will be building a scheduler that takes in a list of student requests and will generate the required courses to meet the demand for those students. In order to do this, we will be building a templated data structure that works like a queue where the juniors and seniors would be inserted into the front of the structure and the other students would be added behind them. We will be building this structure using a dynamically allocated array.

3. Assignment Description

Class 1 – Tqueue – This is a very important class. It is used to manage each Classes roster and waitlist. It is templated so that it could store the information about whatever it is designed to hold (in this case Student pointers). As a dynamically allocated array, the Tqueue has a maximum capacity (which is passed to it) You will need to implement normal queue functions such as Enqueue (which inserts in the end), Dequeue (which removes from front), Sort (which sorts the content in the Tqueue – NOTE: For the sort function, you may need to write it specially to use pointers), capacity functions (including IsEmpty, IsFull, and Size), an overloaded[] operator (to get specific location in the Tqueue), and ClearData (which removes all data from the Tqueue). Finally, you must implement the copy

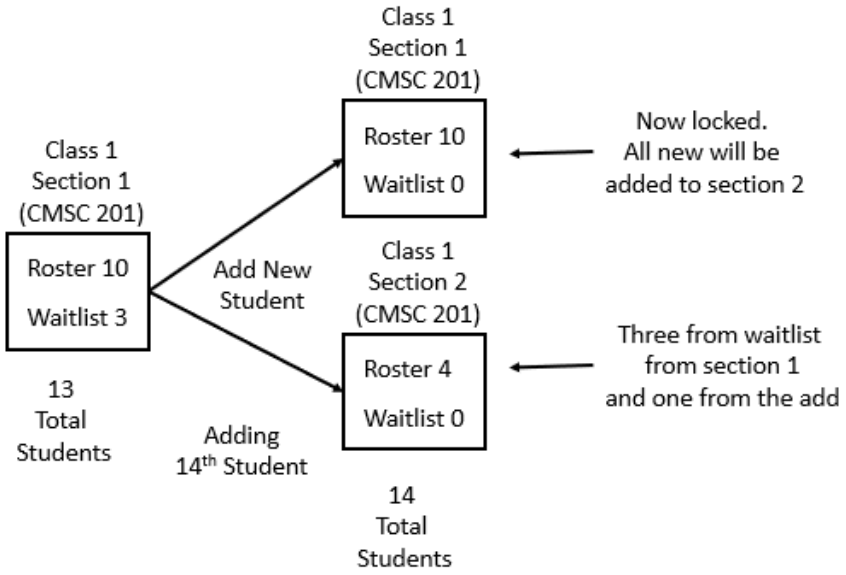
constructor and assignment operator in this class. There should be absolutely **NO** references to anything about classes or students in Tqueue!

You should implement this class by itself and then test it completely before using it. There is sample test code at the bottom of Tqueue.cpp. You can uncomment it as you code functions to test it incrementally. Do not forget how we must implement templated classes!

Class 2: Class – For this project, a Class represents a specific offering of a course. A Class object is made up of a name (like CMSC 201), a section (integer starting at 1), and two Tqueues (one named m_roster and one named m_waitlist). The m_roster represents the number of students in the class (starts capped at 10 per class). The m_waitlist represents the students waiting for the class (starts capped at 3 per class). When a class's roster and waitlist are both full, a new section of that class is opened and everyone in the waitlist is moved to the new class. Otherwise, read the function descriptions for more details in Class.h.

Once a class's waitlist is full, a new section of that class is created, and the students from the waitlist are moved to the new class's roster, the original class is closed. The waitlist from the first section will never be repopulated.

For example, if class 1 – section 1 has 10 students on the roster and 3 on the waitlist when we add another student to class 1, a new class (section 2) would be created and the three from the waitlist and the new student would all four be moved to the roster of section 2. The figure below describes how it should work:



Class 3: Scheduler – This is the class that loads the file of students and loads students into specific classes. It is called directly from driver.cpp. The main menu manages the scheduling tool and allows the user to display all classes (and sections), rosters for specific sections, search for a specific string (searches first and last names) and sorts all rosters. Scheduler does most of

the heavy lifting of the waitlist mechanics listed above as it loads a file. For more details, check out Scheduler.h.

Class 4: Student – This is the class that contains all the information about each student. Each student has a first name, last name, and student ID. There is an overloaded << operator and a < operator (for sorting).

The input files are formatted with first name, last name, student id, and the course.

```
Thane,Durham,46205927,CMSC201
Blossom,Mcfarland,28382208,CMSC202
Juliet,Ware,97530887,CMSC203
```

There are four test files (proj5_test1.txt, proj5_test2.txt, proj5_test3.txt, and proj5_test4.txt) each increasingly complicated (and large). The first test file is named proj5_test1.txt and has the student's names be numbers such as One One or Six Six. This is to help with debugging the **m_roster** to **m_waitlist** process described above.

4. Requirements:

This is a list of the requirements of this application. For you to earn all the points you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standards (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **<iostream>, <fstream>, <iomanip>, <vector>, <cmath>, <ctime>, <cstdlib>, <sstream>,** and **<string>**. You should only use **namespace std**.
- Using the provided files, **Tqueue.cpp, Student.h, Class.h, Scheduler.h, makefile, proj5_test1-5.txt and the driver.cpp file** write the program as described. (Finish **Tqueue** first though!) You must use a **Tqueue** to build a **Class** which is done in **Scheduler.h.**
- As a reminder, **Tqueue.h** is templated and all functions must exist in ONE file (**Tqueue.cpp**). **Class** uses **Tqueues** to manage the roster and waitlist but could easily be modified to use practically any type of data. **Tqueue** must include a functioning **copy** constructor and **assignment** operator.
- Class member variables must be **private** for project 5.
- All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an

integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.

- The code must not have memory leaks.