3460:210 Project Part D

# **Project Assignment Part D: My Computer Lab**

**Overview**

The purpose of this part of the project is to make sure that you know how to write a program that declares and uses a queue, makes use of a map, some error handling, and leverages file usage and file operations.

**PROGRAM SPECIFICATION**

The universities really liked the speed that the latest version using a linked list gave them, and there have been several requests for enhancements and for some new functionality. These changes are included in what we will deliver in rolling out part D.

Recap of current system

The computer lab is a system that hosts users of several universities’ computer labs, and allows users of the labs to log into and log off from the available machines. The computer lab system also keeps track of the machines that are available and those that are in use, allowing users to locate an existing computer that is free as well as free the computers (remove users) upon the user’s logout.

Specifications for the program

You have 8 universities under contract. These labs contain computer stations that are hold the work stations numbered as shown in the table below:

|  |  |
| --- | --- |
| Lab Number | Computer station numbers |
| 1 | 1-19 |
| 2 | 1-15 |
| 3 | 1-24 |
| 4 | 1-33 |
| 5 | 1-61 |
| 6 | 1-17 |
| 7 | 1-55 |
| 8 | 1-37 |

The universities would like to add the ability for the users to be able to have their requests held when the labs are full, and additionally when a lab machine becomes available to have those held requests, the next waiting user’s request, added to their particular linked list. The user that has the oldest request for each lab should be the next one to get a spot, and so on. For the task of doing this the software architect has recommended using a queue. When the software architect recommends something, we have to implement it. Only one queue will maintain the requests for all the labs (not individual queues for each lab), so a bit more work is involved here. The user and all of their input must be placed on the queue, and their associated lab must be tracked as well. For the input, you might be considering if you could use your lab user’s class for the queue. You might also think about implementing the queue as a linked list. These sound like good ideas for the structure. One thing that is different in this waiting process from normal login is that the user no longer has the choice of which work station. As requests are held on this queue they will be slotted into the next available work station when one opens up, and thus any input they had regarding a desired station can be ignored. Again in all this queue creation and processing the lab is key, and that has to be captured and held (along with name and time). For the operations/processing, a critical part might be when the request or action happens that frees up a spot and simultaneously places them from the waiting list into a lab, because at that time their entry on the queue should be removed as they are inserted into their desired lab. Your program design will need to consider not only *how* to perform these operations, but importantly *when*. For example, when a station on a list that was previously full becomes available, and someone is waiting to get a computer station for that particular lab, you will need to trigger the activities that handles these movements. Your queue class should use the STL version of the queue for this implementation (have it hold your lab user class). You will need to make updates to the check capacity function as well. Consider changing the messaging to let the user know that they will be added to the next available station. When they are added, of course, a message should appear as well.

Next, let’s implement error handling in our program. At a minimum, write the logic to trap errors in **two** different parts of your program (your choice what and where). You must explicitly call out in your code comments where you are using error handling or you will not receive credit ☹.

Next, add an option on the menu (option #5), that will interactively print the log on the console, displaying all of the items in a uniform way. Code the function(s) to print the log. The formatting is up to you, but it must display the following info:

* Timestamp
* Action - I (display login), and O (display logout)
* UserID - pad with leading zeroes
* User time
* User name

… and must be neat. Don’t forget to change your logic for the menu to handle the new choice, making a #6 to quit the program.

Finally, do this last step *after* you have made all of the above updates.

The software architects have also been looking at potential bottlenecks, and have asked for our help. After we successfully implemented this latest release, the architects are recommending that we analyze all of the code, code design, and coding structures to find and make changes that will positively impact the software’s efficiency. Your task will be to go through and identify **five** inefficient areas in the program, and subsequently update the program with the correction/optimization. These can be code constructs, rearrangements of code or rearrangements of function calls, etc… These should be well thought out, and valid. Changing something without solid reasoning will not yield any points. For each of these changes the requirement is to place a comment line exactly as follows: //OPT # (where number is 1 through 5). Follow that with comments that articulate and clarify what you found, and why you changed it. Keep and include as part of the comments the exact code that you changed (before version). Then follow all these comments with the updated code. The five areas are totally up to you to find and change, but it must follow good reasoning and is limited to the code only (not the data structures). Remember, you must explicitly include this in the 5 places in your code where you are updating to receive any credit ☹.

*Design of the structures*

We will continue to use the following code that creates a constant fixed array of length 8 for the labs. This is our array that holds the number of possible work stations. NUMLABS is also created to enable us to add or subtract a lab. Using constants here allows our program to be much more dynamic in terms of modification efforts and control. This array is also used for the allocation of the linked list (the sizes).

*Design of the application*

Everything else in your system should remain intact, and the functionality should be the same. Carefully consider your redesign effort strategy to maintain your functionality, and to change code as minimally as possible. Test thoroughly, and test incrementally.

Once again, there is **a lot to change** for the new release: 1) create and maintain a single queue that holds all of the overflow of users seeking labs that are full, with logic to check the queue against the linked list for any availability; 2) add some basic error handling, 3) display the log file and add a new menu option to support the log (with formatting), and; 4) Identify and change 5 areas in your code that will gain more efficiency. Also, you can combine your new code into your existing application file, or create more application files.

Submission Instructions – for programming solutions

On Brightspace, go to the matching Assignments for the **PROJECT PART-#**, where # is the number or character of the project part assigned (eg., 5 for project part 5), and submit your cpp, and any hpp files (if they exist). Unless otherwise specified, they may be submitted under any name that you prefer (such as main.cpp).

*Last updated 5.22.2017 by Will Crissey.*

*Be aware that programming falls under all of the rules of plagiarism. Be careful when using any coding found in the outside world that is not your own. Any evidence of plagiarism is subject to sanctions like forfeits, suspension, and even ejection, as determined by the Department of Student Conduct and Community Standards.*