3460:210 Project Part A

# **Project Assignment Part A: 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 declare and uses pointers and does basic input and output.

**PROGRAM SPECIFICATION**

You have signed a contract with several major universities to run their computer labs for their respective students. To get you started with your own computer labs, you will need to write some software to support your new business endeavor.

Introduction

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 can run many computer labs. For now, we will have 8 universities under contract. These labs contain computer stations that hold the actual physical work stations that are 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 |

We are going to build each lab as an array. We also will be using a strategy to make these structures which allows them to be different sizes, and this is known as a jagged structure. We will be using a separate array for each university’s computer lab, and those arrays will be different sizes based upon the number of labs required by each university. For example, The University of Akron may have 44 lab stations while Case Western University may have 28, and so on.

We are going to create an additional data structure to help out with our programming efforts by reducing the amount of coding that we have to do. We will build an array which holds the values (above) of the maximum number of possible stations for each given computer labs. These represent the sizes of the jagged member arrays. So this array will simply be used to build the 2D jagged arrays (see design of structures below). Let’s consider this static array to be a control support structure.

Each student or user has a unique five-digit ID number. Whenever a user logs on, the user’s ID, the lab number, and the computer station number are transmitted to (input into) your system. For example, if user 49193 logs onto station 2 in lab 3, then your system receives user ID 49193, lab number 3, and computer station number 2 as the input data. Similarly, when a user logs off a station, then your system receives the lab number and computer station number.

Your program is used to track, by lab, which user is logged onto which computer, and additionally the program has the capability to maintain those labs’ computer stations as per the menu-driven requests. The array of lab sizes (one for each lab) holds the values of the maximum number of possible stations for each given lab, and then those sizes are used to build our jagged arrays, that we will dynamically allocate using pointers that point to the elements that each hold the station number values, i.e., the user ID. All labs are of the type int. More details are provided under the design of structures sections. Default values for stations should set to -1 (this indicates it is free or empty).

A sample view of the lab arrays is given here, where user 49193 is logged into station 2 in lab 3 and user 99577 is logged into station 1 of lab 4 then your system might look as follows:

|  |  |
| --- | --- |
| Lab Number | Computer station numbers |
| 1 | 1: empty 2: empty 3: empty 4: empty …… 19: empty |
| 2 | 1: empty 2: empty 3: empty 4: empty … 15: empty |
| 3 | 1: empty 2: 49193 3: empty 4: empty ……… 24: empty |
| 4 | 1: 99577 2: empty 3: empty 4: empty …………… 33: empty |
| 5 | 1: empty 2: empty 3: empty 4: empty …………………..… 61: empty |
| 6 | 1: empty 2: empty 3: empty 4: empty .… 17: empty |
| 7 | 1: 89098 2: 67890 3: empty 4: empty …………………. 55: empty |
| 8 | 1: 12345 2: 67899 3: empty 4: empty ………….… 37: empty |

*Design of the structures*

To make our program more dynamic, you are given the following code that will create a constant fixed array of length 8 for the labs. This array holds the number of possible work stations. This is a simple array of integers. 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.

**// Global Constants**

**// Number of computer labs**

**const int NUMLABS = 8;**

**// Number of computers in each lab**

**const int LABSIZES[NUMLABS] = {19, 15, 24, 33, 61, 17, 55, 37};**

Note: you will need to research creating a jagged array. Array #1 holds the pointers to each member array, call it Array #2, which holds the lab user IDs.

Next establish an array to hold the reference (pointers) for each of the computer stations. This array, an array of pointers, is used in the dynamic allocation for each of the computer stations whose size is determined by the LABSIZES array. The next step would be to dynamically allocate member arrays pointed to by our reference array. Again, each of these holds the information for the respective computer stations. Thus, these are arrays that function as two dimensional. This is a key concept in our project’s implementation that must be followed.

The structure is shown in the figure below. This structure is sometimes also called a ragged array since the columns are of unequal length. Ragged or jagged, this array is an array of arrays. The member arrays can be of different sizes. If you visualize it as output, it produces rows of jagged edges. For our solution we created the member arrays as dynamic arrays.

|  |
| --- |
|  |
|  |
|  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

When we allocate these arrays now, we will use the appropriate pointer from the 8 pointers in the labs array, and thereafter we will store the values into the dynamically allocated memory (the second array), which is also known as the member array.

So, for example, looking at the ragged array above, let’s examine how that works for the first array. The first member array has 6 slots, and so the code should be labs[0] = new int[LABSIZES[0]]. This says the first pointer in the labs array, the array of pointers, is now pointing to a new dynamic array allocated for 6 slots (that is what value would be found at LABSIZES[0]). By virtue of using the NUMLABS value and the values in the LABSIZES array, you can do this all with nested loops.

Looking again at the example, the first array (left) works as the row, and the array with 6 slots (right) works as the columns. The row array are the pointers, and the second array holds the values. You can access this by using the combined [row] [col] processing.

*Menu design*

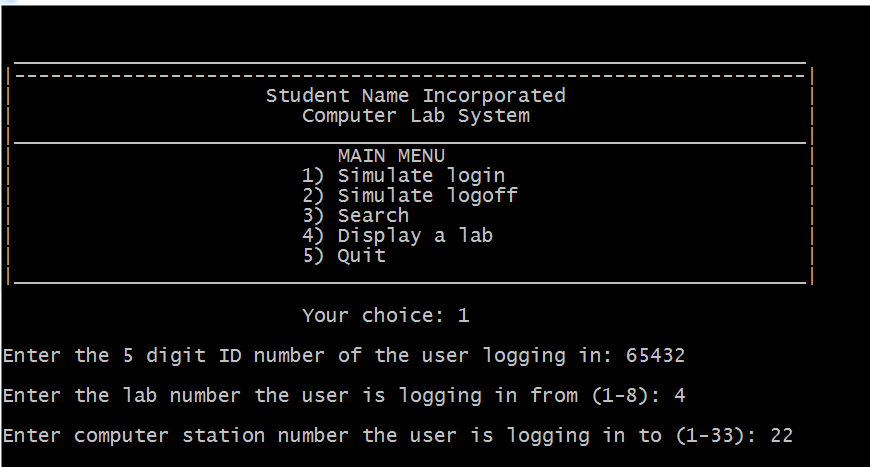
Create a repeating menu that allows the administrator to simulate the transmission of information by manually typing in the login or logoff data. Whenever someone logs in or logs out, the array should be updated. Along with the options to log in and log out, include a display option that offers the user the opportunity to print any one of the four labs on the screen. Also, include a search option so that the administrator can type in a user ID and the system will output what lab and station number that user is logged into, or “that user is not logged on” if the user ID is not logged into any computer station.

#### The user interface design

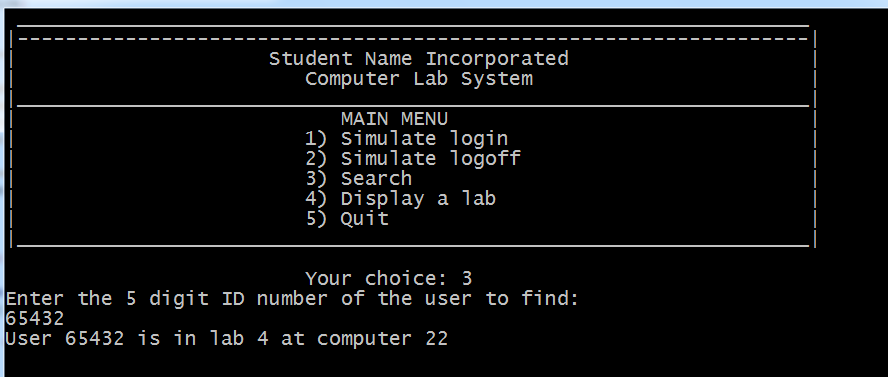
The user interface must be built as described. Again, any and all entries should be validated. Also, any login actions that find the user seeking an already filled computer should be rejected, and an appropriate message should be displayed such as “this computer station is already in use”.

Again, make use of manipulators such as setw for the output and match the output.

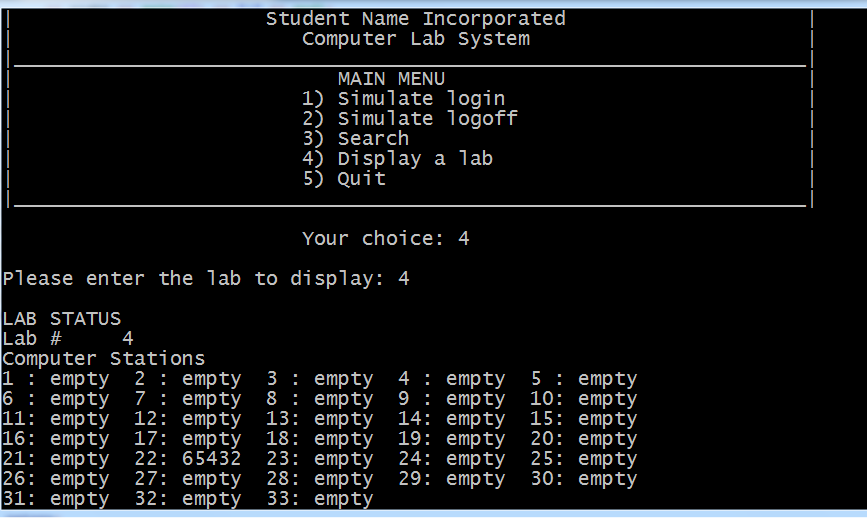
Here is an example of the menu, and the process of login:



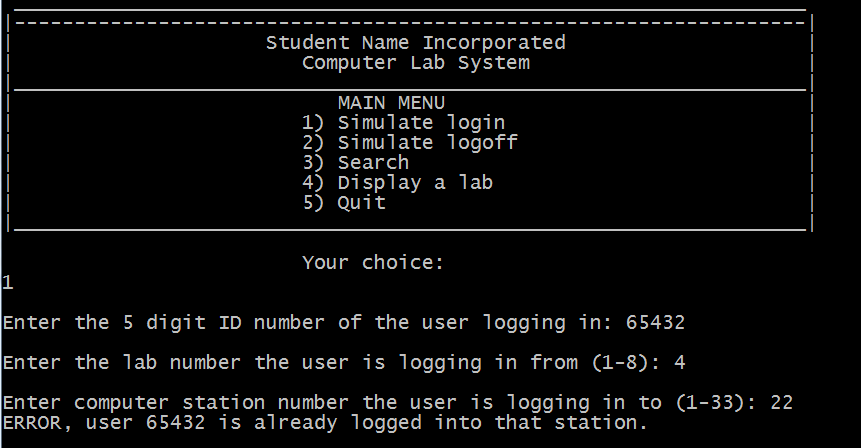
Here is an example of the menu, and a search request:



Here is an example of a display request:



And finally, an example of trying to add to a previously filled slot:



Note that any user ID can be reused. In other words, we can add 65432 to as many free slots as we wish to.

Normal validation is expected for all entries, and the user ID entries are limited to numbers only, and should always be 5 in length. Properly decompose your program into functions. NO Separate Compilation Files are required at this time ☺

Make use of manipulators for the output. Replace the Student Name heading with your own name (first, last, both…up to you).

Submission Instructions – for programming solutions

On Springboard, 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.*