**3460:209 PROGRAMMING PROJECT: CS1 HOSPITAL, Bed Assignments**



**T**he CS1 hospital receives a variety of new patients each day. At various times of the day patients are held in a temporary room until there is a determination made where the patient should be placed for their visit. The bed assignments are dynamically added throughout the day. There also are changes and updates from a file that have to be made. These rearrangements are reflected and placed on a grid that contains all of the current day’s room assignments.

You have been hired to program the daily floor updates for the hospital using the input provided from the head nurse and a file that contains the pre-existing grid of the floors. There are 5 floors with 8 rooms on each floor. The pre-existing grid is actually a two-dimensional array that contains the following numeric codes:

1. Vacant
2. Check out
3. Occupied
4. Nurse Station
5. Transfer
6. Utility room

These indicate the status for each room and this status must be considered as updates are made to the grid. Initial updates (checkouts and transfers) will have to be made to the floors (grid) first, and then the new patients are inserted into the vacant rooms as an ad-hoc request made by the user at the nurse’s station.

Here are the basic requirements:

* A file is setup with the current grid of rooms for your program to load.
* An occupied room, the utility room, and the Nurse Station rooms on the grid are not available, and should be protected from any updates.
* When a room is vacant it can be filled with a transfer or with a new patient.
* The check outs are patients checking out and will be done first, and these check outs should be updated to be vacancies.
* Transfers are to be handled next. Transfers are to be moved to the most recent available room, and their existing room from which they transferred is made available by switching it to a status of vacant. You may have to transfer patients to a different floor.
* After the transfers and check outs have been completed, new patients may begin to move in.
* The program will display the grid of rooms two times: 1) upon reading the file and; 2) after the transfers and check outs but before the user input.
* The final processing is to open the program up for input from the nurse station:
  + You will allow the nurse station input to continue until they choose to end the program or there are no more vacancies. You can choose to let the user quit any way that you wish to accomplish that.
  + When accepting the user input, you are required to validate the input and also you will need to display the total number of rooms that are currently available. You must limit the user’s input to not be greater than that value.
  + Each time you accept the user’s input, you should process the updates and redisplay the grid of rooms.

Summary:

* Open the file for reading.
* Build and display the initial grid of rooms from the input file.
* Remember that room 1, floor 1, is (0,0) on the grid.
* Perform the checkout updates.
* Perform the transfer updates.
  + Move the transfers (***important****: see instructions* *below*\*).
    - Calculate the number of vacancies.
    - Move the transfers using the pointer to the vacant room.
    - Display the current grid of rooms.
* Rebuild and redisplay the grid of rooms.
* Repeatedly ask the user for any new patient input (display the threshold and limit the number) until they quit or there are no more vacancies.
  + Open and get the input from the nurse’s station.
  + Insert the new patients requested by the user into the grid of rooms (***important****: see instructions* *below*\*) by using the pointer to the vacant room.
  + Rebuild and redisplay the grid of rooms.

Printing the grid of rooms

* You read the floor grid file into a 2-dimensional array of chars. The values are initially integers for the sole purpose of file reading and should be converted to char. This is how they will appear when you display it on the screen.   
  + Transfers are represented by a ‘T’.
  + The Check outs are represented by a ‘C’.
  + Nurse work stations are represented by a ‘W’.
  + Occupied rooms are represented by an ‘O’.
  + A vacant room is represented by a ‘V’.
  + The utility rooms are represented by an ‘X’.
* There are different ways to handle this so as long as it makes sense, I won’t interfere. If you want advice, see me during office hours or ask in class.

Code:

* Use a two-dimensional array of chars to represent the room grid.
* Make sure that your input from the nurse’s station is valid (it is in the range of vacancies and it must be numeric).
* At a minimum you are **required to create the following functions**:
  + printGrid
    - … Display the current grid of rooms with current data on who is in what position.
  + readGrid
    - ... Reads in the initial grid from Beds.txt file.
  + checkOuts
    - … Move the check outs.
  + transfers
    - ... Perform the transfers and build the dynamic array of pointers. Also, this function also will compute the available number of (vacant) rooms.
  + newPatient
    - ... Perform the new patient adds passing the new patient input from the nurse station and passing the dynamic array of vacancies count for updating.
  + newPatientInput
    - ... function for the input from the nurse's station.
* This is a complicated project, but to help us along all of these function prototypes are all included in the template for this project, called Final Project template. It is a text file that you probably want to use as a starting point for your program. Notice that the functions should be in the main program, and not in separate compilation files (hpp).You may add more includes, namespace, functions, etc…, as you see fit, however, **do not rename or remove** the functions that were provided.
* Make sure you use call by value and call by reference properly.
* Make sure you close the input file when you are finished reading it.
* Make sure you delete the dynamic array when the program terminates.
* Use a header comment with your name and the description of the program.
* Use pre and post condition comments for each function.
* Use prototypes for the functions (pre & post comments go with them).
* Include all your functions at the end of the program.
* Use appropriate code comments.
* Use appropriate variable names.
* Use constants where appropriate.
* Use good coding practices (i.e., spacing, indentation, etc.). Refer to the Programming Rubric for details located on Springboard: Table of Contents -> main() -> { Course Orientation ->RubricProgrammingforAssignments

Transfers

For moving transfers, your function is required to create a char array of pointers. This array will hold the addresses of the vacant positions. You will allocate an array the size of the number of vacancies and load the dynamic array with the pointers to all of the current vacancies. Build the array using the last vacancy first (we will process in a LIFO sequence later). The use of a dynamic array of pointers is required, and creates a scenario whereby pointers are held in memory and have a pointer pointing to them! Finally, in this same function, you will move all of the transfers to the available rooms (and mark them as OCCUPIED) using the dynamic array of pointers. You must do this by performing a swap of the transfer addresses with the vacant ones. This is important, because your program will not perform the proper order later during the new patient processing. Start from the back of the array of pointers and continue to the front for these operations until all transfers are done. DO NOT REPROCESS the array of pointers.

Nurse Station Updates

For new patients, your function is required to use the char array of pointers. In case you forgot, this array was built by the transfer function and holds the addresses of the vacant positions. You will ask the user for input, validate the input by making sure it is numeric, and by ensuring it is within the range of the current vacancies. Once validated then you next will call the function for new patient adds. These two functions run iteratively so a requirement is to allow the user to quit. Also, this process of adding patients is done in a LIFO sequence by using the dynamic array of pointers in conjunction with the two-dimensional grid. It’s interesting now to notice the order (as illustrated below). The last in was the last transfer, the next to the last in was the next to the last transfer, etc… This means your program, if working correctly in our design, will use the transferred rooms first, and that is because we ordered them intentionally as last updated first used. You will move the requested number of new patients to the available rooms (and mark them as OCCUPIED) using the dynamic array of pointers, and not the grid. You must start with the last available room and work backwards through the array of pointers. Each time you change a room to be OCCUPIED, set the pointer to be null (use nullptr). When there are no more rooms available (i.e., no pointers left), the program should display the message as seen in the example and terminate and return a code of 4. Otherwise, the program should display “\*\*\* More rooms are available today \*\*\* “ and terminate with return zero.

# Your output should look like this:

# *Note the behavior – the positions taken first correlate to the last activities (the 5 transfers), and then takes the order of the original activity (which was the build).*

# 

RUBRIC



Submission Instructions – for projects

On Springboard, go to the matching Assignments for the **Project #**, where # is the appropriate number of the project that is assigned (eg., Project 1), and submit the program (cpp) to Springboard and upload. Make sure it is the program only. There are ***no separate compilation files*** **required** for this project so include your functions in the main program. You may use the name main.cpp, or any other suitable name of your choice.

**Projects will not be graded after 11:59 p.m. on the due date.**

*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.*

*Original version created by Will Crissey, Jr. February 21, 2014.*