Assignment 3-1-2: C++ Pointers to Functions and Standard Library Functions

Assignment Instructions:

- You must use the virtual environment that you set up in Exercise 1-1-3 for this assignment.
- Each program must be in a separate .cpp file. If you aren't sure how to create a new .cpp file, refer to the Create a New File section in the setup instructions document. Make sure each file is named as instructed in the question.
- Each program must include appropriate input and output messages.
- Be sure to compile and test each program to be certain it works as expected. If you aren't sure how to compile and run a C++ program, refer to the Build and Execute Program section of the setup instructions document.

Important notes:

- At the top of your .cpp file, please include a comment with your **full name**. If your section uses **Lightweight Teams**, add the names of the teammates whom you worked with to the same comment.
- Add your own **individual comment** for each function / major portion of code that you add, briefly explaining what that part does.
- If you are asked to submit screenshots and your submitted screenshots do not match with your program's actual behavior, we will consider that to be a **violation of academic integrity** and pursue it accordingly.
- Make sure to **organize and format** your code in a consistent way.
- If you refer to any online resource to understand a concept, see examples of the use of a particular syntax, etc., add a comment **citing** that resource (i.e., specify website name and link).
- You must only submit **.cpp** files. If you have multiple .cpp files, upload them individually and **not** as a zip / compressed file.
- No screenshot(s) will mean no grade for this assignment.

Assignment Objectives:

- The purpose of this activity is to become familiar with or refresh concepts related to C++ pointers to functions and the C Standard Library

Assignment Resources:

- http://www.cplusplus.com/reference/cstdlib/qsort/ (*Provides an overview of the library Quick sort function and an example of its use. Note: the example uses an array of a basic data type, whereas this assignment uses an array of a* ***user defined*** *data type; make sure to apply your knowledge about how user defined structs are accessed*)
- http://www.cplusplus.com/doc/tutorial/operators/
- Review the **void pointers** section in the following tutorial: http://www.cplusplus.com/doc/tutorial/pointers/
- http://www.cplusplus.com/faq/sequences/arrays/sizeof-array/
- https://www.tutorialspoint.com/cplusplus/cpp_sizeof_operator.htm
- http://www.cplusplus.com/reference/cstring/strncmp/
- Here is a C++ visualizer that you may find useful as you write programs in C++. It allows you to quickly step through your program and visualize the output at each step.

Assignment Tasks:

1. **(1 point) Setup**
    a. To download the *skeleton* code for the assignment, copy-paste the following command ***into the terminal window*** to pull the project repository from GitLab:

    git clone https://cci-git.charlotte.edu/jbahamon/ITSC_3146_A_3_2_2.git

    b. Change directory into the newly created directory (folder) named ITSC_3146_A_3_2_2

    c. Compile and run the *QuickSort_Skeleton.cpp* program, your output should be similar to the following:

    d. Examine the code. Make sure to find the *// TODO* comments.

    e. Add a prompt to ask the user to input their *last* name (no spaces ; please use underscores if needed), age and a balance of their choice. Create a new entry in the patient_list array with this information [**Important note**: you can statically modify the code to have the patient_list array hold an extra entry ; no need to do any dynamic memory allocation to accommodate the new entry].

    f. **Rename** the program file to *QuickSort_99999.cpp*, where *99999* should be replaced with your Student ID (800#). For the rest of this assignment, use that renamed cpp file.

Take a screenshot of a sample output and upload the picture as part of your assignment submission.

2. (5 points) Display Function

a. Implement a function named `displayPatientList` that prints the contents of the *patient_list* array to the console.
b. Add code to call this function. Note that there are multiple places where this function needs to be called. Look for the *// TODO* comments to find the correct locations.
c. Compile and test your program.

Take a screenshot of a sample output and upload the picture as part of your assignment submission.

3. (10 points) Sorting the array by *Age*

a. Implement the code to sort the contents of the *patient_list* array based on the value stored in the *age* field. To do this you will need to implement code that relies on the *qsort* function from the C Standard library (see http://www.cplusplus.com/reference/cstdlib/qsort/). As shown in the reference, this code requires two parts:
   i. A function named `comparePatientsByAge` that compares two *patient* elements, based on the value stored in the *age* field.
   ii. A call to the *qsort* function, which includes the array to be sorted, the number of elements in the array, the size of each array element and the function used to compare the array elements.
b. Add code to call the *qsort* function, using the **age comparison** function that you implemented. This code should be placed just under the appropriate *// TODO* comment in *main()*.
c. Compile and test your program.

Take a screenshot of a sample output and upload the picture as part of your assignment submission.

4. (10 points) Sorting the array by *Balance Due*

a. Implement the code to sort the contents of the *patient_list* array based on the value stored in the *balance* field. To do this you will need to implement code that relies on the *qsort* function from the C Standard library (see http://www.cplusplus.com/reference/cstdlib/qsort/). As shown in the reference, this code requires two parts:
   i. A function named `comparePatientsByBalance` that compares two *patient* elements, based on the value stored in the *balance* field.
   ii. A call to the *qsort* function, which includes the array to be sorted, the number of elements in the array, the size of each array element and the function used to compare the array elements.

b. Add code to call the *qsort* function, using the **balance comparison** function that you implemented. This code should be placed just under the appropriate *// TODO* comment in *main()*.

c. Compile and test your program.

Take a screenshot of a sample output and upload the picture as part of your assignment submission.

5. (10 points) Sorting the array by Patient *Name*

a. Implement the code to sort the contents of the *patient_list* array based on the value stored in the *name* field. To do this you will need to implement code that relies on the *qsort* function from the C Standard library (see http://www.cplusplus.com/reference/cstdlib/qsort/). As shown in the reference, this code requires two parts:
   i. A function named `comparePatientsByName` that compares two *patient* elements, based on the value stored in the *name* field. Because name data is stored in an array of characters, you cannot use the relational operators (<, >, <=, …) to do the comparison, instead you should **use a function from the C Standard Library** to determine the

order of the names, see (http://www.cplusplus.com/reference/cstring/strncmp/).

ii. A call to the *qsort* function, which includes the array to be sorted, the number of elements in the array, the size of each array element and the function used to compare the array elements.

b. Add code to call the *qsort* function, using the **name comparison** function that you implemented. This code should be placed just under the appropriate *// TODO* comment in *main()*.

c. Compile and test your program.

Now, your output should include the original patient list and three sorted patient lists.

Take a screenshot of a sample output and upload the picture as part of your assignment submission.

Caution: Before you submit, make sure that you have followed all the instructions under Assignment Tasks and Important notes and that you have taken screenshots as indicated in the assignment.

Assignment Submission Items:

The files that need to be submitted for this assignment are the following:

- Renamed QuickSort_Skeleton.cpp
- The necessary output screenshots for the cpp file.

Note: No screenshot(s) will mean no grade for this assignment.