CST8234 – C Programming F17 (Assign. 1)

Programming Exercise

In this assignment, we will demonstrate what we have learned with respect to

- Using structs and arrays
- Sorting
- Code reuse

Statement of the problem

We'll be building a simulation of a student enrolment program.

To start, you will use a list of TWELVE students.

You will represent each student by their student ID that is a randomly assigned 5-digit number, and a nested name structure that contains their first and last name.   You can arbitrarily pick a bunch of unique names for your students.

You will create THREE courses.

You will represent each course by a name (of your choosing, e.g., "Perl Programming"); a description (of your choosing, e.g., "Learn the fundamentals of programming scripts in Perl, and be able to..." ; a course code that will consist of the string "CS" and a random three-digit course number (e.g., "CS193"); and the maximum number of students that can be registered, which will be randomly set, for each course, to a number between 4-8.

Additionally, each course will maintain the list of the students currently registered for it, and a list for those that are on the waiting list. There is no limit to the number of students who may on the waiting list for a course.

You will then go through the list of students, TWICE, and each pass you will randomly try to register each student in one of the three courses. This will result in the student being either registered (if there's room) or put on the waiting list. A student cannot be in the same course more than once.

You will then print out the courses' information, including registration and waiting lists. List of courses will be sorted by the courses' codes. You will show the number of students on each list and the capacity.   Lists of students will be sorted by the students' ID's.

All students will always be identified by their 5-digit student ID following by their name in the "last name, first name" format (e.g., "78232 - Trump, Donald")

All courses will always be identified by the course code followed by the course name, (e.g., "CS123 - Perl Programming").

In the descriptions above, “random” or “randomly” means that the values will be different with each execution of the program.

You will not ever have more than one copy of a student record or course record in memory. I.e., all lists will contain pointers to the source structures.

Optional bonus marks: Finally, you will prompt for a student ID, and print out what courses that student is registered for, or on the waiting list for, where each course is specified by its course code and name. Entering a zero for a student ID will exit the program.

Follow the formatting in the example below, EXACTLY. Pay attention to the nuances. Here is a sample execution (example shows optional bonus querying).

```
$ ./lab4.exe
CS123 – Perl Programming
Learn the fundamentals of programming scripts and be able to install Perl modules.
Registered Students (5/5):
* 00449 - Putin, Vladimir
* 11111 - Merkel, Angela
* 78232 - Donald, Trump
* 82264 - Kim, Jong-un
* 99111 - Zuma, Jacob
Waiting List (2):
* 05532 - Trudeau, Justin
* 60003 - May, Theresa
```

(other courses not shown for brevity)

```
Enter a student ID: 78232
Trump, Donald is registered for:
* CS123 – Perl Programming
* CS155 – Python for Beginners
On waiting list for:
* None

Enter a student ID: 1111
No student found with ID 1111

Enter a student ID: 11111
Merkel, Angela is registered for:
* CS123 – Perl Programming
On waiting list for:
* CS155 – Python for Beginners

Enter a student ID: 60003
May, Theresa is registered for:
* None
On waiting list for:
* CS123 – Perl Programming
* CS155 – Python for Beginners
```

Requirements

1. Create a folder called algonquinUserID1_algonquinUserID2_A1 (e.g., "mynam00123_yourna45678_A1") that represents the two members of your team. Do all of your work in this folder, and when complete, submit the zipped folder as per the "Lab Instructions" posted on Blackboard.
2. Write a program that will implement ALL of the requirements, explicit and implicit, listed in the "Statement of the problem" above.
3. You must distribute your functions in a meaningful manner across **multiple .c** files.
    a. The .c files should contain functions that represent sensible groupings of functionality
    b. You must define .h files as appropriate
4. Each function must have a header comments that explain what it does, and describe/explain its inputs (if any) and return value (if any)
5. You may NOT use any global variables that are defined in a different file. I.e., you have to pass arrays as function arguments!
6. You must use a 'makefile', with the **`CC_FLAGS`** set to "**`-g -ansi -pedantic -Wall -w`**"), and **`OUT_EXE`** set to "**`assignment1`**"

Marking

This assignment is out of 40

- 15 for coding correctness (i.e., correct results)
- 10 for appropriate and efficient logic (i.e., no spaghetti code!)
- 5 for sensible distribution of well-contained functions between files
- 10 for clear comments and coding convention (not necessarily K&R, but it should be clean and consistent)
- 5 marks (optional bonus) for your implementation of the querying

You can also lose marks for incorrect submission (e.g., including unnecessary files in the zipped folder), compiler warnings, typographic errors (including in comments).

You will DEFINITELY lose marks for duplicating code between your functions.

You will lose marks for overly long functions. Each function should do a specific task. Other functions should call those specific tasks.

You will lose marks for using more than the minimum number of global variables. Most variables should be local to a function, and passed between functions as arguments.

Submission

When you are done, ONE person on the team will submit your program to Blackboard, as per the instructions for each Lab.

You ***must*** include a makefile, as part of your submission! When grading your reports I will unpack your zip file and type '`make`'… and if I don't end up with a 'assignment1.exe' to run, ***I will not grade your assignment***.