EE 441 HOMEWORK #2

Section Assigner

Due: December 18, 2021, 23:59

Questions: Please use the forum "*Homework #2*" for your questions.

Introduction

As a teaching assistant, your first task of the semester is to divide students into three laboratory sections: Wednesday, Thursday, and Friday each of which has limited capacity. Since students have different course schedules, assigning them randomly is not an option. Therefore, you have asked them to submit their preferences based on which you can assign them to the eligible sections. However, there are an exponential number of ways to assign students and it is not easy to know which one is the best. To choose the best one, you have decided to code some of them and make your computer do all the work to finish the task as fast as possible. Now, you are going to implement your first grouping algorithm by using class, stack, and queue data structures.

Input - Output in C++

Your program will read section capacities and student preferences from a text file one by one. When the process is completed, the program will write into an output file which sections students have been assigned to. To be able to read from and write to a text file, you should include <fstream> library. Then you can use 'ifstream' and 'ofstream' classes which are called as stream classes.

An example for writing "example" to an output txt file named 'out.txt' is given below.

```
ofstream outputtxt;
outputtxt.open("out.txt");
outputtxt << "example";
outputtxt.close();
```

An example code segment for reading from an input txt file named 'in.txt' is given below. It reads and outputs lines one by one for every iteration in while loop.

```
ifstream inputtxt;
inputtxt.open("in.txt");
char output[100];
if (inputtxt.is_open()) {
        while (!inputtxt.eof()) {
                inputtxt >> output;
                cout << output;
        };
};
inputtxt.close();
```

Input File Format

- The name of the input file will be 'preferences.txt'.
- The first 3 lines must be reserved for section capacity. The order of the sections is random.
- The remaining lines should start with student ID and continue with section preferences, all separated by a comma.
- Students will have at least 1 preference.
- The section written right after the student ID has the highest priority for that student and the last section has the lowest one.
- Capital letters W, T, F will be used for section days Wednesday, Thursday, and Friday, respectively.

Example:

```
F,3
T,3
W,2
101,W
102,W,T
103,W,T,F
104,W,F,T
105,T
106,F,W
107,W,T,F
108,W
109,T,F
110,T,W,F
```

Output File Format

- The name of the output file will be 'results.txt'.
- Students assigned to a section will be shown by their student IDs under the name of that section day.
- Students who have not been assigned to any section will be indicated by their student IDs under 'Unassigned'. If all students have been assigned, just write '-'.

Example:

Wednesday:
108
101

Thursday:
103
102
105

Friday:
107
104
106

Unassigned:
109
110

Required Data Structures

Note: You must implement these data structures from scratch by yourselves. You are not allowed to use STL etc.

A) "Student" Class:

- You are going to implement a "Student" class which will be used to store & access related information about students such as student ID and preferences.
- When a new student is received i.e., a line is read from the input file, a new "Student" object must be created.
- It must be able to return the number of eligible sections for a student by taking into consideration the fullness of the sections.
- It must be able to return the most preferred section of a student that is not full.
- Any other feature you think necessary to use in your design...

B) "Section" Stack:

- You are going to implement a "Section" stack to store "Student" objects.
- You will use one "Section" stack for each section i.e., three stacks in total.
- When a student is assigned to a section, their "Student" object will be pushed into the corresponding "Section" stack.
- The maximum number of "Student" objects that a "Section" stack can hold is given in the input file as section capacity.
- It must be able to indicate if it is full or not.
- Any other feature you think necessary to use in your design...

C) "Waiting" Queue:

- You are going to implement a "Waiting" queue to store "Student" objects.
- You will use only one "Waiting" queue.
- When a student is not assigned to any section, s/he will be put in the "Waiting" queue and will wait to be placed in later steps.
- Any other feature you think necessary to use in your design...

Section Assigner Algorithm

Below a simple heuristic algorithm, although it is not finding the optimum solution, is given. First of all, "Section" stacks must be initialized with their capacities indicated in the input file.

Read each student from the input file one by one. If a student can only be placed into one section, then they must be assigned to it as long as the related section capacity permits. Otherwise, place them into the waiting queue. After a new student is assigned to one of the sections, students waiting in the queue must be considered repeatedly until no more admission of students from the queue is possible. If a student in the waiting queue has only one eligible section, assign them to it. Otherwise, put them at the end of the queue. BE CAREFUL! you can use only one queue & before continuing to read a new student, the position of the students with respect to each other in the queue MUST REMAIN THE SAME AS BEFORE! When students in the queue are all checked, then proceed to read a new student from the input file.

When the students in the input file are all read, students with more than two or no eligible sections will be waiting in the queue. These students will be considered repeatedly until no more admission is possible. Students will be placed based on the priority of their preferences.

When the waiting queue is empty or has only students who are ineligible for any section, the program stops after it prepares an output file by popping students from stacks and queue as described above.

Note: You can use the above input-output examples to check the correctness of your code.

Regulations

1) Use **Code::Blocks IDE** and choose **GNU GCC Compiler** while creating your project.
2) Name your Project as "eXXXXXXX_HW2", where X's are your **7-digit student ID number**. Send the whole project folder compressed in a rar or zip file. Name your submission as "eXXXXXXX_ee441_hw2.rar". You will **not** get full credit if you fail to submit your project folder as required.
3) The homework must be written in **C++** (**not** in C or any other language).
4) Your C++ program should follow object-oriented principles, including proper class and method usage, and should be correctly structured, including private and public components. Your work will be graded on its correctness, efficiency, clarity, and readability as a whole.
5) You should insert comments in your source code at appropriate places without including any unnecessary details. **Comments will be graded**.
6) You should **not** call any external programs in your code.
7) **Check** what you upload. Do not send corrupted, wrong files or unnecessary files.
8) Programs giving compilation errors will **not** be evaluated!
9) The homework is to be prepared **individually**. Group work is **not** allowed. Your code will be checked for cheating.
10) The design should be your original work. However, if you partially make use of a code from the Web, give proper **reference** to the related website in your comments. Uncited use is unacceptable.
11) **METU honor code is essential.** Do not share your code. Any kind of involvement in cheating will result in a **zero grade**, for both providers and receivers.

12) **Late submissions** are welcome, but are penalized according to the following policy:
    - 1-day late submission: HW will be evaluated out of 70.
    - 2-days late submission: HW will be evaluated out of 50.
    - 3-days late submission: HW will be evaluated out of 30.
    - Later submissions: HW will **NOT** be evaluated.