CSCI-1200 Data Structures — Spring 2022
Homework 2 — Schedule Classes

In this assignment you will parse and compute statistics from course schedules. Please read the entire handout before starting to code the assignment.

Input Format

You will parse an input file with information for zero or more courses. An example line might look like:

```
64859 CSCI 4030-01 RANDOMIZED_ALGORITHMS MR 10:00AM 11:50AM DARRIN_239
```

Each line contains several fields, separated by spaces. You can assume that values have no white spaces, i.e. a course name like "RANDOMIZED ALGORITHMS" is illegal. There are 8 fields in each line: a CRN (unique ID, e.g. 64859), a department code (e.g. CSCI), a course code (e.g. 4030-01), the course title (e.g. RANDOMIZED_ALGORITHMS), the days this course occurs on (e.g. MR), the time the course starts (e.g. 10:00 AM), the time the course ends (e.g. 11:50AM), and a room (e.g. DARRIN_239). Each entry will contain all 8 fields.

Each letter in the days represents one day, with M being Monday, T being Tuesday, W being Wednesday, R being Thursday, and F being Friday. Multiple days can appear on one line in the input, as shown in the example above. Time is always two digits for the hour, a colon, two digits for the minute, and then either "AM" or "PM".

There may be multiple lines for one course, so you will want to study the provided input and output carefully before beginning. We *strongly recommend* that you use the `iostream >>` operator to parse this file and not use `getline` or other C-style file I/O commands.

File I/O and Command Line Arguments

Your program will run with several command-line arguments, the first being the name of the input file described above and the second being the name of the output file where you will write either nicely formatted tables or an interesting statistic (described below). Example input and output files posted on the course website.

Here are examples of four valid command lines to your program that demonstrate the different usages. These are described in more detail below.

```
scheduling.exe simple2.txt out_simple2_room.txt room
scheduling.exe simple2.txt out_simple2_room.txt room ACADMY_AUD
scheduling.exe simple2.txt out_simple2_dept_ECSE.txt dept ECSE
scheduling.exe simple2.txt out_simple2_custom.txt custom
```

We have provided you with several schedule datasets, primarily based off of the Fall 2018 registration data from SIS. We have done some processing to make your task simpler.

Room-based Output

If the third argument is `room` then you will be outputting statistics about courses in each room. There is an optional fourth argument, if present you should only print information about courses in the room with the name matching the fourth argument. In the above example all the courses are in "ACADMY_AUD" so the output will look the same with or without the fourth argument. If the room cannot be found, your output should look like the examples using "empty.txt" . Your spacing must match exactly - there are two spaces in between each column and columns must be wide enough to fit the data.

When outputting more than one room, the tables should be printed alphabetically by room name. Within each table, courses should first be sorted by day of the week (Monday should come before Tuesday, Tuesday

before Wednesday, etc.). Courses on the same day should be sorted by what hour the course starts at, with the earlier courses coming first. If two or more courses happen in the same time and place, you should sort those courses by course code, i.e. "1100-01" would come before "1100-02". If the course numbers also match, you should sort by department name, i.e. "CHEM" would come before "CSCI".

Department-based Output

If the third argument is `dept` then a fourth argument is mandatory. The fourth argument specifies a department code (keep in mind this can be anything as long as it does not have whitespace). If no courses for the department code are found, the output should look the same as for the "empty.txt" case.

Similar to the room-based output, the columns in the table that you output should be wide enough to fit the data and should have two spaces in between each column. Sorting should be done by course code first, then by earliest course day (e.g. Monday should come before Tuesday), and then by **latest** starting hour if there is a tie for course code and day.

Creative Statistic

The last part of the assignment is a chance for you to be creative. Brainstorm a new interesting statistic that can be calculated and presented from this data. (There is no sample output for this, because there is no autograding for this part beyond whether or not you did it. The TA will manually grade the output for quality.

Examples include tabulating when each department "opens" (has its first class), perhaps comparing across different days, or seeing if some rooms only get heavy usage on certain days. Your new statistic should only be printed if the third argument is `custom`. **NOTE:** You MUST make a creative statistic to get full credit, even if you are not interested trying for the extra credit.

Extra credit will be awarded to particularly interesting statistics that require clever programming. The most important task for this part of the assignment is to write a concise description (< 100 words) of your new statistic. Put this description in your *plaintext* `README.txt` file along with any other notes for the grader. Be sure to tell the grader which dataset best demonstrates your new statistic, and include a sample of the output. Feel free to create your own dataset and include it with your submission.

Useful Code

To control the formatting of your tables, you'll want to read up on the various I/O manipulators: `std::setw(int)`, `std::setprecision(int)`, `std::fixed`, `std::left`, etc. You'll need to `#include <iomanip>` to access these tools. And don't forget about the `sort` function that can be used to order the contents of a `vector`.

If you add `-std=c++11` to your compile line, you can use C++11 features. Submitty will use this flag, if you do not want any C++11 features this will not affect you. However, students on Homework 2 often find `std::stoi` to be useful, and this requires C++11.

We will not formally cover the concept of "include guards" until Tuesday's lecture. If you are writing more than one class, you may run into strange compiler errors when you compile everything. This may be due to a problem with including your class files, which can be solved as follows: for a header file called `myclass.h` add these two lines at the very top of the header file:

```
#ifndef __MYCLASS_H
#define __MYCLASS_H
```

and at the very bottom of your .h file, add this line:

```
#endif
```

If you want to make a vector of vectors of a type (let's use `int` in this example) called vectorname, you can do so by writing `std::vector<std::vector<int> > vectorname` and then indexing as usual. `vectorname[0]`

will return a `vector<int>` in this example. The most important detail is to remember to put a space between the two closing angle brackets. Some compilers will interpret `>>` as the input operator, even though it's obvious to a human reader that this was not the intent.

Program Requirements & Submission Details

Your program should involve the definition of *at least one class* that has its own .h and .cpp files, named appropriately. We have provided a series of series of datasets to aid in your program development and debugging. The majority of points for this homework assignment will be awarded for programs that work correctly for the simple examples.

Do all of your work in a folder named `hw2` inside of your Data Structures homeworks directory. Use good coding style when you design and implement your program. Be sure to make up new test cases and don't forget to comment your code! Please use the provided template `README.txt` file for any notes you want the grader to read. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss the problem or error messages, etc. with anyone, please list their names in your `README.txt` file. If by Wednesday 11:59:59PM any version of your HW2 that you've submitted has earned 5 points between Test Cases 3, 4, 5, 6, and 7, you will earn you a 1 day extension for HW2.