CSCI-1200 Data Structures — Fall 2024
Homework 7 — Registration Maps

Ben Bitdiddle has invited you to join his new Rensselaer Center for Open Source (RCOS) team project. He as been inspired by the existing and massively popular YACS (Yet Another Course Scheduler) and QUACS (Questionably Accurate Course Scheduler) projects. His project is named BBEAPS (Ben Bitdiddle's Exceptional Academic Planner and Scheduler). He has expanded the functionality of the other projects – instead of picking courses for a single term, BBEAPS facilitates planning a full four year undergraduate degree!

In this assignment you will design and implement efficient data structures so that BBEAPS can maintain the registration data for the entire university! Departments will be able to schedule faculty to courses and have sufficient lead time for the university to hire new faculty or upgrade classrooms as needed – based on student demand for specific courses. He wants to adapt to the size of the incoming class of undergraduates, *which has varied from 1500-2000+ in the last five years!*

We will work with the actual semester, prerequisite, and degree requirements data originally scraped and freely shared online by the QUACS team: `https://github.com/quacs/quacs-data`. Note that data quality has a huge impact on the accuracy of this tool. **CAUTION: Output of this tool is not a substitute for reading the official published requirements of your own degree program and ensuring you have the necessary prerequisites for all courses in which you enroll.**

By using STL's binary search tree based containers (`map` and/or `set`) we can make this system quite efficient and elegant. *Please carefully read the entire assignment before beginning your implementation.*

Data Format

Here's a small piece of the course information file, `fall22.txt`, which is the actual schedule for RPI's Fall 2022 term:

```
...
55032 CSCI 1100 01 Computer_Science_I 32 4 [ M 1400-1520 T 1000-1150 R 1400-1520 R 1800-1950 ]
55350 CSCI 1100 02 Computer_Science_I 32 4 [ M 1400-1520 T 1000-1150 R 1400-1520 R 1800-1950 ]
55351 CSCI 1100 03 Computer_Science_I 32 4 [ M 1400-1520 T 1200-1350 R 1400-1520 R 1800-1950 ]
...
55037 CSCI 1200 01 Data_Structures 36 4 [ T 1400-1550 W 1000-1150 R 1800-1950 F 1400-1550 ]
57469 CSCI 1200 02 Data_Structures 36 4 [ T 1400-1550 W 1000-1150 R 1800-1950 F 1400-1550 ]
55038 CSCI 1200 03 Data_Structures 36 4 [ T 1400-1550 W 1200-1350 R 1800-1950 F 1400-1550 ]
...
```

Each line contains information for one section of a course. The information from left to right on that line: the "CRN", a 5 digit identifier for the course section. The CRN is unique for that term (no other course or section has the same id for that term). Note that the CRNs will be recycled and re-assigned to other courses in future terms. Next is the 4 letter department code. "CSCI" is the Computer Science department, "MATH" for the Mathematics department, etc. Then is the 4 digit course number within the department and the 2 digit section. The we have the title for the course. The actual title has been edited to replace each space character with the underscore to ease parsing. Note that the title of the course may change or may be abbreviated differently in different semesters or in the prerequisite file (described next). *The title is for human readability, the department code + course code is more consistent and reliable.* Then we have the maximum capacity / enrollment for that section and the number of credits students will received when they pass the course. Finally, the weekly meeting times for the course are listed between square brackets `[ ]` . Each meeting time consists of a letter: `M`=Monday, `T`=Tuesday, `W`=Wednesday, `R`=thuRsday, and `F`=Friday and the time range in military time.

Next, we have a portion of the prerequisites file, `conservative_prereqs.txt`:

```
...
CSCI 1200 Data_Structures has prereqs: [ CSCI 1100 Computer_Science_I ]
CSCI 2200 Foundations_Of_Computer_Sci has prereqs: [ CSCI 1200 Data_Structures and
          MATH 1010 Calculus_I ]
...
CSCI 4210 Operating_Systems has prereqs: [ CSCI 2300 Introduction_To_Algorithms and
          CSCI 2500 Computer_Organization and ECSE 2660 Cptr_Arch,_Nets_&_Os ]
...
MATH 1020 Calculus_II has prereqs: [ MATH 1010 Calculus_I ]
...
```

Each line contains the information for one course. If the course has no prerequisites (e.g., Computer Science I and Calculus I have no prerequisites) that course does not have its own row in the file, but it may appear in the file as a prerequisite for another course.

IMPORTANT NOTE: One limitation / source of data error within the conservative version of the prereq data is that the prerequisite information does not distinguish *and* vs. *or* in the prerequisites. For example CSCI 2200 requires "*both CSCI 1200 and MATH 1010*". Other courses listing two prerequisites might accept "*either course A OR course B*" of students who wish to register. And some courses have more complicated English descriptions – for example the actual prereqs for CSCI 4210 is "*CSCI 2300 and either CSCI 2500 or ECSE 2660*". Furthermore this conservative and simplified file does not represent that BIOL 1010 and BIOL 1015 are *co-requisites* (a student is expected to take both classes in the same semester). And it also ignores the complication of *cross-listed courses*, when a single course is listed with multiple course codes in multiple departments – a student can take either version to satisfy the prerequisites of future courses.

For this initial implementation, Ben has decided to make BBEAPS overly conservative in processing all prerequisites. Students must satisfy ALL listed prereqs in order to register for a course. Furthermore, Ben is ignoring the possibility of AP credit, transfer credit, and requesting and receiving permission of the instructor to *override the prerequisite* for a course (take the course WITHOUT receiving credit for the prerequisite course in a prior term) or to *override the maximum registration capacity* of a section. Also, Ben hopes by ignoring the existence of "Summer Arch" and "Arch Away" terms, that will just go away.

Finally, here is an example of the so-called *named degree requirements* for the CSCI major, stored in the `simplified_degree_requirements.txt` file:

```
...
CSCI [ CSCI 1100 , MATH 1010 , PHYS 1100 , CSCI 1200 , MATH 1020 , BIOL 1010 , BIOL 1015
       CSCI 2200 , CSCI 2500 , CSCI 2300 , CSCI 2600 , CSCI 4210 , CSCI 4430 ]
...
```

This list is significantly incomplete. The actual CSCI major requires four additional in-major course electives selected from a large list of topics courses in different concentrations. Furthermore, the additional science and math electives and HASS courses are not represented in this list. Also students must complete a minimum of 128 total units (averaging 16 credits per term) in order to graduate. For this homework we are also ignoring the fact that degree requirements can change from year to year – students are expected to complete the requirements that were in place the year they first enrolled as a student.

Input/Output & Basic Functionality

Your program will read from `std::cin` and write to `std::cout` and `std::cerr`, but we expect you will *redirect* the input (& output) to trick your program into reading from & writing to files – see the course webpage: "Helpful C++ Programming Information". Here's how you might call the program on the small example with errors and separate the standard output and standard error:

```
./bbeaps.out < example_with_errors.txt > output.txt 2> stderr.txt
```

Each action line in the input file begins with a keyword signaling which operation should be performed:

`load_courses Fall 2022 fall22.txt` This operation will load the course offerings from the `fall22.txt` file and make those courses available in the Fall 2022 term.

`load_prereqs conservative_prereqs.txt` This operation will load the prerequisite data from the `conservative_prereqs.txt` file.

`enroll Ben_Bitdiddle CSCI 16` In this operation, we add the student `Ben_Bitdiddle` and specify that he is following the `CSCI` degree requirements and will take a maximum of `16` credits per term.

`register Ben_Bitdiddle Fall 2022 55032` This operation will attempt to register Ben for a course in the `Fall 2022` semester. Specifically for the course with the CRN `55032`. Various checks should be performed with this information and appropriate error messages printed to `std::cerr` if there were problems. Some potential issues include: the CRN is invalid for the specified term, Ben is already planning to take that course in another semester, Ben has not added the necessary prerequisite courses to his schedule in the prior terms, Ben is attempting to add a section of the course that does not have an open seats, or a meeting time of this new course conflicts with the meeting time for a course he has already selected. In all of these cases (and some additional error situations), the operation will fail to add the course to his academic plan and an error will be sent to `std::cerr`. Samples of the expected error messages are shown in `stderr_example_with_errors.txt`.

`print_student_registration Ben_Bitdiddle` This operation will print Ben's schedule for each term, showing the courses he successfully registered for in chronological order, the credits for the term, and visualize his weekly course schedule of meeting times by day of the week in half hour increments hour. See the details for formatting this output in `output_small_example.txt`.

`print_course_registration Spring 2023 CSCI 1200` This operation prints the data for a specific course, so the instructor can see the names of all of the students, organized by course section. Note that the names are in chronological order of when the students added the course. See the details for formatting this output in `output_small_example.txt`.

Sample input and output files are available on the course website. Additional files with larger amounts of synthetic data will be added so you can perform light stress testing of your implementation. To receive full credit on the assignment, please follow these examples exactly. To see if your program is performing perfectly, you may use the Unix command line library program, `diff`, which takes two files as arguments and outputs the differences between them.

Performance & Order Notation

You should carefully consider the performance of each of the BBEAPS operations and choose data structures to achieve both correct output *and* efficient performance.

Let t be the number of terms (semesters) used by the input data file. Let c be the number of different courses offered in a single semester. And n be the maximum (or average) number of sections offered for a single course. Each course has a maximum (or average) of p prerequisite courses. The maximum (or average) number of meetings times for a single course is m. Let s be the number of students in the input file and e be the maximum (or average) number of students who are enrolled in a section of a course – this is also the capacity of the course when many or most of the sections are full or nearly full. We will assume that the typical student takes courses over 8 terms/semesters, most courses are worth 4 credits, and students need 128 credits (or more) to graduate. Some students may choose to take more than 16 credits per semester, but they don't take more than $\sim$22 credits in a semester.

What are the relative sizes of these variables for a large university? What are the current values of these variables at RPI? Which of these variables are going to have the most dramatic impact on the performance

of your program? You should aim to achieve sub-linear (log or constant time) expected running time for each operation with respect to these variables when possible. *Hint: That means you should use maps and/or sets!* In your `README.txt` file include the order notation for each operation in terms of the variables above. You will also write a few sentences to justify each of your answers.

You are not explicitly required to create any new classes when completing this assignment, but please do so if it will improve the program design, readability, and/or code organization. We expect you to use `const` and pass by reference/alias as appropriate throughout your assignment. We have provided a partial implementation of the main program to get you started with parsing and printing. You may use none, a little, or all of this, as you choose, but we strongly urge you to examine it carefully. *Yes, you are allowed to modified the provided code.*

Data Structure Diagram

As part of your electronic submission, you must include a neat diagram illustrating the overall data structures you selected. Follow the conventions from lecture for diagramming maps, sets, lists, vectors, pairs, and custom classes. You are encouraged to explicitly label the types in your diagram as well. Use this diagram to communicate design choices you made and how this data structure works. Consider diagramming the `small_example.txt` example, but you may simplify/omit some of the more tedious details.

The accepted file formats for this diagram are .pdf, .png, or .jpg. Name your file `diagram.pdf` or `diagram.png` or `diagram.jpg` (depending on the filetype). You may draw this diagram with pen/pencil & paper and scan or take a photo with a camera or you may use an electronic drawing/diagram software. You will be graded on neatness, clarity, and amount of informative detail included in the picture. This diagram will be worth approximately 7 points of the total HW grade.

Bring your finished or nearly finished diagram to your lab on Wednesday October 30th, 2024 and discuss the design with a TA/mentor to receive a 1 day deadline extension on this homework.

Extra Credit

For extra credit, you may implement the degree clearance operation:

`load_degree_requirements simplified_degree_requirements.txt` This operation will load the degree requirements data from the `simplified_degree_requirements.txt` file.

`degree_clearance Ben_Bitdiddle` This operation checks Ben's current academic plan with the degree requirements of his selected major and prints helpful information about any problems with the schedule that will prevent graduation.

Submission

Use good coding style when you design and implement your program. Be sure to write your own new test cases and don't forget to comment your code! Use the provided template `README.txt` file for 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 this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.