Middle East Technical University Department of Computer Engineering

CENG 242

Programming Languages

Spring 2021-2022

Programming Exam 6

author: Merve Asiler

Due date: 27 May 2022, Friday, 23:59

1 Problem Definition

In this exam, you are going to construct METU's student-course system. You are going to register students to METU, open courses, add students to them, grade the students' exams. Finally, in case that there is a cheat action in an exam, you will detect and extract seating plan. In this system, there will be 3 main classes:

- The `class Course` representing the courses in METU,
- The `class Student` representing the students in METU, and
- The `class Metu` representing METU itself.

2 Classes

2.1 `class Course, class OpenCourse, class CourseInstance`

1. `class Course`
   Each object of this class represents a base course which is constructed just by a string name. They can be considered as the fixed predefined courses in METU. They may require other `Course` objects as prerequisite. When a course is to be opened in a semester, an `OpenCourse` object is defined by taking a `Course` object as a base.

2. `class OpenCourse : public Course`
   This class is derived from the `Course` class. Objects of this class are specialized forms of the `Course` object from which they are structured, and they represent that course opened for a specific semester. For instance, `Course ceng242("PL")` represents a base for all PL courses whereas `OpenCourse open_ceng242(ceng242, "2021-2", ...)` represents the PL course opened in the semester 2021-2. In the example, `...` are used to refer other arguments. In fact, all the implementation details are given in the .cpp files, yet let's shortly mention the general structure here. `OpenCourse` objects are also constructed with a quota, which is the upper limit for the number of students it can be given. They also take candidate student lists when constructed. These are the lists of the students who

want to take the course, yet it may not possible for each of them to take it. A student is able to take the course if only s/he took its prerequisite course(s) before and the course quota is not exceeded. Also, among the students, 4th-year-students (seniors) has priority over 3rd-year-students (juniors), 3rd-year-students has priority over the 2nd-year-students (sophomores) and 2nd-year-students has priority over the 1st-year students (freshmans). Therefore, as the quota permits, you should give the course depending on this priority.

3. `class CourseInstance : public OpenCourse`
   This class is derived from the `OpenCourse` class. Objects of this class are specialized forms of the `OpenCourse` object from which they are structured, and they represent that open course taken by a specific student. For instance, `OpenCourse open_ceng242(ceng242, "2021-2", ...)` represents the PL course opened in the semester 2021-2 whereas `CourseInstance course_instance(open_ceng242, student_S)` represents that course instance taken by the `student_S`. It holds the grade taken by `student_S` for the course PL.

2.2 class Student, class Freshman, class Sophomore, class Junior, class Senior

Don't worry, the workload is not heavy as it seems. For the completeness of the story, there have been defined 5 classes in this category, yet you only need to implement simple (short) functionalities for Student, Freshman and Sophomore and the other parts are just copy-paste.

1. `class Student`
   Each object of this class represents a student which is constructed just by id, name and department information. Nevertheless, for the objects of this class, at which education year of his/her studentship the student is registered is not known. You can assume that they are in prep class. They are not allowed to take course (namely, there is no defined method for this purpose). However, since they may pass the class and upgrade to Freshman, Sophomore, Junior or Senior status later, some methods such as listing the courses they took, or getting their GPA exist in this class. Their implementation details are given in .cpp files.

2. `class Freshman : public Student`
   This class is derived from the `Student` class. Objects of this class represent the students who are registered at their 1st year. They are allowed to take any course if they satisfy the course conditions (the things explained in `class OpenCourse` section, i.e. quota, priority, prerequisites). For that purpose; `bool addCourse(const OpenCourse&)` method is defined.

3. `class Sophomore : public Freshman`
   This class is derived from the `Freshman` class. Objects of this class represent the students who are registered at their 2nd year. From now on, they can start working as an intern. For that purpose; `void doAnInternship(int)` method is defined.

4. `class Junior : public Sophomore`
   This class is derived from the `Sophomore` class. Objects of this class represent the students who are registered at their 3rd year. Semantically, they are allowed to select elective courses now, and `void selectElectiveCourse()` is defined for that purpose. **However, you do not need to implement anything in that method.** It is just put to complete the story :)

5. `class Senior : public Junior`
   This class is derived from the `Junior` class. Objects of this class represent the students who are registered at their 4th year. Semantically, they are allowed to graduate now in the case they satisfy the graduation conditions, and `bool graduate()` is defined for that purpose. **However, you do not need to implement anything in that method.** It is just put to complete the story :)

2.3 `class Metu`

This class represents METU itself. It is responsible of registering students, defining and opening courses, upgrading students which means moving the student to his/her next academic year when s/he passed the current year, and extracting the schema of cheating to apply university regulations. In the pdf, only the last functionality will be explained since the others are trivial as it can be understood from their names and their implementation details are given .cpp files.

2.3.1 Seating Plan Extraction

In case that a cheating activity is detected from the exam results, the seating plan is extracted and presented as an evidence to the university jury. In that scenario, you are responsible of extracting the seating plan. Seating plan consists of a 2D array, in which each location represents a seat. The challenge is, there does not exist a given plan before the exam, so the students are placed randomly (as they wish). The plan is extracted by taking statement of each student participating the exam. In his/her statement, each student admits only the friend from whom s/he cheated by giving her/his seating position relative to himself/herself. This statement is implemented by `addCheatInfo()` method. For example, `addCheatInfo(1234567, 7654321, "|")` means that the student with id 1234567 sat in front of the one with id 7654321. Another example is `addCheatInfo(1234568, 7654320, "-")` means that the student with id 1234568 sat to the left of the one with id 7654320. As you see, only two types of sitting is given to computation system: "—" represents vertically successive sitting and "-" represents horizontally successive sitting. Since all the students declare their statements in random order, the seating plan information is collected as mixed data. However, in the end, you are expected to extract the 2D array structure properly and print the seating plan exactly. Examine the example below:

Example:

```
setClassroomSize(7, 7);  // Initially, # of rows and columns in the class is given.

addCheatInfo(18, 35, "-");
addCheatInfo(18, 13, "|");
addCheatInfo(7, 19, "|");
addCheatInfo(23, 26, "-");
addCheatInfo(51, 36, "-");
addCheatInfo(36, 47, "-");
addCheatInfo(47, 30, "|");
addCheatInfo(30, 11, "-");
addCheatInfo(82, 80, "-");
addCheatInfo(65, 80, "|");
addCheatInfo(1, 65, "-");
addCheatInfo(12, 13, "-");
addCheatInfo(35, 10, "|");
addCheatInfo(10, 45, "-");
addCheatInfo(19, 20, "-");
addCheatInfo(65, 58, "-");
addCheatInfo(58, 51, "-");
addCheatInfo(32, 49, "-");
addCheatInfo(24, 32, "|");
addCheatInfo(23, 24, "|");
addCheatInfo(12, 26, "|");
addCheatInfo(30, 29, "|");
addCheatInfo(45, 7, "|");
addCheatInfo(7, 3, "-");
addCheatInfo(4, 6, "|");
addCheatInfo(2, 6, "-");
addCheatInfo(49, 2, "-");
addCheatInfo(51, 39, "|");
addCheatInfo(39, 35, "|");

printSeatingPlan();
```

... prints the following as the output:

```
1    65   58   51   36   47   X
82   80   X    39   X    30   11
X    X    18   35   X    29   X
X    12   13   10   45   X    X
23   26   X    X    7    3    X
24   X    X    4    19   20   X
32   49   2    6    X    X    X
```

... where `X` is printed to represent the empty seats in the classroom of size 7x7. It is guaranteed that each student id is unique and there will be no given erroneous information.

3 Grading

Although only the methods given below will be graded, all the other methods should also be implemented for proper system execution.

- `float Student::getGPA() const; ........................................ 6 points`

- `bool Sophomore::getInternshipStatus(int); ............................. 6 points`

-
  ```
  OpenCourse& Metu::openCourse(const Course&, string, int,
                               vector<Freshman*>, vector<Sophomore*>,
                               vector<Junior*>, vector<Senior*>); ....... 20 points
  ```

- `const vector<int> OpenCourse::showStudentList() const; ............... 20 points`

-
  ```
  Sophomore* Metu::upgradeStudent(Freshman&); .......................... 6 points
  Junior* Metu::upgradeStudent(Sophomore&); ............................ 6 points
  Senior* Metu::upgradeStudent(Junior&); ............................... 6 points
  ```

- `Seating Plan Extraction ............................................... 30 points`

- `Memory Leak ....................................... 15% penalty over total grade`

4 Regulations

1. **Implementation and Submission:** The template files "*.h" and "*.cpp" are available in the Virtual Programming Lab (VPL) activity called "PE6" on OdtuClass. At this point, you have two options:

   - You can download the template files, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   The second one is recommended. However, if you're more comfortable with working on your local machine, feel free to do it. Just make sure that your implementation can be compiled and tested in the VPL environment after you submit it. "*.h" and "*.cpp" files are given to you so that you can work on your local machines. If you choose first option, you have to submit these files as well but they will not be included into evaluation process. There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Library:** Using any C++ library other than the already givens is FORBIDDEN.

3. **Programming Language:** You must code your program in C++. Your submission will be compiled with g++ on VPL. You are expected to make sure your code compiles successfully with g++ using the flags -pedantic.

4. **Cheating:** We have zero tolerance policy for cheating. People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

   Important Note: The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your actual grade after the deadline.