6. PRACTICAL EXAMINATION

Below you will find a problem statement similar to what you can expect to receive during the practical examination. The problem statement will in general follow the requirements set out between Lab 5 and Lab 14, will require a graphical user interface (using Qt) and writing specifications, tests and the implementation of layered architecture.

***Observations:***

1. Solving the following problem statement completely should be possible for you in a time span of 3 hours.

2. You are encouraged to bring your own laptop to the exam. You are free to use your preferred IDE. Make sure your IDE is set up correctly and it works! Make sure that Qt works!

3. You are allowed to use Qt Designer, if you want to.

4. The problem must be started from an empty workspace. You are allowed to use the following sites for documentation, but nothing else:

- http://doc.qt.io/qt-5/

- http://en.cppreference.com/w/

- http://www.cplusplus.com/

6.1. Problem statement

**Write an application for teachers, which simulates shared grading, as follows:**

1. The information about the students is stored in a text file, each student having: **an id** (integer), **a name** (string), **a group** (integer), **a grade** (double) and the **name of the teacher** that graded the student (string). The file is manually created and the data about the students is read when the application starts.

2. There is a main teacher, who can add and remove students.

3. Another file contains information about the teachers who can grade students. Each teacher has *a* **name** (string) *(and the groups that he/she has activities with)*. This information is also read from the file when the application starts.

4. When the application is launched, a new window is created for the main teacher, which shows all the students, sorted by their groups and by their names. **(1.5p)**

5. Also, when the application is launched, a new window is created for each teacher who can grade students, having as title the teacher’s name. This window will show all the students, sorted by their groups and by their names. (*Each such window will only show the students belonging to the groups that specific teacher has activities with.*) **(1p)**

6. The main teacher can add and remove students. When a student is added, the teacher must input the name and the group. The grade will automatically be set to 1 and the grading teacher to the empty string. **(1p)** When a student is removed, the application must show a confirmation dialog, to

prevent the teacher from deleting students by mistake. If the teacher confirms, the student is deleted. **(1p)**

7. When a modification is made by the main teacher (a student is added or removed), all the other teachers will see the modified list of students in their windows. **(1p)**

8. The teachers can grade students. When a student is graded by a teacher, the main teacher, as well as all the other teachers will see the modifications: the new grade and the name of the teacher that made the modification. **(1p)** If a teacher tries to grade a student that has already been graded (by some other teacher), the operation fails and the teacher is informed with a message. **(0.5p)**

9. When the application is finished, the students’ file will be updated. **(0.5p)**

**Non-functional requirements**

1. Use STL to represent you data structures.

2. Use an object *Student* to represent the necessary data.

3. Use a class *GradingRepository* to manage your teachers and your students.

4. Use a class *GradingController*, which has the role of a controller.

5. Create a custom defined exception class to handle the constraint “If a teacher tries to grade a student that has already been graded (by some other teacher), the operation fails” and use objects of this class. This constraint will be imposed by *the controller*: the function that modifies a grade will have this as a precondition and if it is broken, a custom exception will be thrown.

**Observations**

**1. 1p - of**

* 1. 2. Specify and test the following functions (repository / controller): **(1.5p)** a. Function which adds a student. **(0.5p)**
  2. b. Function which removes a student. **(0.5p)**
  3. c. Function which updates the student’s grade and grading teacher. **(0.5p)**
  4. 3. Use a layered architecture. If you do not use a layered architecture, you will receive 50% of each functionality.
  5. 4. If you do not read the data from file, you will receive 50% of functionalities 4, 5 and 6.
  6. 5. If you create a console based application, you will receive 40% of each functionality.

6.2. Advice for the practical examination

1. Implement a problem similar to the one in the example (Section **Problem statement**). Time yourself while solving it, make sure you can implement at least some of the functionalities in the allotted time (3 hours). This way you can detect where your difficulties are and you can improve yourself.

2. Build your application incrementally: one step at a time, **compile frequently**.

3. Only add one function in one step, so that you can easily revert to a functional version, in case something doesn’t work.

4. **Do not ignore the errors**, solve them before continuing with writing source code.

5. Do not ignore warnings, sometimes these can indicate errors in the program.

6. **Do not implement functionalities that are not required**. By doing this, you might waste valuable time and **the source code will not be graded, only the functional requirements**!

7. If there are issues that you cannot solve, try finding alternative implementations such that you can still test your code.

8. For the practical examination, build a Qt empty project and make sure that it compiles and that you can execute it ( + that you have all the include paths set properly).