Assignment 3

Forfall 11. sep. 2020 av 23.59 **Poeng** 100 **Tilgjengelig** til 4. okt. 2020 i 23.59

Denne oppgaven ble låst 4. okt. 2020 i 23.59.

Introduction

Note: Do not start on this assignment if you haven't mastered classes yet. Wait for the lecture if you have to.

This is the first assignment that will give you a more realistic attempt at writing more complex applications.

These require *design* before you start coding. Think about where you are going to go before jumping into code. Draw on paper if necessary.

Start with the basic building blocks. A tip here is to design classes that represent real world entities, and the functionality that affects them. Put these in separate files. The bigger picture should be thought out before you start writing the code that binds all the smaller parts together.

Try to identity nouns and verbs in the text. Nouns often become classes, verbs become functions or state (variable values).

You're going to display a lot of menus with choices in them. Try to make classes and/or functions to do this so you can reuse code.

Handle errors early. If an error in a function is critical so the function can't continue, just return from it instead of nesting scopes. Use boolean return codes to notify the caller of errors. Typical: True is OK, false is error.

3.1

Your task is to write an application that keeps track of:

- Students: id, name, e-mail
- Tests: id, course name, test name, date/time
- Test results: test id, student id, grade

Details:

- IDs should be assigned automatically, starting from 1
- IDs are not reused even if an item is deleted (think database auto increment)
- Any student can take any number of tests
- Students are not always present at tests

Use the ideas we have discussed in class to design classes that keeps track of all this information and a way to store them while the application is running. Follow my user interface to pass the tests:

```
Please choose:

1. Add student
2. Edit student
3. Remove student

4. Add test
5. Edit test
6. Remove test

7. Add / edit test result
8. Remove test result
9. Show all info
10. Exit
```

The choice numbers must match. The text in the menu does not. Once an operation is completed the application should go back to the menu, ready for another choice.

Each choice has to display information needed to perform that choice. As an example, "Edit student" must show a list of students and let you pick one by number, starting from 1 sorted by id.

Add / edit / remove test result should ask for a student picked by number starting from 1 (sorted by id), and a test picked by number starting from 1, sorted by id, then finally ask for the grade. If the student already has a grade it will be replaced, if he/she doesn’t the grade is added.

In all edit / remove menus the last choice means back to the menu (do nothing). Example: There are 2 students. Remove student let’s you choose 1 or 2 (for the students), or 3 for back.

Removing a student should also remove all results for that student. The same for other data relationships. The model should never be inconsistent. Adding a test result to a student should not display tests the student has already taken, and so on.

Expected output (all of these must be displayed on choice 9):

```
Student details:
student id = 1, name = Christian Auby, email = christian.auby@uia.no

Test details:
test id = 1, course name = DAT220, test name = Exam, date = 2015.12.14

Test results:
student id = 1, course name = DAT220, test id = 1, test name = Exam, grade = 4
```

Areas of interest:
Cpp files and header files. Classes, operator overloading, std::list, std::vector, std::map

Optional: std::fstream (optional for storing the information)

Important: The select student and select test menus (used on edit and remove menus) must show a short version of the details above. Otherwise the edit and delete tests will fail because my tests will see the old version of the student/test.

3.2

Design a templated utility class called SuperMath that implements the following functions:

- add
- subtract
- multiply
- divide

The application must then read in two numbers from the user and call each function with those two numbers.

```
Expected output (2 and 4):
Add: 6
```

```
Sub: -2
Mul: 8
Div: 0.5
```

Areas of interest:
classes, templates, header files

Delivery

1. Test locally

see *Local testing* in Canvas

Test the solution on your own computer. You can fix your code and retest as many times as you want.

2. Deliver and test in Bamboo

See *Delivery and Bamboo testing* in Canvas

Finally deliver and test the solution in Bamboo. This is what updates your score in Canvas which is the score I can see. You can test as many times as you want here as well.