Object Oriented Programming - Classes

Fundamentals on Computing for Robotics,

Rational calculator

In the Github repository (subfolder `src/assignment_05_classes_materials`) you will find the implementation of a rational number as an abstract data type.

As a first step, implement a calculator (similar to a previous assigment) with this data type, in which the user will introduce a bynary mathematical expression with rational numbers (such as `1/3 + 1/2`) and the program will show its result on screen (e.g. `5/6`).

The program will interactivelly ask for expressions until one expression has two zero operands (both rational numbers are zero).

Implement this behavoir in a file in the same directory `calculator.cpp`, as well as a `CMakeLists.txt` that compiles everything. Rename the folder as `rational-struct`.

Rational class

Reimplement the `Rational` data type so that it becomes a class, and all the related functions become methods (the `rational` function should become the constructor).

For that, duplicate the whole folder (including `rational.h`, `rational.cpp` and `calculator.cpp`) and rename it as `rational-class`.

Modify the calculator so it works with the class version of the rational number.

Rational operators

Once again, reimplement the `Rational` data type so that it is a class (same as previous section) but it works with operators. Particularly, except for `is_zero` and the constructors, everything else should be an operator:

- `<<` and `>>` for input/output (these need to be implemented as functions, they cannot be methods).
- `+`, `-`, `*` and `/` as arithmetic operators (these need to be implemented as methods).

For that, duplicate again the whole folder and rename it as `rational-operators`.

Modify the calculator so it works with the operators. Notice that the source code of this calculator looks very similar to the one you implemented for real numbers.

Submission

The source code that solves the proposed problem should be in three separate folders, one per task: `rational-struct`, `rational-class` and `rational-operators`.

Each of these subfolders should contain four files: `rational.h`, `rational.cpp`, `calculator.cpp` and `CMakeLists.txt`. They should compile using `cmake`.

Before submitting, remember to talk to your teacher and have a **small interview** describing and answering questions about the job you have done and the decissions you have made. This interview is *20%* of the total grade of the assignment, while the submitted material is evaluated for the other *80%*.

All source code files, must include a comment on top with the names and NIAs of the students involved on their development. Then, all source code files, including directory structure, must be compressed into a single `zip` file with the following naming:

- **`rational_<nia1>_<nia2>.zip`**, where `<nia1>` and `<nia2>` are the NIAs of the involved students, if the work has been done in pairs. In this case, **only one** of the students must submit the work in Moodle.
- **`rational_<nia>.zip`**, where `<nia>` is the NIA of the involved student if the work has been done individually.

The compressed file must not contain anything else besides the source code files. Particularly, do not submit any binary file, neither executable, library nor object. The submission of the `zip` file must be done through the corresponding task in Moodle before the stablished deadline.