Homework. 6: Intro to classes in C++

Ignacio Vizzo, E-Mail ivizzo@uni-bonn.de

Handout : 25.05.2020
Handin: 12.06.2020 at 23:59:59 (CET)

General rules

1. You need to provide the build system for this homework. This means, you need to provide as many **CMakeLists.txt** files as you think is needed.
2. Your code will be checked using `clang-format`. If your code in not properly formatted, then the tests will not run.
3. Your code will be static-analyzed by `clang-tidy`, if your code does not comply with the analysis, the tests will not run. If you are using VSCode with the reccomended setup, most of the errors will be visible while you program.
4. **ALL** tasks should be solved within the `homework_6` folder, no need to create `task_x` folders.
5. The design of how to solve this exercise is up to you. The **ONLY** requirement to pass the tests are:
    - You should provide one header file(put it where you preffer) and call it `homework_6.h`.
    - All the functions of this exercise must be accessible form this header file
    - You should guarantee this header file can be included thorugh your build system.
6. **ALL** your implementations must lie under the namespace **igg**

A C++ classes: igg::Image (10 points)

In this exercise you will implement a simple version of an image class that can store grayscale values. The end result would be a class that:

- Can be filled from disk from a `*.pgm` file.
- Can be written to a `*.pgm` file.
- Can compute a histogram over its pixels.
- Can be resized.

Note: You will not need to implement the actual reading/writing to disk for now. This functionality is provided to you via a small library `libio_tools` that you can find in the homework files. Notice that you have the library implementation, but as usual, you are in charge of how to build this library and properly link it agains your code.

Optional: You are not forced to use this library, you can write your own implementation of this functionality from scratch, if you wish. Use your own `API`, etc. I will "review" your implementation but it wont be subject to any type of tests.

A.1 image class

Create a class with the name `image` with the following functionality:

- Image can be created empty or of any size, i.e. there are constructors:
    - `Image();`
    - `Image(int rows, int cols);`

  Make sure the `data_` gets resized to accommodate all the requested elements.
- Size of an image can be accessed with getter functions
    - `int rows();`
    - `int cols();`

  for variables `rows_` and `cols_`. Make sure to be using `const` correctly for the getter functions.
- Pixel values can be accessed and modified through the function `at(int row, int col)`. Make sure this function can be called in both of the scenarios:
    - `int val = image.at(row, col);`
    - `image.at(row, col) = 255;`

  Make sure you use `const` where needed.
- The `Image` class has a single `std::vector` to store two-dimensional data. You will need to compute a single index from `row` and `column` values to store and retrieve pixel values to and from the `data_` vector. Make sure your image stores pixel data in row-major order, i.e. every row is stored sequentially in `data_`. For more info see: `https://en.wikipedia.org/wiki/Row-_and_column-major_order`

A.2 Reading and writing to disk

- Implement functions to read and write the data from disk using the provided library as a proxy. Your class must have the following functions:
  - `bool FillFromPgm(const std::string& file_name);`
  - `void WriteToPgm(const std::string& file_name);`

  These functions should convert the data stored within the `Image` class to and from `ImageData` struct and call functions `ReadFromPgm` and `WriteToPgm`.

A.3 histograms

Compute a histogram over the pixels. This function should take as input the number of bins in the histogram. A histogram counts how many pixels fall into each bin and provides a vector of these values normalized by the total number of pixels. For example, a histogram with two bins will store a normalized count of all pixels with the value below 255/2 in the bin number 0 and a normalized count of all the pixels with value higher than 255/2 in the bin number 1.

The interface should be as follows:

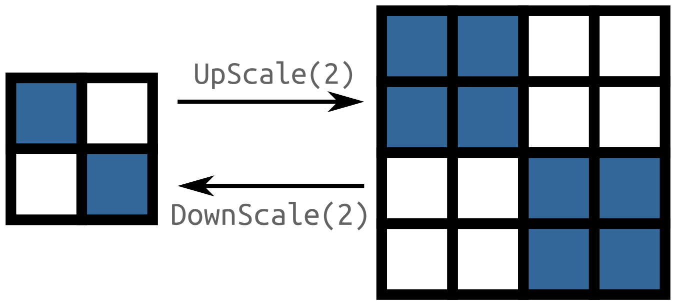
```
std::vector<float> ComputeHistogram(int bins);
```

Note again, that you must make a decision if the function should be `const`

A.4 Resizing the image

- `void DownScale(int scale);`
- `void UpScale(int scale);`

The `scale` is an integer factor. For example `DownScale(2)` should result in an image half the size of the original, while `UpScale(2)` in an image twice bigger than the original.
When downscaling, you must just pick every k pixel depending on the `scale` parameter.
When upscaling some pixels will not have a value. Fill these pixels using the nearest neighbor algorithm.
An example result is illustrated below.



A.5 *Tips*

1. The interfaces provided above are stripped from `const` modifiers. It is part of this exercise to think where `const` is appropriate and add it where needed.
2. Use Unit Tests to evaluate your work. The evaluation script will inject our custom tests into your framework and will run those tests against your code. Do not remove `tests` folder from the project.