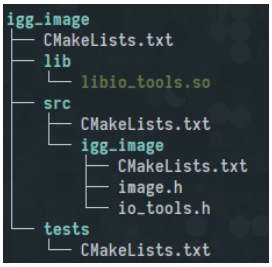
Homework. 2: Intro to classes in C++

Igor Bogoslavskyi, E-Mail igor.bogoslavskyi@uni-bonn.de

Handout : Wed, 18.04.2018
Handin: Wed, 09.05.2018

In this exercise you will implement a simple version of an image class that can store grayscale values. You are given an initial project sceleton, where you will need to fill in existing files and to add new files so that the below presented functionality is met.



To start:

1. Unpack the provided archive into the `homework_2` folder
2. All your code must be in `igg_image` folder, no need for `task_x` folders

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

You will not need to implement the actual reading/writing to disk for now. This functionality is provided to you via a shared library `libio_tools.so` (`libio_tools.dylib` for a Mac) that you can find in `lib/` folder of the project sceleton `igg_image` along with the appropriate header file `io_tools.h`.

1. **(2 points)** Create a library 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 accomodate 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 fucntions.
   - 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 scenarious:
     - `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-dimentional 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`

2. **(2 points)** Reading and writing to disk.

   - Link your code with the given `libio_tools.so(.dylib)` library. Use `find_library()` CMake function to use the provided library with your code.
   - 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` from the file `igg_image/io_tools.h`

3. **(2 points)** 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`.

4. **(4 points)** Resizing the image:

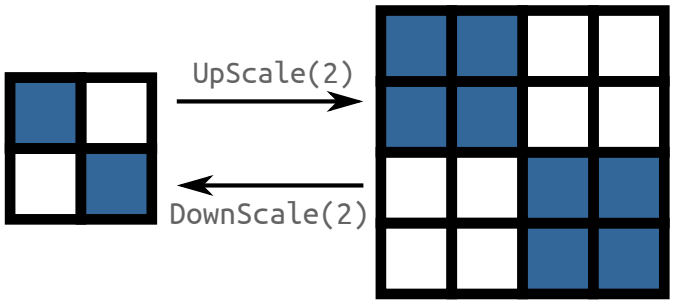
   - `void DownScale(int scale);`
   - `void UpScale(int scale);`

   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.



IMPORTANT: 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.
IMPORTANT: Use Google 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.