Mini Gimp

In the Github repository (subfolder `src/assignment_06_polymorphism_materials`) you will find the implementation of simple tool with very simple brushes to paint images. The tool is not the important concept of this assigment, but still, you can play with it. For that, you need to compile it using the CMakeLists.txt file, either using the command line or Visual Studio Code. This has been compiled and executed on both Windows and Linux distributions, but let your teachers know if you have any issue when compiling it for your particular system.

When you execute it, it will open a simple window in which you can draw by clicking the left button of the mouse and dragging it along the window. The right mouse button does the exact same thing but with a thicker brush.

You can use the left and right arrows of your keyboard to change the brush being used. When you do that, you will see that the title bar of the application changes to describe the brush being used. Right now you have four very simple brushes: red, green, blue and grayscale.

By passing a command line parameter on execution, it is also possible to load a specific image instead of opening an empty black canvas. We provide you a public domain image to do that, but you can use one of your own. From the folder `src/assignment_06_polymorphism_materials` you can use the following command line:

```
build/minigimp -input image.bmp
```

You can play for a while with the tool. You can also have a look at the source code if you want, but we do not expect you to fully understand all of it.

Defining new brushes

Your main task for this assignment is to **define new brushes**. Brushes are defined through inheritance, that provides the necessary polymorphism for each of them having a specific behavior. The superclass is `Brush` and can be found in the file `brush.h`. Each of your new brushes will need to reimplement the two methods that you see there:

- The `name` method returns a string that contains the name of your brush. Note that the same brush might have different names depending on some of its attributes.
- The `edit` method receives a reference to the `r` (red), `g` (green) and `b` (blue) channels of a pixel on the image, which go from 0 to 255. A black color will be (0,0,0), while a green color will be (0,255,0).

Each of your new brushes will be a new class, with its specification in a header (`.h`) file and its implementation in a `.cpp` file. You can get inspiration by the brushes already in the syste,:

- `Color`, defined in `color.h` and implemented in `color.cpp`.
- `Grayscale`, defined in `grayscale.h` and implemented in `grayscale.cpp`.

You can use any of those as a template if you want, but let your new brushes do whatever you want them to do. Be creative, be original, but if you aren't, here you have several ideas: brightening (or

darkening) a pixel, a rainbow brush that changes its colors after every use, a brush that gives random colors from a set of them to each pixel, a brush that saturates or colorizes a pixel…

Once you have defined a new brush, first you need to make sure that its implementation, on a `.cpp` file, is compile. For that, you will need to go to the `add_executable` line on the `CMakeLists.txt` file and add there your `.cpp` file.

In order to the tool to use it, you will need to include it also as part of the vector of available brushes. Go to the `brushes()` method of the `brushes.cpp` file and, following a syntax similar to the other brushes there (`bs.push_back(std::make_unique<?>(...))`), add your brush as well. In fact, if you have parametrized your brush with some attributes that can be set on the constructor, you can add your brush several times with different attributes (similar to the `Color` brush).

You can then recompile and experiment with your brush on the application. It is simply there, and you have not needed to know and understand the whole source code. The only thing you have needed to understand is the class you need to inherit from and what is expected from each of the methods. That's the key finding of inheritance-based polymorfism: you can focus on specific parts of the code without the need of having a grasp of the whole codebase. Also, your code is more mantainable because it is functionally separated.

Submission

The submitted source code should at least include all your new brushes, the new `CMakeLists.txt` and the new file `brushes.cpp`.

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:

- **`minigimp_<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.
- **`minigimp_<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.