Operating in Color

Due Tuesday, February 11 at 8 a.m.

CSE 1325 - Spring 2020 - Homework #3 - 1 - Rev 1

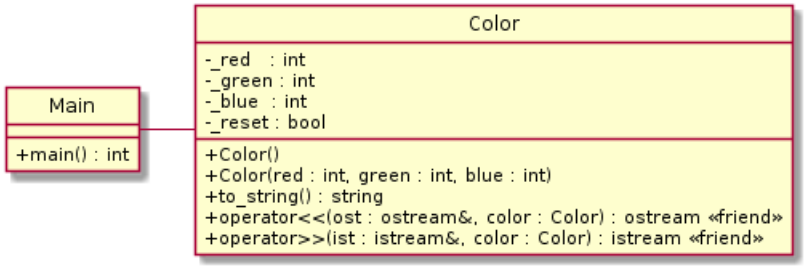
Assignment Overview

Last week we created a Color class to *encapsulate* the red, green, and blue components of an RGB color specification. We added (among others) a `Color::colorize(std::string text)` method to add ANSI escape codes to the parameter text to set its color appropriately.

But that looks suspiciously like a C function (ugh!). Let's *refactor* into a more C++ish `std::cout << red << "I'm red!";` to provide streaming (and other) operators.

Full Credit

In your git-managed Ubuntu Linux 18.04 directory **cse1325/P03/full_credit** (capitalization matters!), update either the Color class that you wrote or the suggested solution - full_credit, bonus, or extreme_bonus version, as you please - as shown in the class diagram.



Remove the `Color::colorize(string text)` method, which will be replaced with the << operator.

Add two friend *functions* - NOT methods! - that overload the << and >> operators.

- **ostream& operator<<(ostream& ost, const Color& color)** will stream out the ANSI escape codes, similar to the prefix added by the previous `Color::colorize(string text)` method.
- **istream& operator>>(istream& ist, Color& color)** will read a color using the `(red, green, blue)` format (e.g., `(255,0,0)`), storing the first integer in _red, the second in _green, and the third in _blue.

Since we'll be setting the color with one object and resetting to defaults in another object, in addition to _red, _green, and _blue integer attributes we'll need a Boolean _reset attribute. If true, operator<< should ignore the other attributes and stream the reset ANSI escape code sequence instead. If false, stream exactly the same ANSI escape sequence as in P02.

Then, write a main() function that:

- Instances your Color class into 3 objects, representing 3 different colors (e.g., `Color red{255,0,0};`). Print the name of each color you instanced *in its representative color* but using the << streaming out operator. Then stream out a color reset (e.g., `Color{}`).
- Instance a Color object named color. Ask the user for a color using the `(red, green, blue)` format, and stream it into color. Then print the color's string representation (the to_string method) *in its representative color*.

Add, commit, and push all files.

```
ricegf@saturn:~/dev/202001/P03/full_credit$ make clean
rm -f *.o *.gch ~* a.out color
ricegf@saturn:~/dev/202001/P03/full_credit$ make
g++ --std=c++17 -c main.cpp -o main.o
g++ --std=c++17 -c color.cpp -o color.o
g++ --std=c++17 main.o color.o -o color
ricegf@saturn:~/dev/202001/P03/full_credit$ ./color
UTA Blue UTA Orange Maroon

Enter color as (red, green, blue):   (128,0,192)
(128,0,192)
ricegf@saturn:~/dev/202001/P03/full_credit$
```

Bonus

In the previous assignment, you called `bool Color::compare(const Color& rhs)` directly to compare magnitudes. This time, override the 6 comparison operators (<, <=, ==, !=, >=, and >) to use the results of the compare method so that the magnitude of each color is compared by, e.g.,

```
if (Color{255,0,0} > Color{0,0,255}) std::cout "Red is brighter than blue!\n";
```

prints "Red is brighter than blue!".

Write the comparison operator overloads as *methods* rather than friend functions of class Color. (Traditionally, these are *inline methods* using the inline keyword and the definition as part of the class declaration in color.h, although we won't count off if you use traditional methods instead) Make the `bool Color::compare(const Color& rhs)` method private.

Then, write a main function that creates 64 random colors on a vector of type Color named colors. Then, *sort <http://www.cplusplus.com/reference/algorithm/sort/>_* the vector by color magnitude, noting that you need only provide *two* parameters (colors.begin() and colors.end()), since objects of type Color are now comparable with your lovely new operators. Finally, print the to_string of each color in that color in sorted order.

Add, commit, and push all files. Additional information that you may find helpful follows.

```
ricegf@saturn:~/dev/202001/P03/bonus$ make
g++ --std=c++17 -c main.cpp -o main.o
g++ --std=c++17 -c color.cpp -o color.o
g++ --std=c++17 main.o color.o -o color
ricegf@saturn:~/dev/202001/P03/bonus$ ./color
(75,18,195)
(38,53,40)
(206,11,5)
(182,2,182)
(90,27,231)
(76,53,68)
(6,65,158)
(131,33,140)
(44,68,126)
(141,49,46)
(58,71,134)
(40,87,79)
```

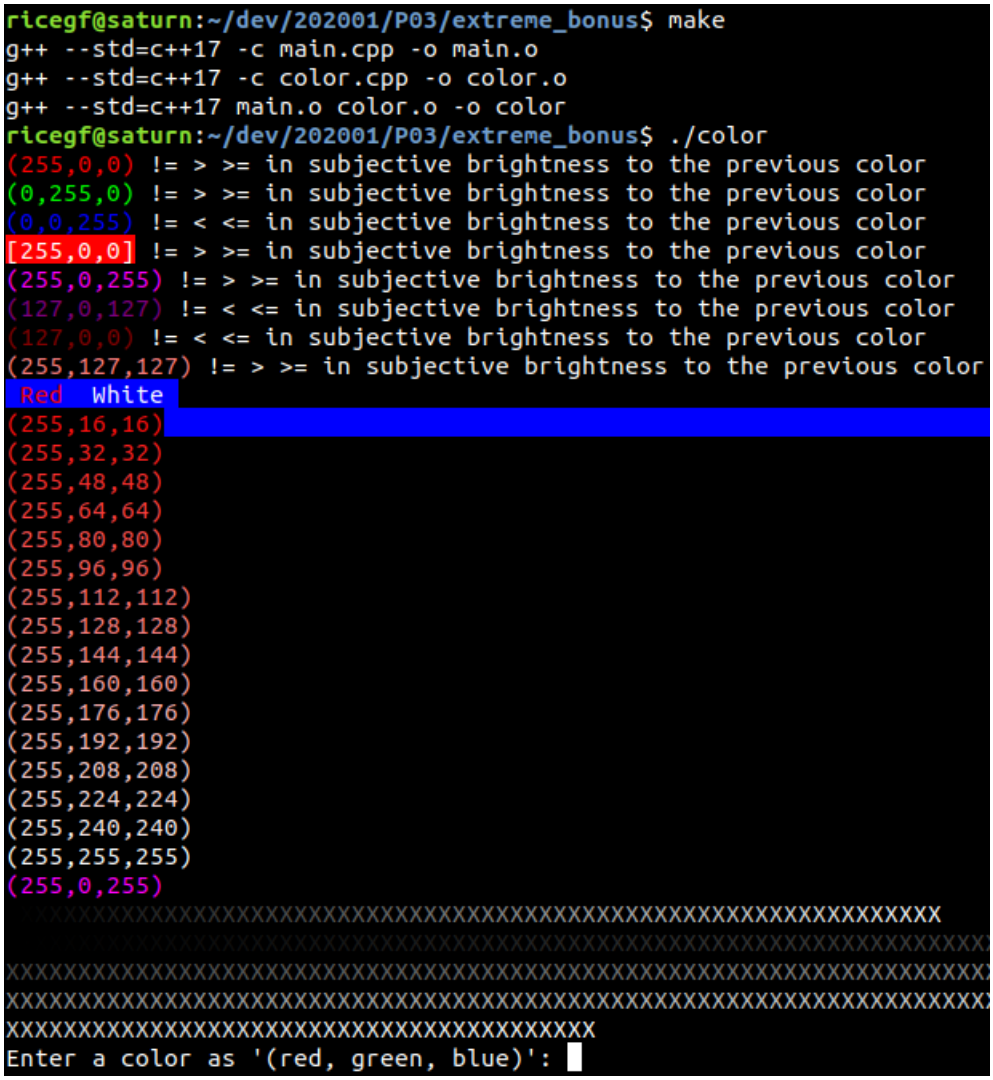
Extreme Bonus

Implement other operators for class Color. Select from the following options:

- Also support *background* colors. A background color is specified using a similar ANSI escape sequence, just replacing "38" with "48" (think ternary operator, perhaps). For example, you might add a `bool _background;` attribute set via an optional constructor parameter. So `Color red{255,0,0}` might be foreground red, `Color bk_blue{0,0,255,true}` background blue, and `std::cout << red << bk_blue << "Hi!"` would print "Hi!" in red text on a blue background. Or you may want to use the *factory pattern* from cse1325-prof/04/code_from_slides/complex.h to specify a background color.
- Support *color addition and subtraction*. When using the + or - operator, simply add or subtract each respective _red, _green, and _blue parameter to create the new color to return, e.g., `Color purple = red + blue;`. Be sure to limit the resulting integers between 0 and 255, inclusive!
- Support *lightening and darkening* colors by adding or subtracting the same integer each of the red, green, and blue components via the + and - operator, e.g., `Color dark_red = red - 128;`. As with the previous option, be sure to limit the resulting integers between 0 and 255, inclusive.
- Support *incremental lightening and darkening* of colors via the ++ and -- operators, which increment or decrement each red, green, and blue component on each call, e.g., `red--;`. Be sure to support both prefix (`++color`) and postfix (`color++`) notation for each operator.
- Support *color blending* by averaging and de-averaging each respective _red, _green, and _blue parameter to create the new color to return using the * and / operators, e.g., `Color dark_purple = red * blue;`. Pay special attention to the algebra of de-averaging, such that `(color1 * color2 / color2) == color1` is logically true (ignoring rounding error).
- Provide *predefined colors* for black, blue, green, cyan, red, magenta, yellow, grey, and white, such that e.g., `Color::GREEN` provides exactly the same color as `Color{0,255,0}`. Also provide `Color::RESET` as a synonym for `Color{}`.

Write a main function that clearly demonstrates the proper operation of each feature you implement. IMPORTANT: It need NOT look anything like the example below! And it can look as random or as organized as you like. But above all, never forget to add, commit, and push all files!

Up to 15 points will be added to your grade for implementing one of these options *well*, 10 points for a second option, and 5 points for each remaining option (155 points max for this homework).


ricegf@saturn:~/dev/202001/P03/extreme_bonus$ make
g++ --std=c++17 -c main.cpp -o main.o
g++ --std=c++17 -c color.cpp -o color.o
g++ --std=c++17 main.o color.o -o color
ricegf@saturn:~/dev/202001/P03/extreme_bonus$ ./color
(255,0,0) != > >= in subjective brightness to the previous color
(0,255,0) != > >= in subjective brightness to the previous color
(0,0,255) != < <= in subjective brightness to the previous color
[255,0,0] != > >= in subjective brightness to the previous color
(255,0,255) != > >= in subjective brightness to the previous color
(127,0,127) != < <= in subjective brightness to the previous color
(127,0,0) != < <= in subjective brightness to the previous color
(255,127,127) != > >= in subjective brightness to the previous color
Red White
(255,16,16)
(255,32,32)
(255,48,48)
(255,64,64)
(255,80,80)
(255,96,96)
(255,112,112)
(255,128,128)
(255,144,144)
(255,160,160)
(255,176,176)
(255,192,192)
(255,208,208)
(255,224,224)
(255,240,240)
(255,255,255)
(255,0,255)
Enter a color as '(red, green, blue)':