Coding in Color

Due Tuesday, February 4 at 8 a.m.

CSE 1325 - Spring 2020 - Homework #2 - 1

Assignment Overview

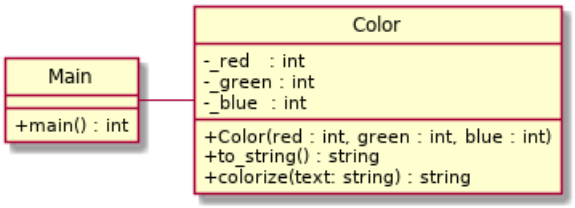
At the dawn of the Age of Video Terminals, around 1969, each terminal had unique special codes to control the cursor and create colorized text. In 1978, to reconcile these Terminals of Babel, the American National Standards Institute (ANSI) proposed a set of standard control codes as ANSI X3.64, which went international in 1983 as ISO 6429.

More important was the release of the ANSI X3.64-compatible Digital Equipment Company (DEC) VT-100 terminal in 1978, which sold in the millions and became the standard terminal for decades. To this day, terminal emulator programs such as the one used by Linux support "VT-100 emulation" and what are now called "ANSI escape codes" (since they rely on the "escape" character, labeled "Esc" on most keyboards).

Although Linux includes an extensive library called ncurses to manage these codes, you learn by doing. Let's write a simple class to put them to use!

Full Credit

In your git-managed Ubuntu Linux 18.04 directory **cse1325/P02/full_credit** (capitalization matters!), create class Color as shown in the class diagram. (We don't cover class diagrams until Lecture 04, but don't panic - all is explained below!)



A color will be represented by the RGB color model, so your class will store *private* attributes _red, _green, and _blue integer each between 0 and 255, inclusive. *Public* executable Color class members are:

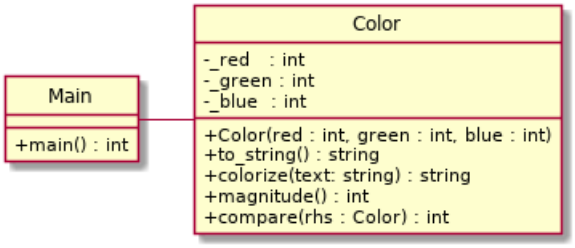
- **Color(int red, int green, int blue)** - The constructor should just store each parameter in its respective attribute *using an initialization list*, with an empty constructor body.
- **std::string to_string()** - This method returns a string representation of the RGB (red then green then blue) color, i.e., for purple it would return "(128,0,128)".
- **std::string colorize(std::string text)** - This method returns its parameter *preceded* by the ANSI escape code for "set foreground color" to _red, _green, and _blue, and *followed* by the ANSI escape code for "reset".

Then, write a main() function that:

- Instances your Color class into 3 objects, representing your 3 favorite colors (e.g., `Color red{255,0,0};`). Print the name of each color you instanced *in its representative color* (e.g., `std::cout << red.colorize("Red") << std::endl;`.
- Ask the user for 3 integers representing red, green, and blue. Instance a Color object to represent that color, e.g., `Color color{r, g, b};`. Then print the color's string representation (the to_string method) *in its representative color*.

Bonus

A class can collect quite a few methods related to the data it manages. Let's add a couple of additional methods, shown in the updated class diagram below.



The two additional executable Color class members are:

- **int magnitude()** - This method returns the "subjective brightness", or magnitude (between 0 for "pitch black" and 255 for "noonday sun") for the color specified by the attributes _red, _green, and _blue. One algorithm to calculate this is to return 21% of red, 72% of green, and just 7% of blue. Watch out for underflow! :-)
- **int compare(const Color& rhs)** - When we compare two objects, the left-hand side (lhs or *this* object) to the right-hand side (rhs), we have 3 possible outcomes: lhs < rhs, lhs == rhs, or lhs > rhs. So compare *the magnitude* of the current object (usually simply called *this*) to *the magnitude* of the rhs object provided as a constant reference parameter, and return 1 if *this* > rhs, 0 if *this* == rhs, or -1 if *this* < rhs.

Update Color::to_string to also display the magnitude of the color, e.g., `(0,100,177:84)` where 84 is the magnitude (on a scale of 0 to 255) of color (0,100,177). Your main function should not need to change. (We'll exercise Color::compare in the extreme bonus section.)

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

Extreme Bonus

Let's make use of the Color::compare method you wrote for the bonus section. Write a new main that does the following.

- Instance a vector of Color objects named colors. Remember, a vector can be specified to hold any type!
- Push at least 3 *different* colors onto it.
- Sort the vector, using Color::compare as the criteria. This is the "interesting part" of the problem that we haven't covered yet. Enjoy!
- Print out the colors in the vector from dimmest to brightest.

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

Hints!

The Professor's cse1325-prof

The professor for this class provides example code, homework resources, and (after the due date) suggested solutions via his cse1325-prof GitHub repository.

If you haven't already, clone the professor's cse1325-prof repository with "`git clone https://github.com/prof-rice/cse1325-prof.git`". This will create a new directory called "cse1325-prof" in the current directory that will reflect the GitHub repository contents.

The cse1325-prof directory doesn't automatically update. To update it with the professor's latest code at any time, change to the cse1325-prof directory and type "`git pull`".

Suggested Makefile

You may use the provided Makefile from `cse1325-prof/P02/full_credit` to build both the examples and your code.

```
CXXFLAGS = --std=c++17

all: example test color

color: main.o color.o
	$(CXX) $(CXXFLAGS) main.o color.o -o color
test: test.o color.o
	$(CXX) $(CXXFLAGS) test.o color.o -o test
example: example.o
	$(CXX) $(CXXFLAGS) example.o -o example

main.o: main.cpp
	$(CXX) $(CXXFLAGS) -c main.cpp -o main.o
color.o: color.cpp
	$(CXX) $(CXXFLAGS) -c color.cpp -o color.o
example.o: example.cpp
	$(CXX) $(CXXFLAGS) -c example.cpp -o example.o

color.gch: color.h
	$(CXX) $(CXXFLAGS) -c color.h -o color.gch

clean:
	rm -f *.o *.gch ~* a.out test color example
```

ANSI Escape Code Example

As an example, and to verify that your terminal does indeed support ANSI escape sequences, try building and running cse1325-prof/P02/full_credit/example.cpp using "`make example`":

```
// This is an example of how colorful text is printed to the console

#include <iostream>

int main() {
   std::cout << "\033[38;2;0;100;177m" // Sets color to 0 red, 100 green, 177 blue
             << "UTA Blue"             // Text to print in the above color
             << "\033[0m"              // Resets terminal to default state
             << std::endl;             // Newline and flushes the output buffer
}
```

This should print "UTA Blue" in a familiar blue color.

Full Credit

Test Color in Your Terminal

To verify that your terminal program supports ANSI code sequences, *first build and run example.cpp as described above*.

Write color.h

Then, in cse1325/P02/full_credit, **write color.h** based on the class diagram.

- Don't forget your guard in color.h (`#ifndef __COLOR_H` and `#define __COLOR_H` at the top of the file, `#endif` at the bottom).
- In the UML, attributes starting with "-" become private variables, e.g., `-_foo : int` becomes `int _foo;`.
- Public methods starting with + become class methods, e.g., `+bar(spam : string) : double` becomes `double bar(std::string spam);`.

Test Compile color.h

You may *test compile color.h file* using `make color.gch` (with the provided Makefile) or `g++ --std=c++17 color.h` otherwise. No errors or warnings means it *may* be correct. No later than once it compiles, **put it under version control** with `git add color.h` and `git commit -m 'P02 color.h first draft'` and `git push`.

Add it to GitHub using `git add color.h`, `git commit 'P02 color.h'` and `git push`.

Write color.cpp

Next, **write color.cpp**. Don't forget to `#include "color.h"`. Remember, `std::string to_string();` from color.h becomes `std::string Color::to_string() { /* your implementation here */ }` in color.cpp. If you forget `Color::`, you'll get an error message something like "variable not in scope".

Test Compile color.cpp

You may test compile a .cpp file using `make color.o` (if you have a Makefile) or `g++ --std=c++17 color.cpp` otherwise. No errors or warning means it *may* be correct. No later than once it compiles, **put it under version control** with) `git add -u`, `git add color.cpp` and `git commit "P02 color.h first draft"` and `git push`.

`git add -u` (for color.h), `git add main.cpp`, `git commit -m 'P02 color.cpp'` and `git push`.

Execute Test

Once you have the Color class compiling, **try a simple test**, e.g., with the following main function from test.cpp using `make test`:

```
#include "color.h"
#include <iostream>

int main() {
   Color red{255,0,0};
   std::cout << red.colorize("Red") << std::endl;
}
```

Debug until it runs well, and then if you made any changes (try `git status .` to be sure), add -u, commit, and push as above.

Write and Debug main.cpp

Once you have some confidence in your Color class, **write your own main function** per the spec.

Build, run, and test, then add, commit, and push.

Done! We'll clone your repository (using the link you posted in Canvas as the solution to the first assignment) and grade it there.

Completing this task earns 100 points. **This must be completed prior to attempting any of the bonus levels.** You will receive no credit for bonus work if you do not complete the full credit assignment first.

Bonus

In **cse1325/P02/bonus**, duplicate your code from cse1325/P02/full_credit. Simply edit color.h to add the two additional method declarations, and write their implementations (their definitions) in color.cpp. Don't forget to modify Color::to_string to also show the color's magnitude, e.g., "".

Note that in calculating the magnitude, you don't need to specify the object for lhs - it's *this*, which is assumed if not specified. So rhs.magnitude() calls the parameter's magnitude method, while simply magnitude() with not preceding object calls this object's magnitude.

Again, no later than once it compiles - add, commit, and push!

Completing this task earns up to an additional 10 points.

Extreme Bonus

A little googling will reveal that <algorithm> includes a sort function that takes 3 parameters.

- colors.begin() - This is essentially a pointer to the first element of your vector (we'll cover it is great detail later in the class)
- colors.end() - Similarly, a pointer to the last element of your vector (not exactly, but close enough for homework #2!)
- less_than - A *function* (NOT a method) that takes two parameters, Color lhs and Color rhs, and returns false if lhs <= rhs or true of lhs > rhs. *You'll need to write this function*, either above main(), providing just its name (sans parentheses) as the third parameter of sort, or perhaps as a lambda (in for a penny, in for an extreme pound).

You do NOT have to use algorithm's sort function, of course. It's an extreme bonus - get creative! And remember, we aggressively give partial credit! :-D

As always - add, commit, and push!

Completing this task earns up to 15 additional points, depending on how well you impress the grader.

Example Output

```
ricegf@saturn:~/dev/202001/P02$ ./full_credit/color
UTA Blue UTA Orange Maroon

Enter red, green, and blue ints:   255 0 255
(255,0,255)
ricegf@saturn:~/dev/202001/P02$ ./bonus/color
UTA Blue 84 UTA Orange 146 Maroon 38

Enter red, green, and blue ints:   255 255 0
(255,255,0:237) 237
ricegf@saturn:~/dev/202001/P02$ ./extreme_bonus/color
(93,23,37:38)
(0,100,177:84)
(245,128,38:146)
ricegf@saturn:~/dev/202001/P02$
```