1 Introduction

Imagine that you have a bunch of cans with paints in many different colors. You are going to build a mixer for paints. The mixer can be loaded with cans and then it can mix paints from a given number of cans.

1.1 Part 1

POINTS: 1

Implement class Paint. Each paint is described by 3 values of red, green, and blue (RGB color space), and by the can fill level. Assume that there is only one size of the cans, and all cans are equal. Values of red, green, and blue channels should be between 0 and 255 inclusive, and the correct value of fill level is between 0 and 1, inclusive.

For example: red: 255, green: 0, blue: 0, fill: 0.5 represents a half empty can of red paint.

Implement a constructor for Paint class which takes 4 parameters: red, green, and blue colors, and fill level. Default values for colors should be 0. Default value for fill level should be 1. If fill level is less than 0 make it 0, if is greater than 1 make it 1.

Hint: use unsigned char for colors.

Implement << operator for Paint class. This operator should allow to print paint in a format: (red,green,blue)[fill%]. For example the half empty can of red paint from the first part would be printed like: (255,0,0)[50%].

Hint: if you used unsigned char type to represent colors, cast them to unsigned int before printing/passing to output stream.

1.2 Part 2

POINTS: 0.5

Implement static method of Paint class GetEmpty(). This method should return an empty can of black paint (all colors set to 0 and fill level also set to 0).

1.3 Part 3

POINTS: 0.5

Implement / operator for Paint class. This operator takes as parameter an unsigned integer and returns a copy of the current paint object but with fill level divided by this parameter. In other words it returns the fraction of the original can with paint. For example (255,0,0)[100%] / 2 is equals to (255,0,0)[50%].

If the parameter is equal to 0, return unmodified copy of the current Paint object.

1.4 Part 4

POINTS: 1

Implement + operator for Paint class. This operator takes as parameter other paint and returns a mixture (sum). Only a part that is a complement of the fill level of the current paint object is mixed ($fill_B$ in equations below should be sometimes decreased). For example if first paint can is filled in 75%, and the second one is filled in 100%, then only 25% of the second can is mixed with the first can. Mix colors independently.

$$newcolor = (color_A * fill_A + color_B * fill_B)/(fill_A + fill_B) \tag{1}$$

$$newfill = fill_A + fill_B \tag{2}$$

1.5 Part 5

POINTS: 2

Implement a Buffer class which works like a stack (LIFO): the last inserted value is returned when retrieving elements from the stack. Use an array of constant maximum number of elements (assume: 30). Implement methods: push - adds paint to the buffer, pop - removes last inserted paint from the buffer and returns it, empty - returns true iff buffer is empty, size - returns current number of elements stored in the buffer.

1.6 Part 6

POINTS: 1

Implement a Mixer class with 2 buffers: the first one for non-mixed paints, the second one for mixed paints. Implement add() method with 2 parameters: a paint and a number of cans to insert to the non-mixed buffer. Implement a size() method that returns the count of all paint cans in the mixer, both mixed and non-mixed. Use class Buffer implemented in part 5.

1.7 Part 7

POINTS: 2

Implement a mix() method of the Mixer class. It takes as parameter the number of non-mixed paints to be used in the mix. This method should use and remove the given number of non-mixed paints and create the exact number of mixed ones in the mixed buffer. This method should return false and abort if the given number is bigger than the number of available non-mixed paints.

Implement a get() method that removes from the mixer and returns a mixed paint. If no mixed paints are available, then non-mixed paints are returned (and removed from the mixer). In case of the totally empty mixer, return an empty can of black paint.