Assignment 3

Constructors, Dynamic Memory Allocation, and Overloading Operators: Creating Color Blobs of Varying Sizes

In this assignment, you will create rectangular color blobs of different sizes. A color blob can be defined as a two-dimensional array of varying width and height where each entry is a color value.

Color has three components (red, green, and blue). A number of different colors can be defined by varying the values of the red, green, and blue components as demonstrated in Table 1. The maximum value of red, green, and blue for a color is **1.0** and the minimum value is **0.0**.

Table 1: Color table for different colors

Color	Red component	Green component	Blue Component
Black	0.0	0.0	0.0
Blue	0.0	0.0	1.0
Green	0.0	1.0	0.0
Cyan	0.0	1.0	1.0
Red	1.0	0.0	0.0
Magenta	1.0	0.0	1.0
Yellow	1.0	1.0	0.0
White	1.0	1.0	1.0

Implementing Color class:

So, a **Color** class will have three member variables, i.e., **red**, **green**, and **blue**, all of type **double**. The default **Color** object has a value of 0.5 for all its member variables, i.e., red, green, and blue. Implement **Color** class with **default constructor**, **constructor with parameter**, **copy constructor**, **copy assignment operator**, **getters**, **setters**, and **friend functions** shown below.

```
1. friend bool operator==(const Color& colorOne, const Color& colorTwo);
2. friend Color operator+(const Color& colorOne, const Color& colorTwo);
3. friend Color operator-(const Color& colorOne, const Color& colorTwo);
4. friend Color operator*(const Color& colorOne, const Color& colorTwo);
5. friend bool operator!(const Color& colorOne);
6. friend ostream& operator<<(ostream&, const Color&);
7. friend istream& operator>>(istream&, Color&);
```

(1) The relational operator "==" takes two **Color** objects as arguments and compares them. If values of member variables **red**, **green**, and **blue** for both **Color** objects are equal, it returns true, otherwise, false is returned.

(2) The arithmetic operator '+' takes two **Color** objects as arguments and adds corresponding **red**, **green**, and **blue** member variables to form a new **Color** object. If values for **red**, **green**, or **blue** exceed **1.0**, values will be clamped at **1.0**.

(3) The arithmetic operator '**-**' takes two **Color** objects as arguments and subtract the second **Color** object from the first one to form a new **Color** object. If after subtraction, values for **red**, **green**, or **blue** become negative, values will be clamped at **0.0**.

(4) The arithmetic operator '*' takes two **Color** objects as arguments and multiplies corresponding **red**, **green**, and **blue** member variables to form a new **Color** object.
(5) The unary operator '**!**' compares whether the color is not black, i.e., **red**, **green**, and **blue** member variables of a **Color** object all are **0**.
(6) '<<' operator prints a **Color** object;
(7) '>>' operator allows the user to set values for **red**, **green**, and **blue** member variables.

Implementing ColorBlob class:

A **ColorBlob** is created when a number of **Color** objects are combined. Figure 1 shows a square **ColorBlob object** of width 4 and height 4. This forms a 2D array which has 4 rows and 4 columns. Thus, the **ColorBlob** in Figure 1 includes 16 Color objects. '**Red**' color has been assigned to each element. When a **ColorBlob** is created, a particular color (same) is assigned to all elements.

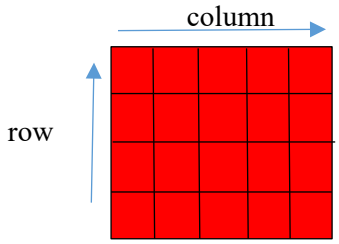

Figure 1: *A square **ColorBlob** where each element is colored red.*

Thus, the width and height of a **ColorBlob** can vary. As the goal is to create color blobs of varying sizes (width and height) and assigning different colors to them, you need to dynamically create two-dimensional arrays of different widths and heights. The member variables of a **ColorBlob** can be defined as follows:

```
int width;
int height;
Color** data;
```

For the **ColorBlob** class, you need to implement **default constructor**, **constructor with parameter**, **copy constructor**, **copy assignment operator**, **destructor**, **move constructor**, and **move assignment operator**. The default width and height of a **ColorBlob** object are 2. So, a default **ColorBlob** object is composed of 4 **Color** objects. A default **ColorBlob** has the same color for each entry. You also need to implement the following **friend functions** for a **ColorBlob** class:

1. friend bool operator==(const ColorBlob&, const ColorBlob&);
2. friend ColorBlob operator+(const ColorBlob &cBlobOne, const ColorBlob &cBlobTwo);
3. friend ColorBlob operator-(const ColorBlob &cBlobOne, const ColorBlob &cBlobTwo);
4. friend ColorBlob operator*(const ColorBlob &cBlobOne, const Color &c);
5. friend bool operator! (const ColorBlob &cBlob);
6. friend ostream& operator<<(ostream&, const ColorBlob&);
7. friend istream& operator>>(istream&, ColorBlob&);

(1) The relational operator "==" takes two **ColorBlob** objects as arguments and if both are of equal height and width, it compares whether the color values for each entry are equal or not. If color values for both **ColorBlob**s are equal, the function returns true, otherwise, it returns false.
(2) The arithmetic operator '+' takes two **ColorBlob** objects. Next, it creates a new **ColorBlob** object from the minimum width and height of the two **ColorBlob** objects. Next, each entry color value of the new **ColorBlob** object is computed as the addition of the color components of the first and second **ColorBlob** objects. As values for red, green, and blue cannot exceed 1.0, so if any of the **red**, **green**, or **blue** components becomes greater than 1.0, it is clamped at 1.0;
(3) The arithmetic operator '-' works as (2) except each color value of the new **ColorBlob** object is the subtraction of the second color value from the first. As colors cannot be negative, if any of the **red**, **green**, or **blue** components becomes negative because of subtraction, that value is clamped at 0;
(4) Each color entry for the new **ColorBlob** is the result of the multiplication of the color entries of a given **ColorBlob** object with the given **Color c**.
(5) Unary operator '!' compares whether all the color entries of a given **ColorBlob** object is black or not. If all entries are not black, it returns true, otherwise, false is returned.
(6) '<<' operator prints all the color entries of a **ColorBlob** object;
(7) '>>' operator allows the user to set different color values for each color entry of a **ColorBlob** object (Figure 2) that was initially created with its default single color.

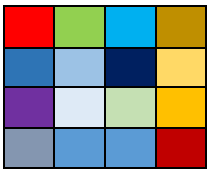

Figure 2: *A square **ColorBlob** with an assortment of colors for its elements.*

Submission:

You need to create two classes: **Color** and **ColorBlob**.

Compile and execute your code with g++ or the Microsoft compiler.

Place your solution in a zipped file named with your last name followed by the first initial of your first name followed by 3 (ex: **YasminS3.zip**) and submit the solution via canvas.

Include at the top of your solution (in comments) your name, what compiler(s) you used.

Your zip should contain the following:

- **Color.h**, **Color.cpp**, **ColorBlob.h**, and **ColorBlob.cpp**.

Attached is a simple main program **colorblob_tester.cpp** which you can use for testing purpose. Below is the output generated by the program when tested with **colorblob_tester.cpp**.

```
syasmin@CSCD110497WP:~/CSCD305/Assignment3$ ./testBlob
cBlob =
[0][0]:(0.5  0.5  0.5)
[0][1]:(0.5  0.5  0.5)
[1][0]:(0.5  0.5  0.5)
[1][1]:(0.5  0.5  0.5)

Now, you can change color entries:
[0][0]:0.3 0.4 0.5
[0][1]:0.6 0.7 0.8
[1][0]:0.9 1.0 1.0
[1][1]:1.0 1.0 1.0
cBlobThree =
[0][0]:(0.8  0.9  1)
[0][1]:(1  1  1)
[1][0]:(1  1  1)
[1][1]:(1  1  1)

cBlobFour =
[0][0]:(0.2  0.1  0)
[0][1]:(0  0  0)
[1][0]:(0  0  0)
[1][1]:(0  0  0)

cBlobFive=
[0][0]:(0.25  0.2  0.225)
[0][1]:(0.25  0.2  0.225)
[1][0]:(0.25  0.2  0.225)
[1][1]:(0.25  0.2  0.225)

It is not black!
cBlobSix=
[0][0]:(0.25  0.2  0.225)
[0][1]:(0.25  0.2  0.225)
[1][0]:(0.25  0.2  0.225)
[1][1]:(0.25  0.2  0.225)

cBlobSeven=
[0][0]:(0.5  0.5  0.5)
[0][1]:(0.5  0.5  0.5)
[1][0]:(0.5  0.5  0.5)
[1][1]:(0.5  0.5  0.5)

syasmin@CSCD110497WP:~/CSCD305/Assignment3$
```

Figure 3: *The output generated by the tester file.*

Submission deadline is **Friday April 25, 11:59 pm**. This assignment weighs **20%** of the course.

Hints:

1. Please review the uploaded example on **Array** to get some idea how **Color** entries (i.e., data) in a **ColorBlob** can be initialized. More constructors will be gradually added to the Array class.
2. Friend functions in **Color** class should be **reused** by friend functions in **ColorBlob** whenever needed!