BBM203: Implementing a Color System with Classes in C++

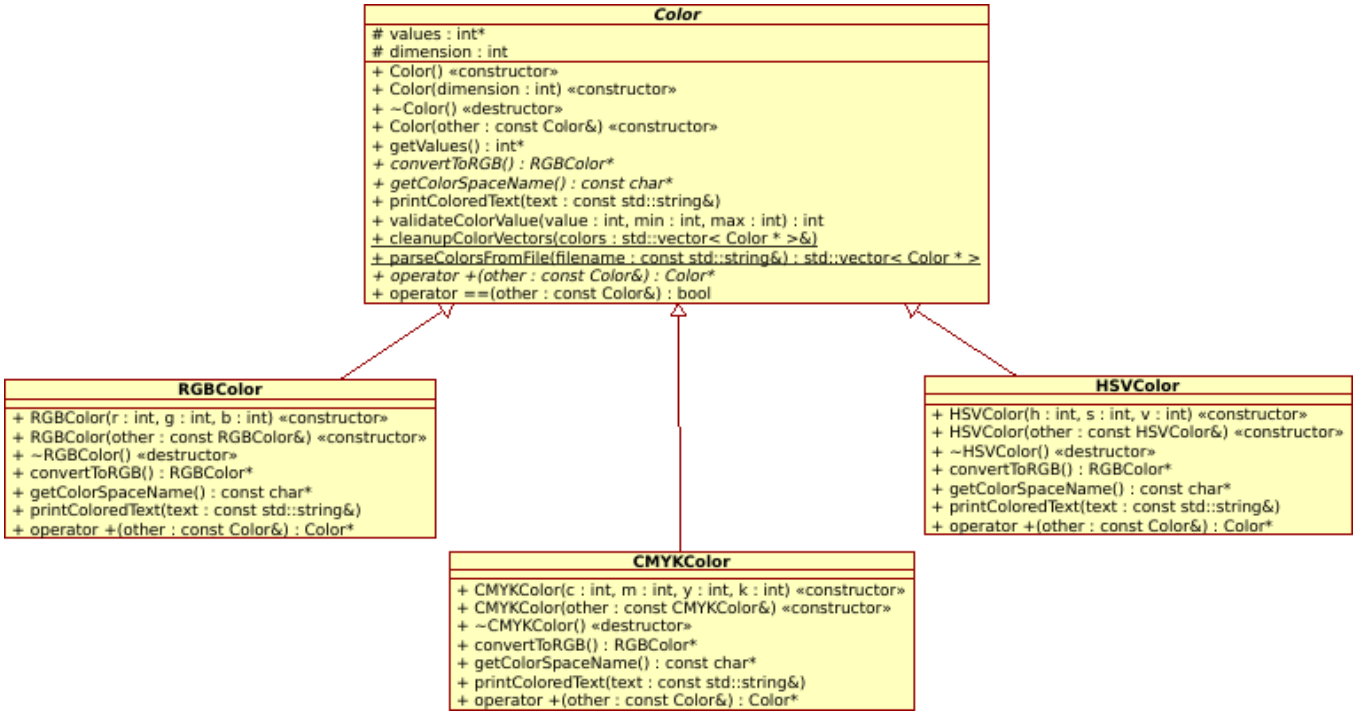

Image 1: Class Diagram of the project

Objective:
This assignment aims to help you understand and practice the following C++ concepts:

- **Classes and Inheritance**
- **Overloading and Overriding**
- **Virtual Functions and Pure Virtual Functions**
- **Dynamic Memory Management**
- **File I/O Operations**
- **Static functions**
- **Forward Declarations**

You will be implementing a system that represents different color spaces (RGB, CMYK, HSV) using a base class called `Color`. Each derived class should implement specific functionalities related to its color space.

Class Descriptions

1. **Base Class: `Color`**

This is an abstract class meant to serve as the base class for all color spaces. It includes common attributes and functionalities that derived classes should extend or override.

- **Protected Attributes:**
  - `int* values;` – Dynamically allocated array to store color values.
  - `int dimension;` – Represents the size of the `values` array.
- **Constructors:**
  - `Color()` – Default constructor. Initializes values as a dynamic array.
  - `Color(int dimension)` – Parameterized constructor to initialize the dimension.
  - `Color(const Color& other)` – Copy constructor for deep copying dynamic memory.
- **Destructor:**
  - `virtual ~Color();` – Virtual destructor for proper cleanup of derived objects.
- **Member Functions (Virtual):**
  - `virtual RGBColor* convertToRGB() const = 0;`
    Pure virtual function to be overridden by derived classes to convert a color to RGB format.
  - `virtual const char* getColorSpaceName() const = 0;`
    Returns a string representing the color space (e.g., "RGB", "CMYK").
  - `virtual void printColoredText(const std::string& text) const;`
    Prints the given text in the terminal using the current color.
  - `int* getValues() const;`
    Returns a pointer to the `values` array.
  - `int validateColorValue(int value, int min, int max) const;`
    Checks if the color value is within the specified range. Adjust if out of bounds.

- **Static Member Functions:**

    - `static std::vector<Color*> parseColorsFromFile(const std::string& filename);`
      Reads color definitions from a file and creates objects of the appropriate type.
    - `static void cleanupColorVectors(std::vector<Color*>& colors);`
      A utility function to clean up dynamically allocated memory in a vector of `Color*`.

- **Overloaded Operators (Virtual):**

    - `virtual Color* operator+(const Color& other) const = 0;`
      Overloads the `+` operator for color blending between different color spaces.
    - `bool operator==(const Color& other) const;`
      Compares two colors based on their `values`.

2. Derived Class: `RGBColor`

- Inherits from `Color`.
- Constructor initializes RGB values (0-255).
- Overrides `convertToRGB`, `getColorSpaceName`, and `operator+`.
- Implements methods specific to RGB color space calculations.

3. Derived Class: `CMYKColor`

- Inherits from `Color`.
- Constructor initializes CMYK values (0-100).
- Implements conversions to RGB using CMYK color formulas.
- Overrides `convertToRGB`, `getColorSpaceName`, and `operator+`.

4. Derived Class: `HSVColor`

- Inherits from `Color`.
- Constructor initializes HSV values (`H` 0-360, `S` and `V` 0-100).
- Implements conversions to RGB using HSV color formulas.
- Overrides `convertToRGB`, `getColorSpaceName`, and `operator+`.

Implementation Requirements

Your task is to:

1. **Create the Classes and Constructors**
   Implement the constructors for all classes as described in the class diagram. Ensure that the constructors handle memory allocation properly.

2. **Implement Member Functions**
   Follow the descriptions for each function and implement them in the respective classes.

3. **Handle Memory Safely**
   Make sure to clean up any dynamically allocated memory in your destructors. Use deep copying in your copy constructor and handle the cleanup correctly in `cleanupColorVectors`.

4. **Overriding and Overloading Functions**
   Implement function overriding to define different behaviors for derived classes. Overload the `+` operator to perform color blending in different color spaces.

5. **File I/O Operations**
   Implement the `parseColorsFromFile` function to read a file with color definitions. The file will have a format such as:

   RGB, 255, 0, 0

   CMYK, 0, 100, 100, 0

   HSV, 0, 100, 100

Additional Information:

- **Do not directly modify base class members in derived classes without calling base constructors.**
- **Implement all pure virtual functions (`= 0`) in derived classes.**

Example File Format

Your input file should contain one color per line with its corresponding values:

RGB, 255, 0, 0

CMYK, 0, 100, 100, 0

HSV, 0, 100, 100

Color values must be within the specified ranges for each color space:

- RGB: 0-255
- CMYK: 0-100
- H (HSV): 0-360
- S, V (HSV): 0-100

Submission Requirements

- Implement all classes and functions following the provided UML Class Diagram in the Image 1.
- Submit b<student_number>.zip with all these in one folder:
    - Color.h
    - RGBColor.h
    - HSVColor.h
    - CMYKColor.h
    - Color.cpp
    - RGBColor.cpp
    - HSVColor.cpp
    - CMYKColor.cpp