CS 3005: Programming in C++

Color Table Class

A color table is an array of colors. It is useful for translating a single number in a range into a color, reliably and repeatedly.

We will use a color table to translate Julia set escape values into colors to create images.

Assignment

Create a class named `Color` and one named `ColorTable`. These classes must have the methods listed below, with the specified behaviors.

Also create or use an existing program that configures a color table, calculates a Julia set, and uses the color table and Julia set to create a PPM and saves the PPM to a file.

**Color** Class Programming Requirements

Your `Color` class must store the following data.

- The integer representation of red, green and blue channels of a color.

Your `Color` class must have the following methods.

**Constructors and Getters**

- `Color();` Sets all color channels to value 0.
- `Color( const int& red, const int& green, const int& blue );` Sets the color channels to the values provided here. No range checking is applied.
- `int getRed() const;` Returns the value of the red channel.
- `int getGreen() const;` Returns the value of the green channel.
- `int getBlue() const;` Returns the value of the blue channel.
- `int getChannel( const int& channel ) const;` Returns the value of the channelth channel. 0 == red, 1 == green, 2 == blue. Returns -1 if the channel is out of range.

**Setters**

- `void setRed( const int& value );` Changes the red channel to `value`. If `value` is less than 0, do not make any changes.
- `void setGreen( const int& value );` Changes the green channel to `value`. If `value` is less than 0, do not make any changes.
- `void setBlue( const int& value );` Changes the blue channel to `value`. If `value` is less than 0, do not make any changes.
- `void setChannel( const int& channel, const int& value );` Changes the channelth channel to `value`. If `value` is less than 0, do not make any changes. 0 == red, 1 == green, 2 == blue. Does not make changes if `channel` is out of range.

**Other Methods**

- `void invert( const int& max_color_value );` Inverts the red, green and blue channels, using `max_color_value`. If `max_color_value` is less than any of the current color channels (red, green or blue), then make no changes. The inversion is completed by subtracting the current value from `max_color_value`. For example: red = max_color_value - red. This only makes sense if red is >= max_color_value. That's why we make no changes if any channel (red, green or blue) is larger than max_color_value.
- `bool operator==( const Color& rhs ) const;` Returns true if `*this` and `rhs` have the same color values. Otherwise, returns false.

**ColorTable** Programming Requirements

Your `ColorTable` class must store the following data.

- A linear collection of `Colors`.

Your `ColorTable` class must have the following methods.
**Constructor and Getters**

- **ColorTable(const size_t& num_color);** Sizes the **Color** collection to **num_color** items.
- **size_t getNumberOfColors();** Returns the number of **Color**s stored.

**Setters**

- **void setNumberOfColors(const size_t& num_color);** Resizes the collection to hold **num_color** items. Previous **Color** contents may or may not be preserved.

**Operators**

- **const Color& operator[](const int& i) const;** Returns the **i**th **Color** in the collection. If **i** is out of range, returns a **static** memory **Color** object with all three channels set to **-1**. For example:

  ```
  { static Color ec(-1, -1, -1); static Color c(-1, -1, -1); c = ec; return c; }
  ```

- **Color& operator[](const int& i);** Returns the **i**th **Color** in the collection. If **i** is out of range, returns a **static** memory **Color** object with all three channels set to **-1**.

**Other Methods**

- **void setRandomColor(const int& max_color_value, const size_t& position);** Assigns the **position**th color random values for all three channels. The random values are between 0 and **max_color_value**, inclusive. If **position** is out of range, no change is made. If **max_color_value** is less than 0, no change is made.
- **void insertGradient(const Color& color1, const Color& color2, const size_t& position1, const size_t& position2);** Change the colors from **position1** to **position2**, inclusive, to be gradients from **color1** to **color2**. If **position1** is not less than **position2**, no change is made. If either position is out of range, no change is made.

**JuliaSet Programming Requirements**

Add the following method to the **JuliaSet** class:

- **void setPPM(PPM& ppm, const ColorTable& colors);** Uses the currently calculated escape values to configure an image in the PPM object. Sets the width and height of the image to match the width and height of the Julia set. Sets the maximum color value to the maximum color value of any color in the color table. For each pixel in the PPM object, sets the color based on the escape value for the pixel. If the color table does not have at least 3 colors, make no changes to the PPM object. Use the color table item at the last position for any pixels with a maximum escape value. Use the color table item at the next to last position for any pixels with a 0 escape value. For all other escape values, use (the escape value) mod (the size of the color table - 2) as the index into the color table.

**Program Programming Requirements**

This program must be written in **julia.cpp**, and the final program must be named **julia**.

The program must:

- Use a menu system.
- Menu loops until user chooses quit
- Options in menu
  - w: set width
  - h: set height
  - x: set minimum x
  - X: set maximum x
  - y: set minimum y
  - Y: set maximum y
  - a: set parameter a
  - b: set parameter b
  - e: set maximum escape value
  - c: calculate all escape values
  - p: assign colors to the PPM
  - s: save the PPM to file
  - S: show Julia set meta data
  - T: resize color table
  - A: assign a color in the color table
  - r: assign a color randomly in the color table
Example Input

This [sample input](#) produces these three images. Note that there is one random color in the color table, so your images will have one color different on the second image.

- [color_example1.ppm](#)
- [color_example2.ppm](#)
- [color_example3.ppm](#)

This [Makefile](#) and [julia.cpp](#) may be used.

Additional Documentation

- Consider using the `std::vector` class template.
- [Color Gradient on Wikipedia](#) (Only marginally useful.)
- [Color Gradient Discussion](#)

Show Off Your Work

To receive credit for this assignment, you must complete the unit tests available in CodeGrinder, and you must upload the source code (.cpp and .h files) and the Makefile to the Canvas submission system.

Additionally, the program must build, run and give correct output.