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 \texttt{Color} and one named \texttt{ColorTable}. These classes must have the methods listed below, with the specified behaviors.

Also create a main program that creates a color table, calculates a Julia set, then uses the color table and Julia set to create a PPM and saves the PPM to a file.

\textbf{Color Programming Requirements}

Your \texttt{Color} class must store the following data.

\begin{itemize}
\item The integer representation of red, green and blue channels of a color.
\end{itemize}

Your \texttt{Color} class must have the following methods.

\textbf{Constructors and Getters}

\begin{itemize}
\item \texttt{Color()}; Sets all color channels to value 0.
\item \texttt{Color(const int& red, const int& green, const int& blue)}; Sets the color channels to the values provided here. No range checking is applied.
\item \texttt{int getRed() const}; Returns the value of the red channel.
\item \texttt{int getGreen() const}; Returns the value of the green channel.
\item \texttt{int getBlue() const}; Returns the value of the blue channel.
\item \texttt{int getChannel(const int& channel) const}; Returns the value of the \texttt{channel}th channel. \(0 == \text{red}, 1 == \text{green}, 2 == \text{blue}\). Returns -1 if the channel is out of range.
\end{itemize}

\textbf{Setters}

\begin{itemize}
\item \texttt{void setRed(const int& value)}; Changes the red channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes.
\item \texttt{void setGreen(const int& value)}; Changes the green channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes.
\item \texttt{void setBlue(const int& value)}; Changes the blue channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes.
\item \texttt{void setChannel(const int& channel, const int& value)}; Changes the \texttt{channel}th channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes. \(0 == \text{red}, 1 == \text{green}, 2 == \text{blue}\). Does not make changes if \texttt{channel} is out of range.
\end{itemize}

\textbf{Other Methods}

\begin{itemize}
\item \texttt{void invert(const int& max_color_value)}; Inverts the red, green and blue channels, using \texttt{max_color_value}. If \texttt{max_color_value} is less than any of the current color channels (red, green or blue), then make no changes.
\item \texttt{bool operator==(const Color& rhs)} const; Returns \texttt{true} if \texttt{*this} and \texttt{rhs} have the same color values. Otherwise, returns \texttt{false}.
\end{itemize}

\textbf{ColorTable Programming Requirements}

Your \texttt{ColorTable} class must store the following data.

\begin{itemize}
\item A linear collection of \texttt{Colors}.
\end{itemize}

Your \texttt{ColorTable} class must have the following methods.

\textbf{Constructor and Getters}

\begin{itemize}
\item \texttt{ColorTable(const size_t& num_color)}; Sizes the \texttt{Color} collection to \texttt{num_color} items.
\end{itemize}
• `size_t getNumberOfColors() const;` 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`.
• `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.

**Program Programming Requirements**

The program must:

• Prompt the user to create a color table. See the example below.
• Prompt the user for the size of the image to create, the dimensions of the plane to calculate, the values of `a` and `b`, and the name of the file to store the PPM in. The order must be the same as shown in the examples.
• Calculate a Julia set using the values given by the user, and a maximum escape count the same as the color table size + 1.
• Create a color PPM using the escape counts from the Julia set as indexes into the color table.
• Save the PPM object to the file specified by the user.
• Use 255 as the maximum color value for all Colors in the table, and for the PPM object.

**Example Execution 1**

```
$ ./main
Color Table options
s) resize color table
  a) assign a color in the color table
  r) assign a color randomly in the color table
  g) assign a color gradient in the color table
q) quit color table operations
Choice: s
Number of colors: 100

Color Table options
s) resize color table
  a) assign a color in the color table
  r) assign a color randomly in the color table
  g) assign a color gradient in the color table
q) quit color table operations
Choice: a
Which position (0-99) 0
Red: 255
Green: 255
Blue: 255
```

```
Color Table options
s) resize color table
  a) assign a color in the color table
  r) assign a color randomly in the color table
  g) assign a color gradient in the color table
q) quit color table operations
Choice: a
Which position (0-99) 99
Red: 255
```
Green: 0
Blue: 255

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: r
Which position (0-99) 1

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: g
First position (0-99) 2
First Red: 0
First Green: 255
First Blue: 0
Second position (0-99) 49
Second Red: 0
Second Green: 0
Second Blue: 0

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: g
First position (0-99) 50
First Red: 0
First Green: 0
First Blue: 0
Second position (0-99) 98
Second Red: 0
Second Green: 255
Second Blue: 0

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: q

Pixel Width 512
Pixel Height 512
Min x -2.0
Max x 2.0
Min y -2.0
Max y 2.0
A 0.285
B 0.01
Filename julia-green.ppm

Produces this image.

Example Execution 2

```
$ ./main
```
q) quit color table operations
Choice: s
Number of colors: 1024

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: a
Which position (0-1023) 0
Red: 0
Green: 0
Blue: 0

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: a
Which position (0-1023) 1
Red: 255
Green: 255
Blue: 255

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: r
Which position (0-1023) 1

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: r
Which position (0-1023) 2

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: r
Which position (0-1023) 3

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: r
Which position (0-1023) 4

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: g
First position (0-1023) 5
First Red: 0
First Green: 0
First Blue: 255
Second position (0-1023) 105
Second Red: 255
Second Green: 0
Second Blue: 0

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: g
First position (0-1023) 106
First Red: 0
First Green: 255
First Blue: 0
Second position (0-1023) 1022
Second Red: 255
Second Green: 0
Second Blue: 255

Color Table options
s) resize color table
a) assign a color in the color table
r) assign a color randomly in the color table
g) assign a color gradient in the color table
q) quit color table operations
Choice: q

Pixel Width 768
Pixel Height 768
Min X -0.2
Max X 0.0
Min Y 0.2
Max Y 0.4
A -0.835
B -0.2321
Filename julia-color.ppm

Produces [this image](https://example.com).

**Additional Documentation**

- Consider using the `std::vector` class template.

**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.