CS 3005: Programming in C++

Color Table Class

Introduction

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

We will use a color table to translate Julia set escape counts or other values in the number grid into colors to create colorful images.

Assignment

In this assignment you will create a class to store a single RGB Color, and a class to store a vector of Color objects in a ColorTable. You will also modify your project to use these classes to provide users the ability to configure and use colors of their own choosing when creating PPM images from fractal escape values.

The ppm_menu program needs to add a few new commands.

The new commands required are:

- grid-apply-color-table) Use the grid values to set colors in the output image using the color table.
- set-color-table-size) Change the number of slots in the color table.
- set-color) Set the RGB values for one slot in the color table.
- set-random-color) Randomly set the RGB values for one slot in the color table.
- set-color-gradient) Smoothly set the RGB values for a range of slots in the color table.

Programming Requirements

Create ColorTable.{h,cpp}

These files will be used to declare and define both the Color and the ColorTable classes.

Color class

Data Members:

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

Methods:

- 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 \texttt{channel}th channel. 0 == red, 1 == green, 2 == blue. Returns -1 if the channel is out of range.
- void setRed(const int& value); Changes the red channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes.
- void setGreen(const int& value); Changes the green channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes.
- void setBlue(const int& value); Changes the blue channel to \texttt{value}. If \texttt{value} is less than 0, do not make any changes.
- 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 == red, 1 == green, 2 == blue. Does not make changes if \texttt{channel} is out of range.
- 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. The inversion is completed by subtracting the current value from \texttt{max_color_value}. For example: \texttt{red = max_color_value - red}. This only makes sense if red is >= \texttt{max_color_value}. That’s why we make no changes if any channel (red, green or blue) is larger than \texttt{max_color_value}. 
bool operator==(const Color& rhs) const; Returns true if *this and rhs have the same color values. Otherwise, returns false.

Additional support functions for the Color class:

std::ostream& operator<<(std::ostream& os, const Color& color); Displays the color to os in the following format: “red:green:blue”. For example, if the color has red = 13, green = 2 and blue = 45, then the output would be “13:2:45”.

**ColorTable class**

Data Members:

- A linear collection of Colors. (Think std::vector.)

Methods:

- ColorTable(const size_t& num_color); Sizes the Color collection to num_color items.
- size_t getNumberOfColors() const; Returns the number of Color's stored.
- void setNumberOfColors(const size_t& num_color); Resizes the collection to hold num_color items. Previous Color contents may or may not be preserved.
- 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. See an example below.
- 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.
- 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. This method should NOT use std::srand(). Add std::srand(std::time(0)); to main().
- 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.
- int getMaxChannelValue(); Finds the largest value of any red, green, or blue value in any color in the table.

Creating a static Color object to return in error cases.

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

**Update NumberGrid.{h,cpp}**

You will add a method to set a PPM object from the grid numbers, using a ColorTable instead of the built-in table with 8 colors. Do not remove the previous method. Just add this one.

Add this method:

- void setPPM(PPM& ppm, const ColorTable& colors) const; Uses the currently stored grid numbers to configure an image in the PPM object. Sets the width and height of the image to match the width and height of the grid. Sets the maximum color value to the maximum color value of any color in the color table (getMaxChannelValue()). For each pixel in the PPM object, sets the color based on the grid number for the pixel. If the color table does not have at least 3 colors, make no changes to the PPM object. The grid number will be used as the index into the color table, with a few special cases: if the grid number is the maximum number, use the color table item with the highest index number; if the grid number is 0, use the color table item with index 0; otherwise use grid number modulus color table size as the index into the table.

**Updates to ActionData.{h,cpp}**

Additional Data Members:

- ColorTable A color table object.
Updated Methods:

- `ActionData(std::istream& is, std::ostream& os)` Needs to initialize the color table to have 16 color table slots. Also should fill the color table with a gradient from \(0,255,0\) to \(255,0,255\).

Additional Methods:

- `ColorTable& getTable();` Returns the color table data member.

**Update** [image_menu.{h,cpp}]

Add the following function declarations to the header file and implementations to the `{cpp} file.

- `void applyGridColorTable(ActionData& action_data);` Uses the new `setPPM` method of the grid to set the output image PPM using color table. Note this is not a replacement for `applyGrid`, this is in addition to that function.
- `void setColorTableSize(ActionData& action_data);` Asks the user for the “Size? “, then applies it to the color table.
- `void setColor(ActionData& action_data);` Asks the user for “Position? “, “Red? “, “Green? “, and “Blue? “. Then uses them to set a color at the specified position in the color table.
- `void setRandomColor(ActionData& action_data);` Asks the user for “Position? “, then uses `setRandomColor()` to set a random color at that position in the color table. Use \(255\) for the maximum color value.

**Update Functions in** [image_menu.{h,cpp}]

- `void configureMenu( MenuData& menu_data );` add the new actions with the names and descriptions listed below.

**Table of New Commands**

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid-apply-color-table</td>
<td>applyGridColorTable</td>
<td>Use the grid values to set colors in the output image using the color table.</td>
</tr>
<tr>
<td>set-color-table-size</td>
<td>setColorTableSize</td>
<td>Change the number of slots in the color table.</td>
</tr>
<tr>
<td>set-color</td>
<td>setColor</td>
<td>Set the RGB values for one slot in the color table.</td>
</tr>
<tr>
<td>set-random-color</td>
<td>setRandomColor</td>
<td>Randomly set the RGB values for one slot in the color table.</td>
</tr>
<tr>
<td>set-color-gradient</td>
<td>setColorGradient</td>
<td>Smoothly set the RGB values for a range of slots in the color table.</td>
</tr>
</tbody>
</table>

**Update** [ppm_menu.cpp]

- `int main();` Add `std::srand(std::time(0));`

**Build Requirements**

- `make` must build the complete programs named `ppm_menu`, `image_file`, `ascii_image`, and `questions_3`.
- `make` must have the `clean` target.

**Additional Documentation**

- [C++ Reference](#)
- [Examples from class](#)
- [Color Gradient Discussion](#)
- [Color Gradient on Wikipedia](#) (Only marginally useful.)
- [Hints on choosing color schemes](#)
- [Paletton color selection site](#)

**Show Off Your Work**

To receive credit for this assignment, you must
complete the unit tests available in CodeGrinder
use git to add, commit and push your solution to your repository for this class.

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

**Extra Challenges (Not Required)**

- Create additional methods in the `ColorTable` class that allow for easy insertion of interesting color patterns. Add the ability to use them from the `imageMenu()`. For example, can you implement a system to use a color and its complement to make a gradient?
- Create a method of `NumberGrid` to find the maximum number stored. Add the ability to set the number of colors in the color table to match this number from `imageMenu()`.
- Try other ways to modify the color system to make good color systems. For example, can you make an HSV based color system that would make setting the color more convenient for designers?