CS 3005: Programming in C++

Color Table

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 grid numbers into colors to create colorful images.

Assignment

The `ppm_menu` program needs to add a few new commands to allow the user to configure the color table, and to use it.

The new commands required are:

- `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.`
- `grid-apply-color-table) Use the grid values to set colors in the output image using the color table.`

This functionality will be implemented by creating a class to store a single RGB `Color`, and a `ColorTable`class to store a vector of `Color` objects.

Programming Requirements

The following files must be updated or created and stored in the `src` directory of your repository.

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 `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 `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 `channel`th 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.
- `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`.

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 int& num_color );` Sizes the `Color` collection to `num_color` items.
- `int getNumberOfColors( ) const;` Returns the number of `Color`s stored.
- `void setNumberOfColors( const int& 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 int& 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()` of `ppm_menu.cpp`.
- `double gradientSlope(const double y1, const double y2, const double x1, const double x2) const;` Calculates a slope from point 1 to point 2, using "rise-over-run" calculation. Be sure to use floating point division operation.
- `double gradientValue(const double y1, const double x1, const double slope, const double x) const;` Calculate the y-value along the gradient from point (x1,y1) to the point at position x.
- `void insertGradient( const Color& color1, const Color& color2, const int& position1, const int& 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. Should use the `gradientSlope()` and `gradientValue()` methods.
- `int getMaxChannelValue( ) const;` Finds the largest value of any red, greeen, or blue value in any color in the table.

Creating a `static Color` object to return in error cases.

```
{
    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 2 colors, make no changes to the PPM object. The grid number will be used as the index into the color table, with a special case: if the grid number is the maximum grid number, use the color table item with the highest index number; 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` and `image_drawing.cpp`

The follow functions must be declared and implemented.

- `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.
- `void setColorGradient(ActionData& action_data);` Asks the user for "First position? ", "First red? ", "First green? ", "First blue? ", "Second position? ", "Second red? ", "Second green? " and "Second blue? ". The uses them to `insertGradient()` in the color table.
- `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.

Update `controllers.cpp`

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

Table of New Commands

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

Update `ppm_menu.cpp`

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

Update `Makefile`

The following commands should work correctly.

- `make hello` - builds the hello program
- `make questions_3` - builds the questions_3 program
- `make ascii_image` - builds the ascii_image program
- `make image_file` - builds the image_file program
- `make ppm_menu` - builds the image_file program
- `make all` - builds all programs
- `make` - builds all programs (same as `make all`)
- `make clean` - removes all .o files, and all executable programs

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

- use git to add, commit and push your solution to your repository for this class.
- successfully pass all unit tests and acceptance tests

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?