Lab 7 – Numeric Conversion

Overview

In this assignment you are going to read common color names and their corresponding numeric values from a group of files. One small issue: the numbers are in the wrong format. They are stored in integers, while typically color values are represented in one of two ways--either in hexadecimal form, or as their 3 separate color channels. For example, the color red might be represented like this:

0xFF0000 as hexadecimal

Red: 255, Green: 0, Blue: 0 as unsigned characters, or

Red: 1.0f, Green: 0.0f, Blue: 0.0f as floats

The integer representation of that color would be 16711680—this is not typically used, as it's not terribly helpful. In this assignment, you are going to convert this not-so-helpful integer into a helpful hex value and RGB value. For more general information on color codes:

https://htmlcolorcodes.com/
https://www.w3schools.com/colors/colors_names.asp

Description

The three files to load are called colors1.txt, colors2.txt, and colors3.txt. Each file contains a list of colors with their name and integer representation of the color. You are to write a small program that loads one or more of these files, converts the values to hex/RGB values, and **sorts** the list of values by the color name.

Storing multiple values in a single variable is a common thing in code. You may do this conserve memory, or to easily pass multiple values around without creating new classes to store them. Very commonly this will be for small values, such as characters or shorts, and they are stored in larger integer variables.

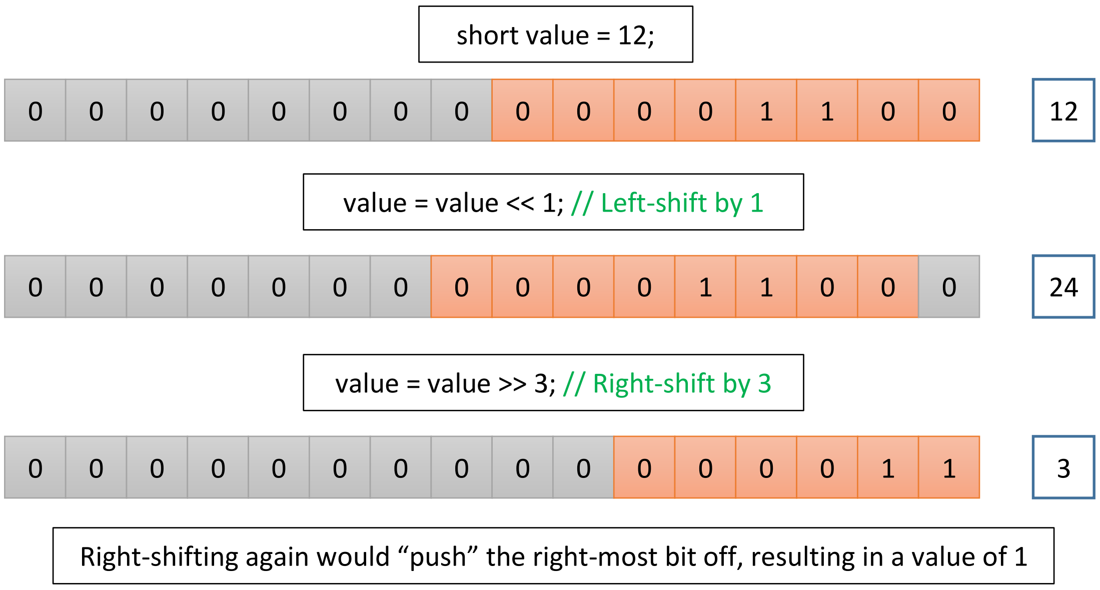
The way to store/retrieve these values is by **bit-shifting**.

Imagine a single byte (i.e. a signed or unsigned character), made up of 8 bits:

The number 12 in binary form: 0 0 0 0 1 1 0 0
The number 255 in binary: 1 1 1 1 1 1 1 1

If you wanted to store these two separate values in one 2-byte short (12 first, then 255), the bytes for that short would look like: 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1. Its decimal value would be 3,327 which, has no obvious connection to either of the two values we're storing. All of memory is like this, but fortunately for programmers we can deal with memory one variable at a time.

If we took a short, and initialized it to 12, its bytes would be 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0. Look at all that empty space on the left! So much room, you can store all kinds of things in there! (All kinds of things, as long as those things are bits.) If you want to store the 12 "on the left" you would left-shift the value. Each time you shift a value, its bits move over as many bits as you specify.



```
short value = 12;
```

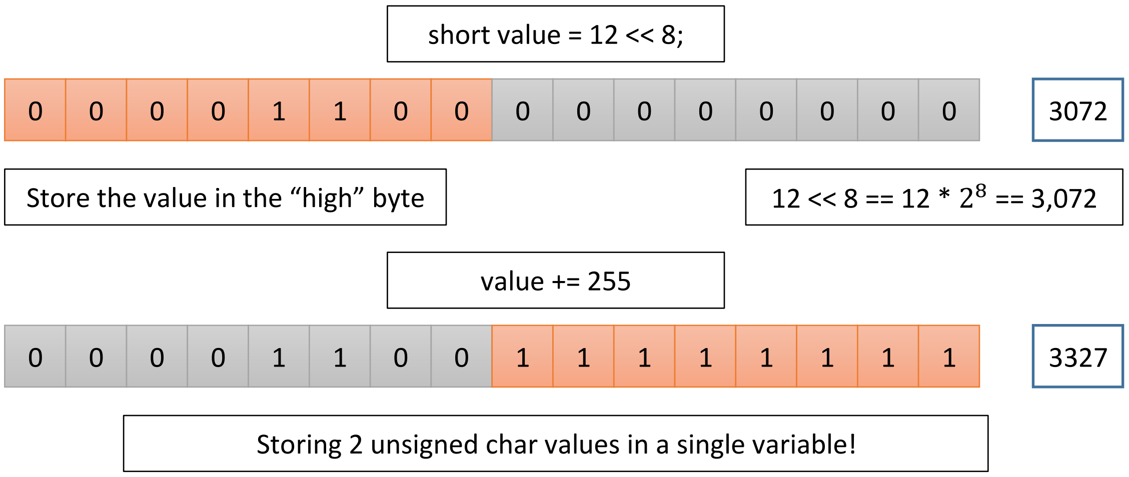
```
value = value << 1; // Left-shift by 1
```

```
value = value >> 3; // Right-shift by 3
```

Right-shifting again would "push" the right-most bit off, resulting in a value of 1

One thing you might notice is that bit-shifting multiplies or divides the value—left-shifting multiplies, while right-shifting divides. The amount of the modification is 2 to power of the number of bits by which you shifted. So left-shifting by 3 multiplies by 2^3, while right-shifting by 2 would divide by 2^2.

If you wanted to store the value of 12 in the "high byte" you would need to move the value over one byte, or 8 bits.



```
short value = 12 << 8;
```

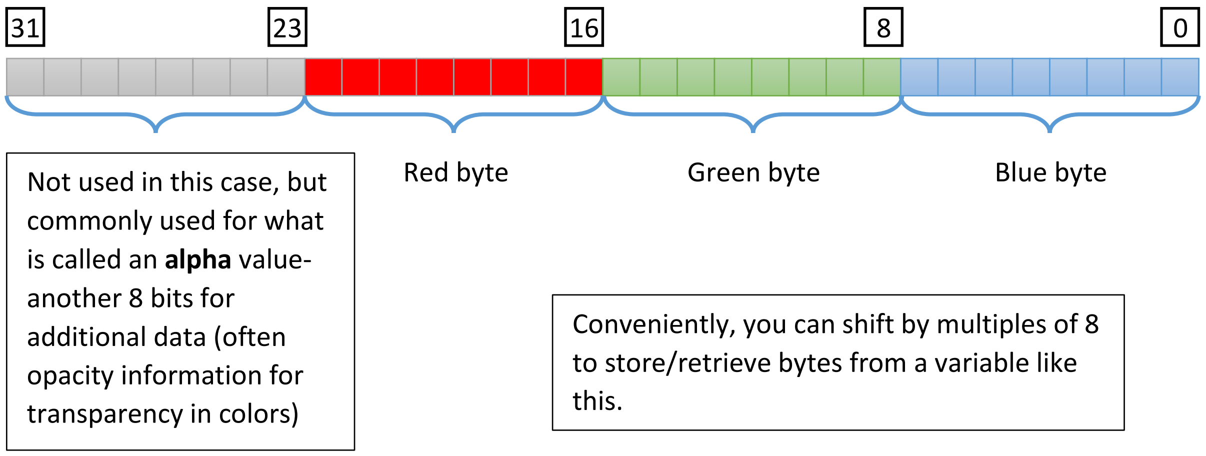
Store the value in the "high" byte

$12 << 8 == 12 * 2^8 == 3{,}072$

```
value += 255
```

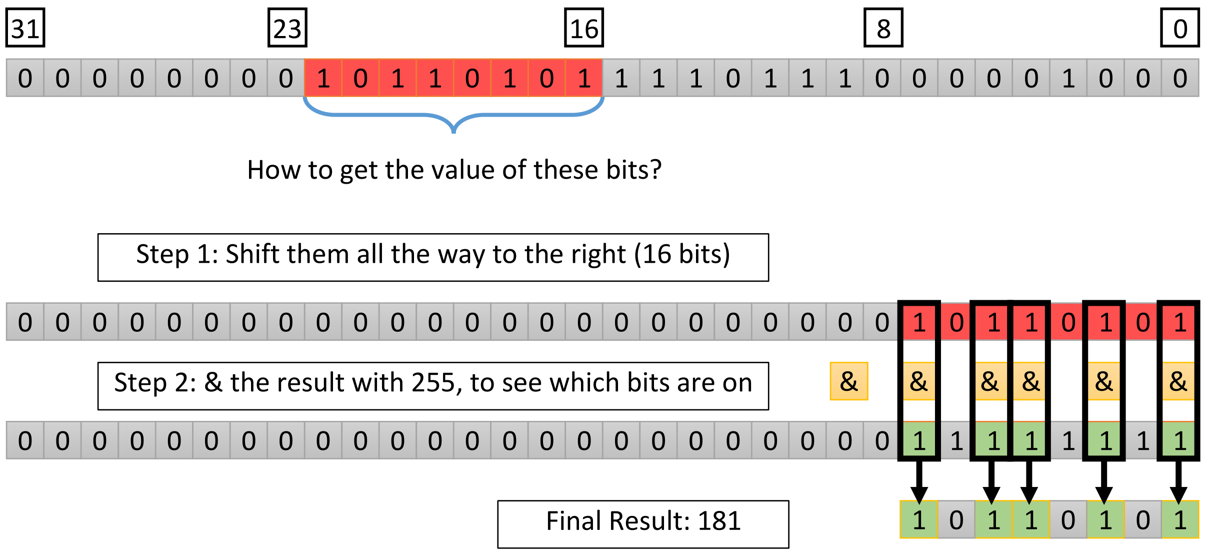
Storing 2 unsigned char values in a single variable!

For this assignment, you will be employing this concept to retrieve 3 unsigned char values from a single integer value. The integer is a 32-bit variable, and you will be retrieving values from bits 0-7 (the green value), bits 8-15 (the blue value), and bits 16-23 (the red value). Visually this would look like the following:



31
23
16
8
0
Not used in this case, but commonly used for what is called an **alpha** value- another 8 bits for additional data (often opacity information for transparency in colors)
Red byte
Green byte
Blue byte
Conveniently, you can shift by multiples of 8 to store/retrieve bytes from a variable like this.

In addition to storing values, you will need to retrieve those byte-values from the variable. This can be done by shifting and comparing to some known value, using the bitwise & operator. The & operator will compare two values, and every bit that is turned on (set to 1) in BOTH values will be present in the final result. For example:



Retrieving the green value would be a similar process, by shifting the original value 8 bits, and the blue value wouldn't need to be shifted at all before the & comparison. After you've shifted and ANDed, you store the value in an unsigned char, and that's it! If you wanted to put the value back in, you could start at zero, and then add the red value left-shifted by 16 bits, the green value left-shifted 8 bits, and then the blue value. If you were using an alpha value, that would be shifted by 24 bits.

Hexadecimal Conversion

After converting your colors to RGB, you will have to store it in a string representing the hexadecimal equivalent. Color values are often represented as hexadecimal numbers, with 2 letters each for the red, green, and blue values. Color values in character form range from 0-255, which can be stored in two hexadecimal digits, 0-FF. The color green would be 0x00FF00, blue would be 0x0000FF, a dark purple color with a value of 93, 0, 106 would be #5D006A.

Hexadecimal is base 16, which means each digit has a value from 0-15, or 0-9, then A is 10, B is 11, C, D, E, and F is 15. The first digit contributes its value to the total value of the number, the second digit contributes 16^1 times the value of the digit to the total of the number, and so on. For example, a value of F3 is $(15 * 16^1) + (3 * 16^0)$, or 243.

Color Class

The Color class you will write for this assignment is pretty simple. You will need to store the **name** and **hex value** of the color as **strings**, and the **RGB values** as **unsigned characters**. You should have the following functions in your class. Any other supporting functions/variables you want to create are up to you.

```
class Color
{
public:
    // Given an integer value, convert it to RGB and Hex values
    void SetValue(int value);
    void SetName(const char *name);

    // Accessors
    unsigned char GetR() const;
    unsigned char GetG() const;
    unsigned char GetB() const;
    string GetHexValue() const;
    string GetName() const;
```