CSC 112 Lab 9
The `Pixel` Class
Spring 2015

Due
3:00pm Friday, 4/10

1 The `Pixel` Class

A pixel is the the smallest unit of a picture that can be controlled. The color of a pixel can be represented using the three component intensities red, green, and blue, which is referred to as the RGB model. For this lab assignment, you will design and implement a class named `Pixel` that stores three `unsigned char` values for the component colors red, green, and blue. Therefore a value for each component color is between 0 and 255. The class must have the following constructors, member functions, and overloads.

- Constructor that will accept three **integer** values for the component colors red, green, and blue. Use default values of 255 for each color. Error checking is necessary since component color values must be within the range of 0 to 255. If a value passed to the constructor is less than 0 then assign the color component 0. Similarly, if a value is greater than 255 then assign the color component 255. The process of limiting a value once it exceeds a threshold is often called *clipping*.
- Accessor functions `red()`, `green()`, and `blue()` that return the corresponding component color value.
- Functions `setRed(int color)`, `setGreen(int color)`, and `setBlue(int color)` set the corresponding component color. You may need to clip the values as described earlier. The functions should return `true` if the value was in range, `false` if clipping was required.

```
Pixel pixel(207, 181, 59);     // old-gold color, not the cigarette...
cout << pixel.red() << '\n';   // prints the red component, which is 207
pixel.setGreen(270);           // value too large, assign green the value 255
```

- `grayscale()` returns the grayscale value of the `Pixel`. The `Pixel` class stores the color using the RGB model. The grayscale version of the color is a single `unsigned char` representing different shades of gray, ranging from white to black. Given a RGB model, the conversion equation is

$$grayscale = 0.2989 \times red + 0.5870 \times green + 0.1140 \times blue$$

- `operator==` return true if the corresponding component colors of the `Pixel` object are equivalent.
- `operator!=` return true if the corresponding component colors are **not** equivalent.
- `operator+` return a `Pixel` object representing the sum of two `Pixel` objects. This overload will add the corresponding color values of the two `Pixel` objects. *Clipping?* Yesssssssss....
- `operator<<` stream insertion operator **neatly** outputs in **color** using the format `[1.0, 2.0, 3.0]`

2 How Do I Test?

Testing will be done using a *tester* program which exercises functionality of the class. The example tester program `driver.cpp` is available at the course web-site, however you will write your own in the future.

3 Programming Points

You **must** adhere to all of the following points to receive credit for this program.

- Turn-in (print-outs and electronically) the files for this program.
- The program must be broken into the following 3 files
  1. `pixel.h` contains the class declaration
  2. `pixel.cpp` contains the member function definitions
  3. `makefile` is the `makefile` for the `Pixel` class and `driver` program, must be commented and have `make clean` option.