Programming Assignment 2:
Color Matcher

Due Thursday, September 14th at 11:59 pm

1. Overview

The purpose of this assignment is to give you experience with creating a larger program, performing mathematical operations using arithmetic operators and functions from the **math** library, use of the **if** and **if-else** constructs, and implementing the algorithm to find a minimum value.

For this assignment, you will start with the skeleton code seen in p2_skeleton.c, found in the p2_dist.zip file distributed with this assignment. As you read through the file you will notice:

- The comment header. You should fill this out with your information and should include a similar header on all your other program submissions for this course.

- A block comment describing the purpose of the program and how to calculate the distance. You'll want to include some of this info in your own header and read it to understand how to calculate the Euclidean distance between two colors – the square root of the sum of the squares of the distances between the individual red, green, and blue values of the two colors. You can delete this comment block before you submit your program.

$$distance = \sqrt{square(red2 - red1) + square(green2 - green1) + square(blue2 - blue1)}$$

- Inside the main method, you'll see that I've defined a palette of 8 colors. Each color has a name and values for red, blue and green. The names are accurate for the RGB values provided.

- Some of you are new to programming, so I've provided a sturdy scaffold on which you can build your program – a series of comment blocks with "TODO" lists. Follow the instructions in the TODO blocks to build your program.

- The only include file in the skeleton is <stdio.h>. You WILL need to add additional include files. For example, to use the **sqrt( )** you'll need to include <math.h>.

- You will create your own Makefile. Because you'll be accessing functions from the math library, you will need to link in the math library using -lm as seen below:

```
gcc -Wall -o p2 p2.c -lm
```

The **-Wall** option that I've added here will give you warnings about non-fatal errors such as unused variables.

In reflecting on your solution you may discover that the program is imperfect in several ways: no input validation, too cluttered with variable definitions, too much repetitive code and thus too easy to make errors in coding. As we move along through the semester, we will not only add functionality to the program, we'll use the imperfections we identify as motivation to incorporate additional features of C, apply our accumulating knowledge of best practices, and refine our implementation.

2. Sample Output

A sample run of the program can be seen below. The highlighted portion is the user input. Your output should be consistent with what you see here.

```
etkraem@joey12:./p2
Enter RGB values for your new color using the format (R,G,B): (124,66,18)
Closest color is initially black, with distance 134.35.
Distance to white is 318.65; closest color is still black.
Distance to red is 127.33; closest color is now red.
Distance to yellow is 202.99; closest color is still red.
Distance to blue is 217.50; closest color is still red.
Distance to orange is 141.15; closest color is still red.
Distance to green is 123.98; closest color is now green.
Distance to purple is 206.23; closest color is still green.
The closest match found in the palette was: green with (59, 171, 7)
```

3. Suggestions for how to proceed:

1. Create a Proj2 sub-directory in the Projects subdirectory of your Mod1 directory.
2. Copy p2_skeleton.c to this directory. Copy p2_skeleton.c to p2.c (or just rename it).
3. Create the Makefile.
4. Run “make” to be sure that the makefile is working. You’ll see many warnings about unused variables (you aren’t using any of them yet!). Remove the -Wall option from your compile command for now. Add it back in later after you have some working code.
5. Work through the “TODO” steps, adding the suggested code.
6. Verify the output as you work along – hand calculate the distance from the user’s color to black, for example. Does the value look right?
7. Add the code for white – is that distance correct? Does the closest value update (or not) as it should?
8. Add the other colors in one by one, compiling and testing along the way.
9. If you get a long list of compiler errors, focus on the first one or two, fix those and recompile, working along until compilation succeeds.
10. If the program does not get the right output, then try printing out intermediate values (the distance between the red components, for example) and confirming them against your hand calculations.
11. If the control flow of the program seems wonky, then try commenting out later chunks of the program, compiling and testing. If the part you have thus far is working, then move the start of the comment block down a few lines and test again … advancing the front until you locate the line or few lines of code that introduce the error .. then read those carefully.
12. Once you have it all working, be sure to test with a variety of values.
13. Submit and test via the Gradescope link.

4. Grading rubric:

___ / 10 Both p2.c and Makefile included in submitted code; no extraneous files

___ / 10 p2.c compiles cleanly using Makefile (with -Wall in compile command)

___ / 10 Header comments are complete

___ / 10 meaningful variable names

___ / 10 appropriate solicitation and collection of input for color name and values

___ / 20 correct computation of distance values

___ / 20 correct selection of closest color

___ / 10 appropriately formatted output

___ / 100

5. Notes on Collaboration

You are required to work individually on this assignment. You can ask questions and get help in piazza, in office hours (by appointment -- just email me and we will set it up!) and in/after class.